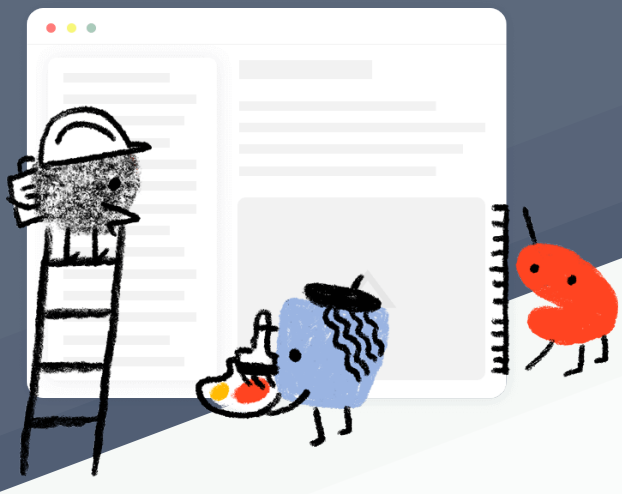
2021 Web Almanac

HTTP Archiveの年次報告書
ウェブの状態レポート



目次

導入

序章 iii

部 I. ページコンテンツ

章 1: CSS 1
章 2: JavaScript 63
章 3: マークアップ 97
章 4: Structured Data 131
章 5: メディア 161
章 6: WebAssembly 199
章 7: サードパーティ 223

部 II. ユーザー体験

章 8: SEO 255
章 9: アクセシビリティ 291
章 10: パフォーマンス 337
章 11: プライバシー 367
章 12: セキュリティ 401
章 13: モバイルウェブ 439
章 14: 能力 479
章 15: PWA 503

部 III. コンテンツ公開

章 16: CMS 535
章 17: Eコマース 571
章 18: Jamstack 615

部 IV. コンテンツ配信

章 19: Page Weight 645

章 20: リソースのヒント 659
章 21: CDN 683
章 22: 圧縮 703
章 23: キャッシング 717
章 24: HTTP 739

付属資料

方法論 771
貢献者 783

序章

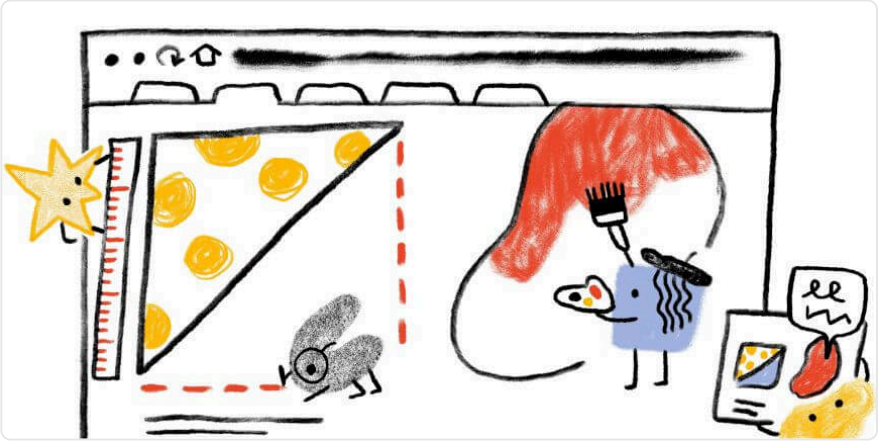
3年前、私はこう考えました。*自分のウェブサイトがどれだけよくできているかは、たくさんのツールで知ることができるが、ウェブ全体の状態を見るにはどこに行けばいいのだろう？*HTTP Archiveのデータセットが洗練されているのと同様に、それが与えてくれる答えは、私たちがそれに尋ねる質問と同じくらい有用である可能性があります。私はWeb開発者ですが、Web開発のすべての分野の専門家ではありませんし、誰もそうなることを期待していません。しかし、私たちは皆、それぞれの専門分野を持っています。多くの人が集まれば、ウェブの現状について適切な質問をすることができ、HTTP Archiveが本当に意味のある形でそれに答えることができるようになるのです。これがWeb Almanacの最初のアイデアでした。

今年で3回目を迎え、100人以上のウェブコミュニティの素晴らしい人々の努力によって実現されました。とくに、今回で3年連続のご寄付をいただいている方々をご紹介します。Barry Pollard、David Fox、Paul Calvano、Brian Kardell、Doug Sillars、Eric Portis、Thomas Steiner、Robin Marx、Alan KentとAbby Tsaiです。このプロジェクトに時間を割いてくれたすべての協力者に感謝しますが、とく初期から参加してくれたこの10人には、大きな恩義を感じています。

2021年版は、今回はじめて取り上げる2つの章を含む、全24章の充実したラインナップで構成されています。Structured DataとWebAssembly の2つです。これらの新しい章は、Web Almanacの範囲を拡大し、より多様なトピックについて読者ベースを教育し、さらに多くの専門グループに実用的なデータを提供するのに役立っています。最終的には、私たちの研究がウェブコミュニティによって真実の共有ソースとして活用され、エコシステムを有意義に改善できることを望んでいるからです。もしあなたがこのリソースを私たちと同じように価値あるものと感じてくださるなら、ウェブの現状に関心を持つ他の人たちと共有してくださるとうれしいです。一緒に、このデータを前向きな変化のための強制力として使っていきましょう。

— Rick Viscomi、Web Almanac編集長

部 I 章 1
CSS



Eric A. Meyer と Shuvam Manna によって書かれた。
Chris Lilley、 Jens Oliver Meiert、 Estelle Weyl、 Brian Kardell、 Adam Argyle と Lea Verou によってレビュ。
Rick Viscomi による分析。
Shaina Hantsis 編集。
Sakae Kotaro によって翻訳された。

序章

CSS（Cascading Style Sheets）は、ウェブ上のページを構築するための3本柱のひとつで、構造を定義するためのHTML、動作やインタラクションを指定するためのJavaScriptと合わせて、三位一体となっています。

2021年版のWeb Almanacでは、前回と比較して、CSSの使い方が「必要だと思うもの」「実際に制作現場で目にするもの」の領域でどのように異なるかをより深く理解できます。CSSの機能をより強固にすることや、CSSで`<div>`を中央に配置することへの挑戦がブログやカンファレンスでの発言、Twitterでの発言などで話題になっていましたが、ウェブ上のページでは、まったく相反する結果が示されていました。これはCSSが十分な年齢になったことで、奇抜なおもちゃに、夢中になるのではなく、安定した状態を維持することに重きを置くようになったという事実を裏付けています。

CSS-in-JSの採用は、クロールされた全ページの3%（昨年比1パーセントポイント増）にまで増加しましたが、最先端のHoudiniの機能は、まだほとんどがチュートリアルやサンプルギャラリーに限られています。レスポンシブ対応は、引き続きもっとも関心の高い優先事項の1つであり、`max-width`と`min-width`はトップのメディアクエリであり、「calc()」は幅を決定するためにもっともよく使用されるCSS関数です。

ユーザーがウェブに集まり続ける中、第二の故郷、ワークスペース、ガレージ、ウサギの穴のような場所であるインターネットを、私たちがどのように描いてきたのかを知るためのデータに飛び込んでみましょう。

使用方法

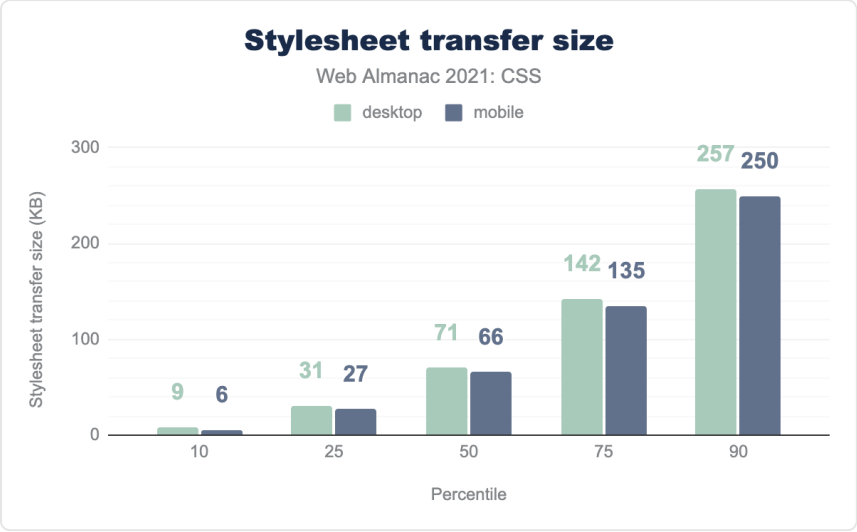


図1.1. ページごとのスタイルシート転送サイズの分布

CSSはほとんどのページでもっとも重いコンポーネントではありませんが、ウェブの他の部分と同様に、年々そのサイズは増加し続けています。ウェブページの中央値では約7KBのCSSがロードされ、上限値では平均4分のMB強となっています。2020年と比較すると、CSSの総重量の中央値は約7.9%増加し、90パーセンタイルは7%弱増加していますが、すべてのパーセンタイルでモバイルCSSがデスクトップCSSよりも少し小さいという昨年のパターンは維持されています。

すべてのページがこのような制約を受けたわけではありません。最大のCSSウエイトを持つページでは、64,628KBの負荷がかかりました。それに比べて、モバイル用の最大のCSSウェイトはわずか17,823KBと、非常にスリムな印象を受けます。

2020年と同様に、ページの重さはプリプロセッサーによって大きく左右されないことがわかりました。デスクトップページの17%、モバイルページの16.5%がソースマップを含んでおり、昨年の15%からわずかに増加しました。ソースマップを含むCSSのシェアが一貫しているのは、ソースマップのシェアが、ソースマップの採用よりもビルドツールの使用によるものであることを示しているように思われます。

どのようなソースマップが使われているかについては、昨年とほぼ同様の結果となりました。

ソースマップの種類	2020	2021
CSSファイル	45%	45%
Sass	34%	37%
Less	21%	17%

図1.2. 2021年と2020年のソースマップの種類。

これは、SassがLessに比べて優位に立ち続けている証拠とも言えますが、その変化は非常に小さいため、統計的にもそうでなくても、有意であると言うのは難しいでしょう。いつものように、時間が解決してくれるでしょう。

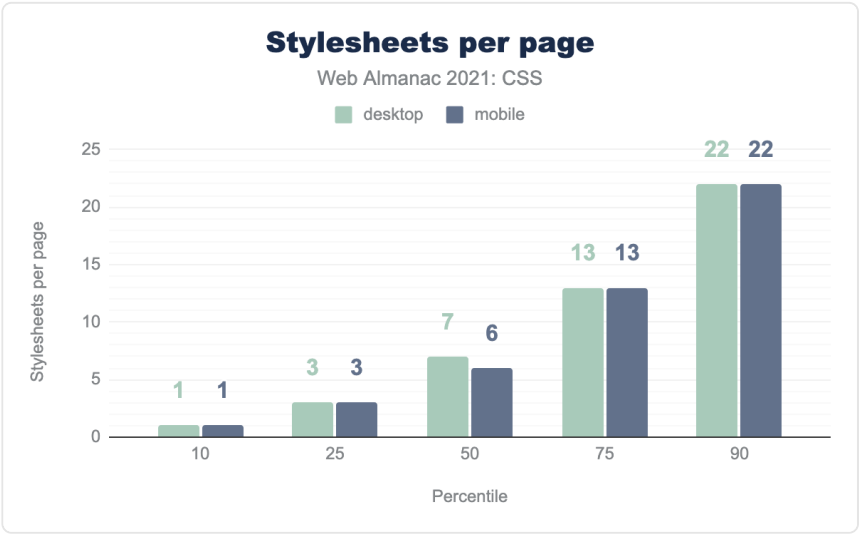


図1.3. ページあたりのスタイルシート数の分布。

埋め込み型、外部型を問わず、1ページあたりのスタイルシート数の平均は、昨年よりもわ

ずかに増加しています。50パーセンタイルから90パーセンタイルまではそれぞれ1つずつ増加しましたが、10パーセンタイルと25パーセンタイルは変化がありませんでした。

2,368

図1.4. ページで読み込まれる外部スタイルシートの最大数です。

信じられないことに、外部スタイルシートの数がもっとも多い今年の記録は、昨年の記録をほぼ2倍上回っています。2020年の1,379に対して2,368です。これを行った人は誰でも、いくつかのファイルを組み合わせてサーバーを休ませてください。

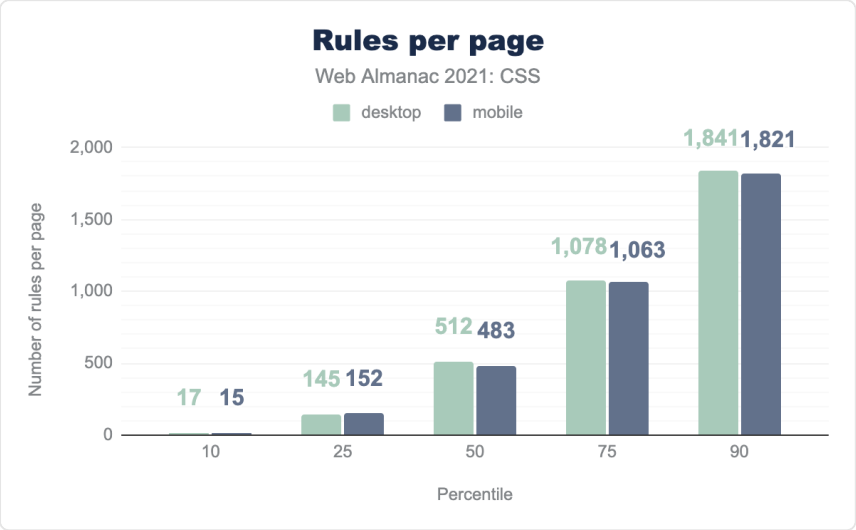


図1.5. ページごとのスタイルルールの総数の分布。

スタイルシートの数は重要ですが、実際のスタイルルールの数はどうでしょうか？　昨年と比較すると、下位のパーセンタイルは少し増加していますが、上位のパーセンタイルはほとんど変化していません。2021年と2020年で異なるのは、ほぼすべてのパーセンタイルにおいて、デスクトップページの方がモバイルページよりも平均してルール数が多いということです。

セレクターとカスケード

カスケードを理解することは、CSSを扱う上で非常に重要なことです。とくに、ある要素に

書いたスタイルがまったく機能していない場合には、なおさらです。

CSSには、クラス、ID、そしてスタイルの重複を避けるために重要なカスケードを使用するなど、ページにスタイルを適用するためのさまざまな方法があります。

クラス名

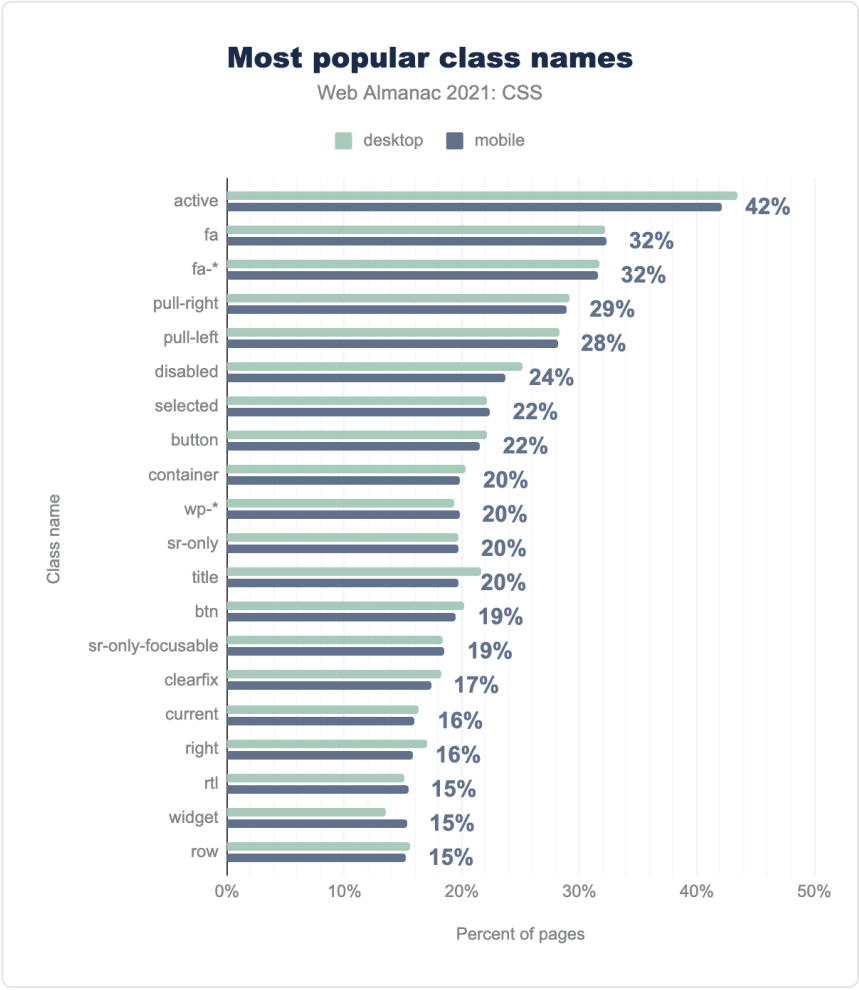


図1.6. もっとも人気のあるクラス名。

昨年同様、もっとも人気のあるクラス名は `active` で、`fa`、`fa-*`（Font Awesomeのプレフィックス）、`wp-*`（WordPressのプレフィックス）のクラス名が非常に強い印象を与

えています。`selected`と`disabled`は昨年と比べて順位が入れ替わっていますが、もっとも心強い変化は`clearfix`が5%減少したことで、フロートベースのレイアウトが衰退し続けていることを示しています。

また、Bootstrapのアクセシビリティ機能である`sr-only-focusable`が配置されているのも心強いですね。これはBootstrapのアクセシビリティ機能で、要素を画面外に配置しても、スクリーンリーダーがアクセスできるようにするものです。

ID

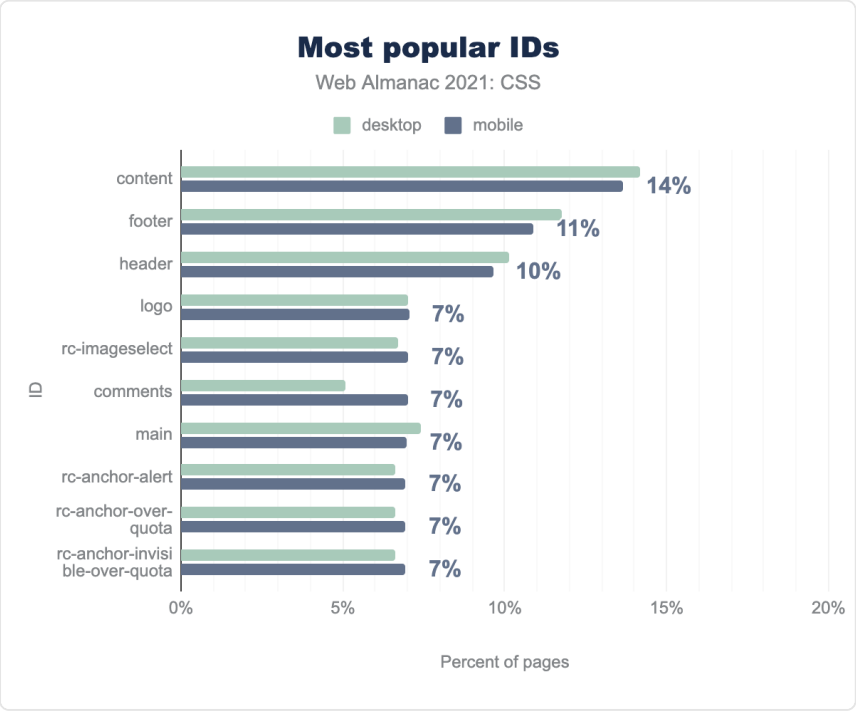


図1.7. もっとも人気のあるID名。

ページは引き続きIDを使用しており、その割合は2020年に見られたものとほぼ同じです。人気のあるID名のリストも一貫しています。”content”が約14%でトップに立ち、”footer”と“header”がそれに続きます。後者の2つのIDは、昨年に比べて約1%減少していますが、これだけで決定的なことは言えず、開発者は可能な限り対応するHTML要素の`<header>`と`<footer>`に置き換えるべきでしょう。

`rc-`で始まるIDは、GoogleのreCAPTCHAシステムの一部であり、そのほとんどのバージョ

ンは、さまざまな方法でアクセスできません[1]。

属性セレクター

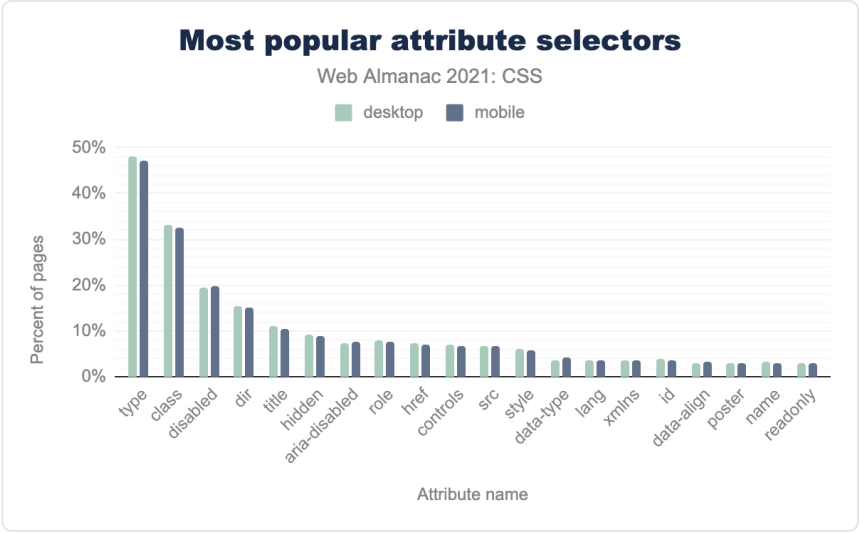


図1.8. もっともポピュラーな属性セレクターです。

もっとも一般的な属性セレクターは引き続き `type` で、チェックボックスやラジオボタン、テキスト入力などのフォームコントロールを選択する際に使用されることが多いです。

擬似的なクラスと要素

擬似クラス、擬似要素ともに、順位や分布は2020年のWeb Almanacから大きく変わっていませんでした。いくつかの順位は変わりましたが、全体的にはほとんど変化がないようです。これが常識の定着を示しているのか、デザイナーの関心事のスナップショットを示しているのか、あるいは単に分析の性質を示しているのかは議論の余地があります。

1. *https://www.w3.org/TR/turingtest/#the-google-recaptcha*

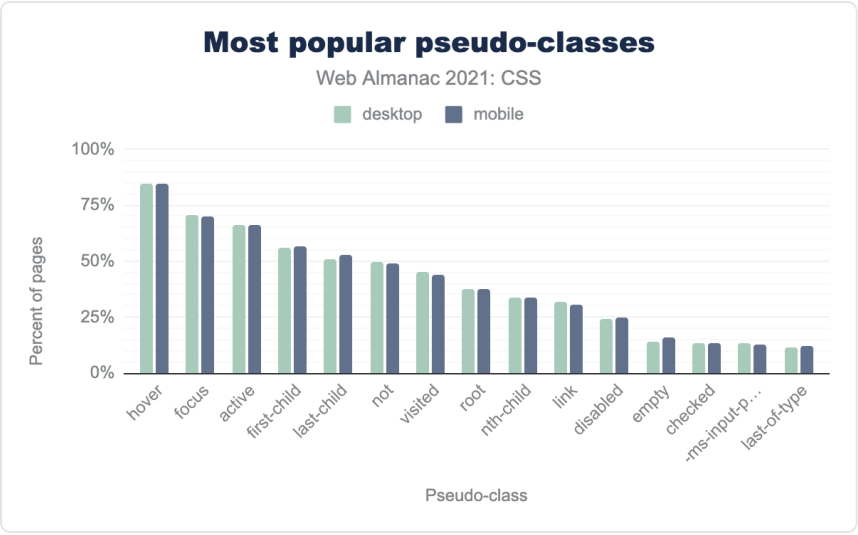


図1.9. もっともポピュラーな疑似クラスです。

2020年と同様に、ユーザーアクション擬似クラスの `:hover`, `:focus`, `:active` がトップ3を占め、いずれも全ページの3分の2以上に登場しています。構造的な擬似クラスも数多く登場しましたが、もっとも興味深い変化は、否定の擬似クラスである `:not()` は `:visited` よりも人気を博し、ページの50%のシェアを獲得したことです。

今年、とくにチェックしたのは、`:focus-visible` という、フォーカスされた要素をユーザーの期待に合うようにスタイルする方法の使用です。この機能は、2020年にChromiumへ搭載され、2021年1月にはFirefoxに搭載され、（発表時点では）Safari 15で実験的なフラグを立てて利用できるようになっています。最近の導入状況を反映してか、分析したページの1%未満にしか登場しませんでした。この数字が今後数年間で変化するかどうかが注目されます。

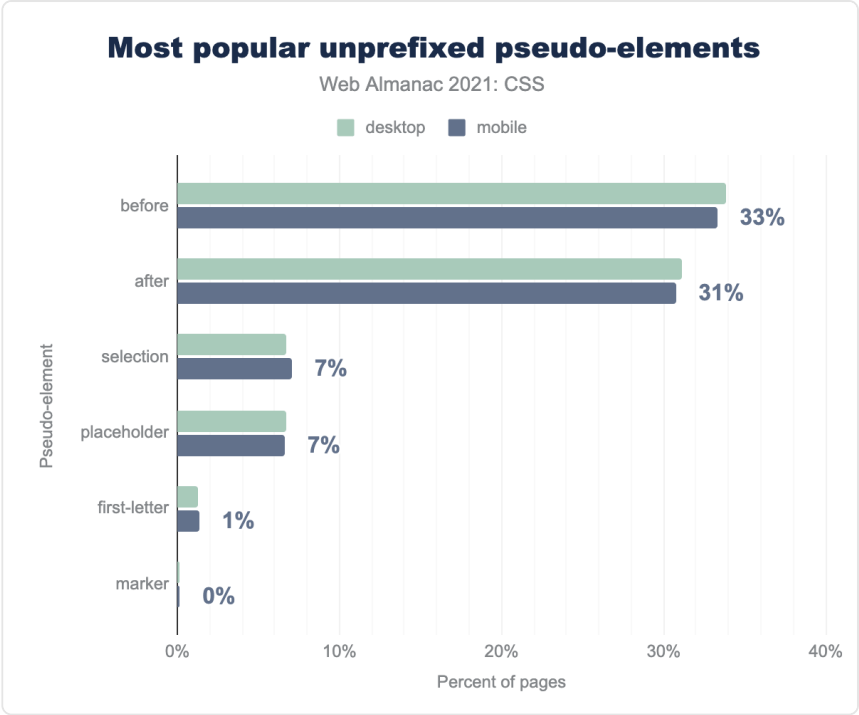


図1.10. もっともポピュラーな接頭辞なしの疑似要素です。

現在使用されている擬似要素のほとんどは、特定のインタフェースコンポーネント、chromeの一部、またはハイライトされたテキストなどを選択するためのブラウザ固有の方法です。これらを除外すると、`::first-letter`が使用されているページは非常に少ないことがわかりますが、それでも`::first-line`よりは多くのページで使用されていることがわかります。`::marker`は、箇条書きやカウンターのようなリストアイテムのマーカーを順序付きリストとして選択する方法で、ページシェアは1%にも満たないが、リストに入っています。ここで注意すべきなのは、`::marker`のクロスブラウザサポートは、比較的新しい[2]ということです（2020年10月）。今後数年間で利用者が増えるかどうかが注目されます。

2. *https://caniuse.com/css-marker-pseudo*

`!important`

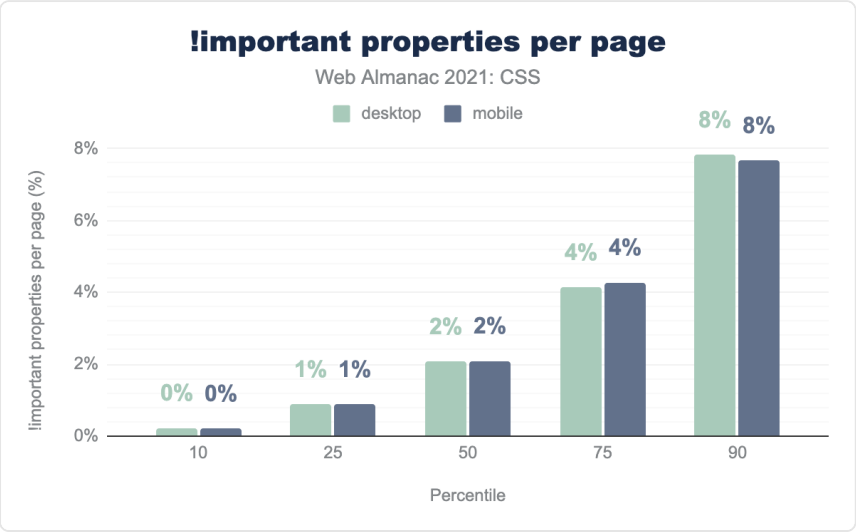


図1.11. `!important` を使ったページルールの割合の分布。

その古い戦斧 `!important` は、2020年のWeb Almanacと比較して、マークされたルールのシェアがほとんど変わらず、ウェブ上で持ちこたえています。

`!important` と書かれたルールが17,990個もあるモバイルページを発見しました。この数字は、もっとも重要なデスクトップページの17,648個のルールを上回っています。これがスクリプトやプリプロセッサのミスによるものであることを願ってやみません。

`!important` が何に適用されるかというと、昨年と同様に `display` であり、その他の項目は2020年と同じ順番であるが、最後の項目を除いて `position` が `float` を上回っています。

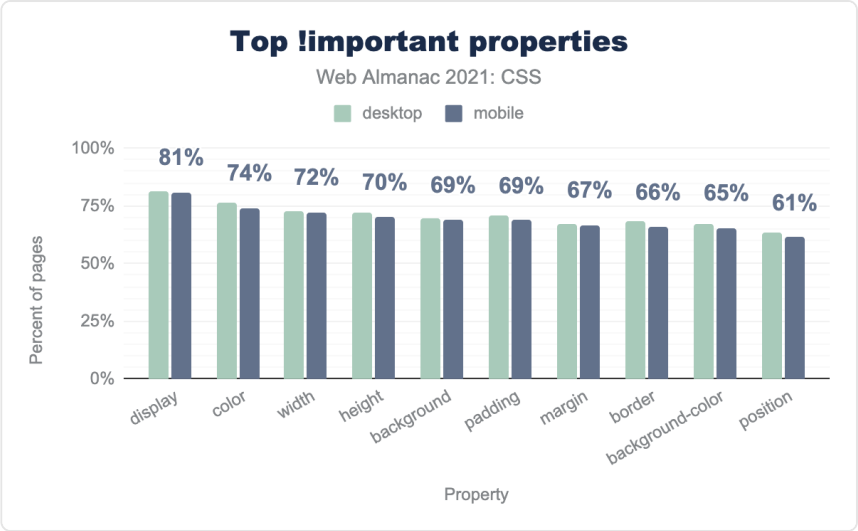


図1.12. `!important` の対象となるもっとも人気のあるプロパティです。

セレクターの特異性

パーセンタイル	デスクトップ	モバイル
10	0,1,0	0,1,0
25	0,2,0	0,1,3 (up 0,0,1)
50	0,2,0	0,2,0
75	0,2,0	0,2,0
90	0,3,0	0,3,0

図1.13. ページごとのセレクターの特異性の中央値の分布。

多くのCSSメソドロジーでは、作者が単一のクラスに限定することを推奨しています。これは、すべてのセレクターの特異性を、より管理しやすい単一のレイヤーに押し込めるためです。たとえば、BEMの手法[3]は、全ページの34%に見られました。セレクターの特異性の中央値の10パーセンタイルは、デスクトップとモバイルの両方の特異性の平均値が（0,1,0）

3. https://en.bem.info/methodology/css/

であることから、この種の考え方のさらなる証拠を示しています。これは昨年の調査結果と同様で、ほぼすべての中央値が一致しています。ただし、モバイルの25パーセンタイルは若干上昇しています。

値と単位

CSSには、値や単位を指定する方法が複数用意されており、設定された長さや、グローバルキーワードに基づく計算などがあります。

長さ

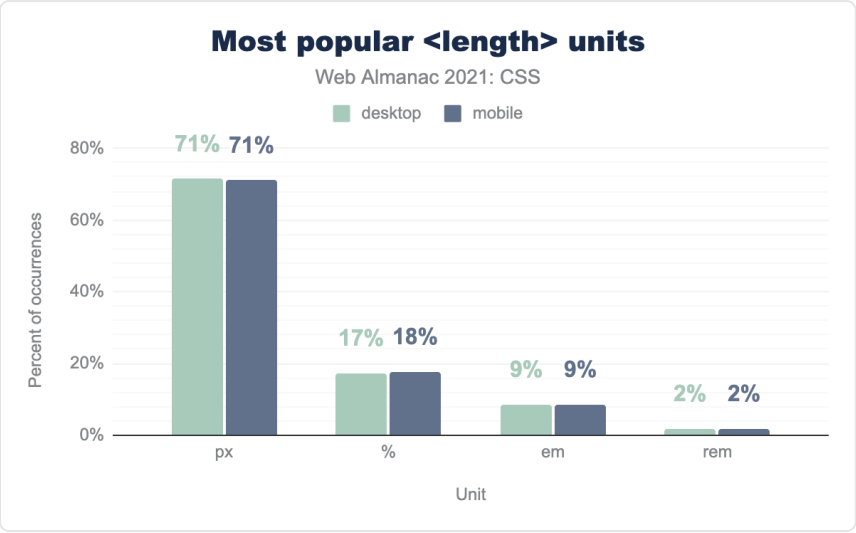


図1.14. もっともポピュラーな長さの単位です。

ピクセルの長さをどう思うかは別として、ピクセルは圧倒的に人気のある長さ単位で、全ページの約71%に登場しています。2位の「パーセンテージ」は、「ピクセル」に圧倒的な差をつけています。

プロパティ	px	<number>	em	%	rem	pt
font-size	(▼1%) 69%	2%	(▼1%) 16%	(▼1%) 5%	(▲1%) 5%	2%
line-height	(▼5%) 49%	(▲3%) 34%	(▲1%) 14%	(▼1%) 2%	(▲1%) 1%	0%
border-radius	65%	(▼1%) 20%	3%	10%	(▲2%) 2%	0%
border	71%	(▲1%) 28%	2%	0%	0%	0%
text-indent	(▼1%) 31%	(▲1%) 52%	8%	(▼1%) 8%	0%	0%
gap	(▼8%) 13%	(▲2%) 18%	(▼1%) 0%	0%	(▲7%) 69%	0%
vertical-align	(▼11%) 18%	12%	(▲11%) 66%	4%	0%	0%
grid-gap	(▲3%) 66%	(▼1%) 10%	9%	(▼1%) 0%	(▼2%) 14%	0%
padding-inline-start	(▼7%) 26%	(▲2%) 7%	(▲4%) 66%	0%	0%	0%
mask-position	0%	0%	(▼1%) 49%	(▲1%) 51%	0%	0%
margin-inline-start	(▼7%) 31%	(▲5%) 51%	(▲1%) 15%	(▲2%) 2%	1%	0%
margin-block-end	(▲1%) 5%	(▲7%) 38%	(▼9%) 56%	0%	(▲1%) 1%	0%

図1.15. プロパティごとの長さタイプの分布

面白くなるのは、さまざまな長さの単位がどのように使われているかを正確に分析することです。一例を挙げると、`line-height`でもっともよく使われる長さの単位はピクセルで、次に`<number>`の値（単位のないゼロ長さの値のすべてのインスタンスを含む）が続きます。また、`vertical-align`や`padding-inline-start`では、`em`がもっとも一般的な長さの単位です。

この表の数字の横の括弧内の正負は、2020年の結果からの変化を示しています。ほとんどの物件で、他の長さの単位に比べて画素数が少なくなっていますが、2つの例外があります。もっとも大きな変更点は`vertical-align`で、供給される値が`baseline`のようなキーワードではなくlengthの場合、選択される単位がpixelsから`em`に11ポイント変更されました。

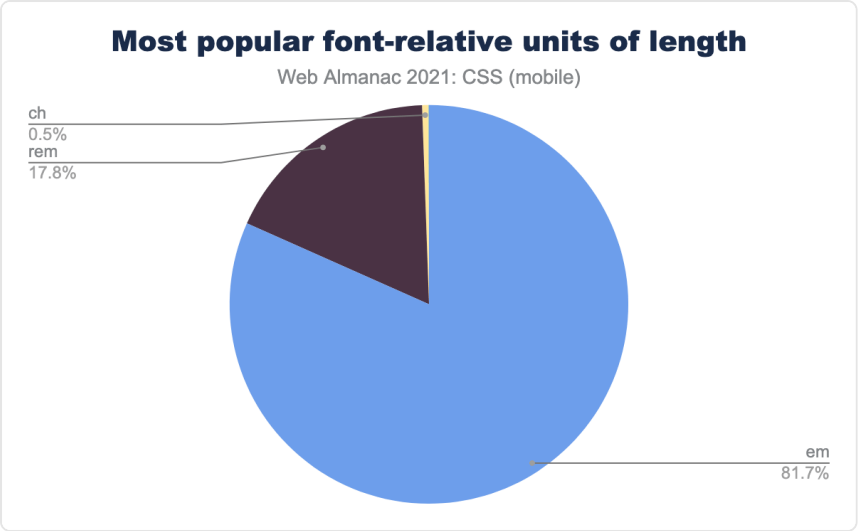


図1.16. もっともポピュラーなフォント、相対的な長さの単位です。

フォントのサイジングに関しては、`em` が `rem` に対して圧倒的な優位性を保っていますが、変化の兆しもあります。2020年から2021年にかけて、`em` から `rem` へ7ポイントの変動がありました。

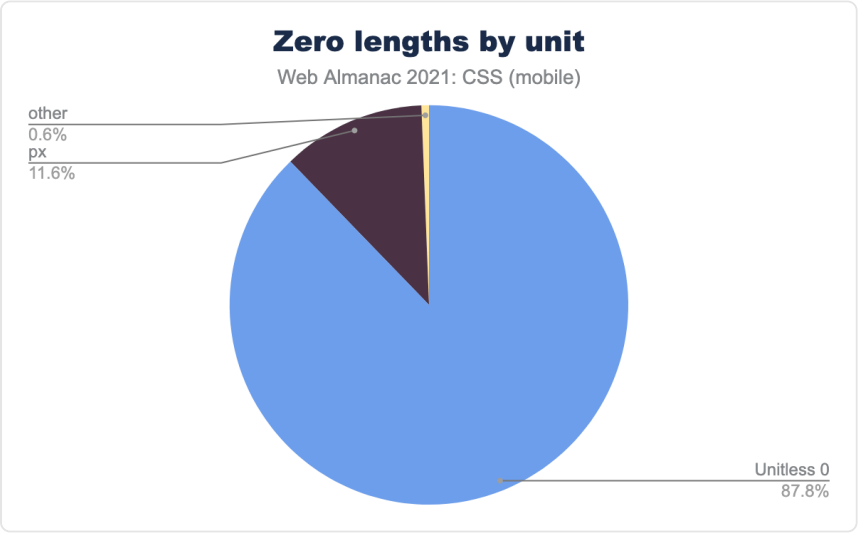


図1.17. ゼロ長の値に使用される単位（またはその欠如）。

裸の `<number>` 単位を許可するプロパティはいくつかありますが（例：`line-height`）、`<length>` 値には、長さがゼロであれば単位を必要としないという特殊なケースがあります。すべての長さゼロの値を調べたところ、ほぼ88%が単位を省略していました。ゼロの長さで単位を含むものは、ほぼすべてがピクセル（`0px`）を使用していました。ゼロの長さには単位が必要なく、単位を含めることはかなり無意味なので、これは嬉しい結果となりました。今後、単位のないゼロの長さのシェアが拡大することを期待しています。

計算方法

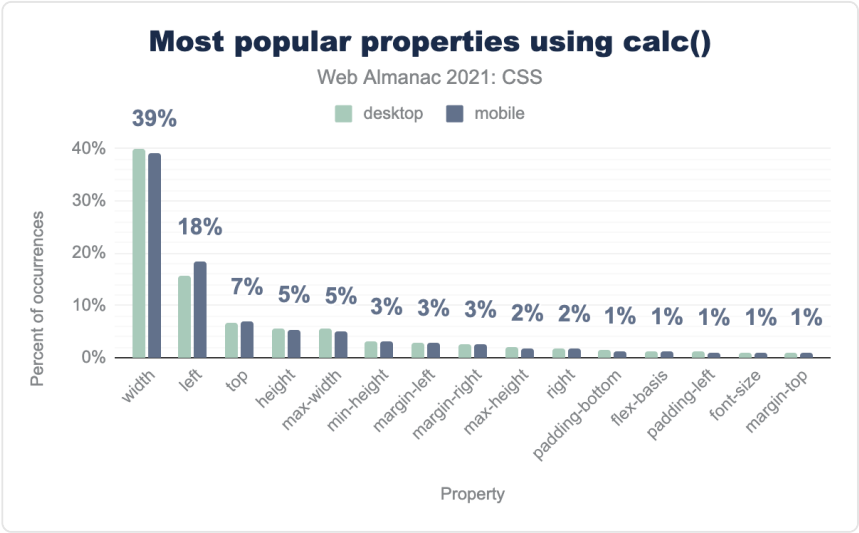


図1.18. `calc()` 関数を使った代表的なプロパティです。

例年通り、`calc()` のもっとも一般的な使い方は幅の設定ですが、`width` に占める `calc()` の値の割合は、2020年と比較して20ポイントも減少しました。これは、`calc()` が `width` に使われることが減ったのではなく、他のプロパティに使われることが増えたことを反映していると考えられます。

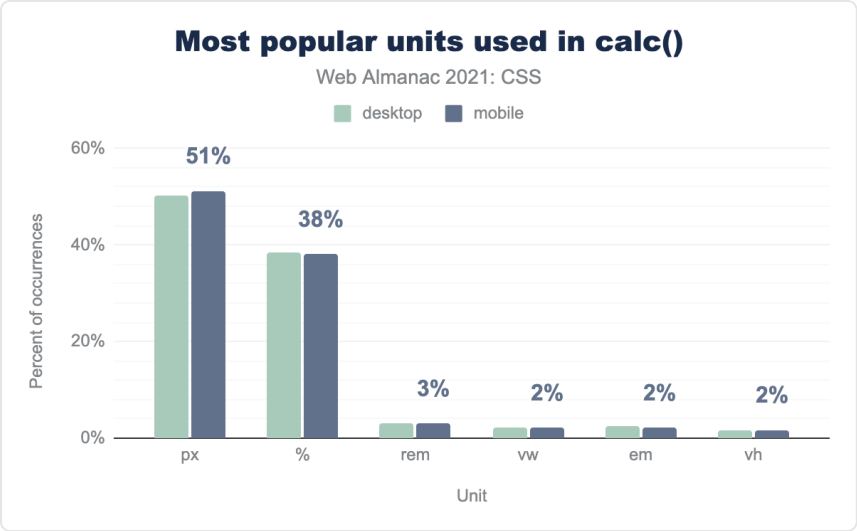


図1.19. 関数 `calc()` で使われるもっとも一般的な長さの単位です。

ピクセル単位は計算上の使用率ではまったく変化がなかったが、パーセントは他の単位のロングテールと比較して少し負けており、2020年から4ポイント低下しました。

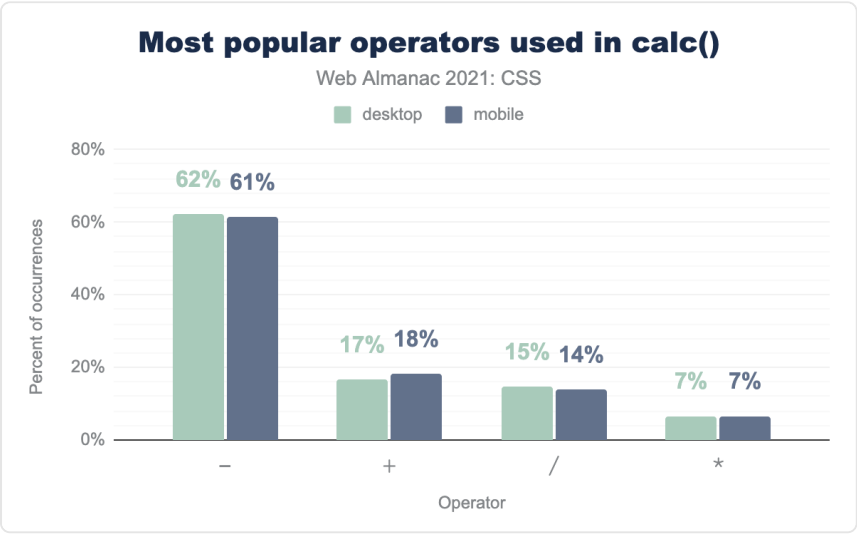


図1.20. 関数 `calc()` で使われる代表的な演算子です。

昨年同様、演算子は引き算が断トツで、シェアはほとんど変化しませんでした。2位と3位の

変化は大きく、足し算が割り算を上回って6ポイント増加し、割り算は同程度減少しました。

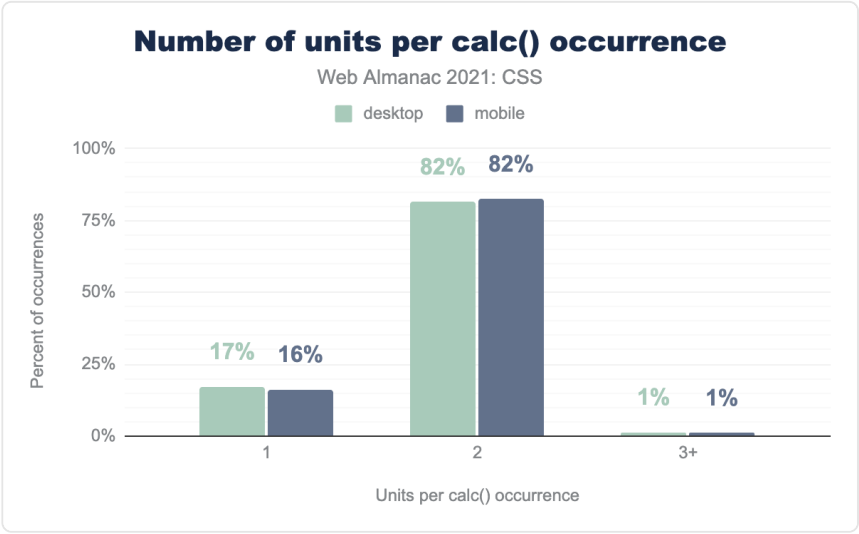


図1.21. 関数 `calc()` で使用されるユニークなユニットの数です。

`calc()` の値は比較的シンプルで、パーセント値の計算結果からピクセルを引くなど、2つの異なる単位を使った計算が圧倒的に多いです。全 `calc()` 式の99%は、1種類または2種類の単位を使用しています。

グローバルキーワード

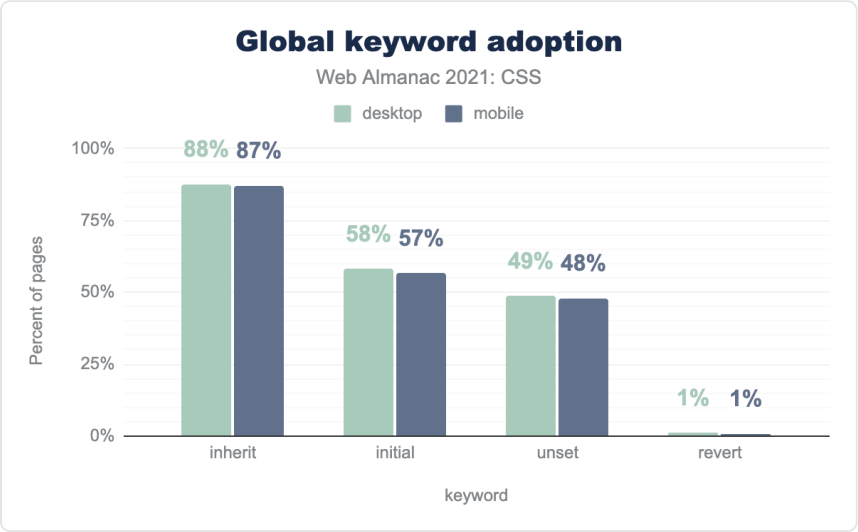


図1.22. グローバルキーワードの値の使い方

2020年版Web Almanacと比較して、`initial`などのグローバルキーワードの使用率が大幅に上昇しました。`inherit`は2〜3ポイントしか上昇していませんが、`initial`は約8ポイント、`unset`は約10ポイント上昇しています。また、`revert`も1ポイント上昇しました。

カラー

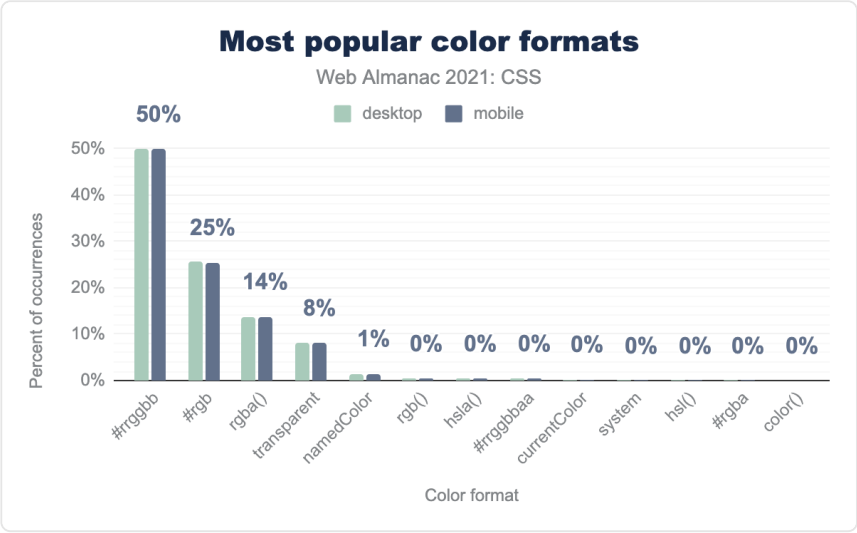


図1.23. もっとも一般的なカラーバリューのフォーマットです。

さまざまな色の値の種類があるにもかかわらず、Netscape 1.1の時代から使われている `#RRGGBB` という構文が、色の宣言の半分に使われています。2位はCSSの革新的な `#RGB` 略記法で、カラーバリューの4分の1を占めています。言い換えれば、すべての色の値の75%は、16進数のRGB構文で表現されているということです。第3位の `rgba()` というフォーマットは、作者が古典的な16進法のフォーマットを超える理由を示しています：アルファ値にアクセスするためです。(実際、どちらのシェアも小さいですが、 `hsla()` は `hsl()` よりも人気があり、 `rgba()` が普通の `rgb()` よりもはるかに一般的であるのと同じです。)

たとえば `rgba(0, 0, 0, 1)` のように、これまで機能的な構文の中で値にカンマが使われていたカラーフォーマットでは、著者はカンマを削除し、スラッシュでカラーとアルファを分けることができるようになりました（例： `rgb(0 0 / 1)` ）。2020年以降、このカンマなしの構文の使用率は倍増し、機能的な色の構文全体の0.12%から0.25%になりました。

キーワード	デスクトップ	モバイル
transparent	82.24%	82.93%
white	7.97%	7.59%
black	2.44%	2.29%
red	2.23%	2.17%
currentColor	1.94%	2.03%
gray	0.68%	0.64%
silver	0.56%	0.55%
grey	0.39%	0.37%
green	0.32%	0.31%
blue	0.15%	0.12%
whitesmoke	0.12%	0.11%
orange	0.12%	0.10%
lightgray	0.08%	0.08%
lightgrey	0.07%	0.07%
yellow	0.07%	0.06%
gold	0.04%	0.03%
magenta	0.03%	0.03%
Background	0.02%	0.03%
Highlight	0.02%	0.03%
pink	0.03%	0.03%

図1.24. もっとも人気のある色の名前キーワード値です。

色の名前領域では、依然として`transparent`が圧倒的な人気を誇っており、色の名前キーワード全体の約82%を占めています。おなじみの`white`、`black`、`red`は約12%で、`currentColor`は2020年と比べて0.5%の増加で5位となっています。

昨年のWeb Almanacでは「`Canvas`や`ThreeDDarkShadow`など、かつては非推奨であったが、現在は部分的に非推奨ではなくなったシステムカラー」が、かろうじて使用されているという記述がありました。これは今でも正しいのですが、奇妙なことに、トップ20にはこのような値が1つ（`Highlight`）ではなく2つあります。とはいえ、どちらもごくわずかなページ数の領域での出来事なので、このような変化は目立たないでしょう。



図1.25. display-p3の色のうち、sRGB空間外にある色の割合。

display-p3色空間の使用率は、2020年に発見された時と同様に、消え入るように少ないままです。これはおそらく、この記事を書いている時点では、Safari（デスクトップとモバイルの両方）でのみサポートされているためです。デスクトップは90ページ、モバイルは105ページと約3倍になりました。モバイルでdisplay-p3を使って表現された色の79%がsRGB色空間では表現できない色だったからです。`color()`関数がブラウザで広くサポートされるようになるまでは、Webは画面が実際に表示できる色の約3分の2を許容するsRGBに留まります。

イメージ

「百聞は一見にしかず」と言いますが、バイト換算すると一桁も、二桁も高いことが、多いのが画像です。JavaScriptで画像を埋め込んだり、HTMLで画像を埋め込んだりと、さまざまなアプローチがありますが、ここではCSSで読み込んだ画像がどのように使われているかを見てみました。

CSSにおける画像のフォーマット

まず、探した画像フォーマットの内訳と、それぞれのフォーマットの出現頻度を紹介します。

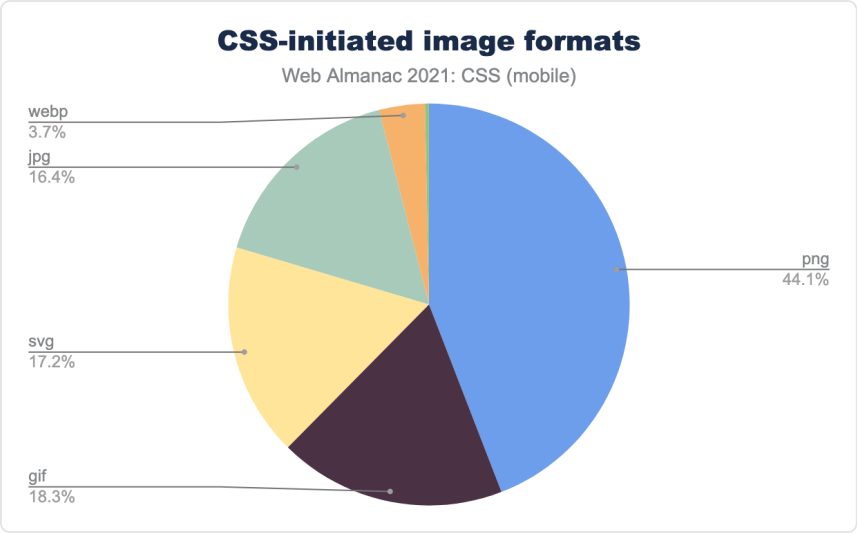


図1.26. CSSで読み込んだ外部画像のフォーマット分布。

PNGが圧倒的な人気を誇り、GIF、SVG、JPGが驚くほど僅差で続いています。かなり新しいWebPフォーマットは、CSSで読み込まれる画像の3.7%しか占めておらず、一番上の小さなスライスは、認識されない値とICOフォーマットに対応しています。

また、アニメーションの有無については確認していません。

なお、今回の分析では、CSSで読み込まれる画像のみを対象としており、HTMLで何が読み込まれているかは確認していません。したがって以下の結果は、Webページの重さ、あるいはCSSの重さ、重くないことを示す指標とはなりません。あくまでも、CSSで読み込まれた画像がページの総重量にどれだけ貢献しているかを示すものです。

CSS内の画像数

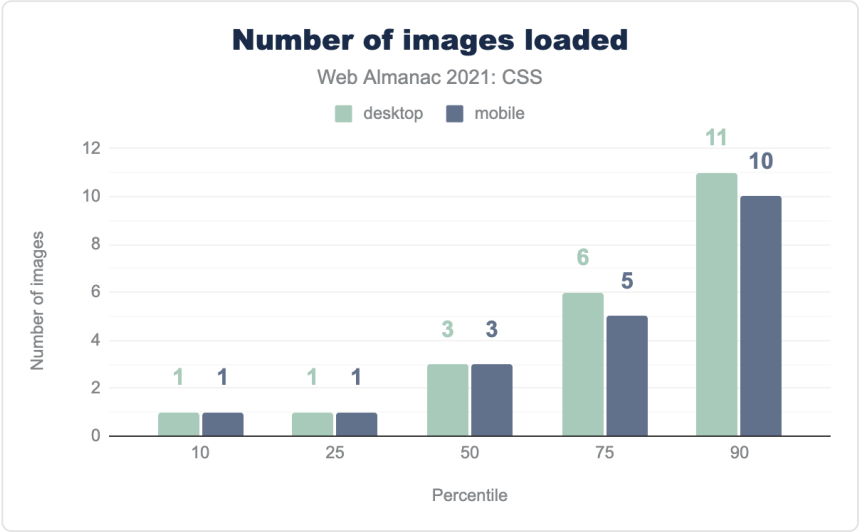


図1.27. CSSで読み込んだ外部画像数の分布。

下位2パーセンタイルはそれぞれ1枚、90パーセンタイルでも10枚程度と、すべての画像タイプにおいて、ほとんどのCSSでは画像の読み込み量は多くないことがわかりました。

6,089

図1.28. ページのCSSで読み込まれる外部画像の最大数。

デスクトップ版のCSSで6,088枚のPNG画像を読み込んでいるサイトがありました。モバイル版では実際に画像が追加され、6,089枚のPNGになっていました。効率化のために、すべての画像が小さく、カラーインデックス化されていることを願っています。

CSSにおける画像の重さ

画像の数もさることながら、その重さも重要です。たとえば、10MBの背景を1枚読み込むのと、100KBの画像を10枚読み込むのとでは、サーバーでの圧縮を考慮しても劣ります。

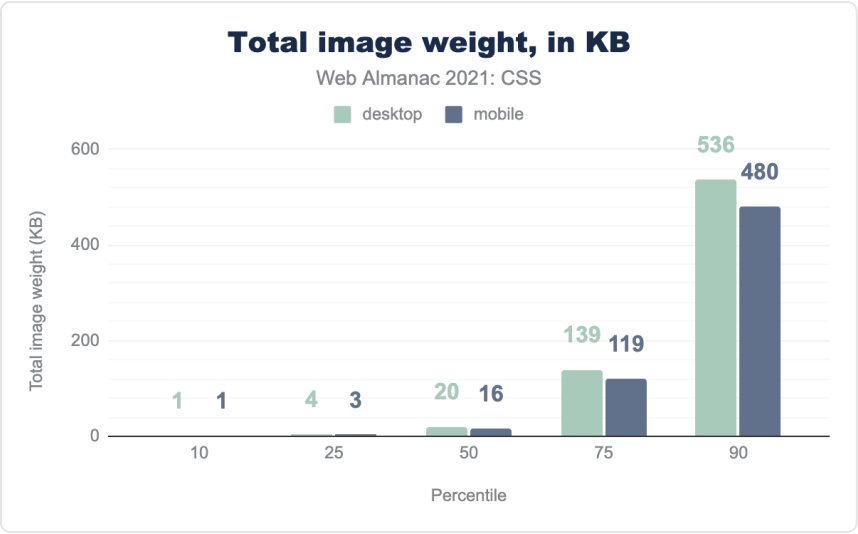


図1.29. CSSで読み込まれた外部画像の総重量（KB）の分布。

中央のページのCSSによる画像の読み込みは、合計16KBほどでした。また、CSSによるモバイルの画像読み込み量は、デスクトップに比べて全体的に一貫してやや少なめで、CSS開発者がモバイルのコンテキストの制限を多少なりとも念頭に置いていることがうかがえます。

314,386

図1.30. CSSで読み込まれたイメージのもっとも重い総量をKBで表しています。

たまにはね。CSSで読み込まれた画像の総重量が314,386.1KBと、GBの3分の1という膨大な量になっているページがありました。

パーセンタイル	JPG	PNG	GIF	(その他)	SVG	WebP
10	4.5	0.7	0.5	0.3	0.4	1.7
25	28.2	2.2	1.7	0.3	0.6	14.2
50	114.3	7.0	3.7	0.3	1.7	39.6
75	350.7	36.4	8.3	48.1	5.4	133.9
90	889.3	173.6	13.0	229.2	20.0	361.8

図1.31. モバイルページでCSS経由で読み込まれた外部画像の総重量（KB）の画像フォーマット別分布。

画像の重さをフォーマット別に分類してみると、興味深いことがわかりました。90パーセンタイルでは、GIF画像がSVGファイルよりも平均的に軽いのです。

また、驚くことではないかもしれませんが、もっとも重い画像フォーマットがJPGであることも興味深いことでした。これはおそらく、ホームページのトップなどでよく見かける大きな写真にはJPGが好まれ、圧縮やその他の最適化トリックを使っても、すべてのピクセルが加算されてしまうからでしょう。

グラデーション

プロパティ	デスクトップ	モバイル
`background`	62%	62%
`background-image`	62%	61%
`-webkit-mask-image`	5%	5%
`--*`	1%	1%
`mask-image`	1%	1%
`border-image`	1%	1%

図1.32. グラデーション画像の値が与えられたプロパティの割合。

CSSグラデーションを使用しているページの割合は、昨年とほぼ同じでした。デスクトップページの77%、モバイルページの76%でした。しかし、グラデーションが使用されているプロパティには変化が見られ、`background`と`background-image`が圧倒的な人気を誇っ

ていましたが、グラデーションの約62%がこのプロパティに割り当てられていました。

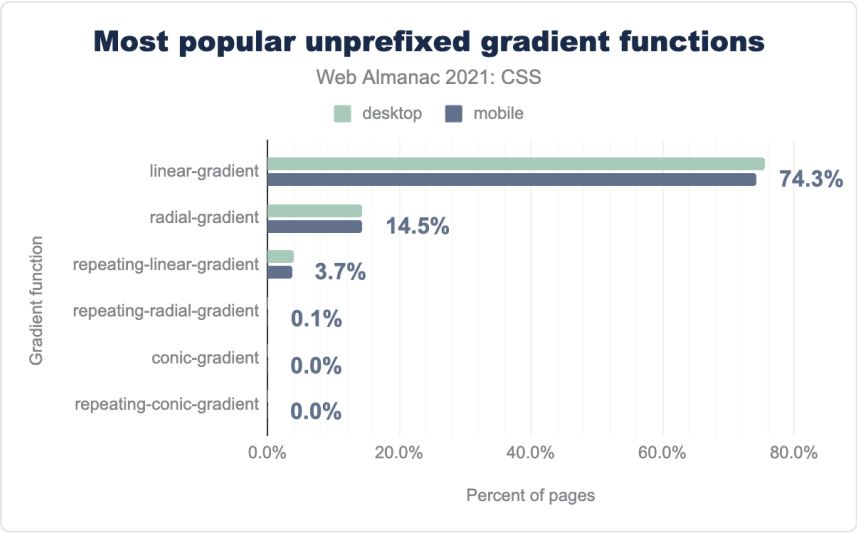


図1.33. グラデーション画像の値の中でももっともポピュラーなタイプです。

線形グラデーションの人気は引き続き高く、2020年版Web Almanac[4]で見られた放射状グラデーションに対する5対1のリードを維持しています。

グラデーションの接頭辞付きバージョン（例：`-webkit-linear-gradient`）を含めると、結果的には昨年のグラフと同じになりました。

その他、グラデーションの値を分析してわかったことがあります。

- グラデーションのカラーストップ数の中央値は、4ストップの90パーセンタイルを除いて、わずか2ストップです。
- ハードカラーストップ（2つのカラーストップを同じ位置に配置したグラデーション）は、グラデーション全体の半分強に存在している。
- カラーストップ補間（別名「ミッドポイント」）は、すべてのグラデーションインスタンスの21%で使用されました。

4. *https://almanac.httparchive.org/ja/2020/css*



図1.34. もっとも多くのカラーストップを持つ線形グラデーションです。

また、グラデーションの複雑さの上限でも、劇的な減少が見られました。昨年、もっとも多くのカラーストップを使用したグラデーションの数は646ストップでした。今年の受賞者のカラーストップ数はわずか81個でした。

レイアウト

ウェブ上のレイアウトにテーブルを使用していた時代からFlexbox、Grid、Multicolumn、そしてfloat、positioning、さらにはCSSテーブルプロパティなど、さまざまな選択肢が用意されている時代になりました。どのようなプロパティと値の組み合わせがあるのか、スタイルシートを簡単に検索してみたところ、次のような数値が出てきました。

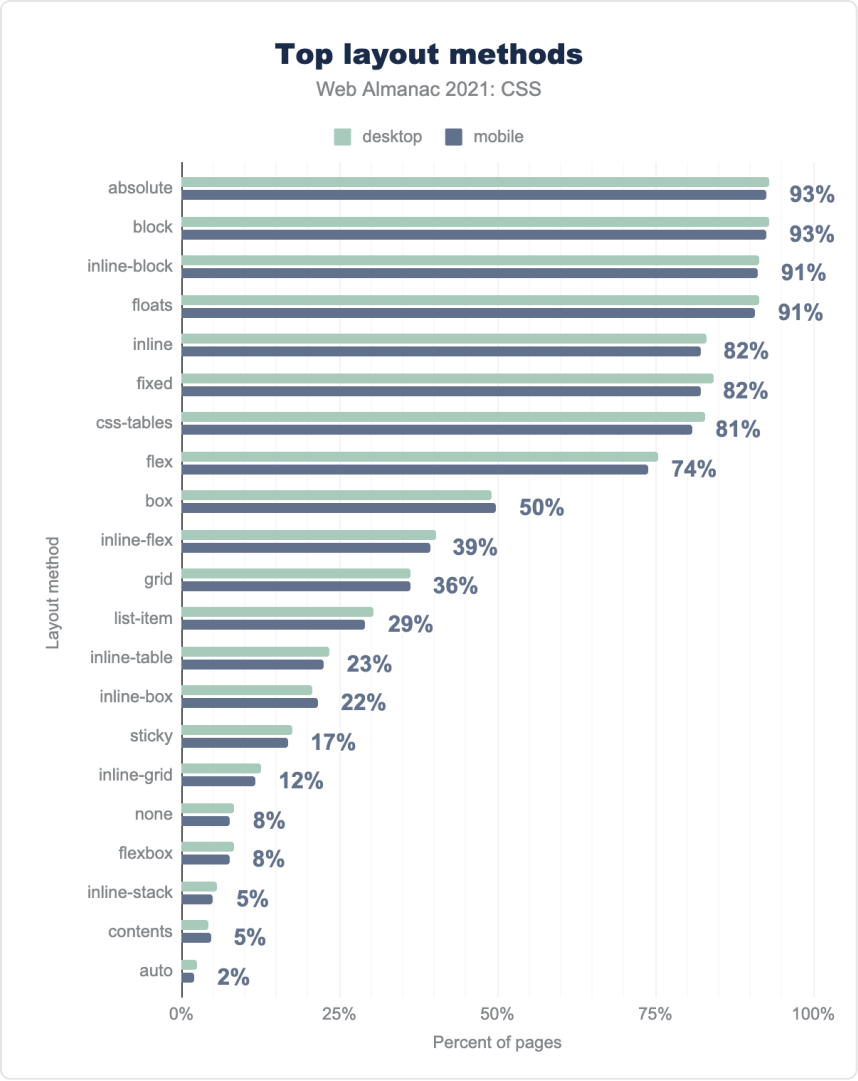


図1.35. もっとも一般的に宣言されているレイアウトタイプです。

分析したページの93%が絶対配置を使ってレイアウトされていると主張しているわけではありません。分析したページの93%が絶対配置でレイアウトされていると主張しているわけではありません！　むしろ、このチャートが示しているのは、分析したページの93%のスタイルに`position: absolute`が登場しているということです。同様に、`display: grid`が36%のページのスタイルに登場しているかもしれませんが、それは全ページの37%がGridページであることを意味するものではなく、単にスタイルシートのどこかにこの組み合わせが

登場しているということです。

このセクションでは、プロパティ値の組み合わせだけでなく、ページでの実際の使用状況の証拠を見て、より詳細な分析をしています。

フレックスボックスとグリッドの採用

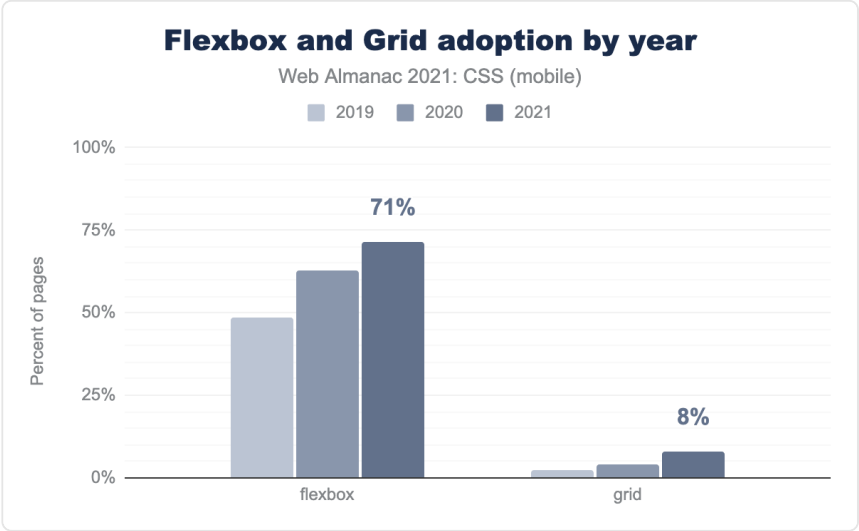


図1.36. モバイルデバイスでのFlexboxおよびGridレイアウトの採用。

Flexboxとグリッドの採用が進み続けています。2019年のFlexboxの採用率は41%、2020年は63%でした。今年、Flexboxはモバイルで71%、デスクトップで73%を記録しました。一方、グリッドは、Web Almanacでは毎年2%から4%、そして現在は8%と倍増しています。前のセクションとは対照的に、ここで計測されているのは、単にスタイルシートにFlexboxやGridのプロパティが含まれているページではなく、実際にFlexboxやGridをレイアウトに使用しているページの割合であることに注意してください。

異なるグリッドレイアウト技術の使用

さまざまなグリッドのプロパティを調べてみると、いくつかの興味深いパターンが見つかりました。

- 全グリッドページの約15%が、グリッドの名前付きエリアを定義するために `grid-template-areas` を使用しています。

- グリッドのテンプレートにある角括弧は、名前付きのグリッド線があることを示していますが、約700万ページの分析のうち、1万ページ弱しか見つかりませんでした。

また、Flexboxのレイアウトを分析し、フレックスの成長値と縮小値をゼロに設定し、すべてのフレックスアイテムの幅をパーセンテージやピクセル幅などの静的なものに設定したレイアウトを確認しました。これらは「グリッドライクのFlexbox」と呼ばれ、Flexboxレイアウトの4分の1強がこの条件を満たしていることがわかりました。分析が複雑であるため、多くのケースを見逃している可能性があります。しかし、デザイナーがグリッドスタイルのレイアウトに強い関心を持っていることは明らかであり、今後数年間でグリッドへの移行は進む可能性があります。

マルチカラム



図1.37. マルチカラムレイアウトを使用しているページの割合。

マルチカラムレイアウトはユーザーが列の一番下までスクロールしてから次の列の一番上まで戻ってこなければならないという、ウェブ上では少々面倒なものですが、分析したページの20%でマルチカラムの使用が検出され、これは2020年版Web Almanacよりも5%上昇しています。これほど多くのページに掲載されていることに驚きを隠せませんし、その採用率が高まっていることにも驚きを隠せません。

ボックスのサイズ

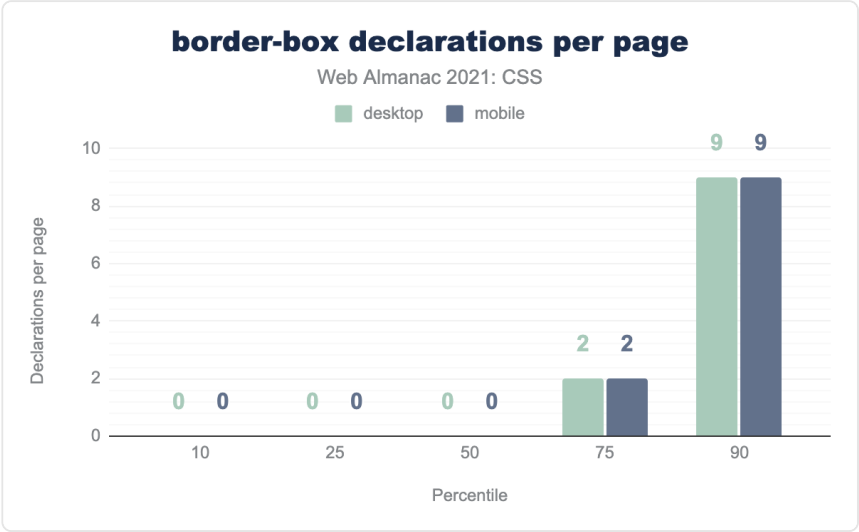


図1.38. 1ページあたりの`border-box`宣言の数の中央値の分布。

W3Cのオリジナルボックスモデルの原則は、引き続き否定されています。`box-sizing: border-box`を使用しているページがどれくらいあるかを調べたところ、2020年から約5%増加し、90%という圧倒的な数字になりました。分析したページの約半数が、ユニバーサルセレクター(`*`)を使って、ページ上のすべての要素にボーダーボックスのサイズを適用しています。このような「1つのサイズがすべてに適合する」アプローチは、ページあたりの`border-box`宣言の数の中央値が下位3つのパーセンタイルで非常に低いことの説明に役立つかもしれません。

また、約4分の1のページでは、チェックボックスやラジオボタンに`box-sizing`を適用しています。

トランジションとアニメーション

アニメーションは引き続き広く使用されており、`animation`プロパティは、分析対象となった全モバイルページの77%、全デスクトップページの73%で使用されています。 さらに人気の高い`transition`は、モバイルページの85%、デスクトップページの90%で使用されています。

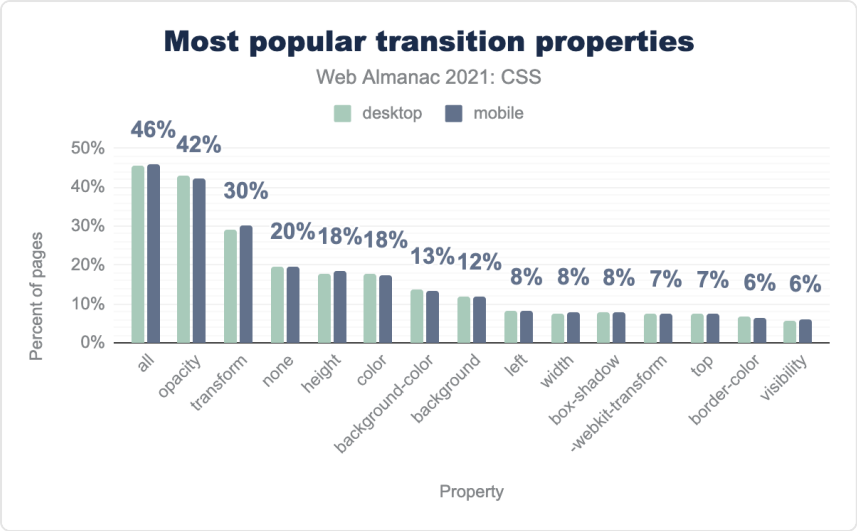


図1.39. トランジション効果が与えられたもっとも人気のあるプロパティです。

これらのトランジションの中で、もっとも一般的なアプリケーションは、`all`キーワードを使ったすべてのanimatable property[5]への適用であり、分析されたページの46%で発生しています（明示的であれデフォルトであれ）。これに続くのは`opacity`で、トランジションを含むすべてのページの42%です。

5. *https://developer.mozilla.org/docs/Web/CSS/CSS_animated_properties*

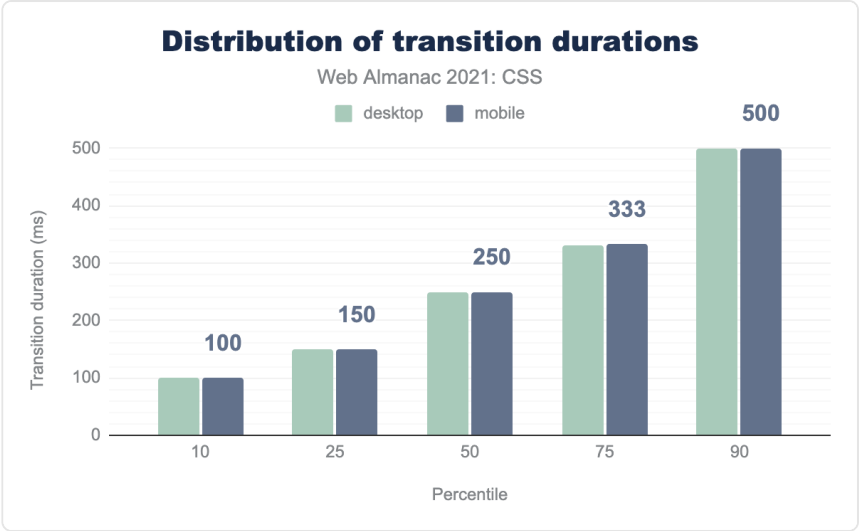


図1.40. 遷移時間の分布

私たちは、これらのトランジションの持続時間と遅延時間を見てみました。90パーセンタイルの場合でも、トランジションの持続時間の中央値はわずか0.5秒でした。

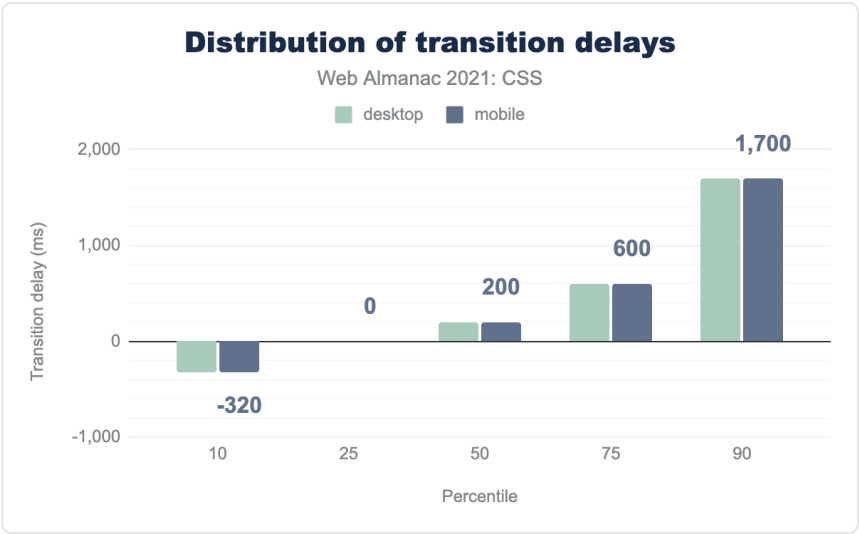


図1.41. トランジションディレイの分布

トランジションの遅延の中央値がもっとも高かったのは1.7秒でしたが、さらに興味深いこ

とに、10パーセンタイルの中央値の遅延は負の3分の1秒とまではいかない程度で、多くのトランジションがアニメーションの途中で開始されていることを示しています（これは負の遅延が引き起こす現象です）。

遷移の継続時間と遅延の範囲を詳しく見てみると、非常に長時間かかることがわかりました。もっとも大きな値は9,999,999,999,996秒で、これは約3億1700万年に相当します。 別の言い方をすると、月がたった1ピクセルだったら[6]水平方向のスクロール遷移にこの期間を使用した場合、1ピクセルだけ右へスクロールするのに2世紀以上かかることとなります。しかし、私たちが発見したもっとも長い移行遅延である31.7 兆年 年に相当するミリ秒の値に比べれば、その差は歴然としています。

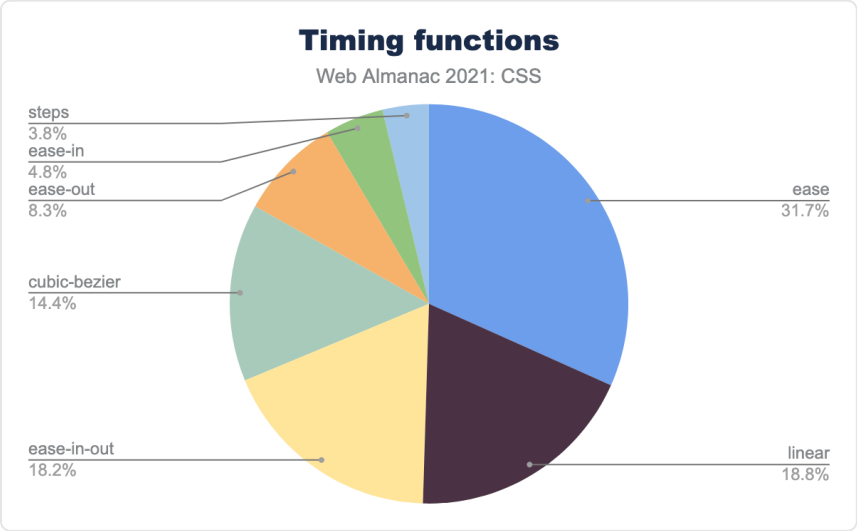


図1.42. 移行タイミング機能の採用

トランジションの際に使用されるタイミング関数については、デフォルト値である`ease`が断トツのトップ。2位は`ease-in-out`と`linear`の事実上の同点でしたが、意外だったのは4位の`cubic-bezier`でした。これはライブラリやある種のツールから来ている可能性が高いと思われます。なぜなら、三次ベジエ曲線の作成方法を手で学ぶことは可能ですが、わざわざそうする人はほとんどいないからです（そうする理由もあまりありませんが）。

なるほど、でも、どんなアニメーションが行われているのでしょうか？ そのために、さまざまなアニメーションラベルを、実行されているアニメーションの種類によって分類しました。たとえば、`fa-spin`、`spin`、`spinner-spin`などと表示されているアニメーションは「回転」アニメーションに分類され、もっとも人気がありました。

6. *https://www.joshworth.com/dev/pixelspace/pixelspace_solarsystem.html*

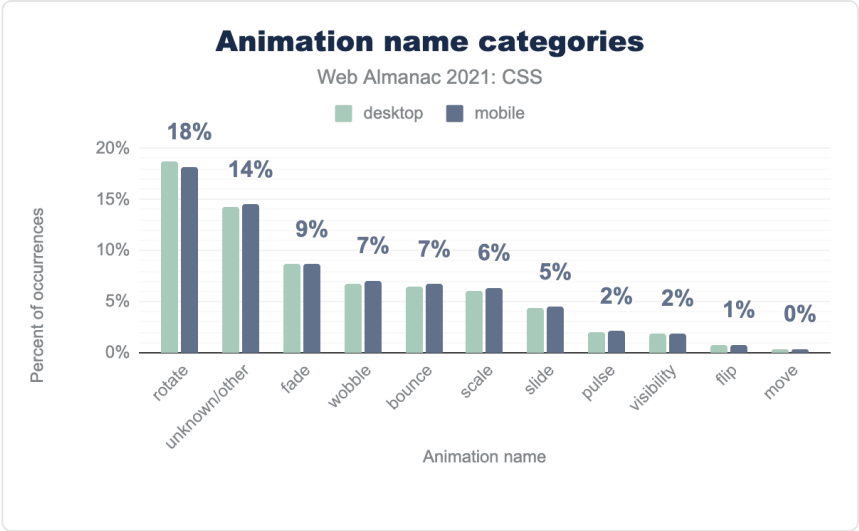


図1.43. もっともポピュラーなアニメーションの種類です。

「不明・その他」の順位が高い理由の1つは、アニメーションのラベル `a` で、名前のついたアニメーション全体の約6〜7%でした。(これにもっとも近いと思われる `b` は2%の普及率でした）。

「移動」や「スライド」スタイルのアニメーションが弱いのは意外かもしれませんが、これらは具体的には `animation` のタイプであることを忘れないでください。このサンプルでは、 `transition` プロパティで駆動するトランジションは表現されていません。多くの単純な動き（およびフェード）はトランジションで処理され、 `animation` はより複雑な効果のために確保されていると考えられます。

レスポンシブデザイン

FlexboxやGridなどの組み込みツールの登場により、ウェブを閲覧する際のさまざまなスクリーンサイズに対応したサイトを作ることが非常に容易になりました。また、media-queriesを使用することで、さらに効果的になります。

使用されているメディア機能

作者がメディアクエリを作成する際には、ビューポートの幅をテストすることがほとんどです。 `max-width` と `min-width` が圧倒的に多いクエリで、これは2020年と同じでした。3位と4位の結果にも順位の変動はありませんでした。

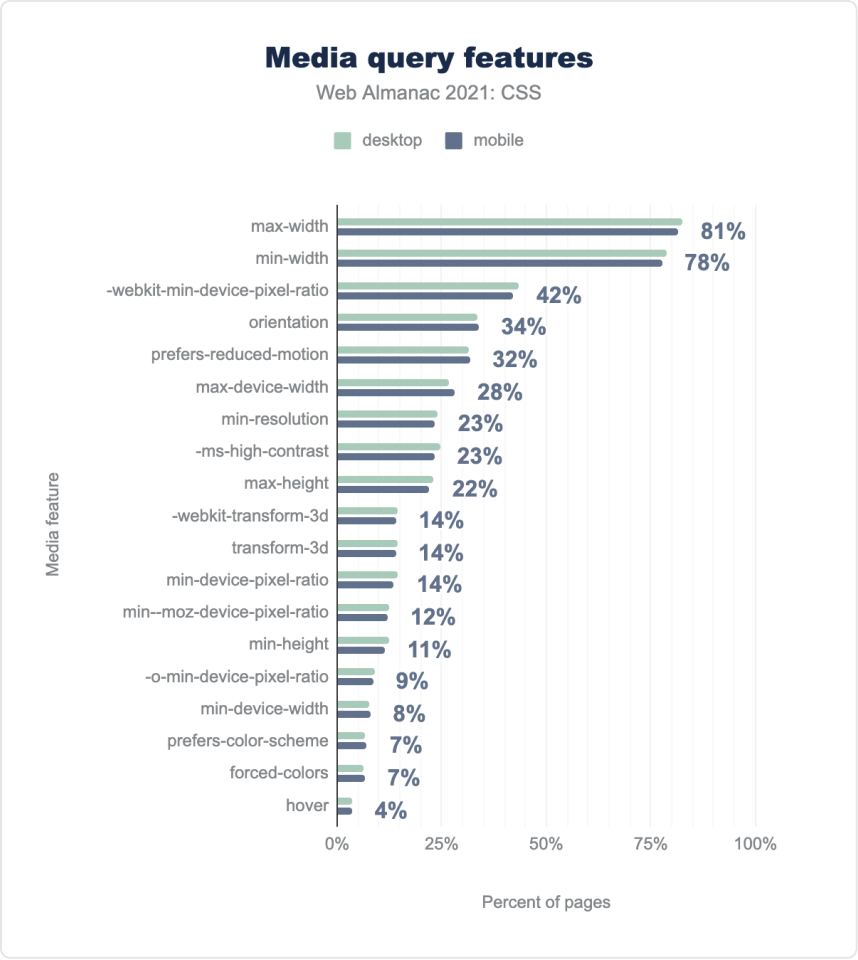


図1.44. メディアクエリとして使用されるもっとも一般的な機能です。

顕著な変化が見られたのは、`prefers-reduced-motion`クエリのランキングでした。このクエリは、2020年には24%のシェアで7位でしたが、今年は32%のシェアで5位に上がり、`orientation`をわずかに上回っています。

また、最下位には新参者が登場しました。昨年19位だった`pointer`は、ディスプレイデバイスの主な入力機構がマウスなどのポインティングデバイスであるかどうかをチェックするクエリで、21位に転落してしまいました。一方、`hover`メディア機能は、20位にランクインしました。`hover`は、ディスプレイデバイスの主要な入力メカニズムが、ページ上の要素にホバー状態を引き起こすことができるかどうかをテストするために使用されます。

どちらのクエリも目的は似ていて、（簡単に言うと）ページを表示するために使用されているデバイスがマウス駆動であるかどうかを調べることです。モバイルファーストのデザイン哲学と組み合わせると、デスクトップのスタイルがモバイルのデフォルトスタイルを上書きするように追加され、`pointer`や`hover`のようなクエリがどのように役立つかがわかります。どちらかが優位に立つかどうかを判断するのはまだ早いですが、今年のトレンドは`hover`に傾いています。

また、今年は`prefers-color-scheme`が登場し、7%となりました。これは、昨年のレポート以降、iOSデバイスがダークモードに対応したためと思われますが、いずれにしても、デザイナーが配色の好みを考慮するようになったことは喜ばしいことです。

一般的なブレークポイント

2020年と同様、もっともよく使われたブレークポイントは767ピクセルと768ピクセルで、これはiPadのポートレートモードの解像度と疑わしいほどよく一致しています。`767px`は最大幅のブレークポイントとして使われることが圧倒的に多く、最小幅の値として使われることはほとんどありませんでした。一方、`786px`は最小値、最大値としてもよく使われていました。

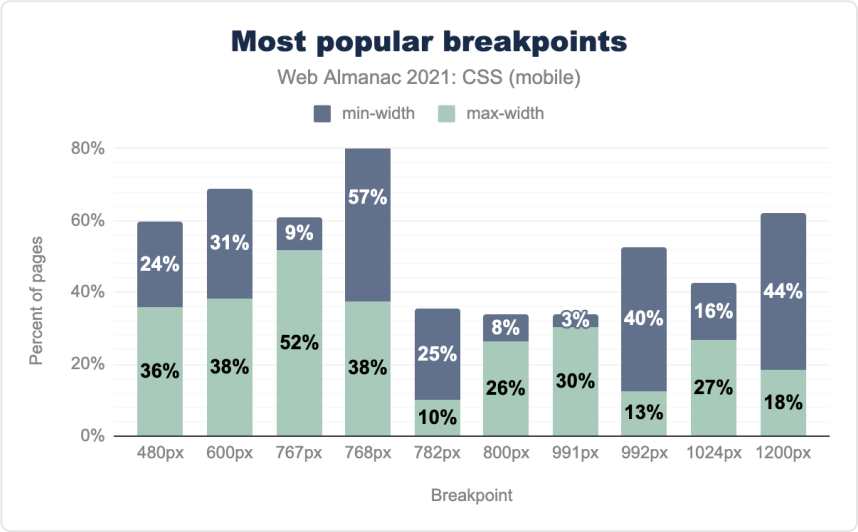


図1.45. もっともポピュラーなメディアクエリのブレークポイントです。

767-768の範囲を超えて、次に人気があったのは600ピクセルと1,200ピクセルで、その次が480ピクセルでした。

ブレークポイントのクエリをすべてピクセルに変換したのかと思われるかもしれませんが、

残念ながら変換していません。今回分析したすべてのブレークポイントのうち、ピクセル以外の値として最初に挙げられたのは`48em`で、ランキングでは76位、デスクトップでは1%、モバイルでは2%のスタイルに現れています。次のemベースの値である`40em`は、85位に見られます。

メディアクエリ内のプロパティ

では、これらのメディアクエリブロックの中で、作者は実際に何をスタイリングするのでしょうか？　もっともよく設定されるプロパティは`display`で、次いで`color`、`width`、`height`となります。

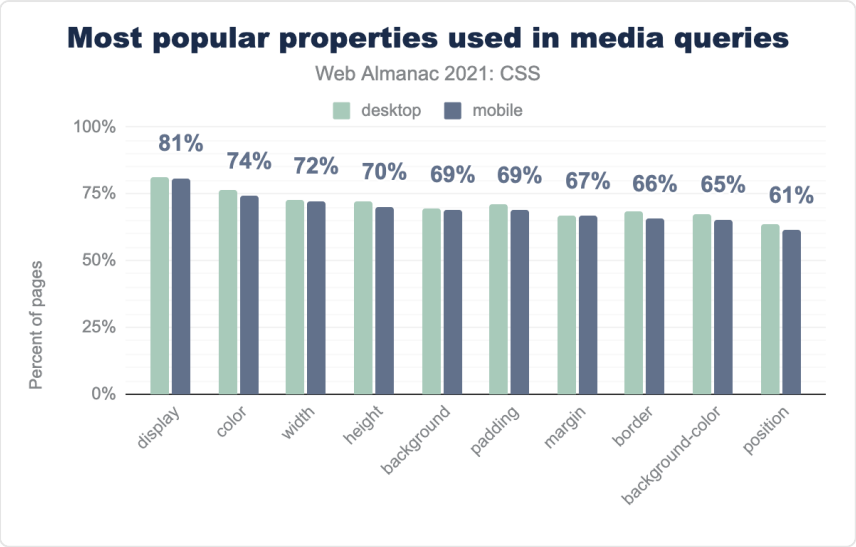


図1.46. メディアクエリで変更するのがもっともポピュラーなプロパティです。

2020年から2021年にかけてのもっとも注目すべき変化の1つは、メディアブロック内に設定されるプロパティとしての`font-size`の凋落です。2020年には、全メディアブロックの73%に登場し、ランキング5位となっていました。今年は全メディアブロックの約6割に登場し、12位にランクインしました。

`margin-right`と`margin-top`はさらに大きく落ち込み、それぞれ8位と9位から25位と17位になりました。このような変化は、共通のフレームワークやソフトウェアの変更を強く示唆しています。WordPressのデフォルトテーマの変更もその一例ですが、これが変化の正確な原因であるかどうかはわかりません。

クエリ機能

クエリ機能（`@supports`）の使用率は伸び続けています。2019年には30%のページが使用していることが判明し、昨年は39%でした。2021年には、ほぼ48%のページが、どのコンテキストでどのCSSを適用するかを決めるために、クエリ機能を使用しています。

では、作者は何を条件にCSSを使うのでしょうか？　もっとも多かったのは「スティッキーポジショニング」で、全体の半数以上を占めています。

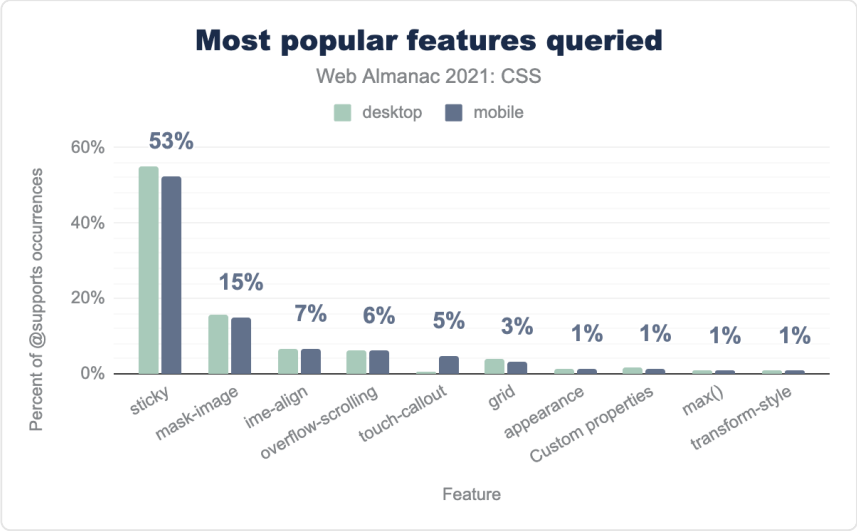


図1.47. CSSの代表的な機能を`@supports`で照会できます。

グリッドをサポートしているかどうかを確認したクエリ機能はわずか3%で、グリッドをサポートしているかどうかを確認したページは261,406ページになります。グリッドレイアウトが使用されていたのは、モバイルページで270万ページ、デスクトップページで230万ページであったことを考えると、この数字が正確であれば、グリッドレイアウトの大部分はフォールバックなしで導入されていると考えられます。

Custom properties

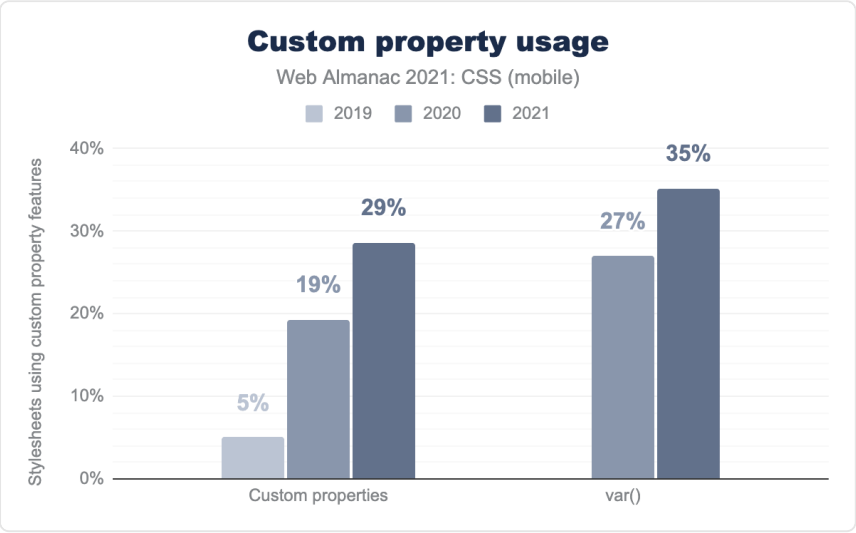


図1.48. カスタムプロパティ使用量の変化（2019-2021年）。

Web Almanacの3年間で、カスタムプロパティ（CSS変数とも呼ばれる）の使用率がもっとも高まったものの1つです。2019年の使用率は全サイトの5%程度でしたが、昨年はモバイルが20%近く、デスクトップが15%と急増していました。今年は、全モバイルページの28.6%、デスクトップページの28.3%でカスタムプロパティが定義されていることがわかりました。さらに、モバイルページの35.2%、デスクトップページの35.6%に、少なくとも1つの `var()` 関数値が含まれていることがわかりました。

ネーミング

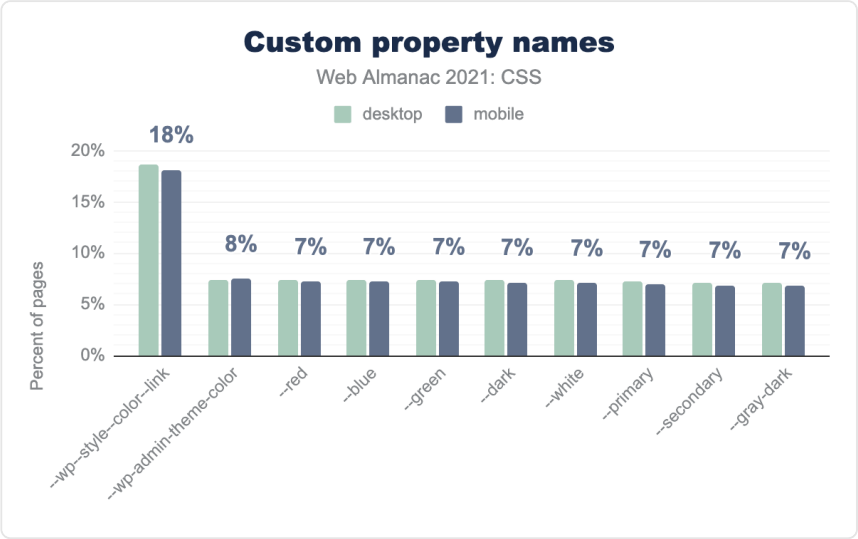


図1.49. もっともポピュラーなカスタムプロパティ名です。

最初に確認したこと、"開発者は自分のカスタムプロパティを何と呼んでいるのか？ "ということでした。結論から言うと、ここではWordPressの普及率が出ていて、WPコアで定義されたリンクカラーリングのカスタムプロパティがトップエントリーになっています。

その後、たくさんの色名が見つかりました。名前のついた色 `blue` がすぐそこにあり、`--blue` へカスタム値を定義する必要があるとは奇妙に思えるかもしれませんが、実際には、開発者は基本的な色名にカスタムシェードを割り当てています。そのため、`--blue: blue` ではなく、`--blue: #3030EA` です。

使用方法

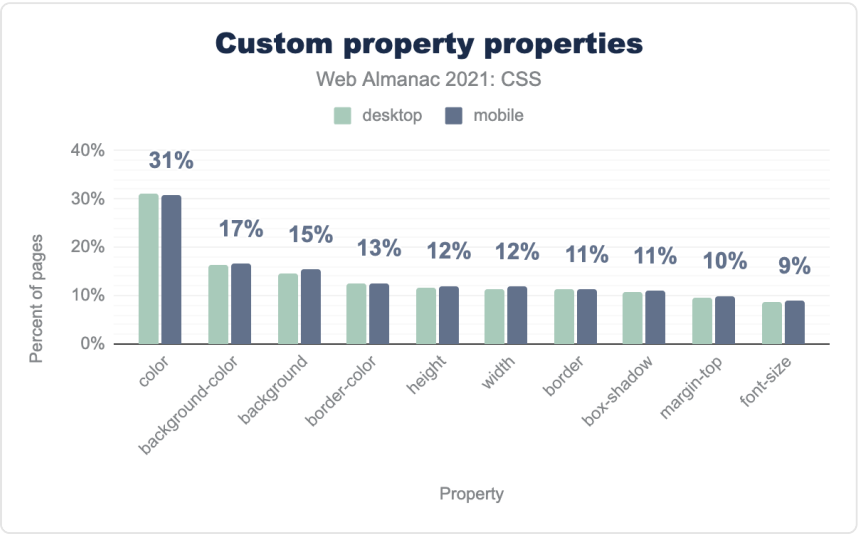


図1.50. custom-propertyの値を与えられるもっとも人気のあるプロパティ。

色にちなんだすべてのカスタムプロパティに加えて、カスタムプロパティの値を受け取る（`var()`関数を使用する）もっとも一般的な4つのプロパティは、いずれも何らかの形で色を設定しています。

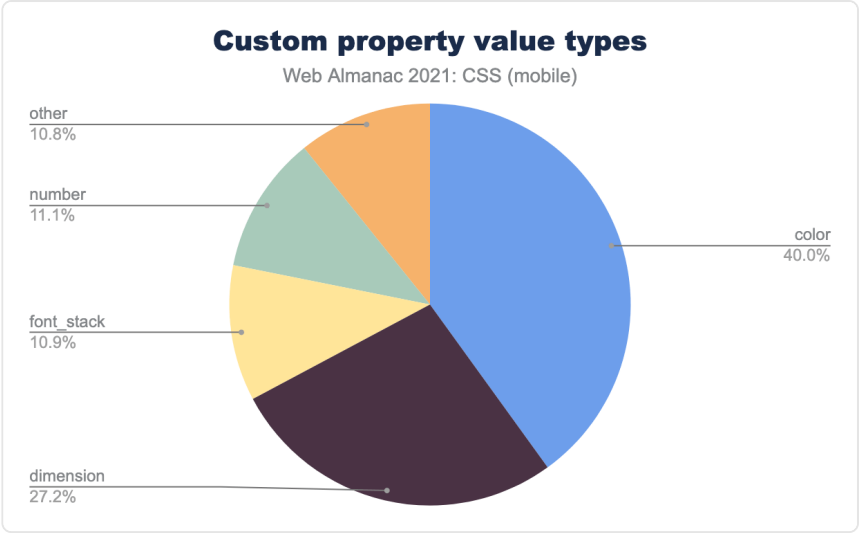


図1.51. カスタムプロパティ値の種類の分布

各カスタムプロパティには、1種類のCSS値が設定されます。たとえば、`--red: #EF2143`は`--red`に色の値を割り当て、`--multiplier: 2.5`は数字の値を割り当てます。その結果、「色」がもっとも多く、次いで「寸法（長さ）」、そして「フォントファミリー」の順で単独、グループ共に人気だということがわかりました。

複雑さ

カスタムプロパティを他のカスタムプロパティの値に含めることは可能です。2020 Web Almanacの例を見てみましょう。

```
:root {
  --base-hue: 335; /* depth = 0 */
  --base-color: hsl(var(--base-hue) 90% 50%); /* depth = 1 */
  --background: linear-gradient(var(--base-color), black); /* depth
= 2 */
}
```

前述の例のコメントにあるように、これらの下位参照を連鎖させればさせるほど、カスタムプロパティの *depth* は大きくなります。

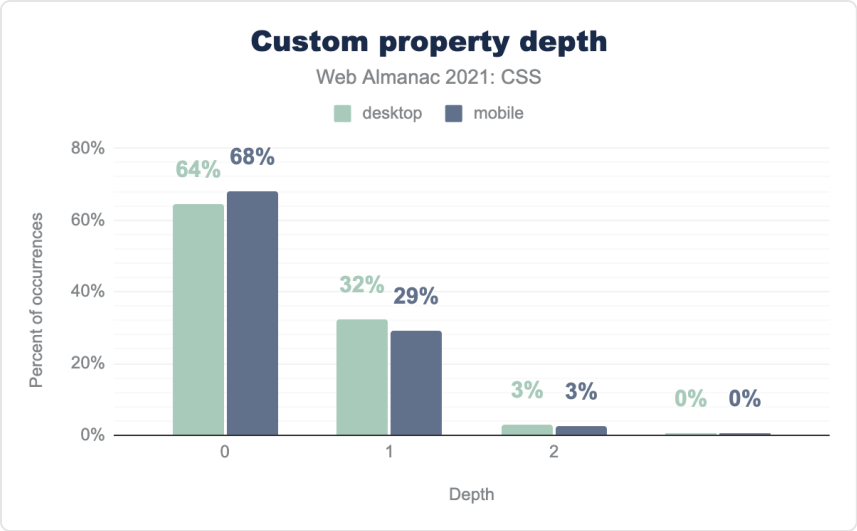


図1.52. カスタムプロパティの深さの中央値の分布

意外かもしれませんが、カスタムプロパティの値の深さは0のものが圧倒的に多く、他のカスタムプロパティの値を自分の値に含めていません。3分の1近くは値の深さが1段階で、2段階以上の値を持つカスタムプロパティはほとんどありません。

2020年と同様に、カスタムプロパティの値が使用されているセレクターも確認しました。約60%がルート要素（`:root`または`html`セレクターを使用）に設定されており、約5%が`<body>`要素に適用されていました。残りは、ルート要素の子孫である`<body>`以外の要素に適用されています。つまり、全カスタムプロパティの約3分の2が、実質的にグローバルな定数として使用されていることになります。これは、昨年の結果と同じです。

国際化

英語は横書きで、文字を左から右に読む。しかし、アラビア語、ヘブライ語、ウルドゥー語などの言語は右から左に書きますし、モンゴル語、中国語、日本語のように、上から下へ縦に書く言語や文字もあります。そのため、非常に複雑な構造になっています。HTMLとCSSには、このような問題を解決する方法があります。

方向性

テキストの方向は、CSSプロパティの`direction`を使って、明示的に強制できます。このプロパティは、全ページの11%の`<html>`要素で使用されており、3%の`<body>`要素で

使用されていることがわかりました（重複した結果をチェックしていないので、重複している可能性があることに注意してください）。

CSSで方向性を設定しているページのうち、`<html>`要素の92%、`<body>`要素の82%が`ltr`（左から右）に設定されていました。全体的に見ると、CSSで方向性を設定しているページのうち、`rtl`（右から左）が使われているのは9%だけでした。これは、ほとんどの言語が右から左ではないことを考えると、大体予想できることです。

論理的・物理的特性

また、国際化に役立つCSSの機能として、`margin-block-start`、`padding-inline-end`などの「論理的」なプロパティや、`text-align`などのプロパティの`start`や`end`などの値があります。これらのプロパティと値により、ボックスの機能を、上、右、下などの物理的な方向ではなく、テキストの流れの方向に関連付けることができます。

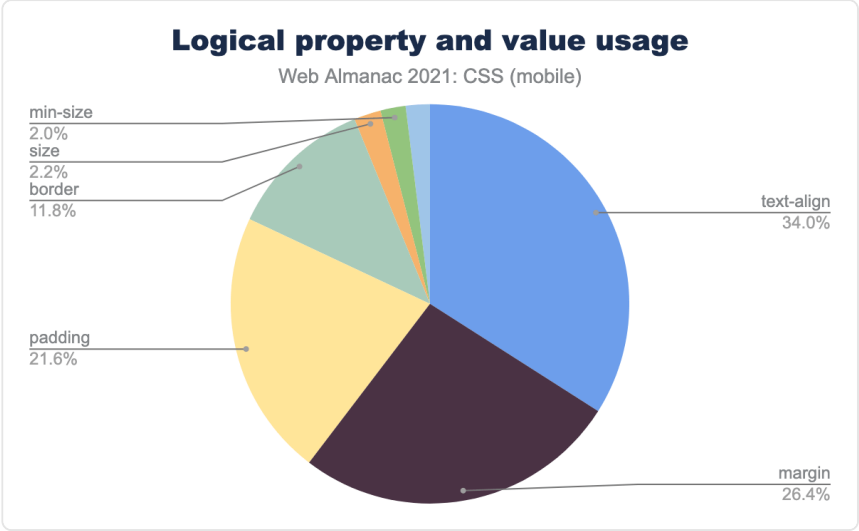


図1.53. 論理的なプロパティのプロパティタイプの分布

2021年半ばの時点で、何らかの論理的プロパティを使用しているページは、わずか4%でした。使っているページのうち、約33%が`text-align`を`start`や`end`へ設定するために使っていました。また、46%ほど（合計）が論理的なマージンやパディングを設定していました。繰り返しになりますが、これらの数字は重複してる可能性があります。

Ruby

CSSは、方向性や論理的な機能に加えて、CSS Rubyによる国際化にも対応しています。これは、インターリニアアノテーション（原文の横に短い文章を並べたもの）のレイアウトに影響を与えるプロパティの集まりです。CSS Rubyの使用率は非常に低く、デスクトップでは8,157ページ、モバイルでは9,119ページと、分析した全ページの0.1%にも満たないことが分かりました。

CSSとJS

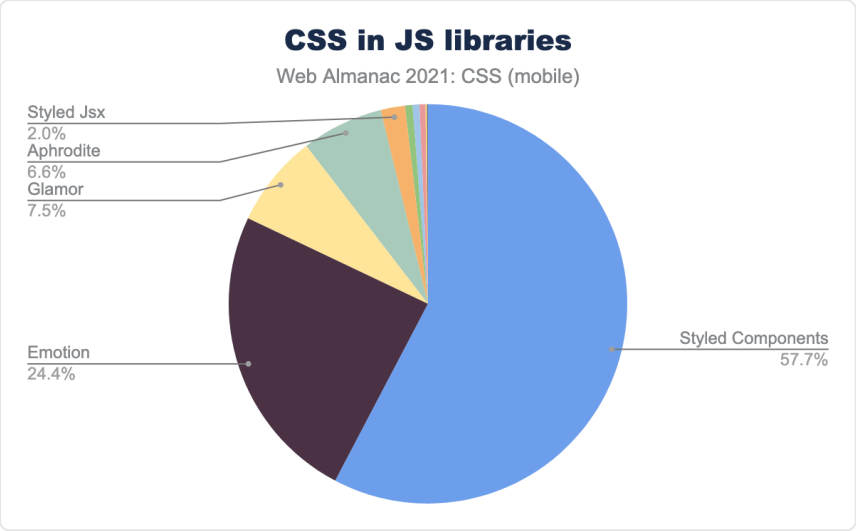


図1.54. CSS-in-JSライブラリの分布。

『CSS in JS」の話題は、少なくともTwitterでの炎上騒ぎの1つや2つにはもってこいですが、自然界での使用は非常に少ない状態が続いています。今年の調査では、ページの約3%が何らかの形でCSS-in-JSを使用していることがわかりましたが、これは2020年の2%から増加しています。さらに、そのほぼすべてがこの目的のために構築されたライブラリによるものであり、その半分以上がStyled Componentsライブラリによるものです。

Houdini

CSS Houdiniは、ある意味ではCSS-in-JSの逆を行くもので、作者がCSSに少しだけJSを混ぜ

ることを可能にします。おそらく、仕様の中核部分の遅い実装[7]（Blinkベースではないブラウザで）が原因で、Houdiniはその地位を確立するのに苦労しています。 We find that it's effectively not used on the open web in 2021: only 1,030 desktop pages and 1,175 mobile pages show evidence of animated custom properties, a feature of Houdini. This is a threefold increase over the 2020 findings, but it looks like it will still be some time before Houdini finds an audience。

Meta

このセクションでは、宣言の繰り返しの頻度や、作者がCSSを書く際にどのようなミスをするかなど、CSSのより一般的な概念を見ていきます。

宣言の繰り返し

2020年版Web Almanacでは、「宣言の繰り返し」の量を分析しています。これは、同じプロパティや値を使用している宣言がいくつあるか、ページのスタイル内でユニークな宣言がいくつあるかを判断することで、スタイルシートの効率を大まかに見積もるための指標です。

2021年の数字が発表され、すべてのパーセンタイルで反復回数の中央値がわずかに低下していることがわかりました。

7. *https://ishoudinireadyyet.com/*

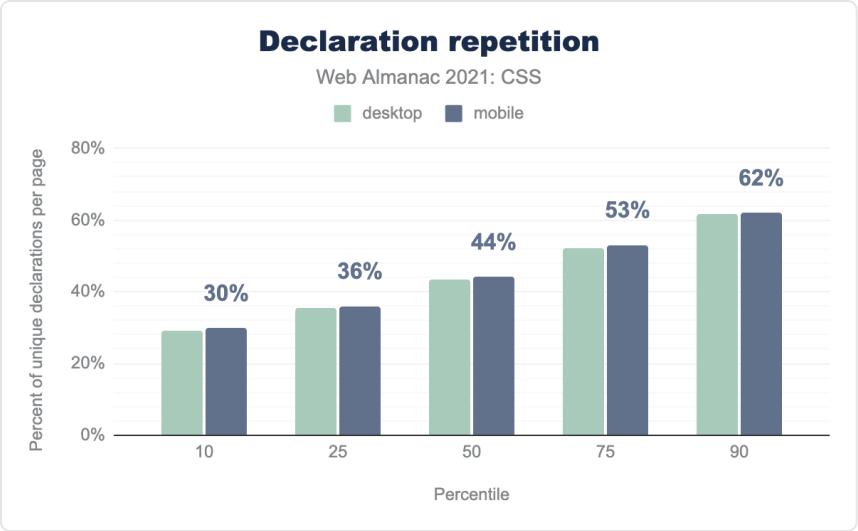


図1.55. ページごとの宣言文の繰り返しの分布。

この低下の度合いは、10%、50%、90%で2%のオーダーなので、これが統計的なノイズである可能性は十分にあります。これを判断する唯一の方法は、今後も分析を継続し、長期的なトレンドをグラフ化することです。

ショートハンドとロングハンド

CSSでは、非常に具体的なプロパティの集合体が、単一の宣言でより具体的なプロパティの値を設定できる単一の「umbrella」プロパティによってもカバーされている部分が多くあります。たとえば、`font`は、`font-family`、`font-size`、`line-height`、`font-weight`、`font-style`、`font-variant`の値を包含します。umbrellaプロパティの`font`は、いわゆる「ショートハンド」プロパティで、作者がある種のショートハンドで多くのことを設定できるようにするものです。これに対応する具体的なプロパティ（たとえば、`font-family`）は、「ロングハンド」プロパティと呼ばれます。

ロングハンドの前にショートハンド

スタイルシートの中で、`background`のような短縮形プロパティと`background-size`のような長音部プロパティが混在している場合は、長音部が短縮形の後に来るのが常にベストです。私たちは、どのロングハンドがもっとも一般的であるかを調べるために、著者がこのようにした例を調べました。

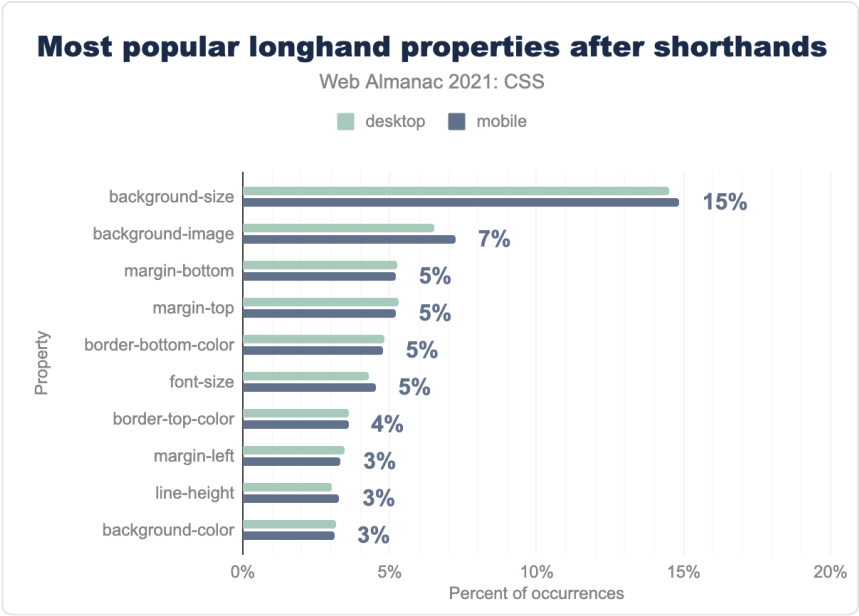


図1.56. 対応する短縮形プロパティの後に表示される、もっとも一般的なロングハンドプロパティです。

2020年と同様に`background-size`が選ばれましたが、昨年はモバイルで41%、今年は15%の割合でしか見られませんでした。

バックグラウンド

バックグラウンドロングハンドのプロパティが前節のチャートの上位にあったので、バックグラウンドショートハンドとロングハンドの使用に目を向けてみました。

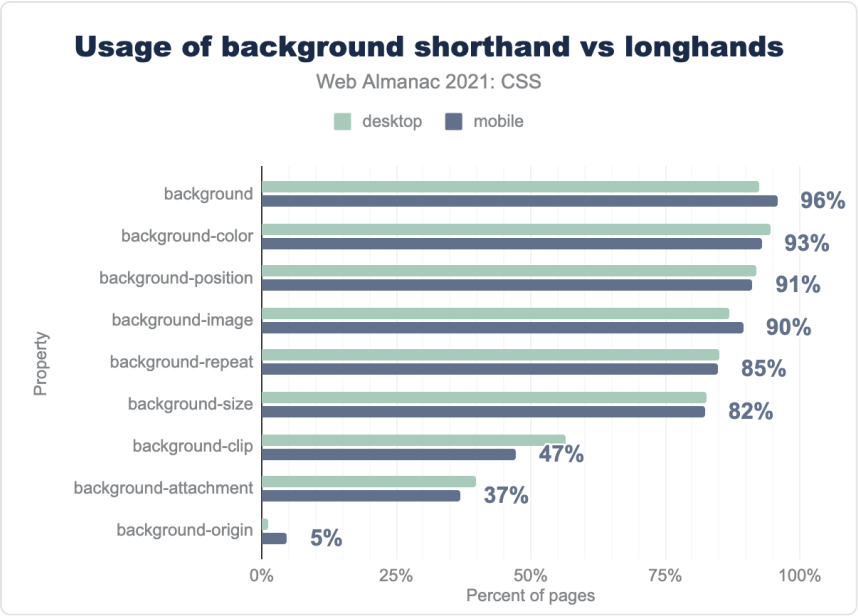


図1.57. もっともよく使われるバックグラウンドのプロパティです。

むしろ、これらを設定していないページがあったことに小さな驚きを感じました。圧倒的に96%のページが`background`というショートハンドを使用していますが、これは1996年のCSS1に遡ります。同年代のロングハンドプロパティも同様で、85%以上のページで適用されていることがわかりました。

しかし、最近の`background-size`は急速に普及しており、82% のページで採用されていることから、作者にとっての有用性の高さがわかります。一方、`background-origin`は、昨年の12%から今年はわずか5%にまで落ち込んでいます。

marginとpadding

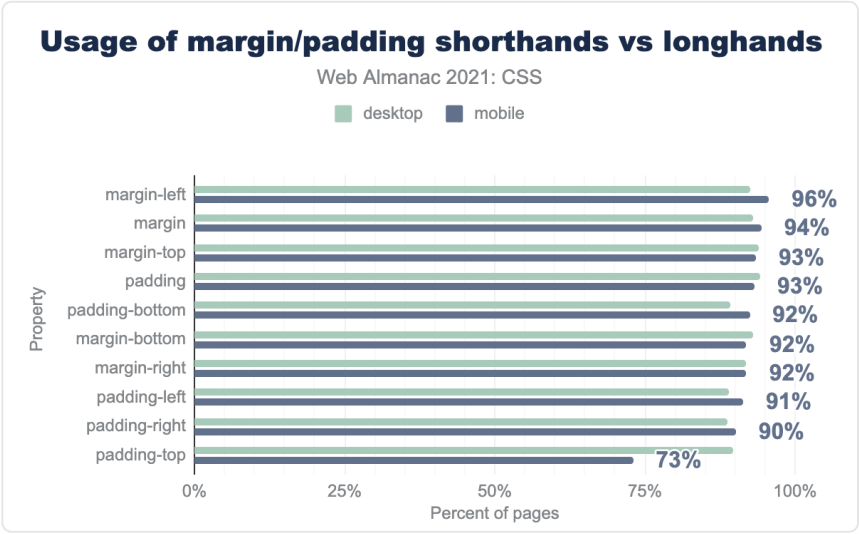


図1.58. もっともよく使われるmarginとpaddingのプロパティです。

続いて、marginとpaddingのプロパティを見てみましょう。背景と同じように、これらのプロパティを設定しているページが多いことよりも、設定していないページがあることの方が驚きです。今年の関心事は、ロングハンドの`margin-left`がショートハンドの`margin`を抑えて1位になったことです。

フォント

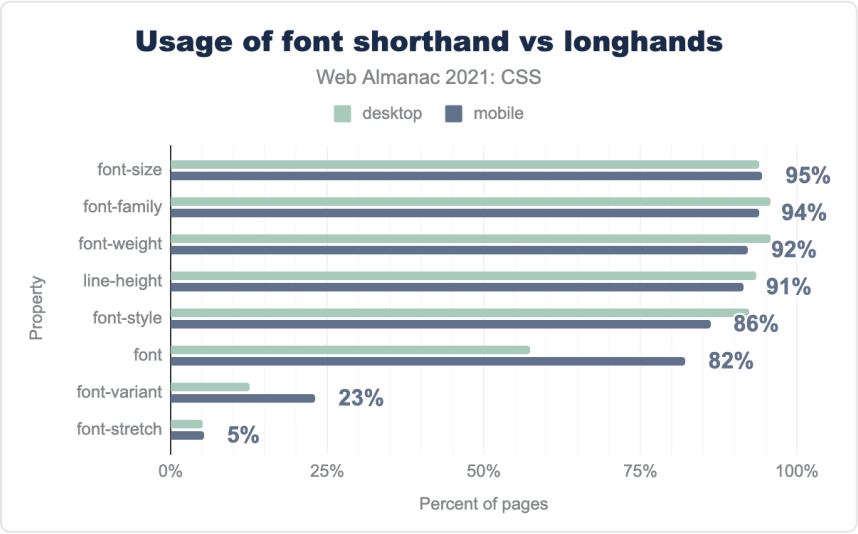


図1.59. よく使われるフォントのプロパティ

2020年と同様に、短縮形の `font` は一般的なロングハンドの `font-weight` と比較して、`font-size` がリードし、昨年の `font-weight` から首位を奪いました。

ここでは、`font-variant` と `font-stretch` という2つの異なったストーリーがあります。`font-variant` は、CSS1の頃から存在していましたが、デザイナーにはあまり浸透しませんでした。おそらく、長い間、`small-caps` を設定できなかったからでしょう。現在では、それとダウンロード可能なフォントを使って多くのことができるようになりましたが、著者はこの能力を活用していないようです。その使用率は今年大きく低下し、2020年の43%から2021年には23%に低下しました。

`font-variant` をもう少し詳しく見てみましょう。モバイルページの23%で使用されていますが、現在省略されているロングハンドのプロパティはほとんど使用されていません。以下は、`font-variant` だけでなく、それに対応するロングハンドをそれぞれ使用しているページの実際の数です。

プロパティ	デスクトップ	モバイル
`font-variant`	3,098,211	3,641,216
`font-variant-numeric`	153,932	166,744
`font-variant-ligatures`	107,211	112,345
`font-variant-caps`	81,734	86,673
`font-variant-east-asian`	20,662	20,340
`font-variant-position`	5,198	5,842
`font-variant-alternates`	4,876	5,511

図1.60. `font-variant` プロパティを使用しているページ数です。

これは、著者がショートハンドのみを使用し、ロングハンドを無視しているということでしょうか。しかし、昨年から `font-variant` の使用が急減していることから、一般的なフレームワークやツールがデフォルトスタイルから `font-variant` を削除したのではないかと考えています。いずれにしても、広くサポートされている多くのフォント機能を、作者は見逃しているのかもしれません。

もうひとつの低得点プロパティである `font-stretch` は、フォントファミリーがワイドまたはナローフェースを利用できるか、そして作者がそれらを利用することを選択するか（または知っているか）に大きく依存しているため、そのシェアが5%（昨年の8%から減少）となったことはあまり驚きではありません。

Flexbox

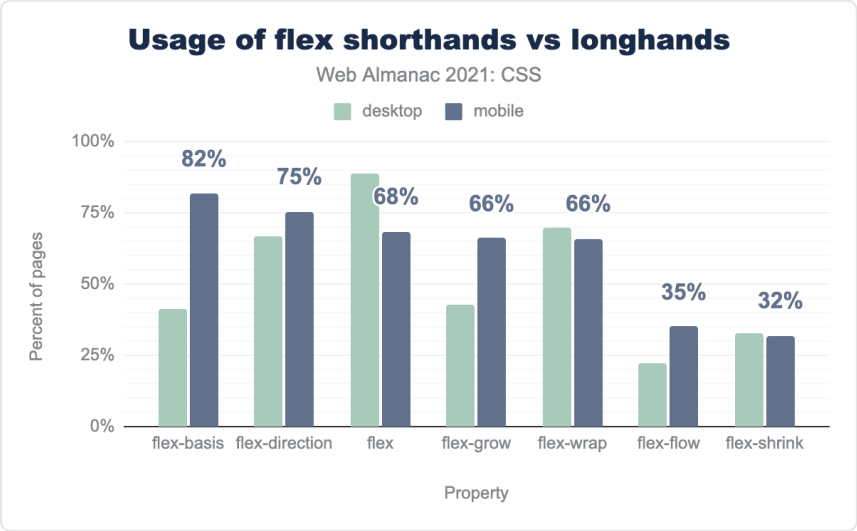


図1.61. もっともよく使われるFlexbox関連のプロパティです。

Flexboxのロングハンドとショートハンドのプロパティの中には、激動の歴史を持つものもあります。たとえば、CSS Flexboxの仕様自体、作者が `flex-grow`、`flex-shrink`、`flex-basis` の使用を避け、代わりに `flex` ショートハンドを使用することを推奨しています。これにより、設定されていないプロパティが適切な値を持つようになります。残念ながら、実際にはこれはうまくいっていないようで、モバイルページでは `flex-basis` が `flex` よりも多く使用されており、その差は10%以上にもなります。

この数字には、モバイルでは `flex-basis` の使用率が2倍になる一方、デスクトップではあまり変化がないなど、昨年と比較して大きな変動があることに留意する必要があります。これは、モバイル開発で使用される共通のフレームワークが変更されたことによるものかもしれませんし、その他の要因によるものかもしれません。

Grid

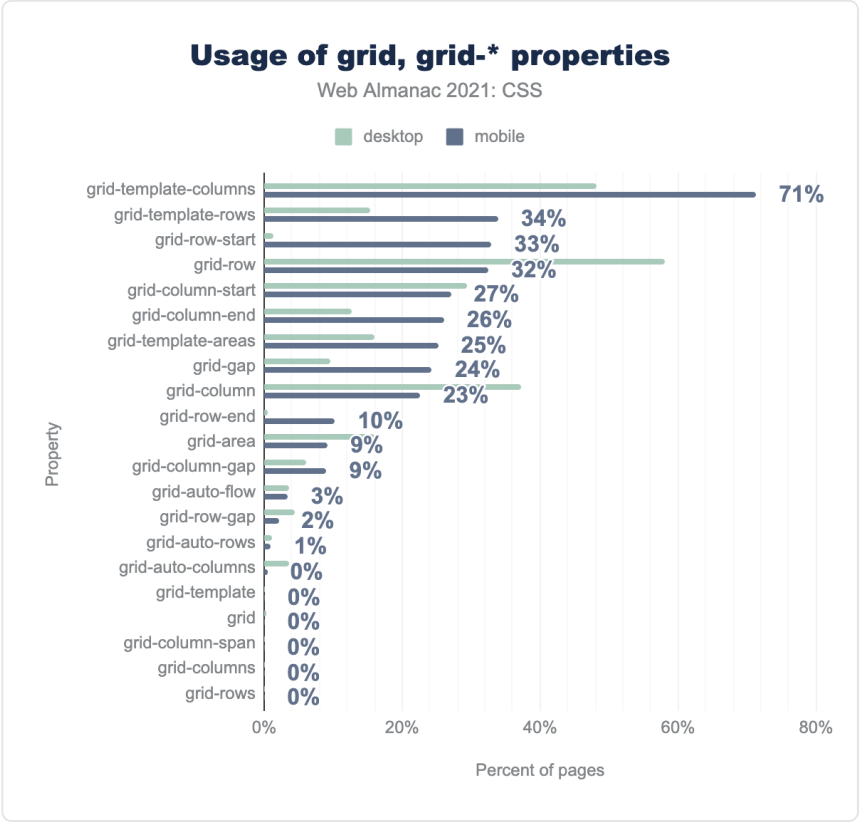


図1.62. もっともよく使われるグリッド関連のプロパティです。

ここ数年の傾向として、グリッドのショートハンドプロパティ（`grid-template`、`grid`など）は、それらが包含するロングハンドプロパティよりもはるかに少ない頻度で使用されていることがわかっています。 実際には、両方とも0%という驚異的な数字で、ランキングでは隣り合っています。残りの略語はすべてこの2つに集約されており、`grid-template-rows`や`grid-column`などのロングハンドのプロパティは広く使用されています。実際、注目すべき使用率の高いロングハンドプロパティは`grid-gap`だけで、モバイルのグリッドページでは24%の使用率となっています。今後、より新しい、そして一般的な`gap`が`grid-gap`を追い越すかどうか、興味深いところです。

CSSの間違い

人は成功からだけでなく、失敗からも多くのことを学ぶことができます。私たちはこの機会に、よくある間違いだけでなく、正しいように見えて正しくないものを探しました。

回復不能なシンタックスエラー

今年の解析作業では、2020年と同様にReworkというCSSパーサーを使用しましたが、より心強い数字が得られました。デスクトップページのわずか0.94%、モバイルページの0.55%に回復不能なエラーが含まれていました。つまり、Reworkでスタイルシート全体を解析不可能なほどひどいエラーです。確かに、回復可能な小さなCSSエラーが発生したページの数は格段に多かったかもしれませんが、今年の回復不可能なエラーの数は、昨年に比べて大幅に減少しています。これは、シンタックスクリーンアップが突然発生したのではなく、Reworkが変更されたことを示していると考えられます。

存在しないプロパティ

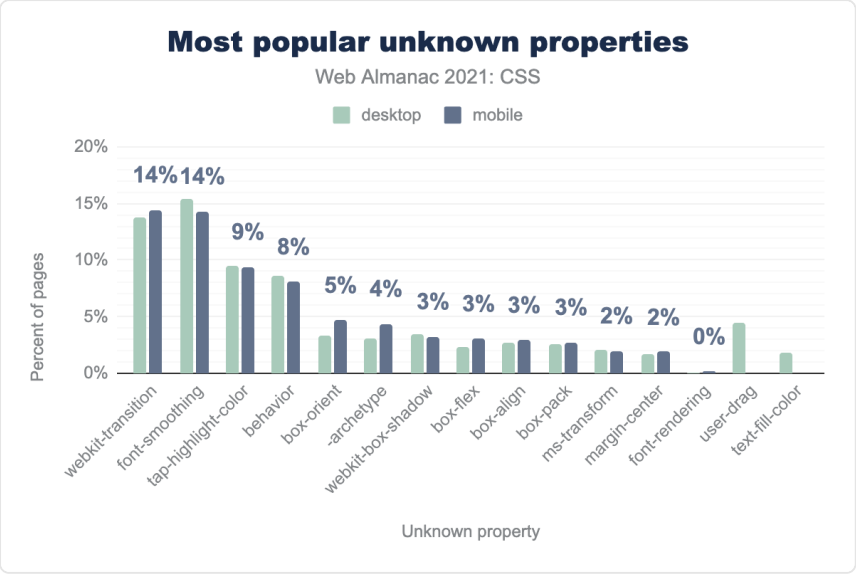


図1.63. もっとも一般的な未知のプロパティです。

私たちがチェックしたいことのひとつに、構文的には有効だが、実際には存在しないプロパティを使用している宣言の存在があります。これは、ベンダー接頭辞付きのプロパティは数

8. https://github.com/reworkcss/css

えませんが、不正なベンダー接頭辞付きのプロパティは含みます。実際、もっとも広く見られた存在しないプロパティは`webkit-transition`（適切なベンダープレフィックスに必要な先頭の`-`がない）で、存在しないプロパティを含むすべてのページの14%に見られました。本質的に結びついていたのは`font-smoothing`で、これは`-webkit-font-smoothing`の接頭辞なしのバージョンで、実際には存在しませんし、すぐに存在する可能性もありません[9]。

ショートハンドの前にロングハンド

本章の前のセクションでは、どのロングハンドプロパティが対応するショートハンドプロパティの後に出てくる可能性が高いかを調べました（たとえば、`background`の後に`background-size`が続くこともあります）。

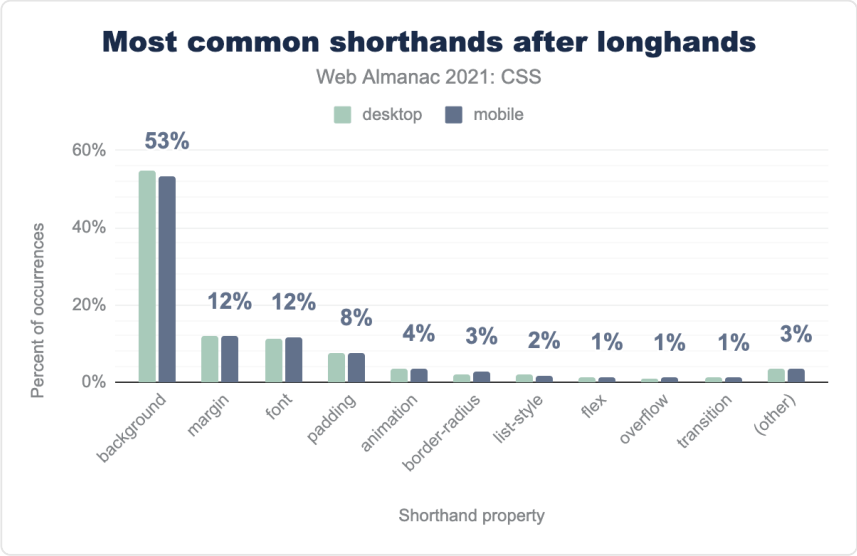


図1.64. 対応するロングハンド・プロパティの後に（不適切に）表示されるもっとも一般的なショートハンド・プロパティです。

ロングハンドの後にショートハンドを置くという逆のことをするのは、気が滅入るほどよくある間違いで、背景のプロパティでよく起こります。ロングハンドの後に対応するショートハンドを置く場合、背景のロングハンドプロパティは、`background`ショートハンドプロパティの値で上書きされます。

9. https://developer.mozilla.org/docs/Web/CSS/font-smooth

Sass

CSSプリプロセッサの大きな利点の1つは、CSS自体に欠けているものを明らかにできるため、将来的にCSSをどのように拡張していくべきかの指針となることです。これは以前にもあったことで、変数はプリプロセッサで非常に人気があったため、CSSは最終的にカスタムプロパティ[10]をそのレパートリーに加えました。色の変更やネストされたセレクターなど、プリプロセッサの他の機能も基本言語に組み込まれています。そこで本章では、現在ウェブ上でもっとも人気のあるプリプロセッサの1つであるSassを、開発者がどのように使っているかを紹介します。

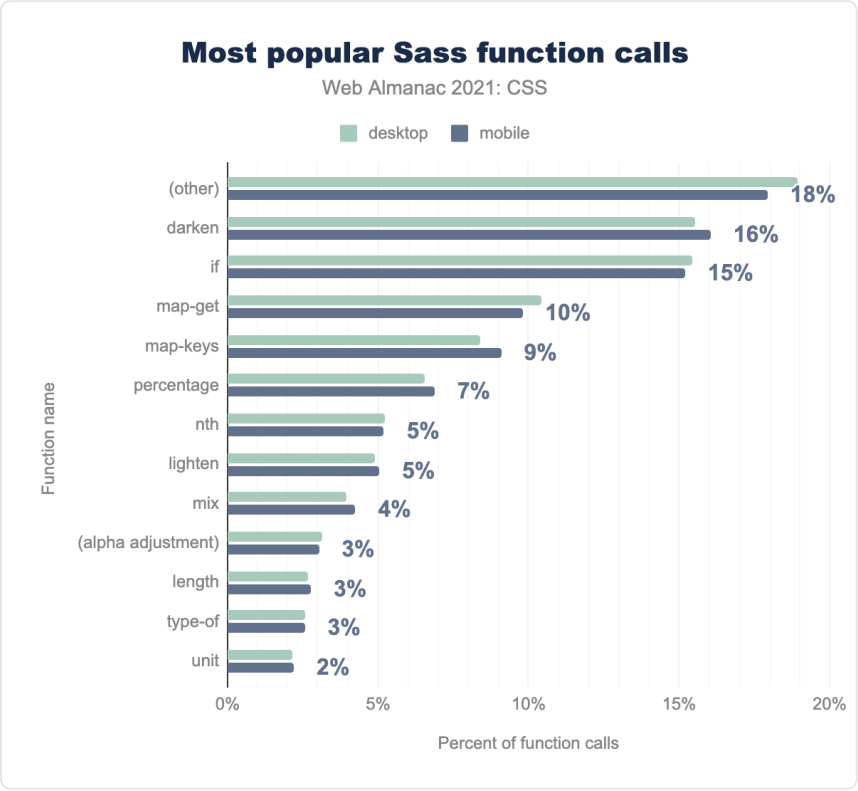


図1.65. もっとも一般的に使用されるSassの関数呼び出しです。

使用されているSass関数は、比率に若干の変更はあるものの、2020年版Web Almanacで見つかったものとほぼ同じでした。タイプ別に分類すると、全Sass関数の28%が色を変更するもの（例：`darken`、`mix`）で、さらに6%が色成分を読み取るために使用されていること

10. *https://www.w3.org/TR/css-variables-1/*

がわかりました（例：`alpha`、`blue`）。

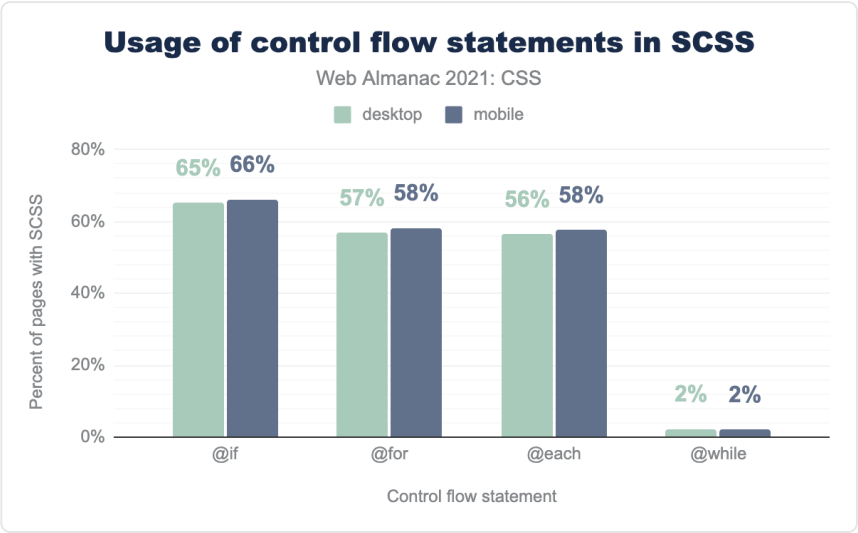


図1.66. もっとも一般的に使用されるSassのフロー制御構造です。

条件付きで動作させたいという要望は、`if()`関数がSass関数全体の15%で3位になったことからもわかります。

この同じ欲求は、`@if`のようなSassのフロー制御構造の使用でより明確に見ることができます。文字通り、すべてのSassスタイルシートの3分の2が`@if`を使用しており、半分以上が`@for`または`@each`（またはその両方）を使用しています。この人気の機能は、最近CSSに追加されました[11]。対照的に、`@while`構造を使用しているのはわずか2%です。

11. *https://drafts.csswg.org/css-conditional-4/#when-rule*

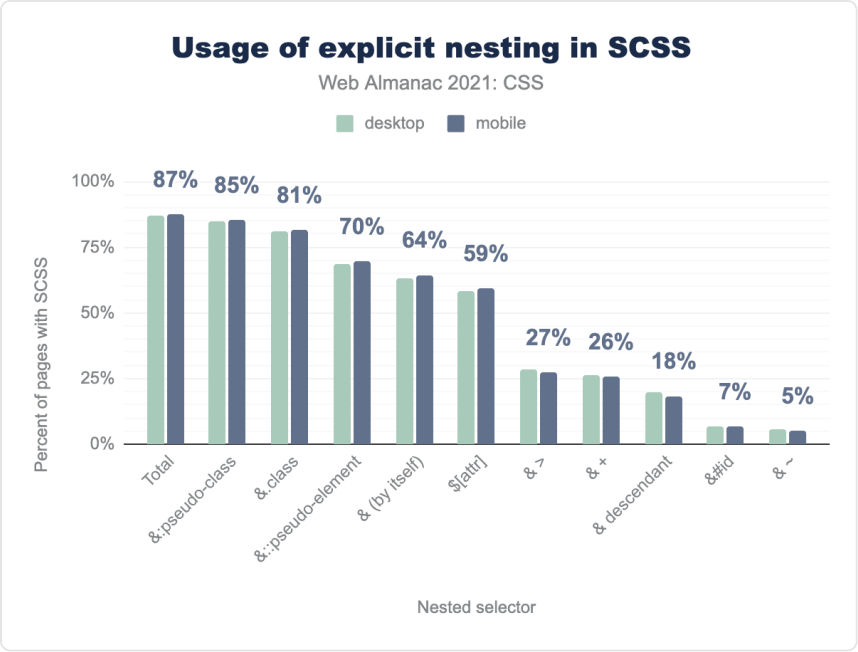


図1.67. Sassでのルールネストの普及について。

Sassのもう1つの大きな魅力は、他のルールの中にルールを入れ子にできるので、繰り返しのセレクタパターンを書く必要がないことです。この機能は、ネイティブのCSSで開発中[12]であり、その理由を我々の分析が示しています。すべてのSassスタイルシートの87%が、検出可能な形式のルールの入れ子を使用しています。特殊文字を必要としない暗黙的なネスティングは測定されませんでした。

結論

最後に、2021年版Web Almanacは、安定しつつも進化し続けるテクノロジーを物語っています。昨年のAlmanacと今年のAlmanacの間に大きな変化はほとんど見られず、明らかに成長しているプラクティスやウェブ機能もあれば、衰退し始めているものもありますが、全体的には継続性が非常に強く感じられました。

これはCSSが停滞しているということでしょうか？　そうではなくて、新しいレイアウト手法が増えたり、大きな新機能が開発されたりしていますが、その多くはプリプロセッサの領域で培われた手法に基づいています。私たちは、CSSが「解決済み」であるとか、「最善の

12. *https://www.w3.org/TR/css-nesting-1/*

方法がすでに確立されている」などとは考えていません。実務者が経験を積むことで、CSSという言語とCSSという実務の両方に変化が訪れるでしょう。その変化は、急激なものではなく徐々に、破壊的なものではなく着実なものになるかもしれませんが、これは成熟した技術に期待されることでもあります。

今後、CSSがどのように成長していくのか、楽しみにしています。

著者



Eric A. Meyer

@meyerweb meyerweb http://meyerweb.com/

Eric A. Meyerは、burger flipper、hardware jockey、大学のウェブマスター、初期のブロガー、オリジナルのCSS Samurai[13]の一人、CSS Working Group[14]のメンバー、コンサルタントとトレーナー、そしてNetscape[15]のスタンダード・エバンジェリストとして活躍してきました。現在は、Igalia[16]のDeveloper Advocateであり、An Event Apart[17]の共同創始者であるJeffrey Zeldman[18]と共に活動しています。中でもEricは、*Design For Real Life*[19]をSara Wachter-Boettcher[20]とA Book Apart[21]のために共同執筆し、*CSS: The Definitive Guide*[22]と、Estelle Weyl[23]がO'Reilly[24]また、初の公式なW3C[25]テストスイートを作成し、microformats[26]の作成を支援しました。



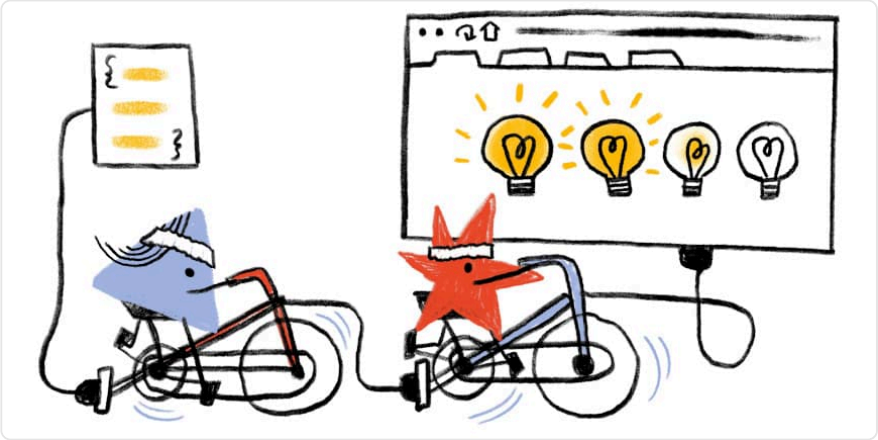
Shuvam Manna

@shuvam360 GeekBoySupreme https://shuvam.xyz

Shuvamはデザイナーであり、doodler[27]、writer[28]、shutterbug[29]であり、software tinker[30]でもあります。現在は、DeepSource[31]でデザインをしたり、Indie-Hackingで、Donethのようなプロジェクトに取り組み、コンピュータが人間とどのように相互作用するかの荒削りな部分を探求しています。

13. *https://archive.webstandards.org/css/members.html*
14. *https://en.wikipedia.org/wiki/CSS_Working_group*
15. *https://ja.wikipedia.org/wiki/%E3%83%8D%E3%83%83%E3%83%88%E3%82%B9%E3%82%B1%E3%83%BC%E3%83%97%E3%82%B3%E3%83%9F%E3%83%A5%E3%83%8B%E3%82%B1%E3%83%BC*
16. *http://igalia.com/*
17. *https://aneventapart.com/*
18. *http://zeldman.com/*
19. *https://abookapart.com/products/design-for-real-life*
20. *https://sarawb.com*
21. *https://abookapart.com/*
22. *http://meyerweb.com/eric/books/css-tdg/*
23. *http://standardista.com/*
24. *https://oreilly.com/*
25. *http://w3.org/*
26. *http://microformats.org/*
27. *https://www.behance.net/shuvammanna*
28. *https://distortedaura.wordpress.com/*
29. *https://www.instagram.com/the_distorted_aura/*
30. *https://github.com/GeekBoySupreme*
31. *https://deepsource.io*

部 I 章 2
JavaScript



***Nishu Goel** によって書かれた。*
Manuel Garcia、 *Minko Gechev*、 *Rick Viscomi*、 *Pankaj Parkar と Barry Pollard によってレビュ。*
Pankaj Parkar、 *Max Ostapenko と Rick Viscomi による分析。*
Rick Viscomi、 *Pankaj Parkar と Shaina Hantsis 編集。*
Sakae Kotaro によって翻訳された。

序章

ここ数年のJavaScript言語の進化のスピードと一貫性には目を見張るものがあります。以前は主にクライアントサイドで使用されていましたが、今ではサービスやサーバーサイドのツールを構築する世界において、非常に重要で尊敬すべき場所となっています。JavaScriptは、より高速なアプリケーションを作成できるだけでなく、ブラウザ内でサーバを実行可能[32]なほどに進化しています。

アプリケーションをレンダリングする際、ブラウザの中では、JavaScriptのダウンロードから解析、コンパイル、実行まで、さまざまなことが行われています。まずはその最初のステップとして、実際にページから要求されるJavaScriptの量を把握してみましょう。

32. *https://blog.stackblitz.com/posts/introducing-webcontainers/*

JavaScriptをどれだけ読み込むか？

「測定することは改善への鍵である」と言われています。アプリケーションでのJavaScriptの使用を改善するためには、検出されているJavaScriptのうち、実際に必要とされるものがどれくらいあるかを測定する必要があります。ここでは、Web設定においてJavaScriptが果たす重要な役割を考慮して、ページあたりのJavaScriptバイト数の分布を理解することにしましょう。

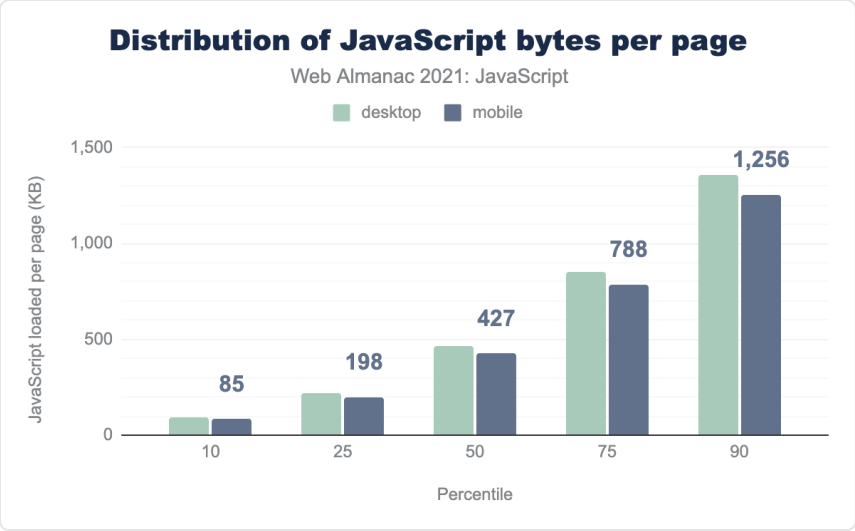


図2.1. ページごとに読み込まれるJavaScriptの量の分布。

50パーセンタイル（中央値）のモバイルページでは427KBのJavaScriptがロードされるのに対し、デスクトップデバイスでロードされるページの中央値では463KBが送信されます。

2019年の結果[33]と比較すると、デスクトップではJavaScriptの使用率が18.4%増加し、モバイルでは18.9%増加していることがわかります。長期的な傾向として、より多くのJavaScriptを使用するようになっていますが、CPUの作業が増えることで、アプリケーションのレンダリング速度を低下する可能性があります。なお、これらの統計は、圧縮応答が可能な転送バイト数を表しているため、実際のCPUコストはかなり高くなる可能性があることをご了承ください。

では、実際にどの程度のJavaScriptがページに読み込まれる必要があるのかを見てみましょう。

33. *https://almanac.httparchive.org/ja/2019/javascript#fig-2*

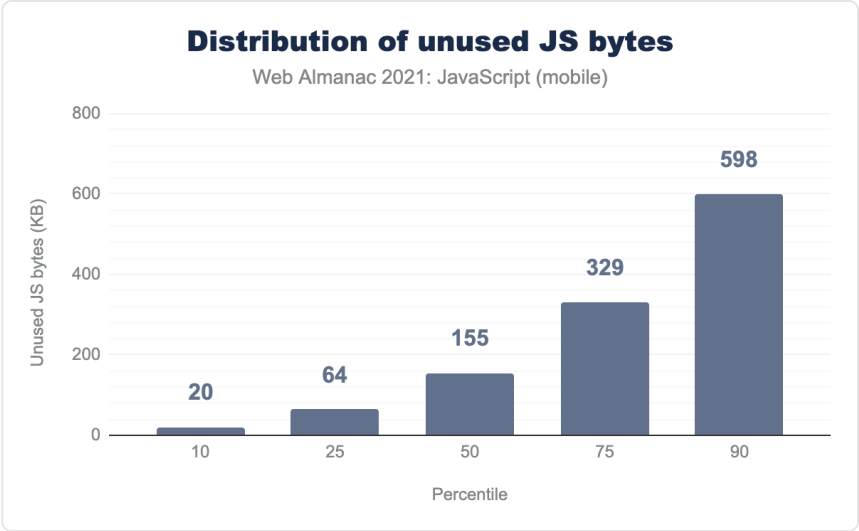


図2.2. モバイルにおけるJavaScriptの未使用バイト量の分布。

Lighthouseによると、モバイルページの中央値では、155KBの未使用のJavaScriptがロードされています。また、90パーセンタイルでは、598KBのJavaScriptが使われていません。

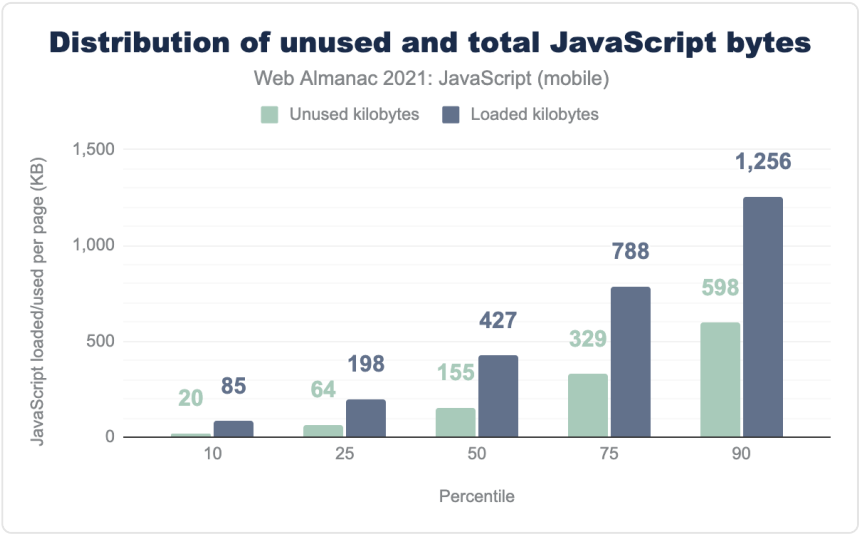


図2.3. モバイルページにおける未使用および総JavaScriptバイト数の分布。

図2.4. 読み込まれたJavaScriptのうち未使用の割合。

言い換えれば、中央値のモバイルページでは、36.2%のJavaScriptバイトが使用されていません。JavaScriptがページの最大コンテンツの描画[34]（LCP）に与える影響を考えると、とくにデバイスの性能やデータプランが限られているモバイルユーザーにとっては、CPUサイクルや他の重要なリソースがムダに消費されていることを考えると、これは非常に大きな数字です。このようなムダは、大規模なフレームワークやライブラリに同梱されている、使用されていない多くの定型的なコードの結果である可能性があります。

サイトオーナーは、Lighthouseを使って使われていないJavaScript[35]をチェックし、ベストプラクティスにしたがって使われていないコード[36]を削除することで、ムダなJavaScriptバイトの割合を減らすことができます。

ページあたりのJavaScriptリクエスト数

Webページのレンダリングが遅くなる要因のひとつとして、ページ上で行われるリクエスト、とくにリクエストをブロックしている場合が考えられます。そこで、デスクトップとモバイルの両方で、1ページあたりのJavaScriptリクエスト数を調べてみることにしました。

34. https://web.dev/i18n/ja/optimize-lcp/
35. https://web.dev/unused-javascript/
36. https://web.dev/i18n/ja/remove-unused-code/

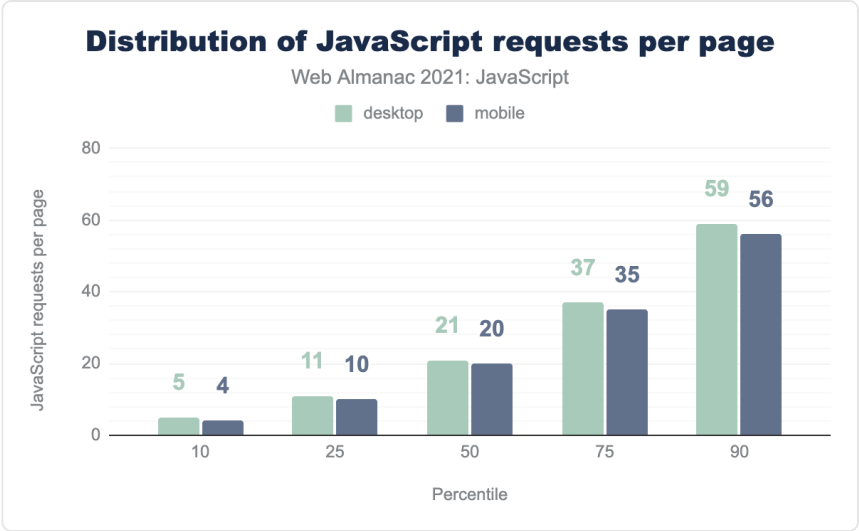


図2.5. ページあたりのJavaScriptリクエスト数の分布。

デスクトップページの中央値では、21個のJavaScriptリソース（`.js`および`.mjs`のリクエスト）がロードされ、90パーセンタイルでは59個のリソースがロードされます。

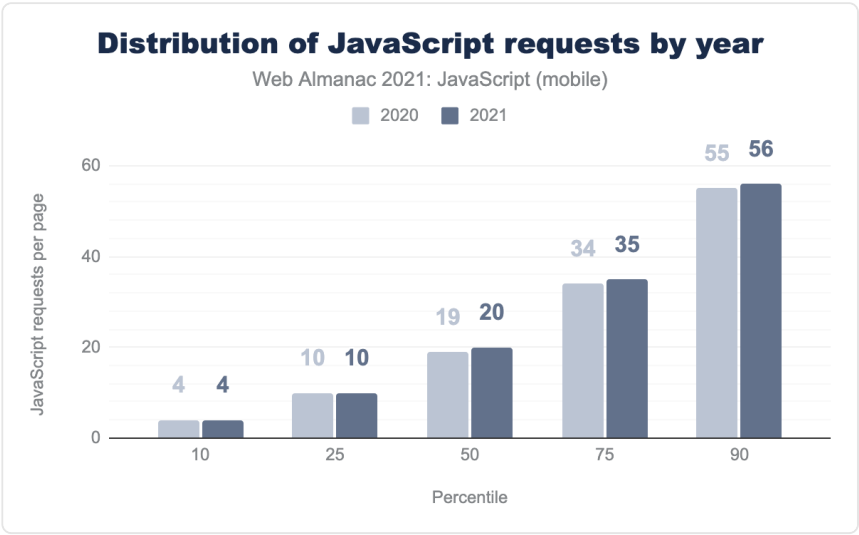


図2.6. 1ページあたりのJavaScriptリクエスト数の年別分布。

昨年の結果[37]と比較すると、2021年に要求されたJavaScriptリソースの数はわずかに増加しており、ロードされたJavaScriptリソースの数の中央値はデスクトップページで20、モバイルで19でした。

傾向としては、1ページに読み込まれるJavaScriptリソースの数が徐々に増えています。これはJavaScriptのリクエスト数が少ないとパフォーマンスが向上する場合もあれば、そうでない場合もあることを考えると、実際に数値を上げるべきか、下げるべきか悩むところです。

ここで、最近のHTTPプロトコルの進歩により、JavaScriptのリクエスト数を減らしてパフォーマンスを向上させようという考えが不正確になってきました。HTTP/2とHTTP/3の導入により、HTTPリクエストのオーバーヘッドが大幅に削減されたため、同じリソースをより多くのリクエストで要求することは、必ずしも悪いことではなくなりました。これらのプロトコルの詳細については、HTTPの章を参照してください。

JavaScriptはどのように要求されるのですか？

JavaScriptはさまざまな方法でページに読み込まれますが、そのリクエスト方法はページのパフォーマンスに影響を与えます。

`module`と`nomodule`

Webサイトを読み込む際、ブラウザはHTMLをレンダリングし、適切なリソースを要求します。ブラウザは、ページを効果的にレンダリングして機能させるために、コード内で参照されるポリフィルを消費します。アロー関数式[38]や非同期関数[39]のような新しい構文をサポートしているモダンブラウザは、動作させるために大量のポリフィルを必要としませんし、その必要もありません。

この時、differential loadingが活躍します。`type="module"`属性を指定すると、モダンブラウザにはモダンな構文のバンドルが提供され、ポリフィルがあったとしても少なくなります。同様に、モジュールをサポートしていない古いブラウザには、`type="nomodule"`属性を指定することで、必要なポリフィルとトランスパイルされたコード構文を持つバンドルが提供されます。HTMLにモジュールを適用する[40]の続きを読んでください。

これらの属性の採用状況をデータで見てみましょう。

37. *https://almanac.httparchive.org/ja/2020/javascript#request-count*
38. *https://developer.mozilla.org/docs/Web/JavaScript/Reference/Functions/Arrow_functions*
39. *https://developer.mozilla.org/docs/Web/JavaScript/Reference/Statements/async_function*
40. *https://developer.mozilla.org/docs/Web/JavaScript/Guide/Modules#applying_the_module_to_your_html*

属性	デスクトップ	モバイル
`module`	4.6%	4.3%
`nomodule`	3.9%	3.9%

図2.7. デスクトップとモバイルのクライアントでの負荷のかかり方の違いの分布。

デスクトップページの4.6%が`type="module"`属性を使用しているのに対し、モバイルページでは3.9%しか`type="nomodule"`を使用していません。これは、モバイルのデータセットの方がはるかに大きいため、最新の機能を使っていない可能性のある「ロングテール」のウェブサイトが多く含まれていることが原因と考えられます。

なお、IE 11ブラウザのサポート終了[41]に伴い、エバーグリーンブラウザは最新のJavaScript構文をサポートしているため、ディファレンシャルローディングは適用しにくくなっています。たとえば、Angularフレームワークでは、Angular v13でレガシーブラウザのサポートの削除を[42]2021年11月に発表しました。

`async`と`defer`

JavaScriptの読み込みは、非同期または遅延の指定がない限り、レンダリングの妨げになる可能性があります。多くの場合、最初のレンダリングにJavaScript（もしくはその一部）が必要となるため、これがパフォーマンス低下の一因となっています。

しかし、JavaScriptを非同期に、あるいは遅延して読み込むことで、このような体験を改善できる場合があります。`async`と`defer`の両属性は、非同期的にスクリプトをロードします。`async`属性を持つスクリプトは定義された順番に関係なく実行されますが、`defer`はドキュメントが完全に解析された後にのみスクリプトを実行するので、指定された順番で実行されることが保証されます。実際に、ブラウザで要求されたJavaScriptにこれらの属性を指定しているページがどれだけあるかを見てみましょう。

41. *https://docs.microsoft.com/en-us/lifecycle/announcements/internet-explorer-11-support-end-dates*
42. *https://github.com/angular/angular/issues/41840*

属性	デスクトップ	モバイル
`async`	89.3%	89.1%
`defer`	48.1%	47.8%
両方	35.7%	35.6%
どちらでもない	10.3%	10.4%

図2.8. `async` と `defer` を使用しているページの割合。

昨年の結果では、同じスクリプトに `async` と `defer` の両方の属性を使用しているウェブサイトがあるというアンチパターンが観察されました。これは、ブラウザがサポートしている場合は `async` にフォールバックし、IE8とIE9のブラウザでは `defer` を使用するというものです。しかし、ほとんどのサイトでは、サポートされているすべてのブラウザで `async` が優先されるため、現在ではこの方法は必要ありません。このパターンでは、ページが読み込まれるまで待つのではなく、HTMLの解析が中断されます。この使用頻度は非常に高く、モバイルページの 11.4%[43] には、`async` と `defer` 属性を持つスクリプトが少なくとも1つ使用されていました。根本原因[44]が判明し、今後このような使い方をしない[45]というアクションアイテムも降ろされたのです。

35.6%

図2.9. 同一スクリプト上で `async` と `defer` 属性が設定されているモバイルページの割合。

今年は、モバイルページの35.6%が `async` と `defer` 属性を併用していることがわかりました。昨年との大きな違いは、初期のHTMLの静的なコンテンツを解析するのではなく、実行時に属性の使用状況を測定するように方法を改善したことによるものです。この差は、多くのページが、ドキュメントがすでに読み込まれた後に、これらの属性を動的に更新していることを示しています。たとえば、あるウェブサイトでは、次のようなスクリプトが含まれていることが判明しました。

```
<!-- Piwik -->
<script type="text/javascript">
```

43. https://almanac.httparchive.org/ja/2020/javascript#how-do-we-load-our-javascript
44. https://x.com/rick_viscomi/status/1331735748060524551?s=20
45. https://x.com/Kraft/status/1336772912414601224?s=20

```
  (function() {
      var d=document, g=d.createElement('script'),
s=d.getElementsByTagName('script')[0];
      g.type='text/javascript'; g.async=true; g.defer=true;
g.src=u+'piwik.js'; s.parentNode.insertBefore(g,s);
  })();
</script>
<!-- End Piwik Code -->
```

さて、Piwikとは何でしょうか？そのWikipediaの項目によると。

> *Matomo（旧Piwik）は、国際的な開発者チームによって開発されたフリーでオープンソースのウェブ解析アプリケーションで、PHP/MySQLのウェブサーバー上で動作します。1つまたは複数のウェブサイトへのオンライン訪問を追跡し、分析のためにこれらの訪問に関するレポートを表示します。2018年6月現在、Matomoは145万5千以上のウェブサイトで使用されており、これは既知のトラフィック分析ツールを持つすべてのウェブサイトの1.3%にあたります...*
>
> *– ウィキペディアのMatomo (ソフトウェア)*[46]

この情報は、我々が観測した増加の多くは、類似のマーケティングおよび分析プロバイダーが、これらの`async`および`defer`スクリプトを以前に検出されたよりも遅くページに動的に注入することによるものである可能性を強く示唆しています。



図2.10. `async`と`defer`属性を併用しているスクリプトの割合。

多くのページがこのアンチパターンを使用しているにもかかわらず、同じscript要素で`async`と`defer`の両方を使用しているスクリプトは全体の2.6%しかないことがわかりました。

46. https://ja.wikipedia.org/wiki/Matomo

ファーストパーティとサードパーティ

JavaScriptをどれだけ読み込むか？のセクションで、モバイルページのJavaScriptリクエスト数の中央値が20であることを思い出してください。このセクションでは、ファーストおよびサードパーティのJavaScriptリクエストの内訳を見てみましょう。

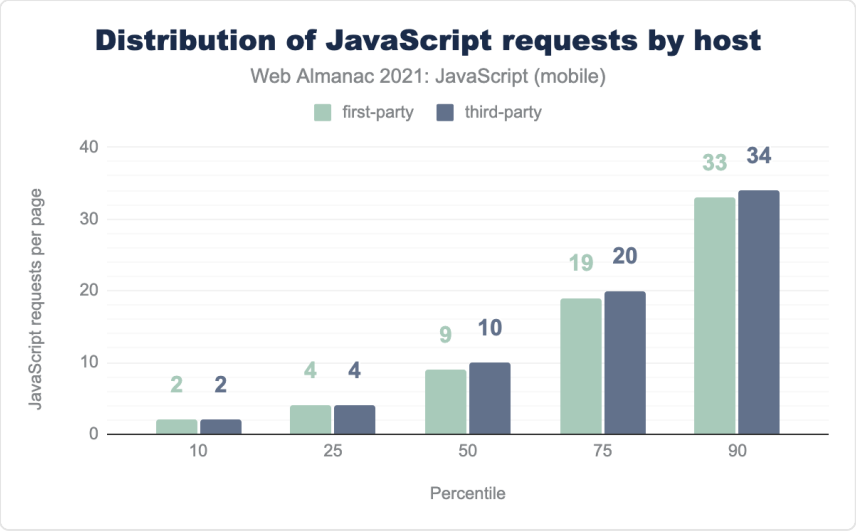


図2.11. モバイルページ1ページあたりのJavaScriptリクエスト数のホスト別分布。

モバイルページの中央値では、サードパーティ製リソースへのリクエストが10件、ファーストパーティ製リクエストが9件となっています。この差は、90パーセンタイルになるほど大きくなります。モバイルページのファーストパーティリソースへのリクエストは33件ですが、モバイルページのサードパーティリソースへのリクエストは34件に上ります。明らかに、サードパーティ製リソースのリクエスト数は、ファーストパーティ製リソースのリクエスト数よりも常に一歩リードしています。

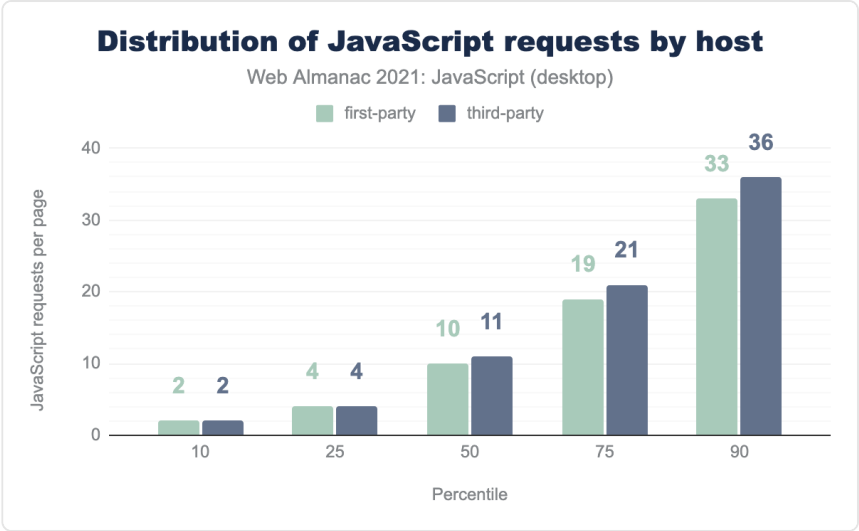


図2.12. デスクトップページあたりのJavaScriptリクエスト数のホスト別分布。

デスクトップページの中央値では、サードパーティ製リソースを11個要求しているのに対し、ファーストパーティ製は10個です。サードパーティ製リソースがもたらすパフォーマンスと信頼性のリスク[47]にかかわらず、デスクトップページとモバイルページの両方で、サードパーティ製スクリプトが一貫して好まれているようです。この効果は、サードパーティのスクリプトがウェブに与える便利なインタラクティビティ機能[48]によるものと考えられます。

とはいえ、サイトオーナーは、サードパーティ製のスクリプトがパフォーマンス良くロードされていることを確認する必要があります[49]。Harry Roberts[50]は、さらに一歩進んで、サードパーティのパフォーマンスとレジリエンスをストレステストすることを提唱しています[51]。

`preload`と`prefetch`

ページがレンダリングされると、ブラウザは与えられたリソースをダウンロードしますが、リソースヒントを使って、ブラウザが使用するいくつかのリソースのダウンロードを他のリソースよりも優先させます。`preload`ヒントは、現在のページで必要となる、より高い優先順位のリソースをダウンロードするようブラウザに指示します。しかし`prefetch`ヒントは、リソースがしばらくしてから必要になる可能性があり（将来のナビゲーションに役立つ）、ブラウザにその能力があるときにリソースを取得し、必要になった時すぐに利用でき

47. https://css-tricks.com/potential-dangers-of-third-party-javascript/
48. https://developers.google.com/web/fundamentals/performance/critical-rendering-path/adding-interactivity-with-javascript
49. https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/loading-third-party-javascript
50. https://x.com/csswizardry
51. https://csswizardry.com/2017/07/performance-and-resilience-stress-testing-third-parties/

るようにしたほうがよいということをブラウザに伝えます。これらの機能がどのように使われるかについては、リソースヒントの章で詳しく説明しています。

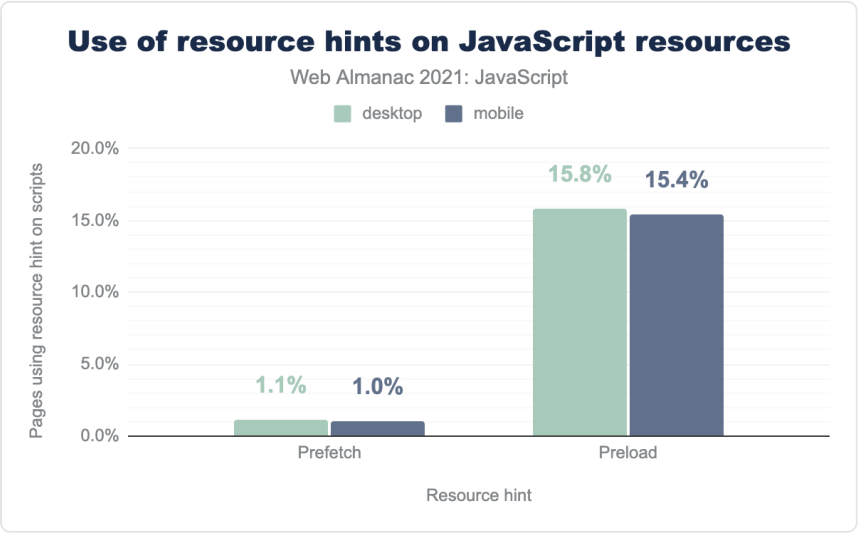


図2.13. JavaScriptリソースでのリソースヒントの使用

`preload`ヒントは、15.4%のモバイルページでJavaScriptのロードに使用されていますが、`prefetch`ヒントは1.0%のモバイルページでしか使用されていません。デスクトップページでは、それぞれ15.8%と1.1%がJavaScriptリソースの読み込みにこれらのリソースヒントを使用しています。

また、ページごとに何個の`preload`と`prefetch`ヒントが使用されているかを確認することも、ヒントの影響に影響を与えるので便利です。たとえば、レンダリング時に読み込まれるリソースが5つあり、その5つすべてが`preload`ヒントを使用している場合、ブラウザはすべてのリソースを優先しようとしますが、これでは`preload`ヒントがまったく使用されていないのと同じように機能してしまいます。

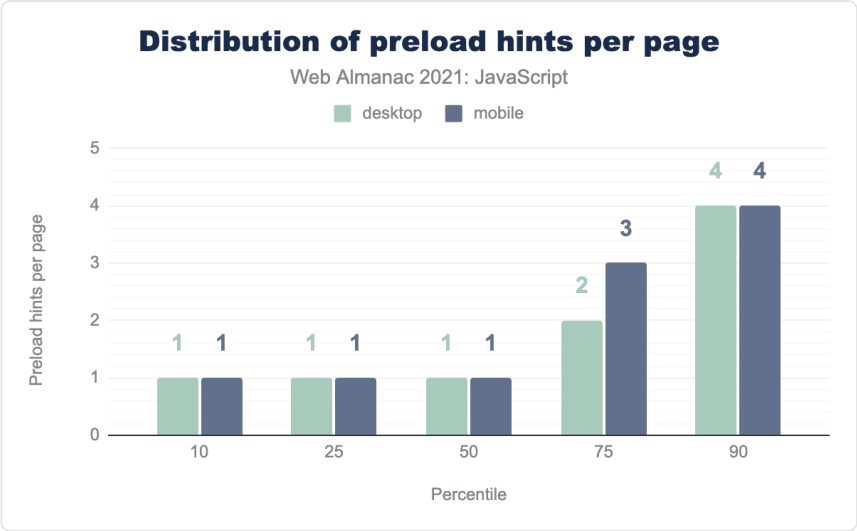


図2.14. ページごとのJavaScriptリソースのpreloadヒントの分布。

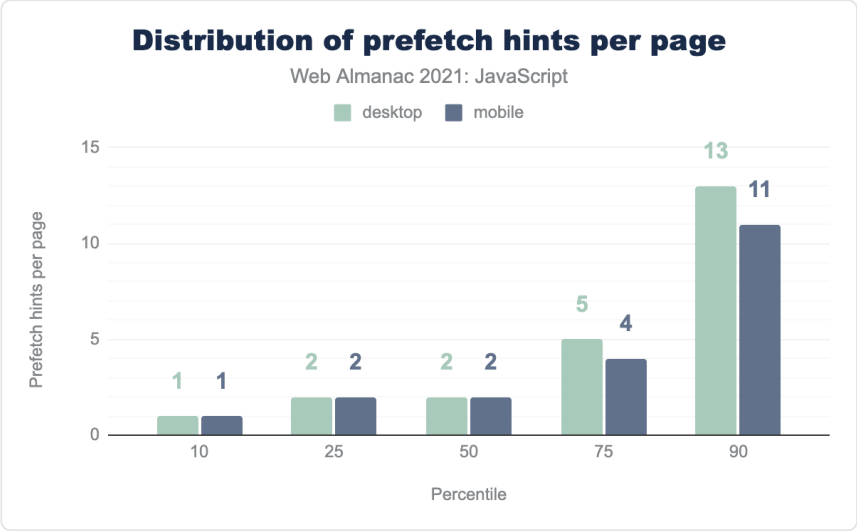


図2.15. ページごとのJavaScriptリソースのprefetchヒントの分布。

中央値のデスクトップページは、1つのJavaScriptリソースを `preload` ヒントでロードし、2つのJavaScriptリソースを `prefetch` ヒントでロードします。

ヒント	2020	2021
preload	1	1
prefetch	3	2

図2.16. モバイルページあたりのJavaScriptリソースに対する `preload` および `prefetch` ヒントの数の中央値の前年比比較。

モバイルページあたりの `preload` ヒントの数の中央値は変わりませんが、 `prefetch` ヒントの数は1ページあたり3つから2つに減りました。中央値では、これらの結果はモバイルページとデスクトップページの両方で同じであることに注意してください。

JavaScriptはどのように配信されるのですか？

JavaScriptのリソースは、圧縮と最小化によってネットワーク上でより効率的に読み込むことができます。このセクションでは、両技術がどの程度効果的に利用されているかを理解するために、両技術の使用方法を探っていきます。

圧縮

圧縮とは、ネットワーク上で転送されるリソースのファイルサイズを小さくするプロセスです。圧縮率の高いJavaScriptリソースのダウンロード時間を短縮するには、この方法が有効です。たとえば、このページで読み込まれている `almanac.js` スクリプトは28KBですが、圧縮のおかげで転送時間はわずか9KBとなりました。ウェブ上でリソースが圧縮される方法については、圧縮の章で詳しく説明しています。

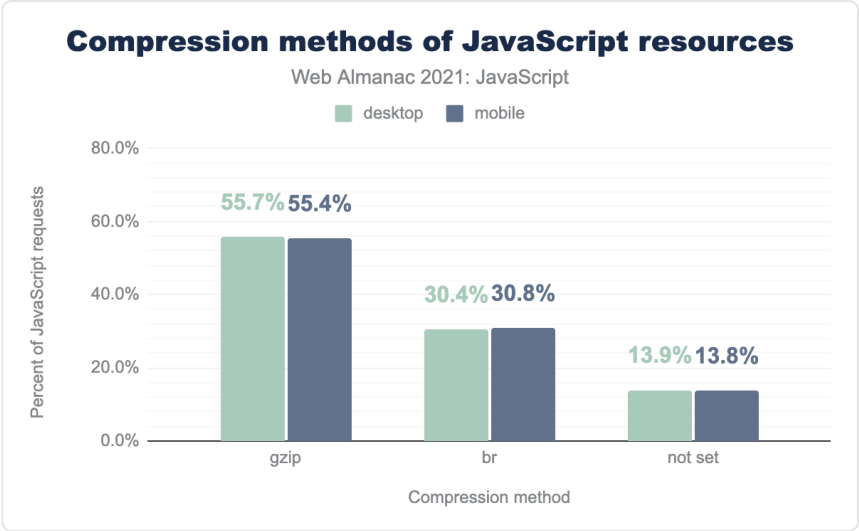


図2.17. JavaScriptリソースの圧縮方法の採用。

ほとんどのJavaScriptリソースは、Gzip[52]、Brotli[53] (br) を使って圧縮されているか、あるいはまったく圧縮されていない（設定されていない）かのいずれかです。モバイルのJavaScriptリソースの55.4%がGzipで圧縮されているのに対し、30.8%がBrotliで圧縮されています。

興味深いことに、2019年のJavaScript圧縮の状況[54]と比較すると、Gzipは10ポイント近く下がり、Brotliは16ポイント上がっています。この傾向は、Gzipと比較してBrotliが提供する圧縮レベルは高く、より小さいサイズのファイルを重視するようになったことを示しています。

この変化を説明するために、私たちはファーストリソースとサードパーティリソースの圧縮方法を分析しました。

52. *https://www.gnu.org/software/gzip/manual/gzip.html*
53. *https://github.com/google/brotli*
54. *https://almanac.httparchive.org/ja/2019/javascript#fig-10*

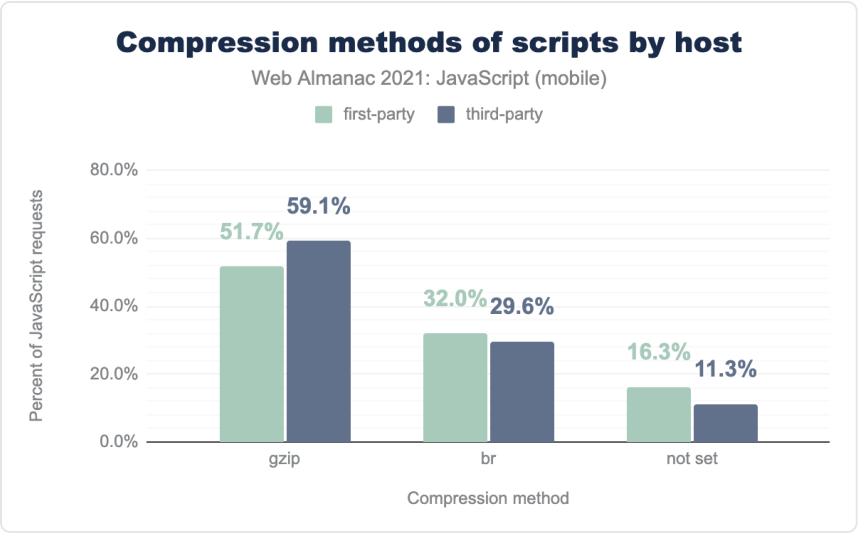


図2.18. モバイルページにおけるファーストおよびサードパーティのJavaScriptリソースを圧縮する方法の採用。

モバイルページ上のサードパーティスクリプトの59.1%がGzip圧縮、29.6%がBrotliで圧縮されています。ファーストパーティースクリプトを見ると、Gzip圧縮は51.7%ですが、Brotliでは32.0%にとどまります。また、圧縮方法が定義されていないサードパーティスクリプトが11.3%存在します。

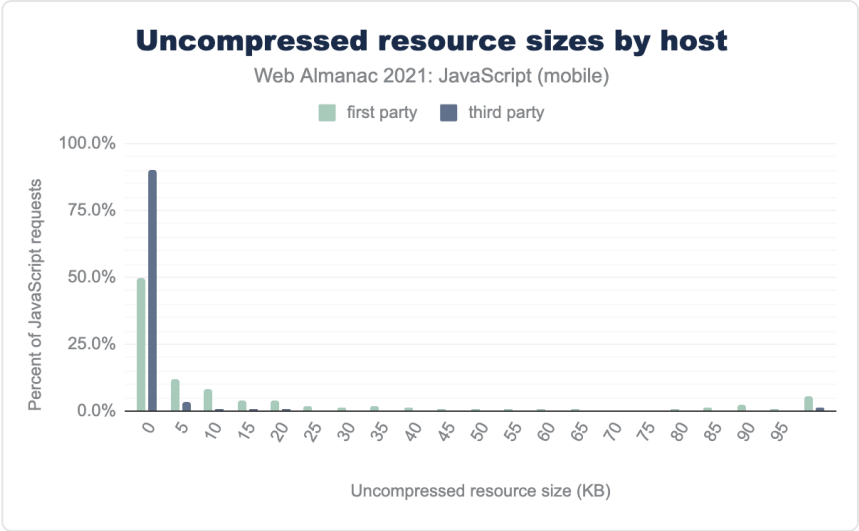


図2.19. ファーストパーティとサードパーティの非圧縮のリソース。

圧縮されていないサードパーティのJavaScriptリソースの90%は5KB未満ですが、ファーストパーティからのリクエストは少し遅れています。このことは、多くのJavaScriptリソースが圧縮されない理由を説明するのに役立つかもしれません。小さなリソースを圧縮することで得られる利益は減少するため、小さなスクリプトはネットワーク上で数バイトを節約することによるパフォーマンス上のメリットよりも、サーバー側での圧縮とクライアント側での解凍によるリソース消費の方が、大きくなる可能性があります。

最小化

圧縮はネットワーク上のJavaScriptリソースの転送サイズを変更するだけですが、ミニフィケーションは実際にコード自体を小さくし、より効率的にします。これにより、スクリプトの読み込み時間が短縮されるだけでなく、クライアントがスクリプトを解析するのに費やす時間も短縮されます。

unminified JavaScript[55]のLighthouse監査では、最小化の機会を強調しています。

55. *https://web.dev/unminified-javascript/*

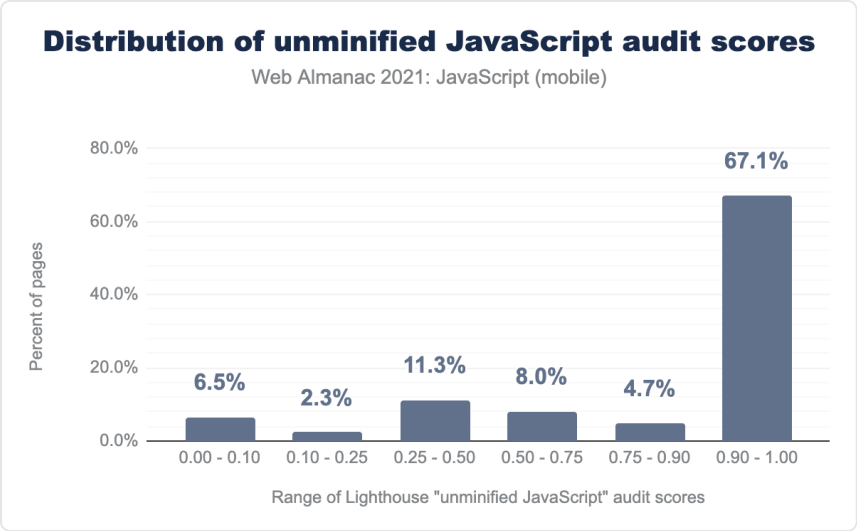


図2.20. 終了していないJavaScriptの監査スコアの分布

ここで、0.00は最悪のスコアを表し、1.00は最高のスコアを表します。モバイルページの67.1%は、オーディットスコアが0.9から1.0の間にあります。つまり、JavaScriptの未最適化のスコアが0.9よりも悪く、コードの最小化をより有効に活用できるページがまだ30%以上あるということです。2020年版[56]の結果と比較すると、「未処理のJS」のスコアが0.9から1.0の間にあるモバイルページの割合は10ポイント減少しました。

今年のスコアが悪化した理由を理解するために、より深く掘り下げて、1ページあたり何バイトが未処理であるかを見てみましょう。

56. *https://almanac.httparchive.org/ja/2020/javascript#fig-16*

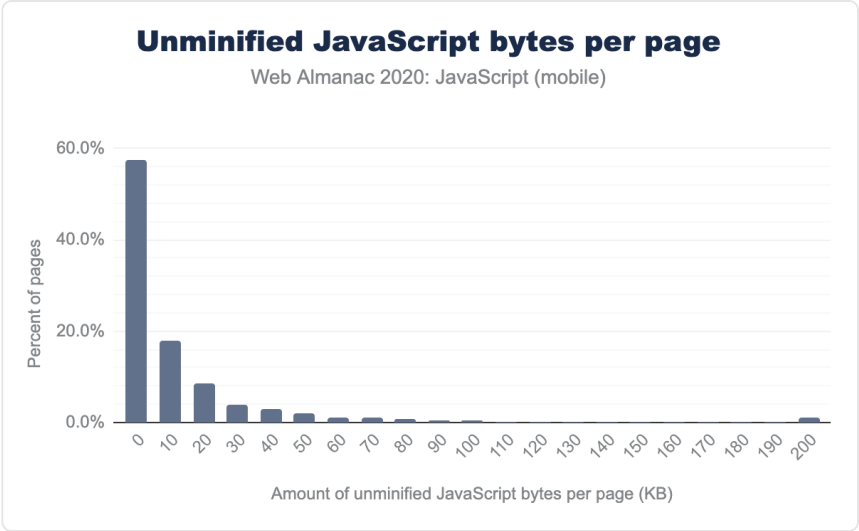


図2.21. ページごとの未処理のJavaScriptの量の分布（単位：KB）。

57.4%のモバイルページでは、Lighthouse監査で報告されたように、0KBの未処理のJavaScriptが使用されています。17.9%のモバイルページでは、0〜10KBの未完成のJavaScriptが使用されています。残りのページでは、未処理のJavaScriptのバイト数が増加しており、前のグラフで「未処理のJavaScript」の監査スコアが低いページに対応しています。

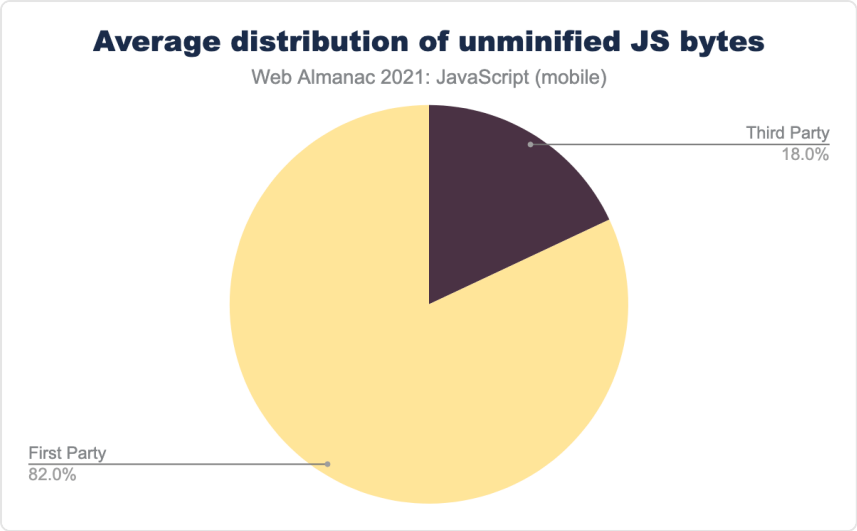


図2.22. 未処理のJavaScriptバイトの平均的な分布。

ホスト別に未処理のJavaScriptリソースを分類したところ、平均的なモバイルページの未処理のJavaScriptバイトの82.0%は、実際にはファーストパーティのスクリプトから来ていることがわかりました。

ソースマップ

ソースマップ[57]とは、JavaScriptのリソースと一緒に送信されるヒントで、ブラウザがそのリソースを最小化してソースコードにマッピングできます。ウェブ開発者にとっては、本番環境でのデバッグにとくに役立ちます。

0.1%

図2.23. `SourceMap` ヘッダーを使用しているモバイルページの割合。

スクリプトリソースに `ソースマップ` レスポンスヘッダーを使用しているモバイルページは、わずか0.1%です。この極端に少ない割合の理由としては、ソースマップを使ってオリジナルのソースコードを本番に出すことを選択するサイトは少ないことが考えられます。

57. https://developer.mozilla.org/docs/Tools/Debugger/How_to/Use_a_source_map



図2.24. `SourceMap` ヘッダーを使用するモバイルページのJavaScriptリソースのうち、ファーストパーティリソースの割合。

JavaScriptリソースの `SourceMap` 使用率の98.0%がファーストパーティに起因しています。モバイルページでヘッダーを持つスクリプトのうち、サードパーティのリソースはわずか2.0%です。

ライブラリーとフレームワーク

多くの新しいライブラリやフレームワークが採用され、開発者とユーザーの体験をそれぞれ独自に改善することが期待されているため、JavaScriptの使用量は年々増加しているようです。あまりにも広く普及しているため、開発者が追いつくのに苦労している様子を表す*framework fatigue*という言葉が作られました。このセクションでは、現在ウェブ上で使用されているJavaScriptライブラリやフレームワークの人気度を見てみましょう。

ライブラリーの利用

ライブラリやフレームワークの使用状況を把握するために、HTTP ArchiveはWappalyzerを使用して、ページで使用されている技術を検出しています。

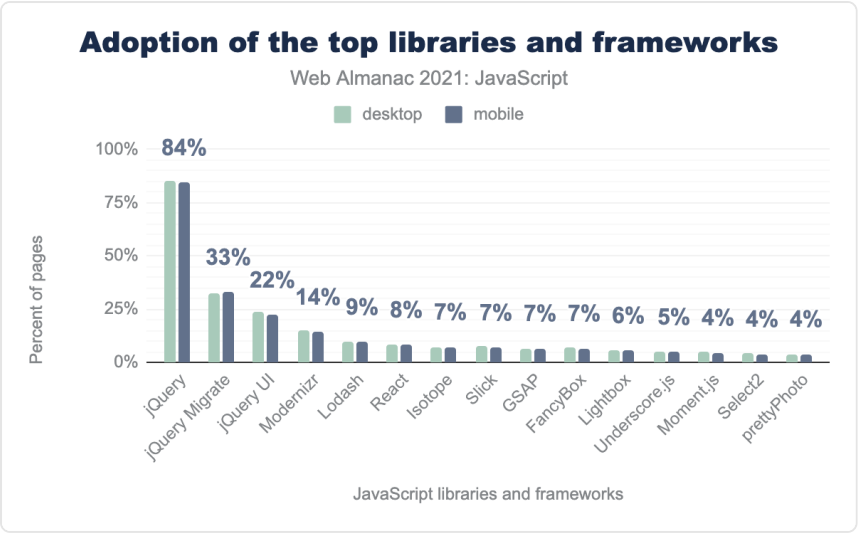


図2.25. JavaScriptのライブラリやフレームワークの使用方法。

jQueryは依然としてもっとも人気のあるライブラリで、モバイルページの84%という驚異的な割合で使用されています。Reactの使用率は、昨年の4%から8%に急上昇しています。Reactの増加は、最近のWappalyzerの検出力向上[58]による部分的なものであり、実際の採用状況の変化を必ずしも反映していない可能性があります。また、jQueryを使用したIsotopeが7%のページで見られ、RequireJSはわずか2%のページで上位から脱落していることも注目に値します。

2021年になっても、なぜjQueryが主流なのか不思議に思うかもしれません。これには主に2つの理由があります。まず、前年よりも強調されている[59]ように、ほとんどのWordPress[60]サイトがjQueryを使用しています。CMSの章によると、WordPressはすべてのWebサイトの約3分の1で使用されているため、これがjQuery採用の大きな割合を占めています。また、他の主要なJavaScriptライブラリの中にも、何らかの形でjQueryを利用しているものがあり、間接的にjQueryが採用されていると考えられます。



図2.26. もっとも普及しているjQueryのバージョンです。

もっともよく使われているjQueryのバージョンは3.5.1で、モバイルページの21.3%で使用さ

58. https://github.com/AliasIO/wappalyzer/issues/2450
59. https://almanac.httparchive.org/ja/2019/javascript#open-source-libraries-and-frameworks
60. https://wordpress.org/

れています。次によく使われているjQueryのバージョンは1.12.4で、モバイルページの14.4%で使われています。バージョン3.0への飛躍は、2020年に行われたWordPress coreへの変更[61]により、jQueryのデフォルトバージョンが1.12.4から3.5.1にアップグレードされたことで説明できます。

併用するライブラリ

では、人気のフレームワークやライブラリが、同じページでどのように併用されているかを見てみましょう。

フレームワークとライブラリ	デスクトップ	モバイル
jQuery	16.8%	17.4%
jQuery, jQuery Migrate	8.4%	8.7%
jQuery, jQuery UI	4.0%	3.7%
jQuery, jQuery Migrate, jQuery UI	2.6%	2.5%
Modernizr, jQuery	1.6%	1.6%
FancyBox, jQuery	1.1%	1.1%
Slick, jQuery	1.2%	1.1%
Lightbox, jQuery	1.1%	0.8%
React, jQuery, jQuery Migrate	0.9%	0.9%
Modernizr, jQuery, jQuery Migrate	0.8%	0.9%

図2.27. 併用されているJavaScriptフレームワークとライブラリの上位の組み合わせ。

もっとも広く使われているJavaScriptのライブラリとフレームワークの組み合わせは、実際には複数のライブラリで構成されているわけではありません！jQueryを単独で使用した場合、モバイルページの17.4%に使用されています。次に多い組み合わせは「jQuery」と「jQuery Migrate」で、モバイルページの8.7%で使用されている。実際、ライブラリとフレームワークの組み合わせのトップ10には、すべてjQueryが含まれている。

セキュリティの脆弱性

JavaScriptライブラリを使用するには、それなりのメリットとデメリットがあります。これ

61. https://wptavern.com/major-jquery-changes-on-the-way-for-wordpress-5-5-and-beyond

らのライブラリを使用する際の欠点として、古いバージョンにはCross Site Scripting[62]（XSS）のようなセキュリティリスクが含まれている可能性があります。Lighthouseは、ページで使用されているJavaScriptライブラリを検出し、そのバージョンがオープンソースのSnyk脆弱性データベース[63]で既知の脆弱性を持っていた場合、監査に失敗します。

63.9%

図2.28. セキュリティ上の脆弱性があるライブラリを持つモバイルページの割合。

モバイルページの63.9%が、既知のセキュリティ脆弱性を持つJavaScriptライブラリまたはフレームワークを使用しています。この数字は、昨年の83.5%から[64]減少しています。

62. *https://owasp.org/www-community/attacks/xss/*
63. *https://snyk.io/vuln?type=npm*
64. *https://almanac.httparchive.org/ja/2020/javascript#fig-30*

ライブラリやフレームワーク	ページ数の割合
jQuery	57.6%
Bootstrap	12.2%
jQuery UI	10.5%
Underscore	6.4%
Lo-Dash	3.1%
Moment.js	2.3%
GreenSock JS	1.8%
Handlebars	1.3%
AngularJS	1.0%
Mustache	0.7%
jQuery Mobile	0.5%
Dojo	0.5%
Angular	0.4%
Vue	0.2%
Knockout	0.2%
Highcharts	0.1%
Next.js	0.0%
React	0.0%

図2.29. 脆弱なバージョンのJavaScriptライブラリまたはフレームワークを含むことが判明したモバイルページの割合。

モバイルページの割合をライブラリとフレームワーク別に分けてみると、脆弱性の減少にはjQueryが大きく関わっていることがわかります。今年のJavaScriptの脆弱性は、jQueryを使用しているページの57.6%で発見されたのに対し、昨年は80.9%[65]でした。本章の2020年版でTim Kadlec[66]が予測[67]したように、*「もし人々がそれらの古くて脆弱なバージョンのjQueryから移行できれば、既知の脆弱性を持つサイトの数は激減するでしょう」*。WordPressは、

65. https://almanac.httparchive.org/ja/2020/javascript#fig-31
66. https://almanac.httparchive.org/ja/2020/contributors#tkadlec
67. https://almanac.httparchive.org/ja/2020/javascript#fig-31

jQueryのバージョン1.12.4からより安全なバージョン3.5.1に移行したことで、JavaScriptの脆弱性が確認されたページの割合が20ポイント減少しました。

JavaScriptはどのように使用されていますか？

さて、JavaScriptの取得方法を見てきましたが、何のために使うのでしょうか？

AJAX

JavaScriptは、サーバーと通信してさまざまな形式の情報を非同期的に受け取るために使用されます。*Asynchronous JavaScript and XML* (AJAX)は、一般的にデータの送受信に使用され、XML以外にもJSON、HTML、テキスト形式などをサポートしています。

ウェブ上でデータを送受信する方法が複数ある中で、1ページあたりに送信される非同期リクエストの数を見てみましょう。

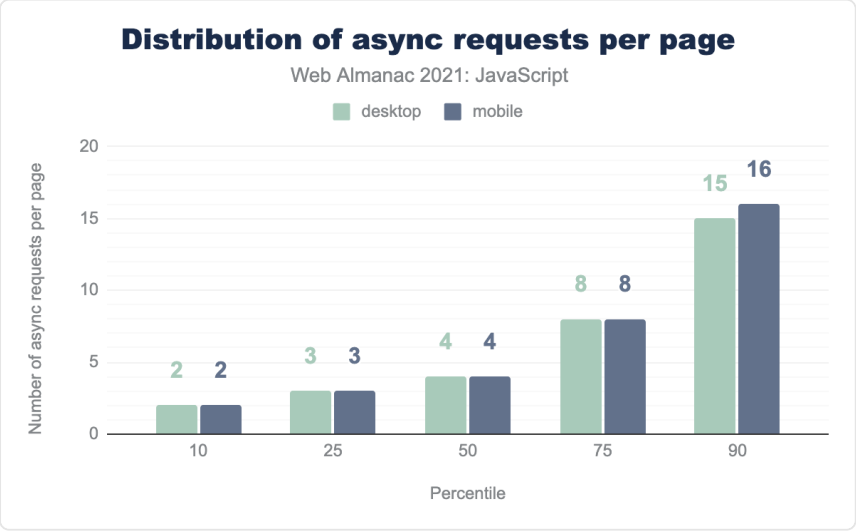


図2.30. ページあたりの非同期リクエスト数の分布。

モバイルページの中央値は4つの非同期リクエストを行っています。ロングテールを見ると、デスクトップページの非同期リクエストの最大数は623で、最大のモバイルページの867の非同期リクエストに抜かれています！

非同期型のAJAXリクエストの代わりに、同期型のリクエストがあります。リクエストをコールバックに渡すのではなく、リクエストが完了するまでメインスレッドをブロックしま

す。

しかし、パフォーマンスやユーザー体験が、低下する可能性があるため、このような行為は推奨されません[68]。また、多くのブラウザでは、このような使用方法についてすでに警告を発しています。今でも同期型のAJAXリクエストを使用しているページがどれだけあるのか、興味をそそられますね。

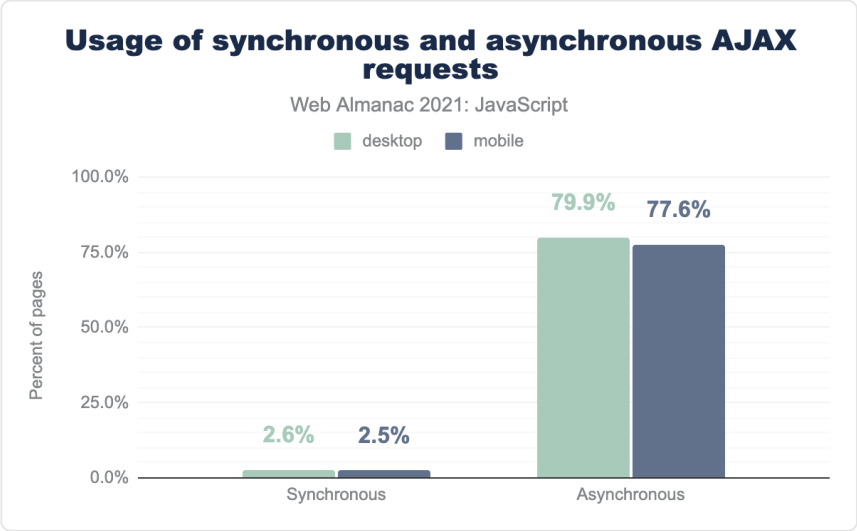


図2.31. 同期および非同期のAJAXリクエストの使用法

モバイルページの2.5%が、非推奨の同期型AJAXリクエストを使用しています。これを踏まえて、過去2年間の結果と比較して、その傾向を見てみましょう。

68. https://developer.mozilla.org/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests#synchronous_request

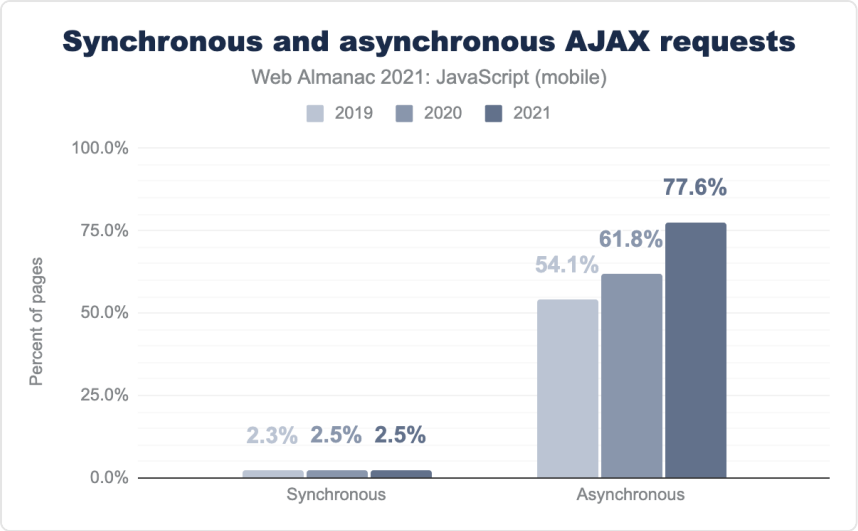


図2.32. 長年にわたる同期および非同期AJAXリクエストの使用状況。

非同期型のAJAXリクエストの使用率が明らかに増加していることがわかります。しかし、同期型のAJAXリクエストの使用率には大きな減少は見られません。

現在、ページあたりのAJAXリクエストの数を知っているので、サーバーからデータを要求するためにもっとも一般的に使用されているAPIについても知りたいと思います。

これらのAJAXリクエストを大まかに3つの異なるAPIに分類し、それらがどのように使用されているかを調べてみましょう。コアAPIである`XMLHttpRequest` (XHR)、`Fetch`、`Beacon`は、Webサイト全体で一般的に使用されており主にXHRが使用されていますが、`Fetch`は人気が高く急成長しており、`Beacon`は使用率が低いです。

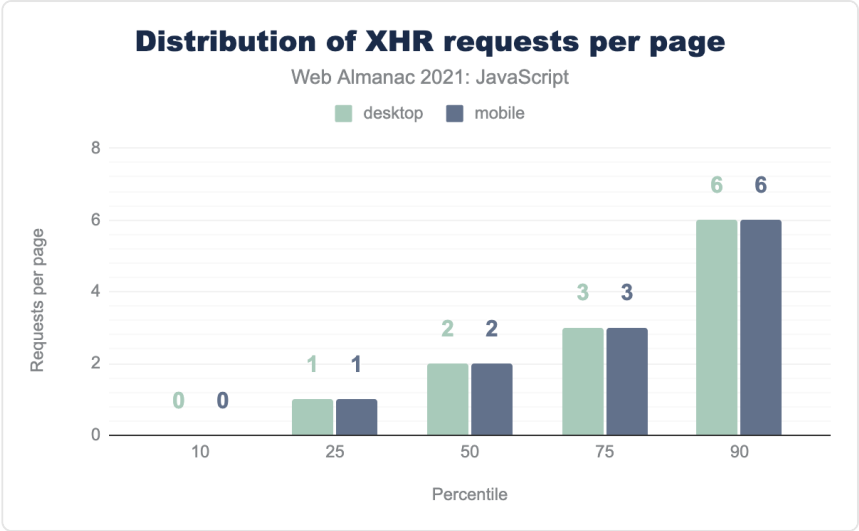


図2.33. ページあたりのXMLHttpRequestリクエスト数の分布。

モバイルページの中央値は2回のXHRリクエストを行いますが、90パーセンタイルの割合では6回のXHRリクエストを行います。

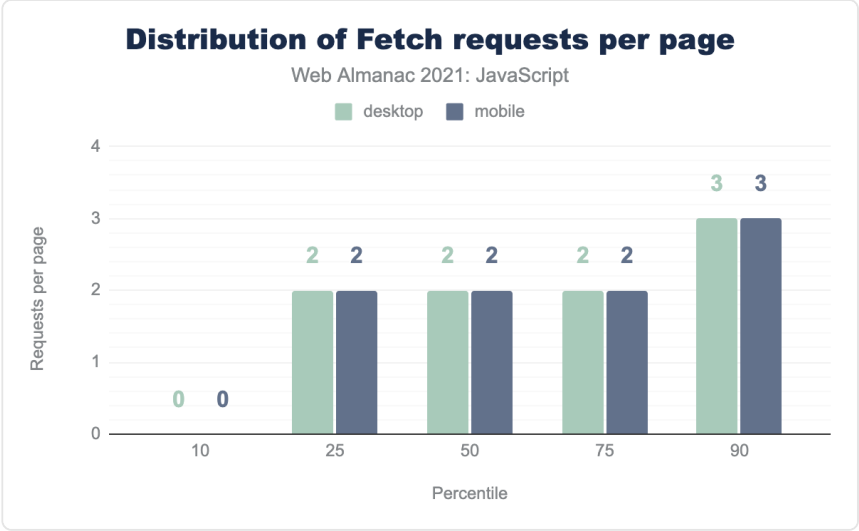


図2.34. ページあたりの `Fetch` リクエスト数の分布。

`Fetch` APIを使用した場合、モバイルページの中央値では2回のリクエスト、ロングテール

では3回のリクエストに達します。このAPIはXHRの標準的なリクエスト方法になりつつありますが、その理由の1つは、よりクリーンなアプローチと、定型的なコードが少ないことです。また、`Fetch`は従来のXHRアプローチよりもパフォーマンス上のメリット[69]があるかもしれません。これは、ブラウザがメインスレッドから大きなJSONペイロードをデコードする方法によるものです。

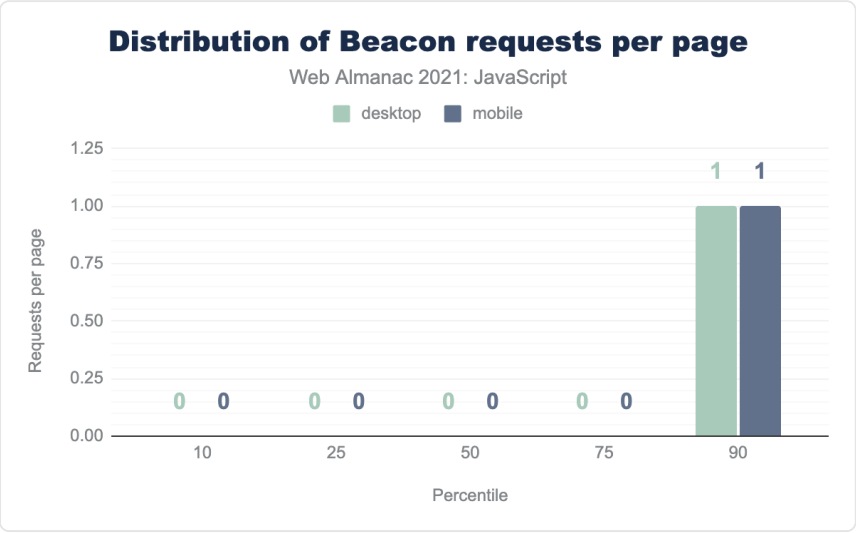


図2.35. 1ページあたりの `Beacon` リクエストの数の分布。

`Beacon`の使用率はほとんどなく、1ページあたりのリクエスト数は0で、90パーセンタイルまでは1ページあたり1つのリクエストしかありませんでした。この普及率の低さを説明する1つの可能性として、`Beacon`は一般的に分析データの送信に使用され、とくに、すぐにページをアンロードされる可能性があっても確実にリクエストを送信したい場合に使用されることが挙げられます。しかし、XHRを使用している場合、これは保証されていません。将来的には、分析データやセッションデータなどのために、XHRを使用しているページの統計情報を収集できるかどうかを試してみるとよいでしょう。

XHRと`Fetch`の採用状況を時系列で比較するのもおもしろいかもしれません。

69. https://gomakethings.com/the-fetch-api-performance-vs.-xhr-in-vanilla-js/

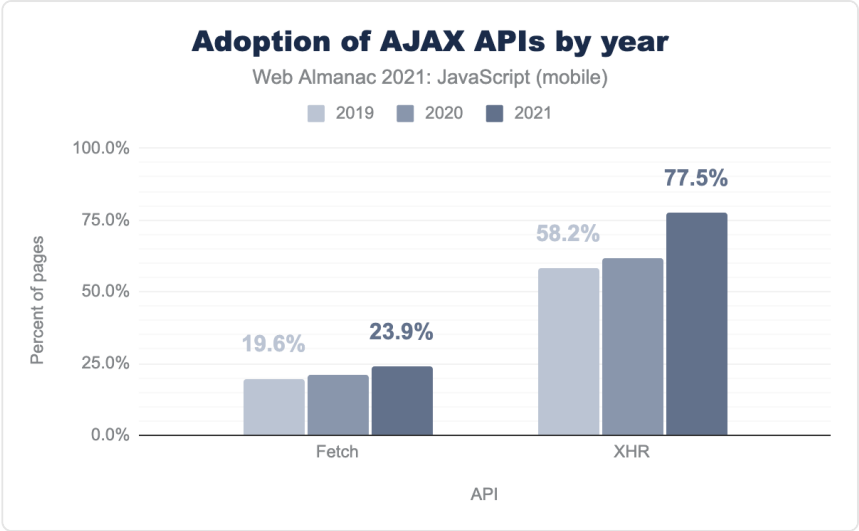


図2.36. 年別のAJAX APIの採用状況。

`Fetch`と`XHR`の両方において、ここ数年で使用量が大幅に増加しています。モバイルページでの`Fetch`の使用率は4ポイント、XHRは19ポイント上昇しています。`Fetch`の採用が徐々に増加していることは、よりクリーンなリクエストと優れたレスポンス処理の傾向を示しているように思われます。

Web ComponentsとシャドウDOM

ウェブがコンポーネント化[70]されていく中で、シングルページアプリケーションを構築する開発者は、ユーザービューをコンポーネントのセットとして考えることができます。これは、開発者が各機能ごとに専用のコンポーネントを構築するためだけでなく、コンポーネントの再利用性を最大限に高めるためでもあります。それは、同じアプリの別のビューにあったり、まったく別のアプリにあったりします。このようなユースケースでは、アプリケーションにカスタムエレメントやウェブコンポーネントが使用されます。

多くのJavaScriptフレームワークが普及したことで、再利用性や専用の機能ベースのコンポーネントを構築するという考え方がより広く採用されるようになったと言ってもいいでしょう。このことは、カスタム要素、シャドウDOM、テンプレート要素の採用を検討する上で、私たちの好奇心を刺激します。

カスタムエレメント[71]は、`HTMLElement`のAPIの上に構築されたカスタマイズされたエレメ

70. *https://developer.mozilla.org/docs/Web/Web_Components*
71. *https://developers.google.com/web/fundamentals/web-components/customelements*

ントです。ブラウザは`customElements`というAPIを提供しており、開発者は要素を定義し、それをカスタム要素としてブラウザに登録できます。



図2.37. カスタムエレメントを使用しているデスクトップページの割合。

3.0%のモバイルページでは、ウェブページの1つ以上の部分にカスタム要素を使用しています。



図2.38. シャドウDOMを使用しているページの割合。

シャドウDOMでは、ブラウザに導入されたカスタム要素のために、DOM内に専用のサブツリーを作成できます。これにより、要素内のスタイルやノードが要素の外からアクセスできないようになります。

0.4%のモバイルページでは、ウェブコンポーネントのシャドウDOM仕様を利用して、要素のスコープ付きサブツリーを確保しています。



図2.39. `template`要素を使用しているページの割合。

マークアップに再利用可能なパターンがある場合には、`template`要素が非常に便利です。`template`要素のコンテンツは、JavaScriptで参照されたときにのみレンダリングされます。

テンプレートは、JavaScriptでまだ参照されていないコンテンツが、シャドウDOMを使ってシャドウルートに追加されるため、ウェブコンポーネントを扱う際に有効です。

テンプレートを採用しているWebページは全体の0.1%にも満たない。テンプレートはブラウザでよくサポート[72]されていますが、テンプレートを使用しているページの割合はまだ非常に低いです。

72. *https://caniuse.com/template*

結論

この章で見てきた数字は、JavaScriptの使用がいかに膨大であるか、また時間の経過とともにどのように進化しているかを理解させてくれました。JavaScriptのエコシステムはウェブのパフォーマンスを向上させ、ユーザーの安全性を高めることに重点を置き、開発者の作業を容易にし、生産性を向上させる新しい機能やAPIを備えて成長してきました。

レンダリングやリソースローディングのパフォーマンスを向上させる多くの機能が、ユーザーに高速な体験を提供するために、より広く活用されていることがわかりました。開発者の皆様は、これらの新しいWebプラットフォームの機能を採用することから始めることができます。しかし、これらの機能を賢く利用し、実際にパフォーマンスが向上することを確認してください。というのも、同じスクリプトで `async` 属性と `defer` 属性を使用した場合のように、APIの中には誤用によって害を及ぼすものがあるからです。

今、私たちが利用できる強力なAPIを適切に活用することが、この数字を今後さらに向上させるために必要です。これからもそうしていきましょう。

著者



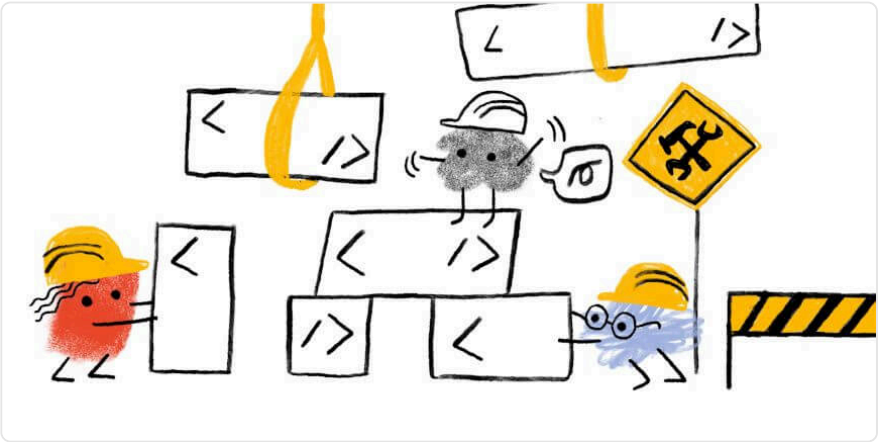
Nishu Goel

@TheNishuGoel NishuGoel https://unravelweb.dev/

Nishu Goelは、Web DataWorks[73]のエンジニアです。WebテクノロジーとAngularのGoogle Developer Expert、Developer TechnologiesのMicrosoft MVPであり、Step by Step Guide Angular Routing (BPB, 2019)とA Hands-on Guide to Angular (Educative, 2021)の著者でもあります。彼女の著作は、unravelweb.dev[74]でご覧いただけます。

73. *http://webdataworks.io/*
74. *https://unravelweb.dev/*

部 I 章 3
マークアップ



***Alex Lakatos** によって書かれた。*
Jens Oliver Meiert、 *Brian Kardell*、 *Shaina Hantsis*、 *Barry Pollard* と *Rick Viscomi* によってレビュ。
Kevin Farrugia による分析。
Rick Viscomi 編集。
Sakae Kotaro によって翻訳された。

序章

Webサイトにアクセスしようとしたとき、何が起こっているのか不思議に思ったことはないだろうか。ブラウザのアドレスバーにURLを入力すると、最初にHTMLファイルがダウンロードされ、解析されます。マークアップはWebの基礎と言えるでしょう。この章では、今日のウェブを成り立たせているレンガのいくつかを見ていくことに専念しました。

過去3年間の分析データをもとに、マークアップの将来、長年のトレンド、新しい標準の採用率などについて、いくつかの疑問点を考えてみました。また、皆様がデータをより深く掘り下げ、私たちの解釈とは異なる方法で解釈してくださることを期待して、データを共有しました。

マークアップの章では、HTMLに焦点を当てます。他のマークアップ言語（SVGやMathMLなど）や『Web Almanac』の他のトピックについても簡単に触れていますが、それらについて

はそれぞれの専用の章で詳しく解説しています。マークアップはウェブへの入り口であるため、章を丸ごと捧げないのは非常に困難でした。

一般

まず、マークアップ文書のより一般的な側面、すなわち文書の種類、文書のサイズ、文書の言語、圧縮などを説明します。

Doctypes

2021年になっても、すべてのページが `<!DOCTYPE html>` などで始まることを不思議に思ったことはないだろうか。DOCTYPEが必要なのは、ブラウザがページをレンダリングする際、"後方互換モード[754]に切り替えてはいけない、その代わりHTML仕様に従うよう最善の努力をするように、ということを伝えるためなのです。

今年は、97.4%のページがdoctypeを持ち、昨年の96.8%からわずかに増加しました。ここ数年を見ると、doctypeの比率は毎年半ポイントずつ着実に増えています。 理想を言えば、100%のウェブページがdoctypeを持つことになるのだが、この調子だと2027年には理想郷に住んでいることになります。

人気という点では、`<!DOCTYPE html>`として知られるHTML5が依然としてもっとも人気で、モバイルページの88.8%がこのDoctypeを使用しています。

Doctype	*デスクトップ*	*モバイル*
HTML ("HTML5")	*87.0%*	*88.8%*
XHTML 1.0過渡期	*5.7%*	*4.6%*
厳密なXHTML 1.0	*1.4%*	*1.3%*
HTML 4.01過渡期	*0.9%*	*0.7%*
HTML 4.01過渡期(癖のある[76])	*0.5%*	*0.5%*

図3.1. 最も人気のあるDoctypeです。

驚くべきは、それから約20年[77]経った今でも、XHTMLはウェブのかなりの部分を占めており、デスクトップでは8%、モバイルは7%弱のページで使用されているということです。

75. *https://developer.mozilla.org/docs/Web/HTML/Quirks_Mode_and_Standards_Mode*
76. *https://hsivonen.fi/doctype/#xml*
77. *https://ja.wikipedia.org/wiki/Extensible_HyperText_Markup_Language*

ドキュメントサイズ

1バイトのデータにもコストがかかるモバイルの世界では、モバイルサイト用のドキュメントサイズの重要性が増しています。また、見たところ、ますます大きくなっているようです。今年、モバイル向けページのHTMLの中央値は27KBで、昨年より2KB増加しています。デスクトップ側では、中央のページのHTMLは29KBでした。

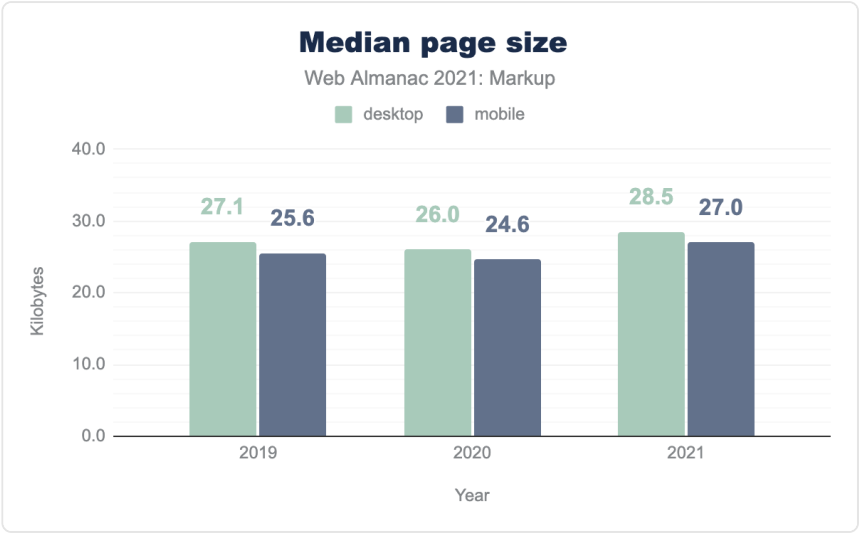


図3.2. ページサイズ前年比の中央値。

興味深かった点は

- 2019年と比較した場合、2020年のページサイズの中央値は縮小していました。上の図を見ると、2020年に凹んだ後、今年は少し増えていますね。
- デスクトップとモバイルの両方で最大のHTML文書は、デスクトップで45MB、モバイルで21MBと、今年に入ってから、それぞれ20MBずつ大幅に減少しているのです。

圧縮

文書サイズが大きくなる中、今年は圧縮についても検討しました。私たちは、文書サイズとそれを転送する際に使用する圧縮のレベルが密接に関係していると考えました。

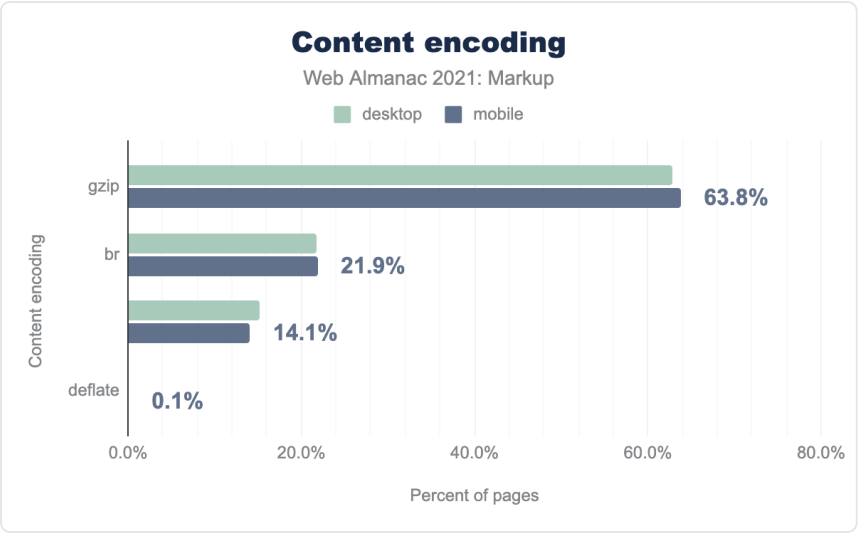


図3.3. コンテンツエンコード方式の採用。

デスクトップ600万ページのうち、84.4%がgzip（62.7%）またはBrotli（21.7%）で圧縮されていることがわかりました。モバイルページについても、85.6%がgzip（63.7%）またはBrotli（21.9%）で圧縮されており、非常によく似た数字となっています。モバイルとデスクトップで割合が若干異なるのは、異なるURLで構成されていることと、モバイルデータセットがより大きいことから、驚くべきことではありません。

特にモバイルの世界では、1バイトのデータにもコストがかかるため、圧縮は重要です。圧縮とモバイルウェブの章で、コンテンツエンコーディングとモバイルウェブの状態についてより詳しく知ることができます。

ドキュメントの言語

私たちは、`html`要素の`lang`属性のユニークなインスタンスに3,598個遭遇しました。この章を書いた時点では、7,139の言語が話されている[78]ので、そのすべてが表現されているわけではないと思わされました。言語タグの構文[79]を考慮すると、さらに少ない数しか残っていません。

78. *https://www.ethnologue.com/guides/how-many-languages*
79. *https://developer.mozilla.org/docs/Web/HTML/Global_attributes/lang#language_tag_syntax*

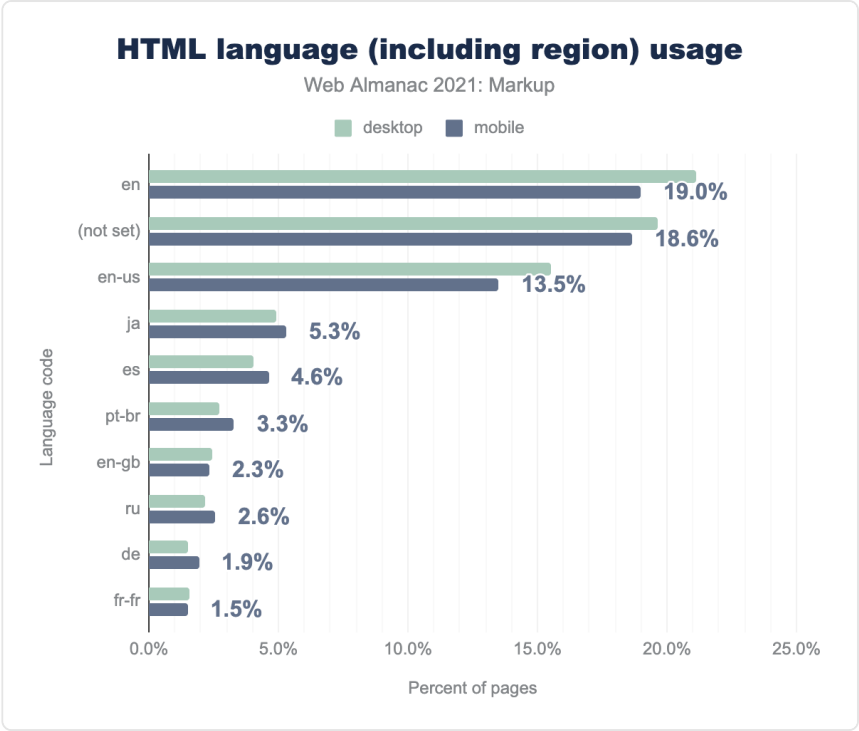


図3.4. リージョンを含むもっとも一般的なHTML言語コードの採用。

Web Content Accessibility Guidelines (WCAG[80]) では、ページの言語を定義して「プログラムからアクセス可能」であることが求められているにもかかわらず、スキャンしたページのうち、デスクトップでは19.6%、モバイルでは18.6% が `lang` 属性の指定なし だったのです。言語は `xml:lang` 要素を含むさまざまな方法で指定できますが、今回はチェックしませんでしたので、スキャンされたページの中にはまだ望みがあるかもしれません。

80. https://www.w3.org/TR/UNDERSTANDING-WCAG20/meaning-doc-lang-id.html

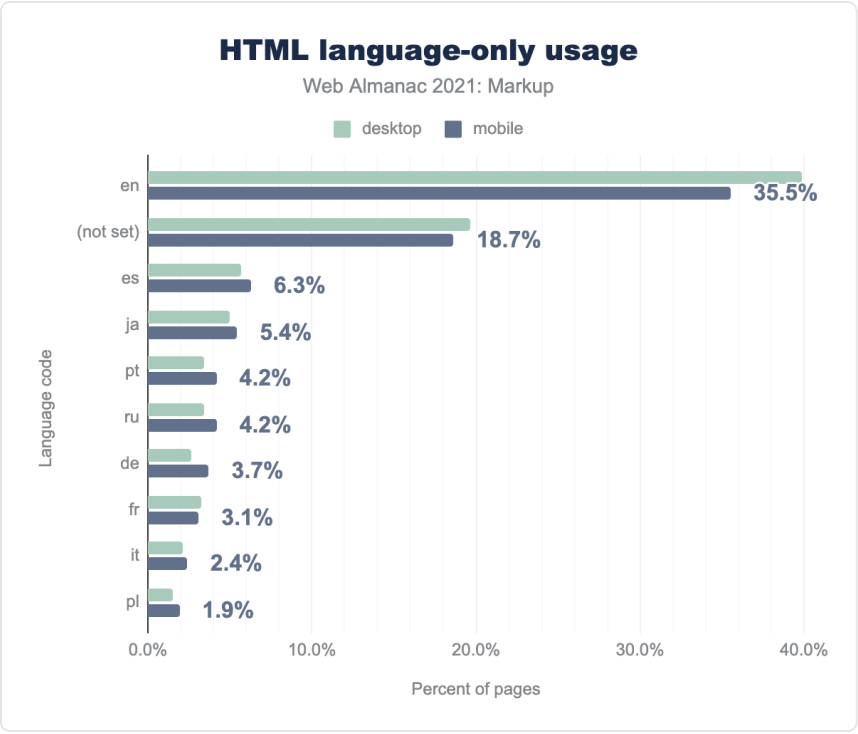


図3.5. リージョンを含まない、もっとも一般的なHTML言語コードの採用。

正規化された言語セットの上位10言語を見ると、興味深い傾向が見られました。

- モバイルは、英語サイトの相対的な割合が低くなっています。なぜそうなるのか、その原因についてチームで議論しています。携帯電話だけを使ってウェブにアクセスする人がいるので、モバイルセットの言語状況が多様化するのかもしれません。筆者は、モバイルページの多くは外出先での利用を想定しており、そのためローカルなページになっていると考えています。
- スペイン語は日本語よりも地域や下付き文字のオプションが多いのですが、2位以下は僅差の争いでした。
- デスクトップとモバイルの空属性の違いと英語には逆相関がある。

コメント

88%

図3.6. HTMLでコメントが1つ以上あるページ。

ほとんどのプロダクションビルドツールには、コメントを削除するオプションがありますが、私たちが分析したページの大半88%には少なくとも1つのコメントがありました。

一般にコメントはコード上で推奨されるものですが、Webページでは特定のブラウザ向けにマークアップをレンダリングするために、条件付きコメントという特殊なタイプのコメントが使われていました。

```
<!--[if IE 8]>

  <p>This renders in Internet Explorer 8 only.</p>

<![endif]-->
```

マイクロソフトは、IE10で条件付きコメントのサポートを打ち切りました。それでも、41%のページには少なくとも1つの条件付きコメントが存在していました。これらが非常に古いウェブサイトである可能性は別として、古いブラウザ用のポリフィリングフレームワークのバリエーションのようなものを使っていると考えるしかないでしょう。

SVGの使用

46.4%

図3.7. HTMLにSVG要素が1つ以上含まれるページ。

今年は、SVGの使い方を取り上げてみたいと思いました。人気のアイコンライブラリでSVGがどんどん使われ、ファビコンのサポートも向上し、アニメーションでもSVG画像が増えてきているので、46.4%のウェブページに何らかのSVGが使われていても不思議ではありません。37.2%がSVG要素を持ち、デスクトップでは20.0%、モバイルは18.4%がSVG画像を使用しており、SVGエンベッド、オブジェクト、iframeのいずれかを含むものはごく僅かでし

た。

SVGはstyle要素と比較するとより多くのユースケースを持っていますが、人気という点では、その数は同等です。SVGは、ページ上の要素の人気度という点では、トップ20からわずかに外れたところに位置しています。

要素

要素は、HTML文書のDNAです。私たちは、Webページという生命体を構成する細胞を分析したいと考えました。ほとんどのページでもっとも人気のある要素、もっとも存在感のある要素、そして廃れた要素は何なのか。

要素の多様性

現在定義され使用されている 112要素[81]（SVGとMathMLを除く）があり、さらに 28 は非推奨[82] または廃止されています。私たちは、1つのページで実際に使用される要素がいくつあるのか、また `div` がどの程度あり得るのかを確認したいと思いました。

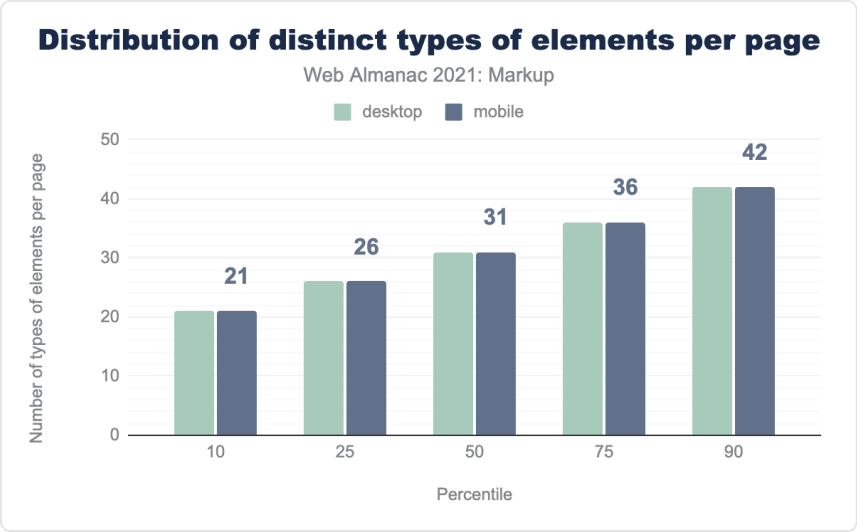


図3.8. 1ページあたりの要素の種類数の分布。

ウェブは `div` でできているわけではありません。モバイルページの中央値は31の異なる要

81. *https://html.spec.whatwg.org/multipage/indices.html#elements-3*
82. *https://developer.mozilla.org/docs/Web/HTML/Element#obsolete_and_deprecated_elements*

素を使用し、合計616の要素を持っています。

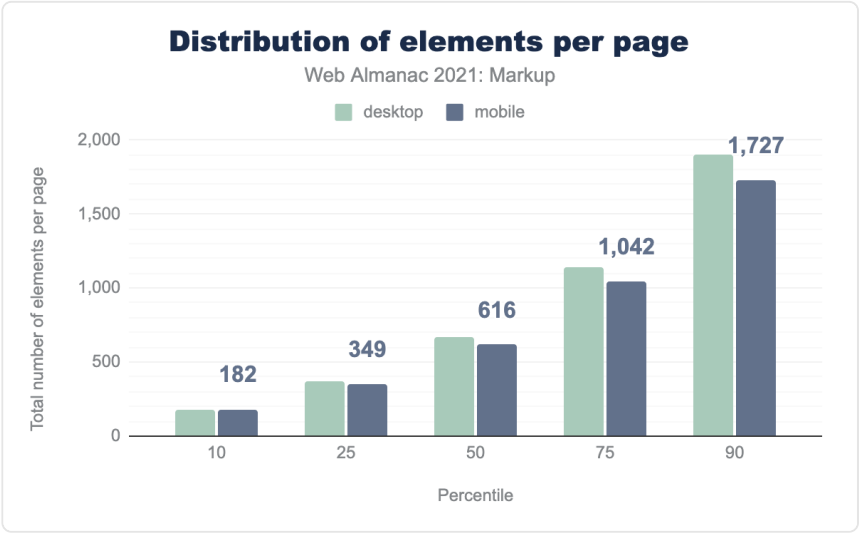


図3.9. 1ページあたりの要素数分布。

中央値のページの要素数はデスクトップで666、モバイルで616だったのに対し、全ページの上位10%ではモバイルで1,727、デスクトップで1,902と3倍近い数に達しています。

要素のトップ

2019年から毎年、『Web Almanac』のマークアップ編では、2005年のIan Hickson氏の著作[83]を参考に、もっともよく使われる要素を紹介しています。この著者は伝統を破ることができなかったので、私たちはもう一度データを見てみたのです。

83. *https://web.archive.org/web/20060203031713/http://code.google.com/webstats/2005-12/elements.html*

2005	2019	2020	2021
title	div	div	div
a	a	a	a
img	span	span	span
meta	li	li	li
br	img	img	img
table	script	script	script
td	p	p	p
tr	option	link	link
		i	meta
		option	i
			ul
			option

図3.10. ページごとの使用頻度の高い要素の進化。

上位6つの要素は過去3年間変わっておらず、`link`要素が7位として確固たる地位を築いているようです。

興味深いのは、`i`と`option`の両方が人気を失っていることです。前者はおそらく、アイコンに`i`要素を悪用するライブラリが、アイコンにSVGを使用するライブラリに取って代わられたからでしょう。今年は、ソーシャルマークアップも盛り上がっているせいか、`meta`要素がトップ10に食い込んできていますね。ソーシャルマークアップについては、この章の後のセクションで見ていきます。スタイル付き`select`要素の増加は、`ul`（順序なしリスト）要素がoption要素よりも人気を博していることを物語っています。

`main`

2021年にコンテンツの作成が急増[84]（おそらく世界で大流行したため）したため、それがコンテンツ要素の採用にも相関しているかどうかを確認したいと思いました。私たちは、

84. *https://wordpress.com/activity/posting/*

`main` が良い指標になると考えました。これは、ページの構造に関するDOMの概念に影響を与えない、情報量の多い要素だからです。



図3.11. 少なくとも1つの `main` 要素を持つモバイルページの割合。

デスクトップページの27.7%、モバイルページの27.9%が `main` 要素を備えていました。人気度では、トップ50の要素の中で34位と健闘しています。114個の要素しかないと思われるかもしれませんが、実際には1000個以上の要素がクエリから返ってきており、そのほとんどがカスタム要素です。

`base`

もうひとつ気になったのは、HTML仕様の厳格なルールに開発者がどれだけ注意を払っているかということです。たとえば、仕様では、 `base` 要素はドキュメントに1つ以上存在してはならないとあります。これは、 `base` 要素がユーザー エージェントによる相対URLの解決方法を定義しているからです。複数の `base` 要素があるとあいまいさが生じるため、仕様では、最初の要素以降のすべての `base` 要素を無視して、それらを使用できなくする必要があります。

デスクトップページを見ると、 `base` はよく使われる要素で、10.4%のページが持っています。しかし、1つしかないのでしょうか？ページよりも5,908個多く `base` 要素があるので、少なくともいくつかのページが複数の `base` 要素を持っていると結論付けることができます。開発者が指示に従うことが得意だと誰が言ったのでしょうか？また、W3Cが提供するMarkup Validation Service[85] を使ってHTMLを検証することをオススメします。

`dialog`

この章では、より議論を呼びそうな要素や新しい要素の採用についても見ていきたいと思います。 `dialog` はその1つで、すべての主要なブラウザがすぐにサポートするわけではありません。デスクトップでは7,617ページ、モバイルでは7,819ページのみがダイアログ要素を使っています。分析したページの0.1%程度に過ぎないことを考えると、まだ採用されていないように見えます。

85. https://validator.w3.org/

`canvas`

`canvas` 要素は、Canvas API[86] または WebGL API[87] と共に使用して、グラフィックスやアニメーションを描画することが可能です。Web上のゲームや複合現実感に使われる主要な要素の1つです。デスクトップページの3.1%、モバイルページの2.6%が使用しているのは当然のことです。デスクトップでの使用率が高いのは、さまざまなデバイスのグラフィック性能や、ゲームやバーチャルリアリティに偏ったユースケースを考慮すれば納得がいきます。

要素の使用頻度

`html`、`head`、`body`、`title`、`meta` 要素はすべてオプションですが、今年もっともよく使われた要素で、いずれも99%以上のページに存在しています。

ブラウザは自動的に `html` と `head` 要素を追加しますが、このグラフでは、クロール時にアクセスできなくなったサイトのために、クロールしたページの0.2% がエラーになっていることに注意してください。

86. https://developer.mozilla.org/docs/Web/API/Canvas_API
87. https://developer.mozilla.org/docs/Web/API/WebGL_API

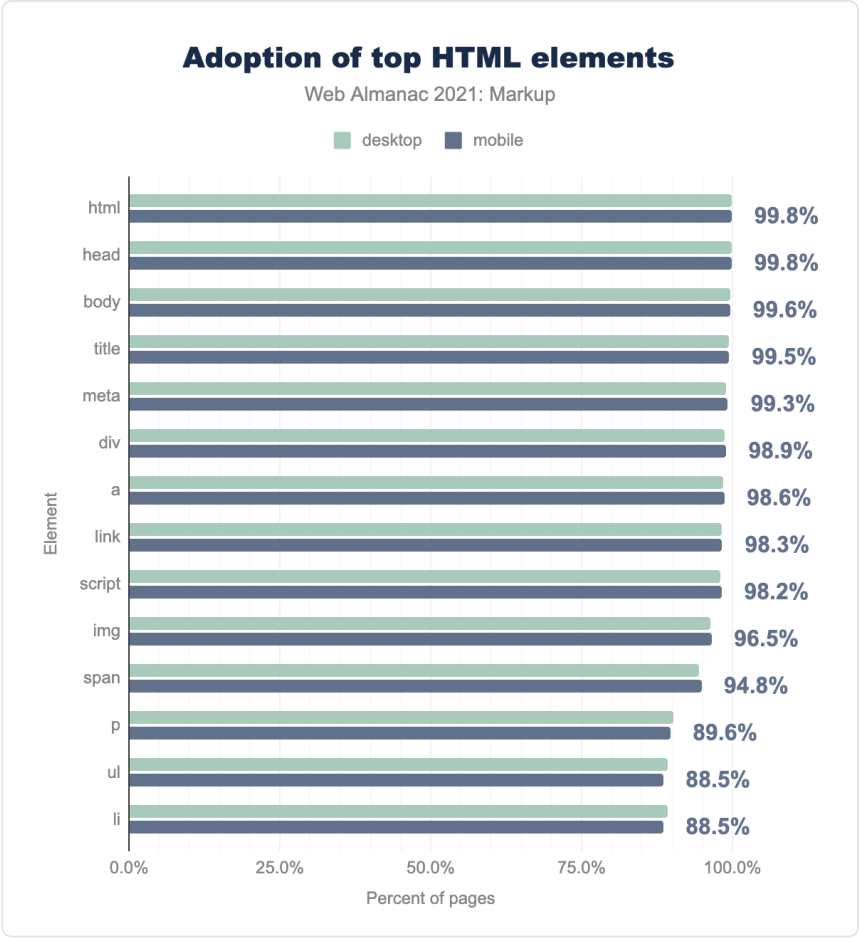


図3.12. HTMLの上位要素の採用。

昨年と比較すると、比率は若干異なるものの、人気の高い要素の順番は変わっていません。もっとエキゾチックな要素はどうでしょうか？

要素	ページ数の割合（モバイル）
`tt`	0.04%
`ruby`	0.02%
`rt`	0.02%

図3.13. モバイルページでの `tt`, `ruby`, `rt` 要素の採用。

興味深いのは、テレタイプテキスト[88]の非推奨要素である `tt` が、東アジアの文字の発音を示すために今でも使われているルビアノテーション[89]とルビ文字列[90]の要素である `ruby` と `rt` より100%人気があることです。

`script`

98.2%

図3.14. 少なくとも1つの `script` 要素を持つモバイルページの割合。

スキャンされたページの98%強が、少なくとも1つの `script` 要素を含んでいます。これは、`script` がページ上で6番目に人気のある要素であることと同じです。昨年と比較すると、`script` 要素の人気は不変のようで、分析された数百万ページでの出現率は97%から98%へとわずかに上昇しています。

51.4%

図3.15. 少なくとも1つの `noscript` 要素を持つモバイルページの割合。

また、51.4%のページが `noscript` 要素を含んでおり、これは一般的にJavaScriptを無効にしているブラウザにメッセージを表示するために使用されます。`noscript` 要素のもうひとつの一般的な用途は、Googleタグマネージャー(GTM)のスニペットです。デスクトップでは18.8%、モバイルは16.9%のページが、GTMスニペットの一部として `noscript` 要素を使用しています。GTMは、モバイルよりもデスクトップで人気があるのは興味深いことです。

88. https://developer.mozilla.org/docs/Web/HTML/Element/tt
89. https://developer.mozilla.org/docs/Web/HTML/Element/ruby
90. https://developer.mozilla.org/docs/Web/HTML/Element/rt

`template`

Web Components仕様の もっとも認知されていないが、もっとも強力な機能[91] のひとつに `template` 要素があります。2013年からモダンブラウザで `template` 要素 がしっかりサポートされているにもかかわらず、2021年には0.5% のページしか使っていませんでした。人気度では、トップ50の要素にすら入っていない。これは、ウェブ開発者にとってのモダンなHTML仕様の普及カーブを物語っていると考えました。

`template` が何をするのかよく分からないという人のために、仕様書を読んで復習しておきましょう。 `template` 要素は、スクリプトによって複製され、ドキュメントに挿入されるHTMLの断片を宣言するために使用されます。もしあなたがウェブ開発者で、この言葉に聞き覚えがあると思うのなら、その通りです。 Angularは `ng-content` 、Reactではportals[92]、そしてVueは `slot` です。現在人気のフレームワークはほとんど同じことを行う非ネイティブ機構が備わっています。これらのフレームワークでは、フレームワーク内で機能を再作成する代わりに、ネイティブの `template` 要素またはWeb Componentsを使用するものと考えていました。

`style`

83.8%

図3.16. 少なくとも1つの `style` 要素を持つモバイルページの割合。

ウェブページを作るとき、3つのものが一緒になります。ひとつはHTMLで、この章を通してそれを見ていきます。2つ目はJavaScriptで、JavaScriptを読み込むために使われる `script` 要素がもっとも人気のあるものの1つであることを前のセクションで見てきました。CSSのインライン化に使われる `style` 要素も同様に人気があることは、さほどショックではないでしょう。スキャンされたモバイルページの83.8%は、少なくとも1つの `style` 要素を持っていました。

ページ内での人気度という点では、0.7%とトップ20に入るのがやっとでした。つまり、1つのページで複数の `script` 要素が人気を集めている一方で、ほとんどのページでは `style` 要素の数が5倍も少ないと考えられます。そして、それは理にかなっているのです。なぜなら、 `script` 要素はインラインと外部スクリプトの両方に使用できますが、CSSでは外部スタイルシートの読み込みに `link` 要素という別の要素を使用するからです。 `link` 要素は `script` 要素よりもわずかに多くのページに存在していますが、出現数の点ではやや劣って

91. *https://css-tricks.com/crafting-reusable-html-templates/*
92. *https://reactjs.org/docs/portals.html*

います。

カスタム要素

また、HTMLやSVGの仕様に現れない要素、それが最新であれ旧式であれ、どのようなカスタム要素が世の中に出回っているのかを調べました。

要素	ページ数	ページ数の比率
`rs-module-wrap`	123,189	2.0%
`wix-image`	76,138	1.2%
`pages-css`	75,539	1.2%
`router-outlet`	35,851	0.6%
`next-route-announcer`	9,002	0.1%
`app-header`	7,844	0.1%
`ng-component`	3,714	0.1%

図3.17. デスクトップページに一部のカスタム要素を採用。

圧倒的に人気があるのはSlider Revolution[93]で、大半の要素がこのフレームワークに起因するものだと言われています。この1年で3倍以上の人気となり、人気のあるテンプレートやサイトビルダーの一部かもしれないと考えるに至りました。僅差で2位は、人気の無料サイトビルダーのWix[94]です。`pages-css`は特定できませんでしたが、なぜ`pages-css`要素が人気なのか、何かアイデアがありましたら、GitHubの編集を提案する で教えてください。

Angular[95]や Next.js[96] 、あるいは以前の Angular.js[97] など人気フレームワークはもっとカスタムコンポーネントを占めていると思っていましたが、`router-outlet` と `ng-component` はカスタムコンポーネント基盤の小さな一部を構成していることがわかります。

廃止された要素

現在、HTMLリファレンスには、28の廃止・非推奨要素[98]が記述されています。私たちは、

93. https://www.sliderrevolution.com/faq/developer-guide-output-class-tag-changes/
94. https://www.wix.com/
95. https://angular.io/
96. https://nextjs.org/
97. https://angularjs.org/
98. https://developer.mozilla.org/docs/Web/HTML/Element#obsolete_and_deprecated_elements

そのうちのいくつが現在も使われているのかを知りたかったのです。圧倒的に使われているのは `center` と `font` で、昨年と比較してその使用率がわずかに減少しているのは喜ばしいことです。

一方、`nobr` と `big` はまだ非推奨ですが、昨年と比較して使用頻度がわずかに増加しています。

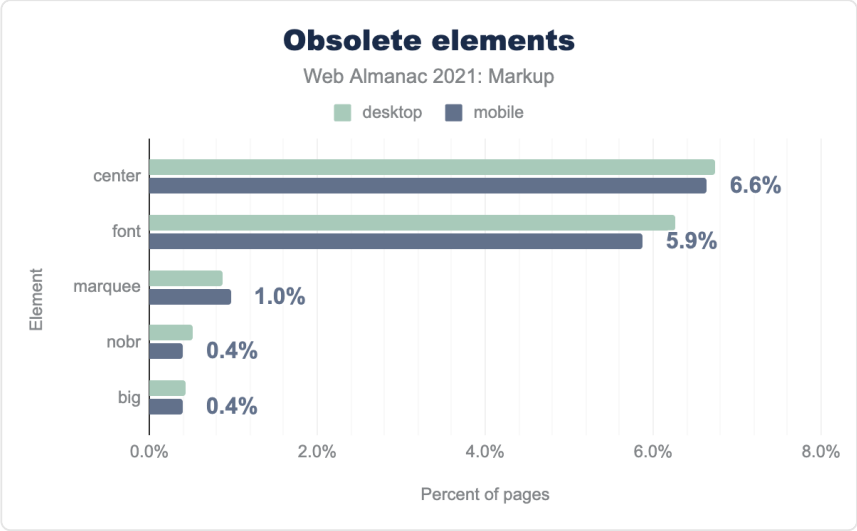


図3.18. 時代遅れのトップHTML要素の採用。

デスクトップと比較した場合、モバイルページの廃止要素の割合は若干異なりますが、順位は変わりません。

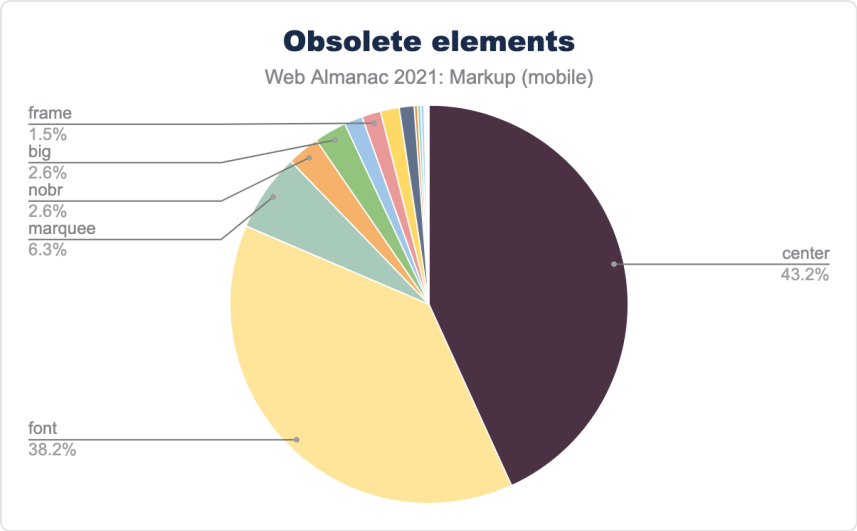


図3.19. 廃止された上位のHTML要素の相対的な採用率。

Googleは2021年になってもトップページに `center` 要素を使用していますが、私たちが判断するつもりはありません。

独自規格・非標準規格の要素

カスタム要素はすべてハイフンが入っていますが、作り物でハイフンがなく、HTML standard[99] に表示されない要素にも遭遇しています。

99. *https://html.spec.whatwg.org/#toc-semantics*

要素	モバイル	デスクトップ
jdiv	0.8%	0.8%
noindex	0.9%	0.8%
mediaelementwrapper	0.6%	0.6%
ymaps	0.3%	0.2%
h7	0.1%	0.1%
h8	<0.1%	<0.1%
h9	<0.1%	<0.1%

図3.20. 非標準要素の採用。

いずれも昨年も存在しており、JivoChat、Yandex、MediaElement.js、Yandex Mapsといった人気のフレームワークや製品に起因するものであることがわかります。また、調子に乗って6個ではヘッダーが足りないという人もいるので、`h7`から`h9`まで。

埋め込み型コンテンツ

要素	デスクトップ	モバイル
iframe	56.7%	54.5%
source	9.9%	8.4%
picture	6.1%	6.0%
object	1.4%	2.0%
param	0.4%	0.4%
embed	0.4%	0.4%

図3.21. コンテンツ埋め込み用要素の採用。

コンテンツは、ページ内の複数の要素を通じて埋め込むことができます。もっとも人気があるのは`iframe`で、それに続くのが`source`と`picture`です。

実際の`embed`要素は、コンテンツを埋め込むための現在のすべての要素の中で、もっとも

人気がありません。

フォーム

フォーム、つまり訪問者から入力を得る方法は、ウェブの構造の一部となっています。デスクトップでは71.3%、モバイルは67.5%のページで、少なくとも1つの `form` が使われていることは、驚くことではありません。もっとも多かったのは、1つのページに1つ（デスクトップでは33.0%、モバイルは31.6%）または2つ（デスクトップでは17.9%、モバイルは16.8%）の `form` 要素があることでした。



図3.22. 1つのページで見つかるもっとも多くの `form` 要素。

また、1つのページがデスクトップで4,018個の `form` 要素、モバイルで4,256個の `form` 要素を持つという極端なケースも存在します。4,000個に分割しなければならないほど価値のある入力とはどのようなものなのか、考えずにはいられません。

属性

要素の動作は属性に大きく影響されます。そこで、ページで使用される属性を調べ、 `data-*` パターンと、 `meta` 要素に人気のあるソーシャル属性を探りました。

トップ属性

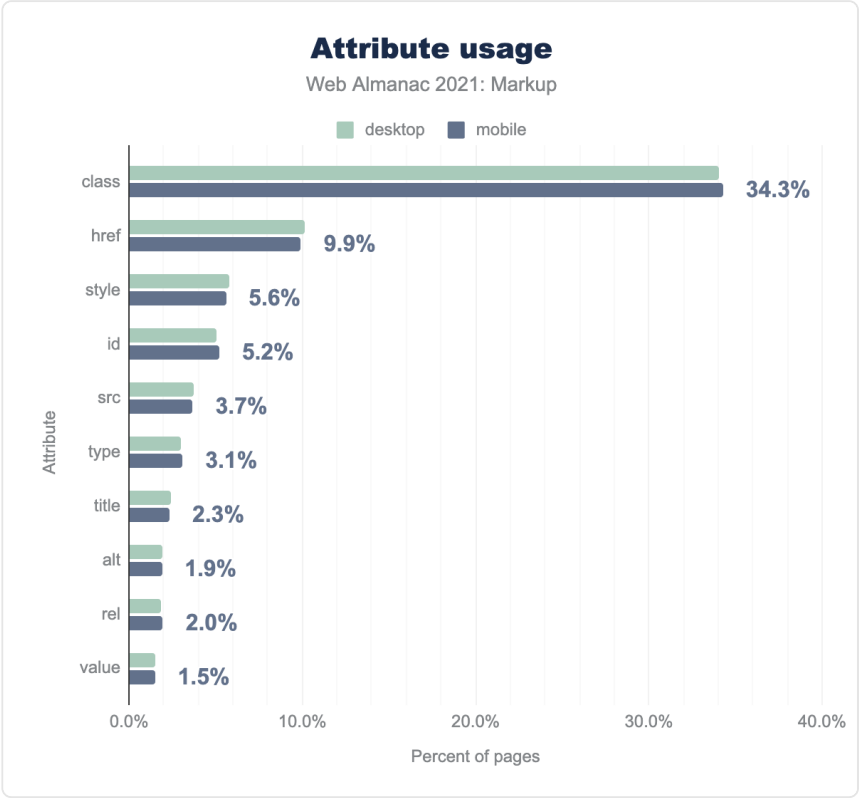


図3.23. もっとも一般的なHTMLの属性です。

もっとも人気のある属性は `class` で、これはスタイリングに使用されることを考えると、驚くことではありません。私たちが調査したページで見つかったすべての属性の34.3%は `class` でした。一方、 `id` の使用率は5.2%と非常に少ない。興味深いのは、 `style` 属性が `id` 属性よりも人気があり、5.6% を占めている点です。

2番目に多い属性は `href` で、9.9%の出現率でした。リンクはウェブの一部なので、アンカー要素の属性がこれほど人気があるのは驚きではありません。意外だったのは、 `src` 属性が、かなり多くの要素で利用できる[100]にもかかわらず、 `alt` 属性の2倍しか利用されていなかったことです。

100. *https://developer.mozilla.org/docs/Web/HTML/Attributes*

メタフレーバー

今年、`meta`要素の人気が落ちてきているので、もう少し詳しく見ていきたいと思います。これらは、機械的に読み取れる情報をページに追加する方法を提供するだけでなく、いくつかの巧妙なHTTP同等機能を実行します。たとえば、ページに対して *コンテンツセキュリティポリシー* を設定できます。

```
<meta http-equiv="Content-Security-Policy" content="default-src
'self'; img-src https://*;">
```

利用可能な属性のうち、`name`（`content`と対になっている）がもっともよく使われています。14.2% の`meta`要素は`name`属性を持ちませんでした。`content`属性と組み合わせて、情報を渡すためのキーと値のペアとして使用されます。どんな情報かというと？

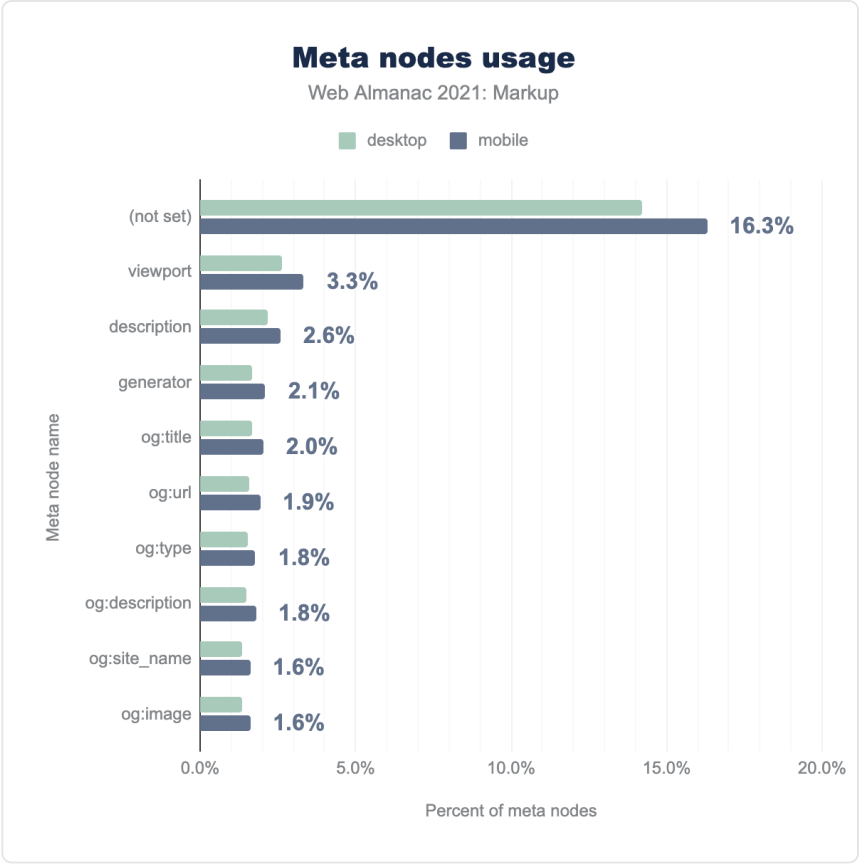


図3.24. もっともよく使われる `meta` ノード名です。

45.0%

図3.25. `initial-scale=1,width=device-width` という値を持つ `meta` ビューポートの割合です。

もっとも人気があるのはビューポート情報で、`viewport` の値としては `initial-scale=1,width=device-width` がもっとも人気があります。スキャンされたモバイルページの45.0%がこの値を使用していました。

2番目に多い組み合わせは、Open Graph[101] meta要素としても知られる `og:*` meta要素で

101. https://ogp.me/

す。これらについては、次のセクションで説明します。

ソーシャルマークアップ

ソーシャルプラットフォームがあなたのページへのリンクをプレビューする際に使用する情報やアセットを提供することは、`meta`要素の一般的な使用例です。

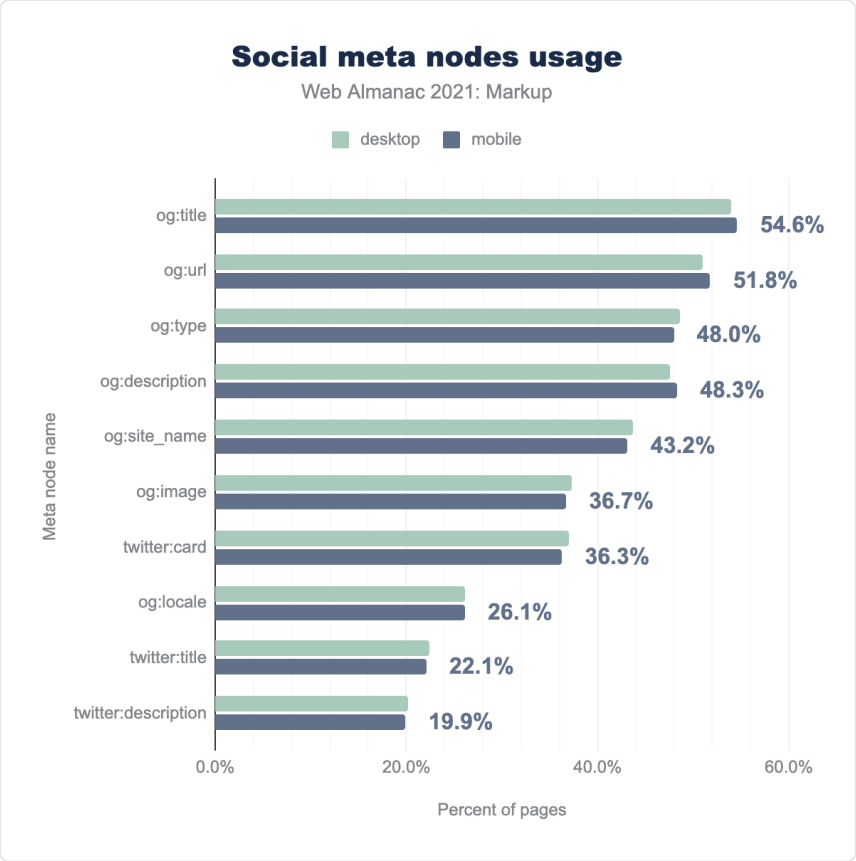


図3.26. ソーシャル `meta` ノードのページ別使用率。

もっとも一般的なのは、複数のネットワークで使用されているOpen Graphの`meta`要素で、Twitter固有の要素はそれに劣ります。`og:title`、`og:type`、`og:image`、`og:url`はすべてのページで必要なので、その使用数にばらつきがあるのは興味深いことです。

`data-` 属性

HTML仕様では[102]、接頭辞に `data-` を持つカスタム属性を使用できます。これは、カスタムデータ、状態、注釈などを保存するためのもので、ページやアプリケーションのプライベートなもので、これ以上適切な属性や要素がないようなものです。

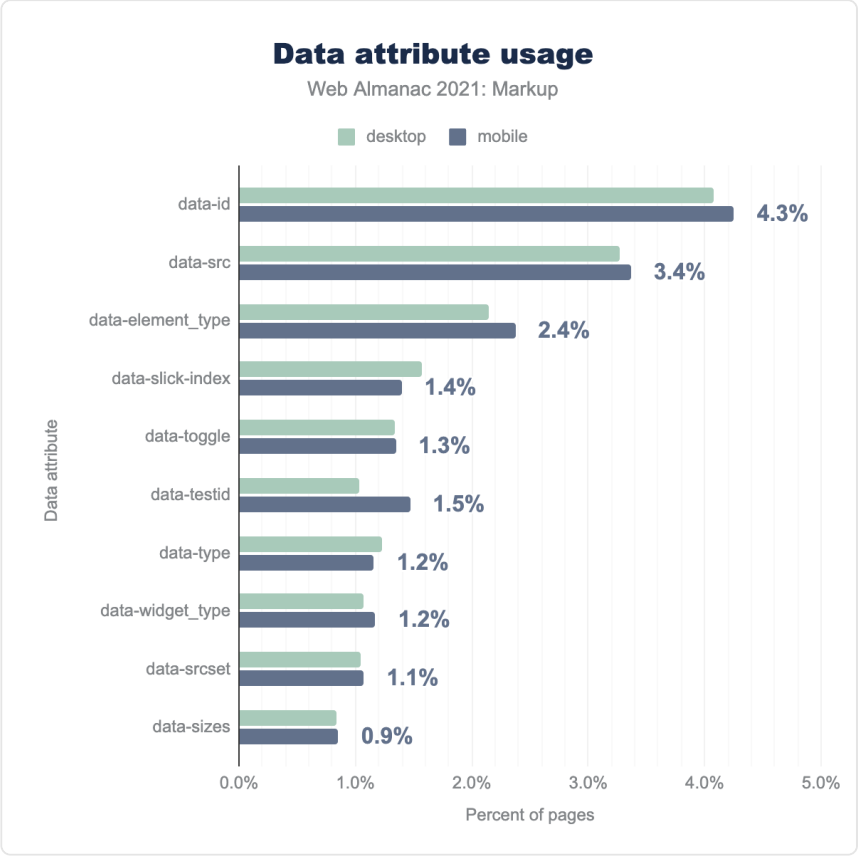


図3.27. もっとも一般的な `data-` 属性です。

もっとも一般的なものである `data-id`、`data-src`、`data-type` は非特定であり、`data-src`、`data-srcset`、`data-sizes` は画像の遅延ローディングライブラリで非常によく使用されるものです。`data-element_type` と `data-widget_type` は、人気のウェブサイトビルダーである Elementor[103] から来ています。

102. *https://html.spec.whatwg.org/#embedding-custom-non-visible-data-with-the-data-*-attributes*
103. *https://code.elementor.com/*

Slick、「あなたが必要とする最後のカルーセル」[104]は `data-slick-index` を担当します。Bootstrapのような人気のあるフレームワークは `data-toggle` を担当し、testing-library[105]は `data-testid` に責任を負います。

その他

これまで、もっとも一般的なHTMLの使用例について、かなりの部分をカバーしてきました。最後にこのセクションを設け、より難解な使用例や、ウェブでの新しい標準の採用について見ていきます。

`viewport` 仕様

`viewport` `meta` 要素は、モバイルデバイスでのレイアウトを制御するために使用されます。あるいは、少なくともそれが登場したときのアイデアでした。今日、一部のブラウザ[106]は、ページを最大500%[107]まで拡大できるように、 `viewport` オプションをいくつか無視しはじめました。

104. *https://github.com/kenwheeler/slick*
105. *https://testing-library.com/docs/queries/bytestid/*
106. *https://www.quirksmode.org/blog/archives/2020/12/userscalableno.html*
107. *https://dequeuniversity.com/rules/axe/4.0/meta-viewport-large*

属性	デスクトップ	モバイル
`initial-scale=1,width=device-width`	46.6%	45.0%
(空っぽ)	12.8%	8.2%
`initial-scale=1,maximum-scale=1,width=device-width`	5.3%	5.6%
`initial-scale=1,maximum-scale=1,user-scalable=no,width=device-width`	4.6%	5.4%
`initial-scale=1,maximum-scale=1,user-scalable=0,width=device-width`	4.0%	4.3%
`initial-scale=1,shrink-to-fit=no,width=device-width`	3.9%	3.8%
`width=device-width`	3.3%	3.5%
`initial-scale=1,maximum-scale=1,minimum-scale=1,user-scalable=no,width=device-width`	1.9%	2.5%
`initial-scale=1,user-scalable=no,width=device-width`	1.89%	1.9%

図3.28. 最も一般的な `meta` のビューポート値を採用。

もっとも一般的な `viewport` コンテンツオプションは `initial-scale=1,width=device-width` で、これはビューポートについて説明している MDN guide[108] で推奨されているオプションなので、驚くことではありません。分析したページの45.0%がこれを使用しており、昨年[109]よりほぼ3%多くなっています。8.2%のページで `content` 属性が空になっており、これも昨年より若干多くなっています。これは、ビューポートオプションの不適切な組み合わせの使用率が減少したことと相関しています。

ファビコン

ファビコンは、ウェブ上でもっとも弾力性のある作品のひとつです。マークアップなしでも機能し、複数の画像形式を受け入れることができます。また、文字通り何十ものサイズがあり、徹底して使用する必要があります。

108. https://developer.mozilla.org/docs/Web/HTML/Viewport_meta_tag
109. https://almanac.httparchive.org/ja/2020/markup#viewportの仕様

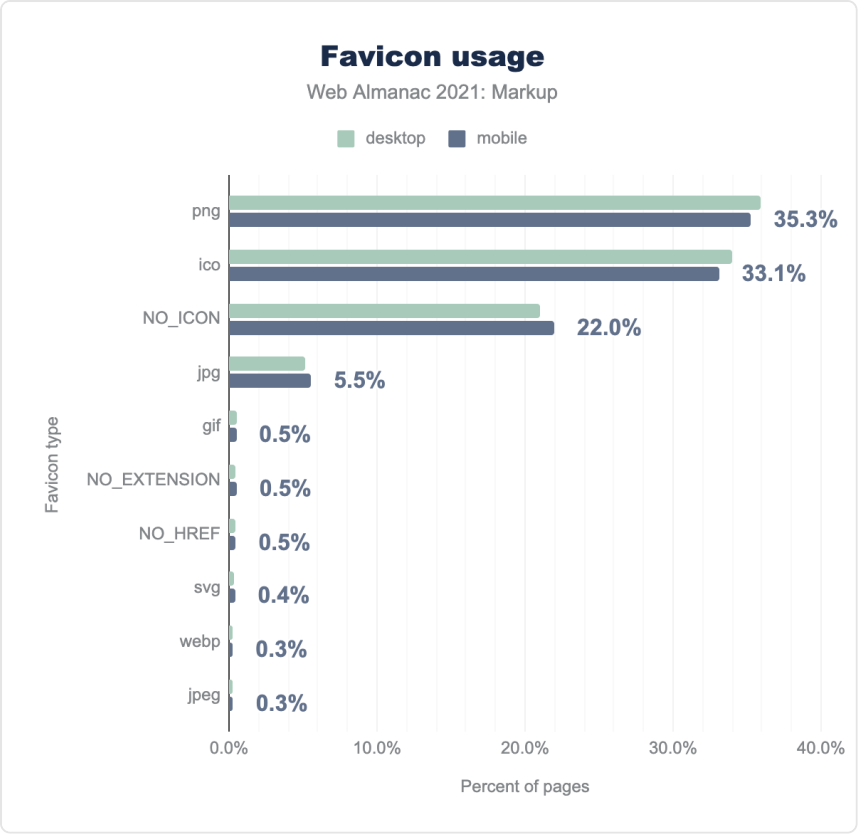


図3.29. もっとも一般的なファビコン形式です。

データを見ると、いくつかの驚きがありました。

- ICOは、ついにPNGにその座を奪われた。
- JPGは、他の不人気な選択肢と比較すると、ベストな選択肢ではないにもかかわらず、いまだに使われているのです。
- ファビコンのSVGサポートがようやく改善され、SVGの人気は今年WebPを追い越しました。

ボタンと入力タイプ

65.5%

図3.30. 少なくとも1つの `button` 要素を持つモバイルページのパーセンテージ。

ボタンは賛否両論ある。何がウェブ上のボタンを構成し、何がボタンを構成しないかについて、多くの意見があります。私たちはどちらかの味方をするわけではありませんが、`button` 要素を指定するセマンティックな方法をいくつか見てみようと思いました。

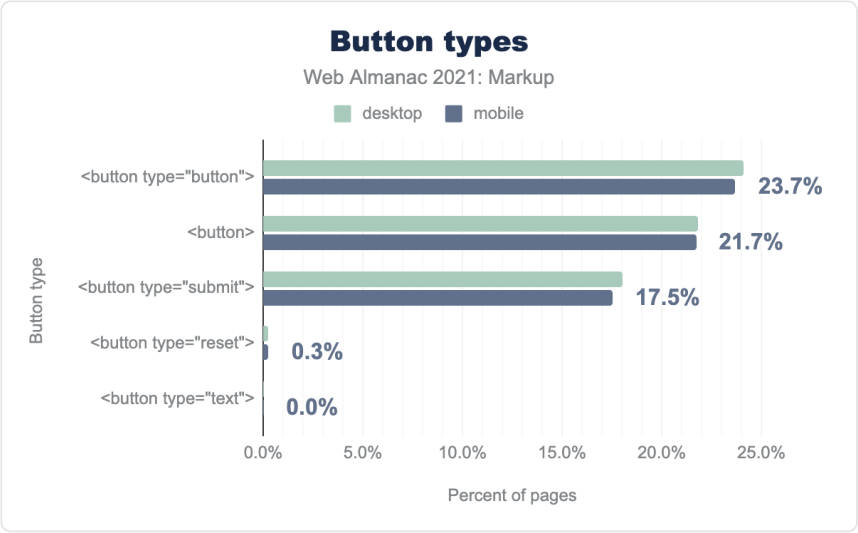


図3.31. もっともポピュラーなボタンタイプです。

昨年[110]と比較したところ、`button` 要素のあるページがかなり増えていることがわかりました。今年は `input` 型のボタンに対するクエリは実行しませんでしたが、ページ上の `button` 要素の使用数は明らかに減少しています。アクセシビリティの章にもボタンに関するセクションがあるので、そちらも読んでみてください。

110. *https://almanac.httparchive.org/ja/2020/markup#button-and-input-types*

リンク

リンク	デスクトップ	モバイル
常に `target="_blank"` を `noopener` と `noreferrer` と共に使用する	22.0%	23.2%
`target="_blank"` を `noopener` と `noreferrer` と一緒に使うことがある	78.0%	76.8%
`target="_blank"` を持つ	81.2%	79.9%
`target="_blank"` が `noopener` と `noreferrer` と共にある	14.3%	13.2%
`noopener` で `target="_blank"` を持つ	21.2%	20.1%
`target="_blank"` を `noreferrer` と共に持つ	1.2%	1.1%
`target="_blank"` から `noopener` と `noreferrer` を取り除いたもの	71.1%	69.9%

図3.32. リンク属性の様々な組み合わせの採用。

リンクはウェブを繋ぐ接着剤です。通常、私たちは、リンクが問題になっている事例を調べたいと考えています。`noopener` と `noreferrer` を使わずに `target="_blank"` を使うことは、長い間セキュリティ上の脆弱性でしたが、現在でもデスクトップページの71.1% とモバイルページの68.9% がこの方法を使用しています。

そのためか、今年になって 仕様変更[111] が行われ、今ではブラウザはすべての `target="_blank"` リンクに `rel="noopener"` をデフォルトで設定するようになりました。

Webマネタイズ

Web Monetization[112] は、Web Platform Incubator Community Group[113] (WICG) でW3C標準として提案されています。これは、クリエイターへの報酬、APIコールへの支払い、重要なウェブインフラのサポートを、オープン、ネイティブ、効率的、かつ自動的に行うための歴史の浅い規格です。まだ初期段階にあり、主要なブラウザでは実装されていませんが、フォークや拡張機能でサポートされており、ChromiumやHTTP Archiveデータセットで1年以上前

111. https://github.com/whatwg/html/issues/4078
112. https://discourse.wicg.io/t/proposal-web-monetization-a-new-revenue-model-for-the-web/3785
113. https://www.w3.org/community/wicg/

から組み込まれています。私たちは、これまでの採用状況を見てみたいと思いました。

1,067

図3.33. Webマネタイズを利用しているモバイルページ数。

ウェブマネタイゼーションでは、一般的にページ上に `meta` 要素を使用し、支払われるお金のためのウォレットアドレスを指定します。ちょっとだけ似ていますね。

```
<meta name="monetization" content="$wallet.example.com/alice">
```

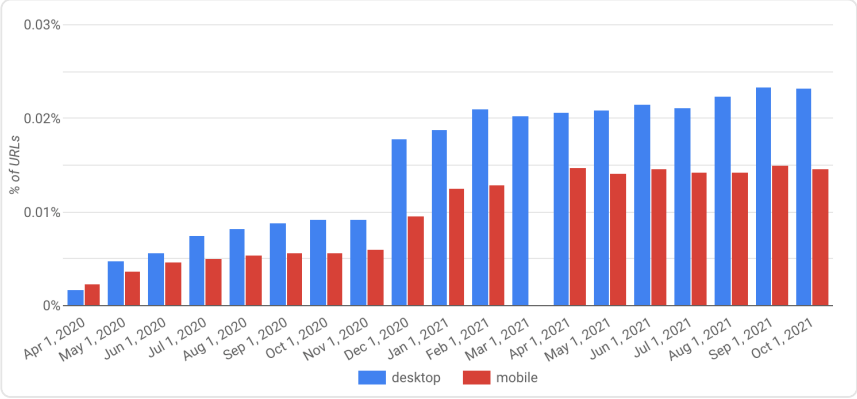


図3.34. Webマネタイズの採用状況の推移。(出典 Chrome Status[114])

割合からするとまだまだ小さい数字ですが、モバイルよりもデスクトップで伸びていることがわかります。HTTPアーカイブのデータセットがいかに大きいか、また、広くネイティブにサポートされている機能であっても、数を増やすのにいかに時間がかかるかを念頭に置いておくことが重要です。今後も時間をかけて、この数字や動向を追っていくのもおもしろいかもしれません。筆者はWeb Monetization標準の編集者として、偏見があるかもしれませんが、試してみてください[115]、無料です。

しばらく前から 問題が生じており[116]、新しいバージョンの仕様では 代わりに `link` を使用します。 `link` バージョンが使われているのはデスクトップ セットで36ページ、モバイル セットで37ページのみで、それらすべてには `meta` バージョンも含まれていました。

114. *https://www.chromestatus.com/metrics/feature/timeline/popularity/3119*
115. *https://webmonetization.org/docs/getting-started*
116. *https://github.com/WICG/webmonetization/issues/19*

現在、エコシステムには2つのInterledger[117]対応ウォレットプロバイダーがあることをわかっているので、それらのウォレットの分布と採用を確認したいと思いました。

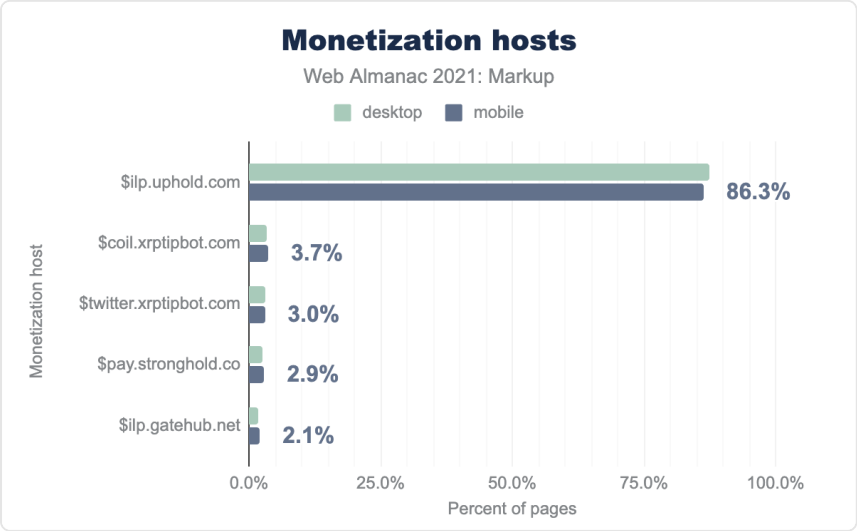


図3.35. もっとも人気のあるWebマネタイズホストです。

UpholdとGatehubが現在のウォレットで、Upholdが圧倒的に優勢なようです。不思議なのは、今年非推奨となったウォレット「Stronghold」が、現役ウォレットプロバイダー「Gatehub」より人気があったことです。これは、Web開発者がWebサイトを更新するスピードの速さを物語っているのではないかと思いました。

結論

この章では、興味深いデータ、驚くようなデータ、気になるデータを指摘してきました。今一度、2021年のマークアップのあり方を振り返ってみよう。

私たちにとってもっとも驚きだったのは、約20年後でも、ウェブのかなりの部分でXHTMLが使われていて、2021年には7%強のページがXHTMLを使っていることでした。

2019年と比較すると2020年のページサイズの中央値は縮小していましたが、今年は2019年の中央値も上回り、トレンドが逆行したような印象です。ウェブが重くなっているのです。またもやです。

モバイルページでは、英語の人気が相対的に低くなっています。その理由は定かでありませ

117. https://interledger.org/

んが、筆者はその理由の可能性を探ることをオススメしたいと思います。

より良いプラクティスを採用しているライブラリと、人気のない要素が直接的に相関しているのは興味深いことでした。 今年は、`i`と`option`の両方があまり使われていません。これは、アイコンライブラリがSVGの使用に切り替えたからです。

ICOがついにPNGに取って代わられ、もっとも人気のあるファビコン形式となったのは素晴らしいことです。同様に、SVGがこの1年でファビコンの使用量を2倍以上に増やしたのを見ると、PNGを追い抜くにはあと10年はかかると思われます。

`doctype`の比率は毎年半ポイントずつ着実に増えている。この調子でいけば、2027年にはすべてのページが`doctype`を持つという理想的な世界になることでしょう。

筆者が気になったのは、新しい規格の採用が遅く、時には10年周期になることもあること、Webページが思うように更新されないことです。

それを踏まえて、2021年のウェブのあり方を考えてみたいと思います。また、毎年新しい規格の採用を増やしている人々の一員になることをオススメします。今日学んだ新しいこと、この章だけでなくこのWeb Almanacの出版物全体で取り上げた多くの標準のうちの1つから始めてみてください。

著者



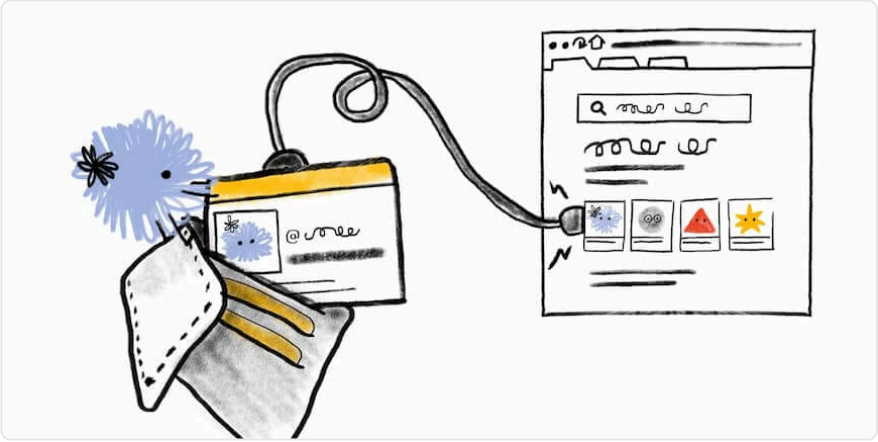
Alex Lakatos

@avolakatos AlexLakatos http://alexlakatos.com/

Alex Lakatosは過去10年間、ブラウザ、通信、FinTechの各組織でオープンウェブの研究に携わってきました。ウェブ技術と開発者支援のバックグラウンドを持ち、Interledger Foundation[118]が開発者向けの製品を構築し、開発者コミュニティ全体と関わりを持つのを支援しています。Twitterで彼にコンタクトを取ることができます[119]。

118. *https://interledger.org/*
119. *https://x.com/avolakatos*

部 I 章 4
構造化データ



Jono Alderson と Andrea Volpini によって書かれた。
Koen Van den Wijngaert と Phil Barker によってレビュ。
Greg Brimble による分析。
Jarno van Driel、 Jasmine Drudge-Willson と Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

ウェブページを読むとき、私たちは「非構造化」されたコンテンツを消費しています。段落を読み、メディアを検証し、消化したものを検討する。このプロセスの一環として、私たちは直感と文脈（主題に精通しているなど）を適用して、重要なテーマ、データポイント、エンティティ、および関係を特定します。私たち人間は、この作業を非常に得意としています。

しかし、このような直感や文脈は、ソフトウェアで再現することが困難です。システムが高い信頼性を持って解析し、重要なテーマを特定し、抽出することは難しいのです。

このような制限は、私たちが効果的に構築・創造できるものの種類を制限し、ウェブテクノロジーの「スマートさ」を制限することになります。

情報に*構造*を導入することで、ソフトウェアがコンテンツを理解するのを*はるかに*容易_に

できます。このためには、ラベルやメタデータを追加して、主要な概念や実体、およびそれらの特性や関係を特定します。

機械が構造化されたデータを確実に、かつ大規模に抽出することができれば、よりスマートな新しいタイプのソフトウェア、システム、サービス、ビジネスが可能になります。

Web Almanacの「構造化データ」の章の目的は、構造化データが現在ウェブ上でどのように使用されているかを探ることです。これにより、現在の状況、課題、および機会についての洞察が得られることを期待しています。

この章は『Web Almanac』にはじめて掲載されたため、残念ながら比較のための過去データが不足しています。今後の章では、対前年比の傾向も探っていく予定です。

主要コンセプト

構造化データは、複雑な風景であり、本質的に抽象的で「メタ」なものである。構造化データの重要性と潜在的な影響を理解するために、以下の主要な概念を調べてみる価値があります。

セマンティックウェブ

公開されたウェブページに構造化データを追加し、それらのページが含む（あるいはそれに関する、あるいは参照する）エンティティを定義すると、リンクデータ[120]の形式ができ上がります。

私たちは、コンテンツに含まれる（あるいは関連する）事柄について、*3 つの組み*という形で*供述*を行います。この **記事** はこの **人** によって **承認** された」、「その **動画** は **猫** に **ついて** だ」というような記述です。

このようにコンテンツを記述することで、機械がウェブページやウェブサイトをデータベースとして扱うことができるようになります。規模が大きくなれば、セマンティックウェブ[121]、すなわち巨大なグローバル情報データベースを構築できます。

120. *https://en.wikipedia.org/wiki/Linked_data*
121. *https://www.techrepublic.com/article/an-introduction-to-tim-berners-lees-semantic-web/*

> *セマンティックウェブとは、ウェブ上のデータを、表示目的だけでなく、自動化、統合、様々なアプリケーション間でのデータの再利用ができるように定義・リンクするという考えを実現するために、W3Cが始めた長期プロジェクトの名称です。*
>
> *– Greg Ross、Tim Berners-Leeのセマンティックウェブ入門*[122]

それは、ビジネス、テクノロジー、そして社会に豊かな可能性をもたらします。

検索エンジン、そしてその先へ

現在、構造化データのもっとも広範な消費者は、*サーチエンジンとソーシャル・メディア・プラットフォーム*である。

ほとんどの主要な検索エンジンでは、ウェブサイトの所有者は、さまざまな種類の構造化データをウェブサイトに実装することによって、さまざまな形式の *豊富な結果*（可視性とトラフィックに影響を与える可能性があります）の資格を得ることができます。

実際、検索エンジンは、構造化データの一般的な採用（および 教育周り[123]）において、ウェブ全体で非常に大きな役割を担っています。この章は、Web Almanac 去年のSEOの章[124]から生まれたということです。近年では、検索エンジンの影響もあり、schema.org[125]が構造化データの選択語彙として普及しました。

さらに、ソーシャルメディアプラットフォームは、コンテンツがプラットフォーム上で共有（またはリンク）されたときに、コンテンツの読み方や表示方法に影響を与えるため構造化データに依存しています。これらのプラットフォームにおけるリッチなプレビュー、カスタマイズされたタイトルと説明文、およびインタラクティブ性は、多くの場合、構造化データによって実現されています。

しかし、ここで見て理解すべきことは、検索エンジンの最適化やソーシャルメディアのメリットだけではありません。構造化データの規模、多様性、影響、そして*可能性*はリッチリザルト、検索エンジンをはるかに超え、schema.orgをはるかに超えるものです。

たとえば、構造化されたデータによって容易になる。

- 複数のページ、ウェブサイト、概念にまたがるトピックのモデリングとクラスタリングが容易になり、新しいタイプの調査、比較、サービスが可能になります。

122. *https://www.techrepublic.com/article/an-introduction-to-tim-berners-lees-semantic-web/*
123. *https://developers.google.com/search/docs/advanced/structured-data/intro-structured-data*
124. *https://almanac.httparchive.org/ja/2020/seo#構造化データ*
125. *https://schema.org/*

- アナリティクスデータを充実させ、コンテンツやパフォーマンスの分析をより深く、水平化して行えるようにする。
- 業務システムやウェブサイトのコンテンツを照会するための統一された（少なくとも接続された）言語と構文を作成すること。
- セマンティック検索：検索エンジン最適化に使用されるのと同じリッチなメタデータを使用して、内部検索システムを構築・管理すること。

私たちの研究結果は、必然的に検索エンジンの影響によって形作られますが、私たちは構造化データの他のタイプ、フォーマット、およびユースケースについても調査したいと考えています。

構造化データの種類とカバー率

構造化データには、多くの形式、標準、構文があります。私たちは、私たちのデータセット全体で、これらのうちもっとも一般的なものについてのデータを収集しました。

具体的には、以下のような構造化されたデータを特定し、抽出しています。

- Schema.org[126]
- Dublin core[127]
- ソーシャルネットワークで使用されるメタタグ:
    - Open Graph[128]
    - Twitter[129]
    - Facebook[130]
- Microformats[131] (とmicroformats2[132])
- RDFa[133], Microdata[134] とJSON-LD[135]

これらは、さまざまなユースケースとシナリオの広い概要を提供し、レガシーな標準と最新のアプローチの両方を含みます（たとえば、microformatsとJSON-LD）。

126. *http://schema.org/*
127. *https://www.dublincore.org/specifications/dublin-core/*
128. *https://ogp.me/*
129. *https://developer.twitter.com/en/docs/twitter-for-websites/cards/guides/getting-started*
130. *https://developers.facebook.com/docs/sharing/webmasters/*
131. *http://microformats.org/*
132. *https://microformats.org/wiki/microformats2*
133. *https://en.wikipedia.org/wiki/RDFa*
134. *https://en.wikipedia.org/wiki/Microdata_(HTML)*
135. *https://json-ld.org/*

さまざまな構造化データ型における具体的な使用方法を探る前に、いくつかの注意点を簡単に調べておきましょう。

データの注意点

1. コンテンツマネジメントシステムの影響

評価したページの多くは、WordPress[136] や Drupal[137] などの コンテンツ管理システム（CMS）を使用しているWebサイトのものでした。これらのシステム、あるいはその機能を強化するテーマやプラグイン、モジュールは、分析対象の構造化データを含むHTMLマークアップを生成する役割を担っていることがよくあります。

つまり、私たちの調査結果は、もっとも普及しているCMSの動作や出力と一致するように偏ることを意味します。たとえば、Drupalを使用している多くのウェブサイトは、RDFaの形で構造化データを自動的に出力しますし、WordPress（かなりの割合のウェブサイトを動かす）はしばしばテンプレートコードにmicroformatsマークアップを含めます。これは、私たちの調査結果の形状に大きく寄与しています。

2. ホームページのみのデータの限界

残念ながら、このデータ収集方法の性質と規模から、分析対象はホームページのみ（つまり、評価した各ホスト名の *root URL*）に限定されます。

そのため、収集・分析できるデータの量が大幅に制限され、収集したデータの種類も偏ってしまうのは間違いありません。

多くのホームページは、より具体的なページへの入り口として機能しているため、今回の分析では、より深いページに存在するコンテンツの種類を過小評価していると考えるのが妥当であろう。その中には、*記事*、*人*、*製品*などの情報が含まれている可能性があります。

逆に、ホームページによくある情報やすべてのページに存在するサイト全体の情報、たとえばウェブページ、ウェブサイト、*組織*に関する情報などは、過剰にインデックスされる可能性が高いです。

3. データの重複

構造化データ形式の性質上、このような分析を大規模かつクリーンに行うことは困難です。多くの場合、構造化データは複数のフォーマットで実装され（重複することも多い）、構文

136. https://wordpress.org/
137. https://www.drupal.org/

とボキャブラリーの境界線があいまいになっています。

たとえば、FacebookやOpen Graphのメタデータは、技術的にはRDFaのサブセットと言えます。つまり、私たちの調査では、Facebookのメタタグを含むページをFacebookのカテゴリと`RDFa`のセクションで識別しています。私たちは、このようなタイプの重複やニュアンスを整理し、正規化し、意味をなすように最善を尽くしてきました。

4. モバイルメトリクス

今回のデータセット全体を通して、構造化データの採用と有無は、デスクトップとモバイルのデータセットでごくわずかに異なるだけです。そのため、ここでは簡潔にするため、主に*モバイル*データセットに焦点を当てて説明します。

タイプ別使用状況

このセットの多くのページで、さまざまな種類の構造化データが、存在することがわかります。

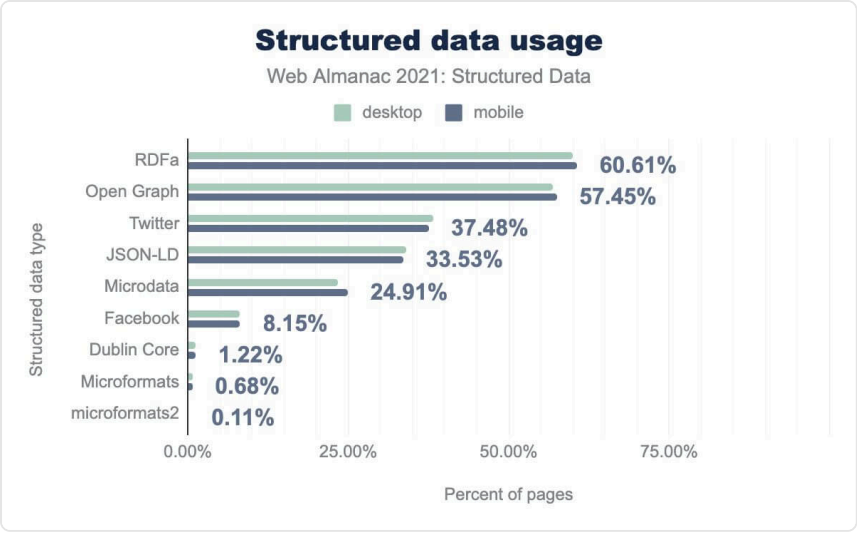


図4.1. 構造化データの利用

また、とくに*RDFa*と*Open Graph*タグは非常に多く、それぞれ60.61%、57.45%のページに表示されていることが分かります。

一方、*Microformats*や*microformats2*などのレガシーフォーマットは、ページの1%未満にしか

表示されません。

シンタックスタイプ別カバレッジ

ある種の構造化データがいつ存在するかを特定するだけでなく、それが記述するデータの種類に関する情報を収集します。それぞれを分解し、各フォーマットと構文がどのように使用されているかを調べます。

RDFa

Resource Description Framework in Attributes[138](RDFa)は、2015年にW3Cが発表したリンクデータのマークアップのための技術である。マークアップに属性を追加することで、Webページ上の視覚情報を補強し、翻訳することができる。

たとえば、ウェブサイトの所有者は、ハイパーリンクに `rel="license"` 属性を追加して、それがライセンス情報ページへのリンクであることを明示的に記述できます。

138. *https://www.w3.org/TR/rdfa-lite/*

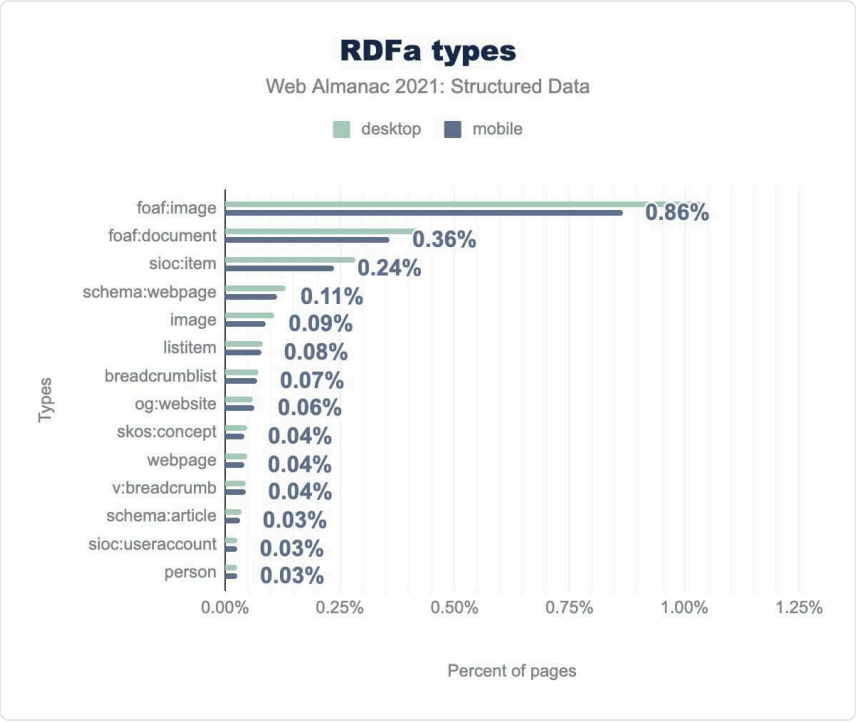


図4.2. RDFaタイプ

RDFaのタイプを評価すると、`foaf:image`構文が他のタイプよりもはるかに多くのページに存在し、データセットの全ページの0.86%以上であることがわかります。これは小さな割合に見えるかもしれませんが、65,000ページを表し、発見されたRDFaマークアップ全体の60%以上に相当します。

この異常値を超えると、RDFaの使用はかなり減少し、断片的になるが、まだ興味深い発見がある。

FOAFについて

FOAF[139] (または “Friend of a Friend”) は、2000年代初頭に作られた、人に関する用語のリンクデータ辞書である。人、*グループ*、*文書*の記述に使用できます。

FOAFはW3CのRDF構文を使っており、原文紹介[140]では以下のように説明されています。

139. *http://xmlns.com/foaf/spec/*
140. *https://web.archive.org/web/20140331104046/http://www.foaf-project.org/original-intro*

> *ある友人たちが興味を持つ事柄について書かれた、相互に関連するホームページのWebを考えてみよう。ウェブ上に現れるそれぞれの新しいホームページは、世界に新しい何かを伝え、事実やゴシップを提供し、ウェブを断絶された情報の断片の鉱山にしています。FOAFは、これらすべての情報を理解する方法を提供します。*
>
> *FOAFの紹介*[141]

Drupal CMSの古いバージョンでは、デフォルトで `typeof="foaf:image"` と `foaf:document` マークアップがHTMLに追加されているため、この結果では `foaf` マークアップが目立つと思われます。

その他の注目すべきRDFaの発見について

FOAFのプロパティだけでなく、他のさまざまな標準や構文がリストに表示されます。

注目すべきは、 `sioc:item` （ページの0.24%）や `sioc:useraccount` （ページの0.03%）など、いくつかの `sioc` プロパティが確認できることです。SIOC[142] は、メッセージボード、フォーラム、Wiki、ブログなどのオンラインコミュニティに関連する構造化データを記述するために設計された標準です。

また、SKOS[143] (または “Simple Knowledge Organization System”) のプロパティである `skos:concept` が0.04% のページで見受けられました。SKOSもその1つで、分類や区分（タグやデータセットなど）を記述する方法を提供することを目的とした規格である。

Dublin Core

Dublin Core[144] は、1995年にオハイオ州ダブリンで開催されたOCLC（Online Computer Library Center）とNCSA（National Center for Supercomputing Applications）のワークショップで考案されたリンクデータ標準と相互運用できるボキャブラリーである。

幅広いリソース（デジタルと物理の両方）を記述できるように設計されており、さまざまなビジネスシーンで利用することができる。2000年以降、RDFベースのボキャブラリーの中で非常に人気が高く、W3Cの採択を受けた。

2008年からはDublin Core Metadata Initiative（DCMI）によって管理され、他のリンクデータ語彙との高い相互運用性を維持している。通常、HTML文書内のメタタグの集合体として

141. *https://web.archive.org/web/20140331104046/http://www.foaf-project.org/original-intro*
142. *https://www.w3.org/Submission/sioc-spec/*
143. *https://www.w3.org/TR/skos-primer/*
144. *https://dublincore.org/*

実装される。

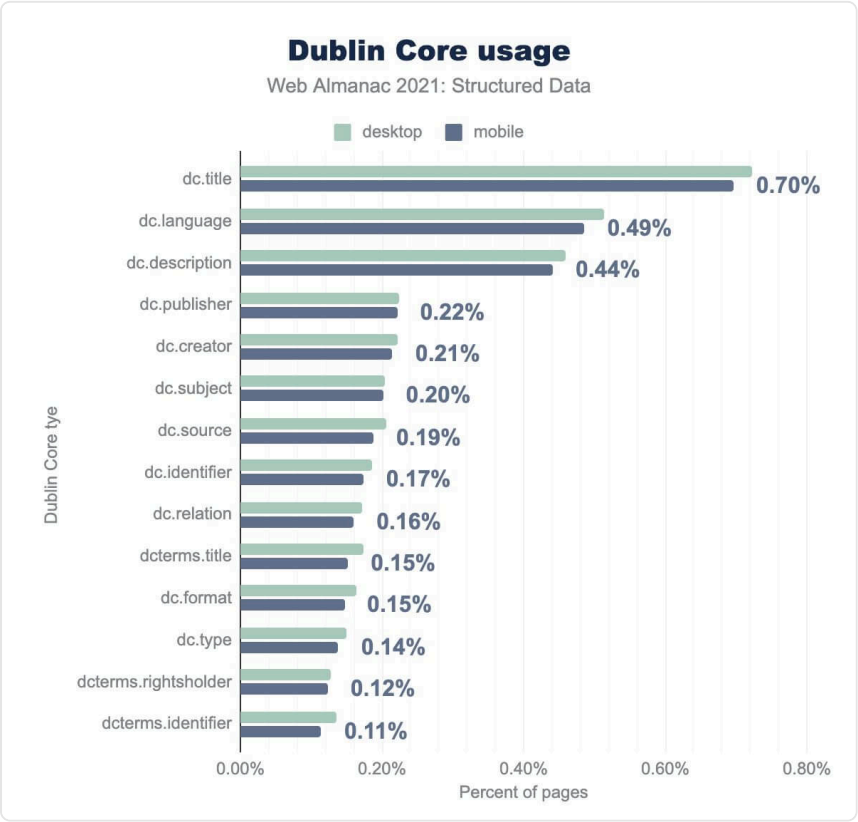


図4.3. Dublin Coreの利用状況

もっとも人気のある属性タイプは`dc:title`（0.70%のページ）であることは、驚くにはあたらない。しかし、`dc:language`が0.49%の普及率で（*description*、*subject*、*publisher*などの一般的な記述子より）次にあるのは興味深いことです。これは、Dublin Coreが多言語のメタデータ管理システムでよく使われていることを考えると、納得がいく。

また、異なる概念間の関係を表現することができる属性である`dc:relation`（0.16%のページ）が比較的多く出現していることも興味深い。

SEOの文脈ではSchema.orgが優勢であると多くの人が思うかもしれませんが、DCの役割は、その概念の幅広い解釈とリンクされたオープンデータ*運動*に深く根ざしていることから、極めて重要であることに変わりはありません。

ソーシャルメタデータ

ソーシャル・ネットワークとプラットフォームは、構造化データの最大の発行者であり消費者でもあります。このセクションでは、構造化データフォーマットの役割、普及率、規模について説明します。

オープングラフ

オープングラフ・プロトコル[145]は、オープンソースの規格で、もともとはFacebookが作成したものです。これは、*コンテンツを共有する*というコンテキストに特化した構造化データの一種で、Dublin CoreやMicroformatsなどの標準を緩やかにベースにしています。

これは、一連のメタタグとプロパティを記述したもので、プラットフォーム間で共有する際にコンテンツをどのように（再）表示するかを定義するために使用することができる。たとえば、投稿に「いいね！」を押したり、埋め込んだり、リンクを共有したりするときに使用します。

これらのタグは通常、HTML文書の `<head>` に実装され、ページの *title*, *description*, *URL*, *featured image* といった要素を定義します。

Open Graphプロトコルは、その後、*Twitter*、*Skype*、*LinkedIn*、*Pinterest*、*Outlook*など、多くのプラットフォームやサービスで広く採用されるようになりました。プラットフォームが共有/埋め込みコンテンツの表示方法について独自の基準を持っていない場合（時には持っている場合でも）、Open Graphタグはデフォルトの動作を定義するためによく使用されます。

145. *https://ogp.me/*

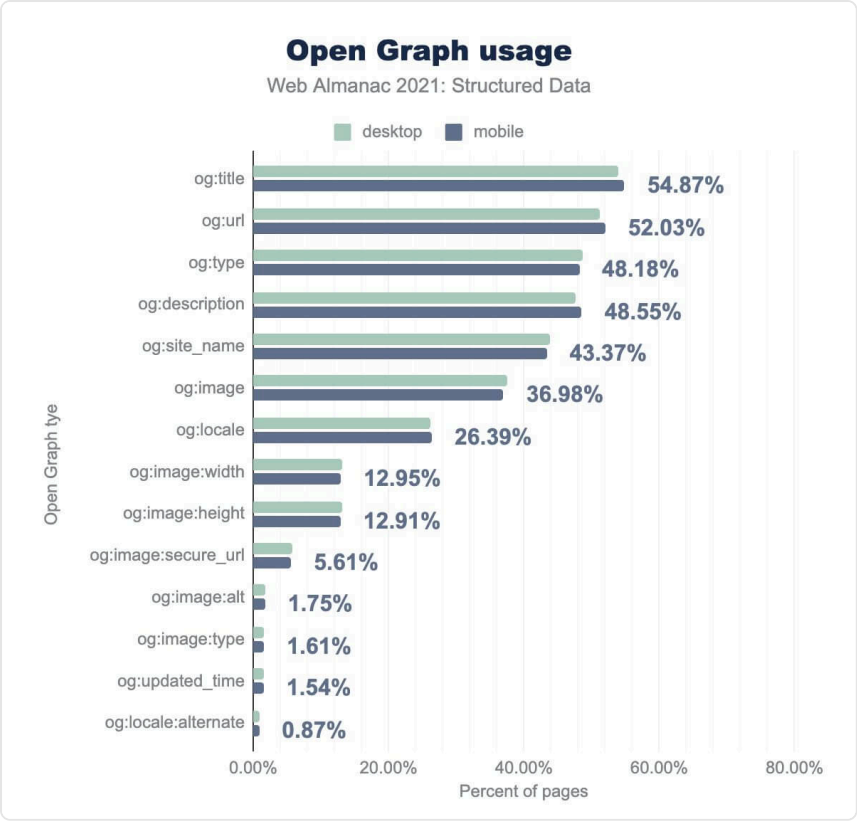


図4.4. Open Graphの利用状況

もっとも一般的なOpen Graphタグは `og:title` であり、54.87%のページで見受けられる。これは、どのような種類のものが表現されているか（例： `og:type` ページの48.18%に掲載）、どのように表現されるべきか（例： `og:description` ページの48.55%に掲載）を記述する関連属性のセットに密接に追随しています。

これらのタグは、サイト上のすべてのページの `<head>` で使用される「定型」タグの一部として一緒に使用されることが多いので、この狭い分布は予想されることです。

やや少ないのは `og:locale`（26.39%のページ）で、これはページのコンテンツの言語を定義するために使用されます。

さらに少ないのは、 `og:image` タグに関するより具体的なメタデータで、 `og:image:width`（12.95%のページ）、 `og:image:height`（12.91%のページ）、 `og:image:secure_url`（5.61% のページ）と `og:image:alt`（1.75% のページ）の形式です。 HTTPSの採用がますます一般的になっている現在、

`og:image:secure_url`（`og:image`の`https`バージョンを識別するためのもの）はほとんど冗長になっていることは注目に値します。

これらの例を超えると使用頻度は急速に低下し、（多くの場合、不正、非推奨、または誤った）タグのロングテールになってしまいます。

Twitter

TwitterはOpen Graphタグをフォールバックやデフォルトとして使用していますが、このプラットフォームは独自の構造化データをサポートしています。TwitterでURLが共有されたときに、どのようにページを表示するかを定義するために、特定のメタタグ（すべて接頭辞が`twitter:`）を使用できます。

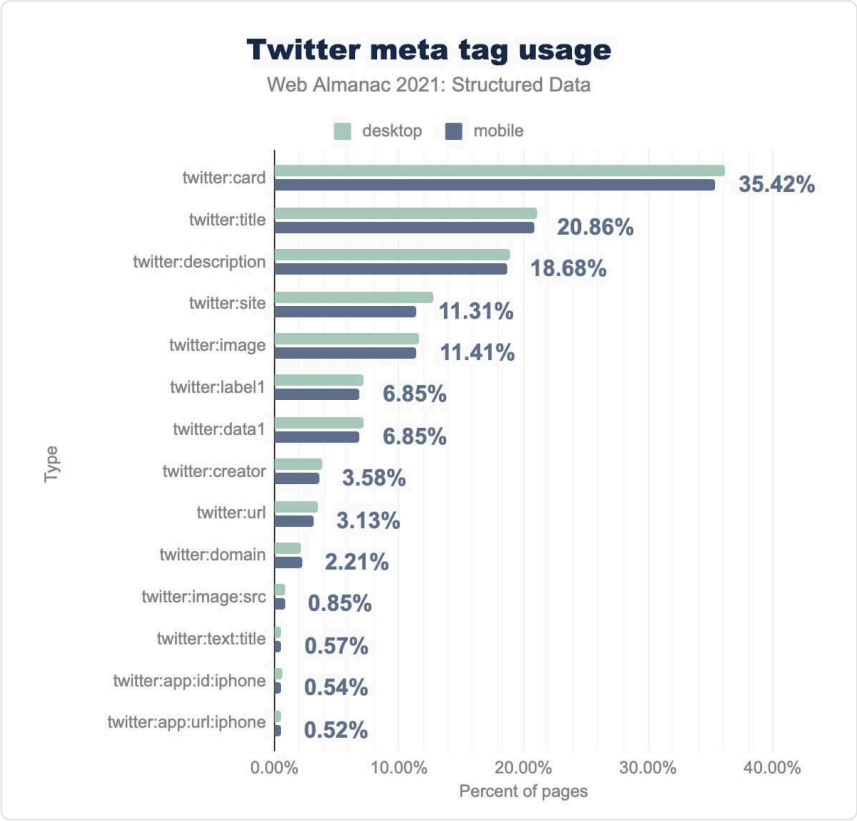


図4.5. Twitterのメタタグの利用状況

もっとも一般的なTwitterのメタタグは`twitter:card`で、全ページの35.42%に見られまし

た。このタグは、プラットフォームで共有される際のページの表示方法を定義するために使用できます（例：*summary*、メディアオブジェクトに関する追加データと組み合わせた場合の*player*など）。

この異常値を超えると、採用率は急降下する。次に多いタグは`twitter:title`と`twitter:description`（どちらも共有URLの表示方法を定義するために使用）で、それぞれ全ページの20.86%と18.68%に表示されます。

これらのタグや、`twitter:image`タグ（ページの11.41%）と`twitter:url`タグ（ページの3.13%）の普及が進まない理由は、Twitterが定義されていない場合、同等のOpen Graphタグ（`og:title`, `og:description`と`og:image`）に戻るためであると理解できます。

また、注目すべきは

- 当該ウェブサイトに関連するTwitterアカウントを定義する`twitter:site`タグ（11.31%のページ）です。
- `twitter:creator`タグ（3.58%）は、ウェブページのコンテンツの作者のTwitterアカウントを定義しています。
- `twitter:label1`タグと`twitter:data1`タグ（ともに6.85%のページに掲載）は、ウェブページに関するカスタムデータと属性を定義するために使用できます。追加のラベル／データのペア（例：`twitter:label2`と`twitter:data2`）も、かなりの数（0.5%）のページに存在しています。

これらの例を超えると使用頻度は急速に低下し、（多くの場合、不正、非推奨、または誤った）タグのロングテールになってしまいます。

Facebook

Facebookは、Open Graphタグに加えて、Webページをプラットフォーム上の特定のブランド、プロパティ、人物に関連付けるための追加のメタデータ（metaタグ、接頭辞は`fb:`）をサポートしています。

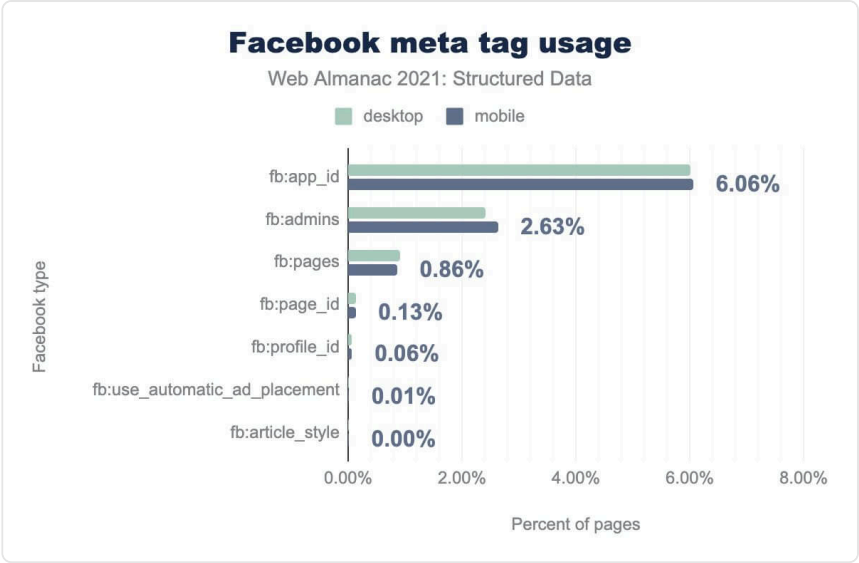


図4.6. Facebookメタタグの利用状況

検出されたFacebookタグのうち、有意に採用されたタグは3つだけです。

これらは `fb:app_id`, `fb:admins`, `fb:pages` で、それぞれ6.06%, 2.63%, 0.86% のページで確認されました。

これらのタグは、ウェブページとFacebookページ/ブランドを明示的に関連付けるため、またはこれらのプロファイルを管理するユーザー（または複数のユーザー）に権限を付与するために使用されます。

また、Facebook社がどの程度サポートしているかは不明です。Facebookはここ数年で大きく変化しており、技術文書もあまり整備されていません。しかし多くのコンテンツ管理システム、テンプレート、ベストプラクティスガイド、そしてFacebookのデバッグツールの中には、まだこれらが含まれ、参照されているものがあります。

MicroformatsとMicroformats2

Microformats（一般に `μF` と略される）は、HTMLにセマンティクスと構造化データを埋め込むためのメタデータのオープンデータ規格です。

これらは、見出しや段落などの通常のHTML要素の背後にある意味を記述する、定義されたクラスのセットで構成されています。

この構造化データのフォーマットは、広く採用されている標準規格（セマンティック

（X）HTML）を再利用してセマンティクスを伝えることを指針としています。公式ドキュメント[146]は、Microformatsを「人間が第一、機械が第二のために設計された」と説明し、「既存の広く採用されている標準に基づいて作られた、シンプルでオープンなデータ形式の集合」であるとしています。

Microformatsは2つのバージョンで提供されています。Microformats v1とMicroformats v2（microformats2）です。2014年3月に導入された後者は、v1を置き換えて優先し、microdataとRDFaの両方の構文から学んだいくつかの重要な教訓を活用しています。

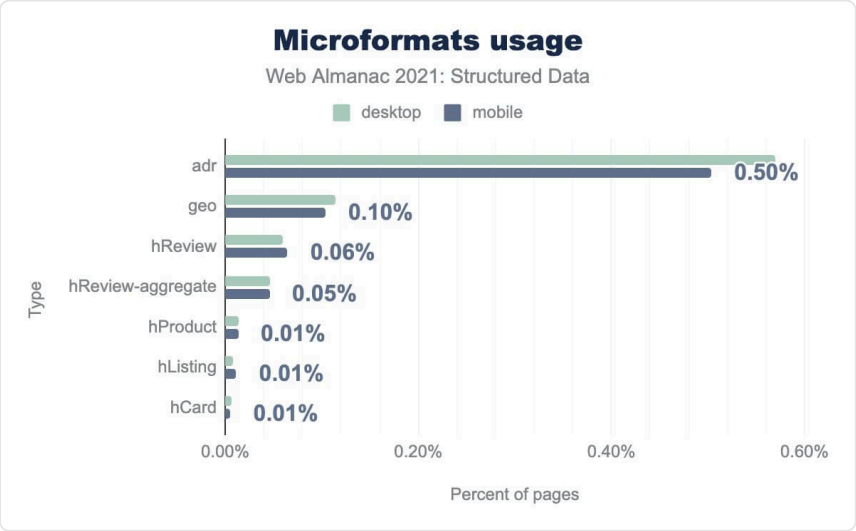


図4.7. Microformatsの利用状況

歴史的に、またその性質上（HTMLの拡張として）、Microformatsは企業や組織のプロパティを記述するために、ウェブサイト開発者によって多用されてきました。とくにローカルビジネスを促進するページ。これは、`adr`プロパティ（ページの0.50%）、レビュー（`hReview`、ページの0.06%）、その他のローカルビジネスとその製品/サービスを特徴付けるための情報が目立つことを説明するのに大いに役立ちます。

146. *https://microformats.org/wiki/what-are-microformats*

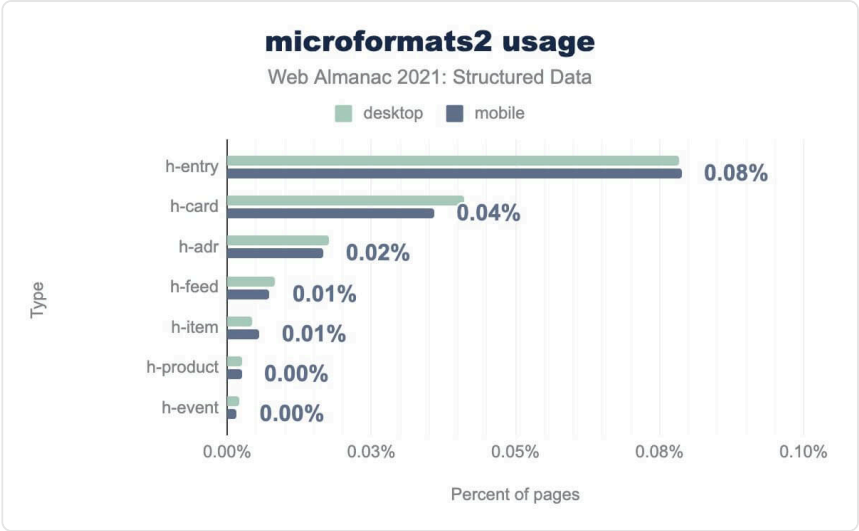


図4.8. microformats2の利用状況

レガシーなマイクロフォーマットとよりモダンなバージョンの違いは大きく、マークアップの使用における行動や嗜好の変化に関する興味深い洞察が得られます。

`adr` クラスが古典的なmicroformatsのデータセットを支配していたところ、同等の `h-adr` プロパティは0.02% のページ上にしか存在しません。この結果は、代わりに `h-entry` プロパティ（ページの0.08% にあり、ブログ記事と同様のコンテンツユニットを記述）と `h-card` プロパティ（ページの0.04% にあり、組織や個人の *名刺* を記述）によって支配されています。

この差の原因として考えられるのは、次の3点である。

- 一般的なクラス名（ `adr` など）のデータは、私たちのmicroformats v1データでは、ほぼ確実に過大評価されています。これらの値が、*構造化*データとより一般的な理由（たとえば、CSSルールを伴うHTMLクラス属性値として）で使われる場合を見分けるのは困難です。
- 一般的なマイクロフォーマットの使用は（種類を問わず）大幅に減少し、他のフォーマットに置き換えられています。
- 多くのウェブサイトやテーマでは、一般的なデザイン要素やレイアウトに `h-entry` (時には `h-card`) マークアップをまだ含んでいます。たとえば、多くのWordPressテーマでは、メインコンテンツコンテナーに `h-entry` クラスが出力され続けています。

Microdata

microformatsやRDFaと同様に、microdata[147]は、HTML要素に属性を追加することを基本としています。microformatsとは異なりますが、RDFaと同様に、定義された意味の集合に縛られることはありません。この規格は拡張可能で、作者はどのボキャブラリーのデータを記述するかを宣言することができる。もっとも一般的なのはschema.orgである。

microdataの限界の1つは、エンティティ間の抽象的で複雑な関係を記述するのが困難なことで、その関係がページのHTML構造に明示的に反映されていない場合です。

たとえば、ある*組織*の*営業時間*を記述する場合、その情報が文書内で同時に、または論理的に構成されていないと難しいかもしれません。なお、この問題を解決するための標準や方法論（たとえば、インラインの `<meta>` タグやプロパティを含めるなど）はありますが、広く採用されているわけではありません。

147. *https://en.wikipedia.org/wiki/Microdata_(HTML)*

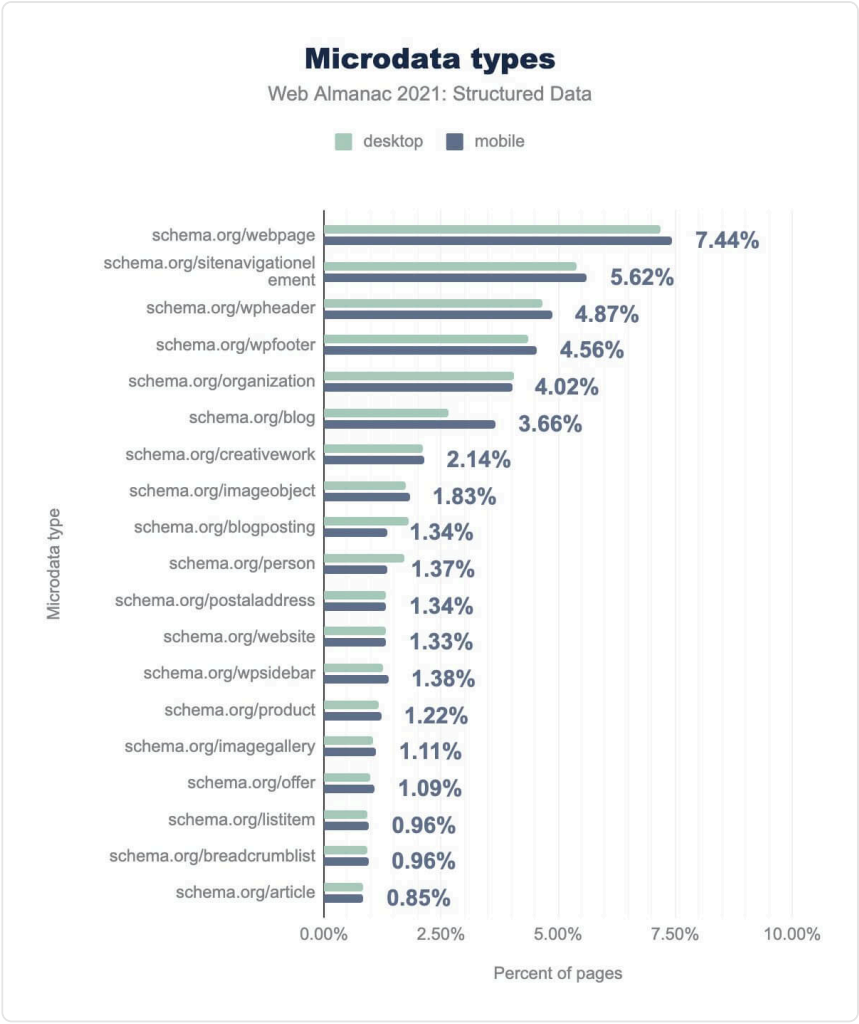


図4.9. Microdataの種類

分析したページでもっとも一般的なmicrodataのタイプは、Webページそのものを記述するもので、`webpage`（7.44%のページ）、`sitenavigationelement`（5.62%のページ）、`wpheader`（4.87% of pages）と`wpfooter`（4.56% of pages）などのプロパティが利用されました。

なぜこのような構造的な記述子がコンテンツ記述子（`person`や`product`など）よりも目立つのかは簡単に推測できます。マイクロデータの作成と維持には、コンテンツ制作者がコンテンツに特定のコードを追加する必要がありますが、それはコンテンツレベルよりもテン

プレートレベルで行う方が、簡単なことが多いのです。

マイクロデータの強みの1つは、HTMLマークアップとの明確な関係（およびその中でのオーサリング）ですが、そのため、マイクロデータを使用するための技術的な知識と能力を持つコンテンツ制作者へのアプローチは限られています。

とはいえ、マイクロデータの種類は幅広く、さまざまなものが採用されていることがわかります。特筆すべきは

- `Organization` (4.02%)は、通常、ウェブサイトを公開する会社、製品の製造会社、著者の雇用主、または同様のものを表します。
- `CreativeWork`（2.14%）は、すべての文章と画像コンテンツ（例：ブログ記事、画像、ビデオ、音楽、アート）を記述するもっとも一般的な親タイプです。
- `BlogPosting` (1.34%)、これは個々のブログ投稿を記述します（一般に、著者として `Person` も特定されます）。
- `Person`（1.37%）これは、コンテンツの作者やページに関係する人（例：ウェブサイトの発行者、出版組織のオーナー、商品を販売する個人など）を表すのによく使われます。
- `Product` (1.22%) と `Offer` (1.09%) は、一緒に使用すると、購入可能な製品を説明します（通常、価格、レビュー、入手可能性を説明する追加のプロパティが付きます）。

JSON-LD

microdataやmicroformatsとは異なり、JSON-LD[148] はHTMLマークアップにプロパティやクラスを追加することによって実装されるものではありません。その代わり、機械可読のコードは、JavaScript Object Notationの1つまたは複数の独立した塊としてページに追加されます。このコードには、ページ上の実体とその関係の説明が含まれています。

この実装は、ページのHTML構造と直接結びついていないため、複雑な関係や抽象的な関係を記述したり、人間が読めるページの内容では容易に得られない情報を表現することがはるかに容易になります。

148. *https://json-ld.org/*

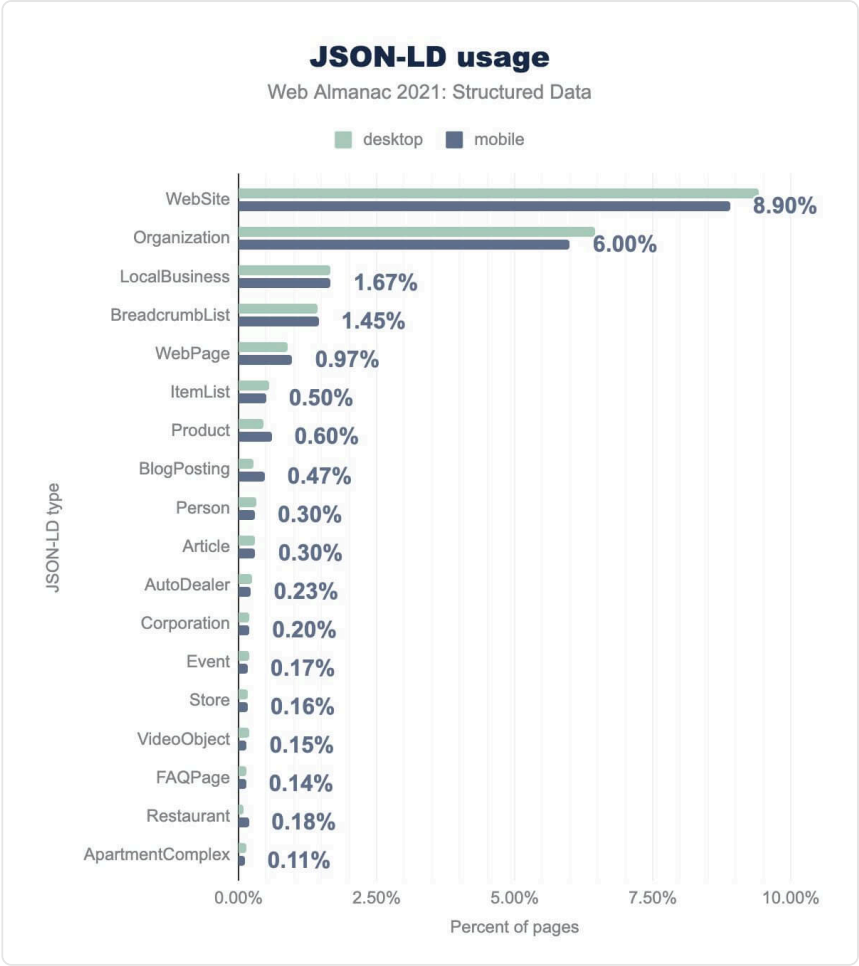


図4.10. JSON-LDの利用状況

予想されるように、今回の結果はmicrodataの利用を評価した結果と似ています。どちらのアプローチもschema.orgの使用に大きく偏っているため、この結果は拡大解釈されるでしょう。しかし、いくつかの興味深い違いがあります。

JSON-LD形式は、サイトオーナーがHTMLマークアップとは別にコンテンツを記述できるため、ページのコンテンツへそれほど厳密に縛られない、より抽象的で複雑な関係を表現することが容易になる可能性があるのです。

このことは、今回の調査結果にも反映されており、より具体的で構造化された記述子がマイクロデータよりも一般的であることがわかる。たとえば

- `BreadcrumbList` (ページの1.45%) は、ウェブサイトにおけるウェブページの階層的な位置を説明します（各親ページについても説明します）。
- `ItemList` (ページの0.5%): *レシピの ステップ*や *カテゴリー の 製品* など、一連のエンティティを記述します。

これらの例以外では、マイクロデータの時と同じようなパターンが引き続き見られます（規模はかなり小さいですが）。Webサイト、地元企業、組織、Webページの構造に関する記述が、広範な採用の大部分を占めています。

JSON-LDの構造および関係

JSON-LDの主な利点の1つは、他のフォーマットよりもエンティティ間の関係をより簡単に記述できることである。

たとえば、*イベント*には、主催する*法人*があり、特定の*場所*で開催され、*オファー*の一部としてチケットが、販売されることがあります。そのイベントを説明する*ブログ記事*には*作成者*があり、といった具合に。このような関係の記述は、他の構文に比べてJSON-LDの方がはるかに簡単で、実体に関する豊かな物語を伝えるのに役立ちます。

しかし、こうした関係は往々にして深く、複雑に絡み合っていくものです。したがって、この分析では、エンティティ間のもっとも一般的なタイプの関係のみを調べ、ツリーや関係構造全体を評価することはしません。

以下は、すべての構造/関係値において、タイプ間のもっとも一般的な接続を、その頻度に基づいて示しています。なお、これらの構造や価値は、より大きな関係性の連鎖の一部であるため、時には重なり合うこともあります。

関係性	デスクトップページの%	モバイルページの%
WebSite > potentialAction > SearchAction	6.44%	6.15%
	5.06%	4.85%
@graph > WebSite	4.89%	4.69%
WebPage > isPartOf > WebSite	4.02%	3.81%
@graph > WebPage	4.01%	3.79%
BreadcrumbList > itemListElement > ListItem	3.93%	3.78%
Organization > logo > ImageObject	2.85%	3.03%
@graph > BreadcrumbList	3.18%	2.99%
WebPage > potentialAction > ReadAction	2.92%	2.71%
WebPage > breadcrumb > BreadcrumbList	2.60%	2.44%
WebSite	2.49%	2.30%
@graph > Organization	2.26%	2.13%
WebSite > publisher > Organization	2.22%	2.09%
Product > offers > Offer	1.47%	1.89%
Product	1.41%	1.73%
@graph > ImageObject	1.80%	1.71%
ItemList > itemListElement > ListItem	1.71%	1.69%
@graph > SiteNavigationElement	1.70%	1.66%
WebPage > primaryImageOfPage > ImageObject	1.67%	1.59%

図4.11. JSON-LD entity relations.

もっとも多い構造は、`website`、`potentialAction`、`SearchAction`スキーマの関係です（構造の6.15%を占めます）。この関係により、Googleの検索結果に*サイトリンク検索ボックス*を利用できます。

おそらくもっとも興味深いのは、次に多い構造（関係の4.85%）が、関係を定義していないことである。これらのページは、もっとも単純なタイプの構造化データのみを出力し、個々の孤立した実体とそのプロパティを定義する。

次に多い構造（リレーションシップの4.69%）は、`@graph`プロパティを導入しています（`Website`の記述と合わせて）。`@graph`プロパティはそれ自体ではエンティティでないですが、JSON-LDではエンティティ間の関係を含み、グループ化するために使用できます。

さらに関係を探っていくと、`WebPage > isPartOf > WebSite`（関係の3.81%）、`Organization > logo > ImageObject`（関係の3.03%）、`WebSite > publisher > Organization`（関係の2.09%）など、コンテンツと組織の関係についてさまざまな記述が見受けられるようになりました。

また、パンくずナビに関連する構造もたくさん見ることができます。

- `BreadcrumbList > itemListElement > ListItem`（3.78%の関係性）
- `@graph > BreadcrumbList`（2.99%の関係）
- `ItemList > itemListElement > ListItem`（1.69%の関係性）

これらのもっとも一般的な構造以外にも、`ApartmentComplex > amenityFeature > LocationFeatureSpecification`(0.1%の関係性）や`AutoDealer > department > AutoRepair`(0.04%の関係性）、`MusicEvent > performer > PerformingGroup`(0.01%の関係性）など、あらゆるエンティティ、コンテンツタイプ、概念を説明する非常に長いテールの関係を見てとることができます。

このような構造や関係は、ウェブサイトのホームページの分析に限られているため、今回のデータセットが示すよりもはるかに一般的である可能性があることを再度確認しておきます。たとえば、何千もの集合住宅を個別に掲載しているウェブサイトが、内側のページでそれを行っている場合、このデータには反映されないということです。

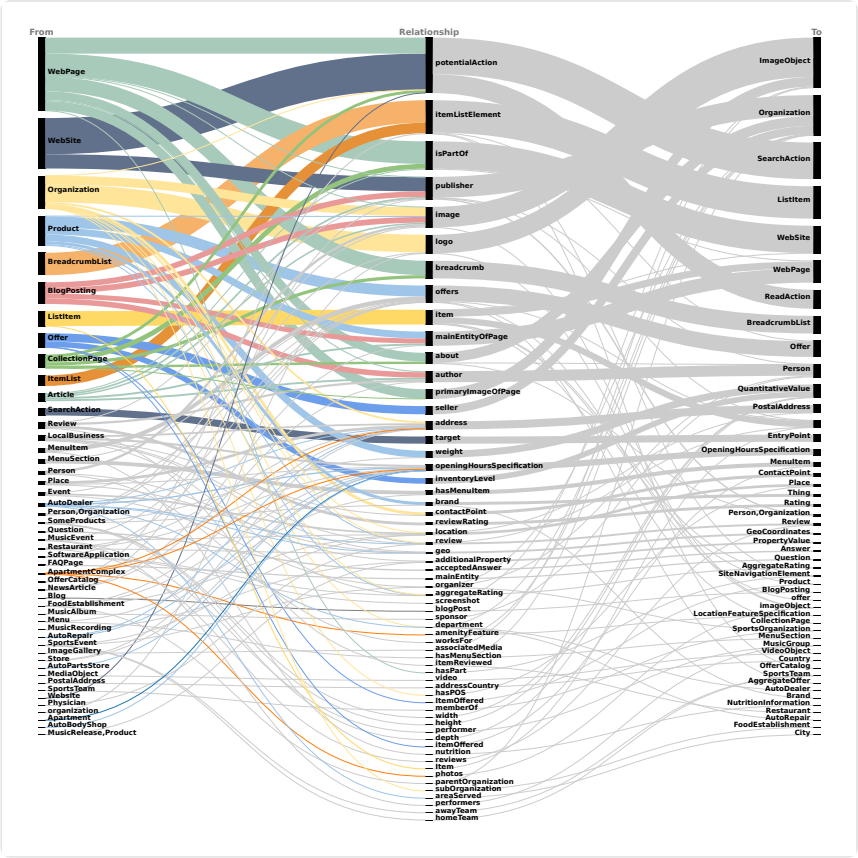


図4.12. JSON-LDのエンティティ関係をSankey図にしたもの。

モバイルページにおけるJSON-LDエンティティ間の相関を示し、フローとして表現することで、エンティティやリレーションシップを視覚的に結びつける図です。各クラスはクラスター内の一意な値を表し、高さはその頻度に比例する。

このグラフでは、分析対象は頻度の高い上位200チェーンに限定しています。

また、このグラフから、一般出版からeコマース、地域ビジネス、イベント、自動車、音楽など、これらのグラフの背景にある分野を概観できます。

関係性の深さ

さらに、モバイルとデスクトップのデータセットにおいて、エンティティ間のもっとも深く複雑な関係を計算しました。

関係が深ければ深いほど、エンティティ（およびそれに関連する他のエンティティ）のより豊かで包括的な記述に等しくなる傾向があります。



図4.13. デスクトップでもっとも深いネスト関係。

深い関係性は

- デスクトップでは、18のネストした接続の深さ。
- モバイルの場合、12個のネスト接続の深さ。

このような構造は、規模が大きくなると記述や維持が困難になるため、手作りのマークアップではなく、プログラムによる出力生成の可能性を示唆するものであると考える価値があります。

`sameAs` の使用

構造化データのもっとも強力な使用例の1つは、ある実体が他の実体と `sameAs` であることを宣言することである。あるものを包括的に理解するためには、多くの場合、複数の場所や形式に存在する情報を消費する必要があります。それぞれのインスタンスが他のインスタンスと相互参照できる方法があれば、「点と点を結ぶ」ことが容易になり、その実体をより豊かに理解できます。

このように、 `sameAs` は非常に強力なツールであるため、ここではもっとも一般的な `sameAs` の使用方法と関係性について、時間をかけて調べてみました。

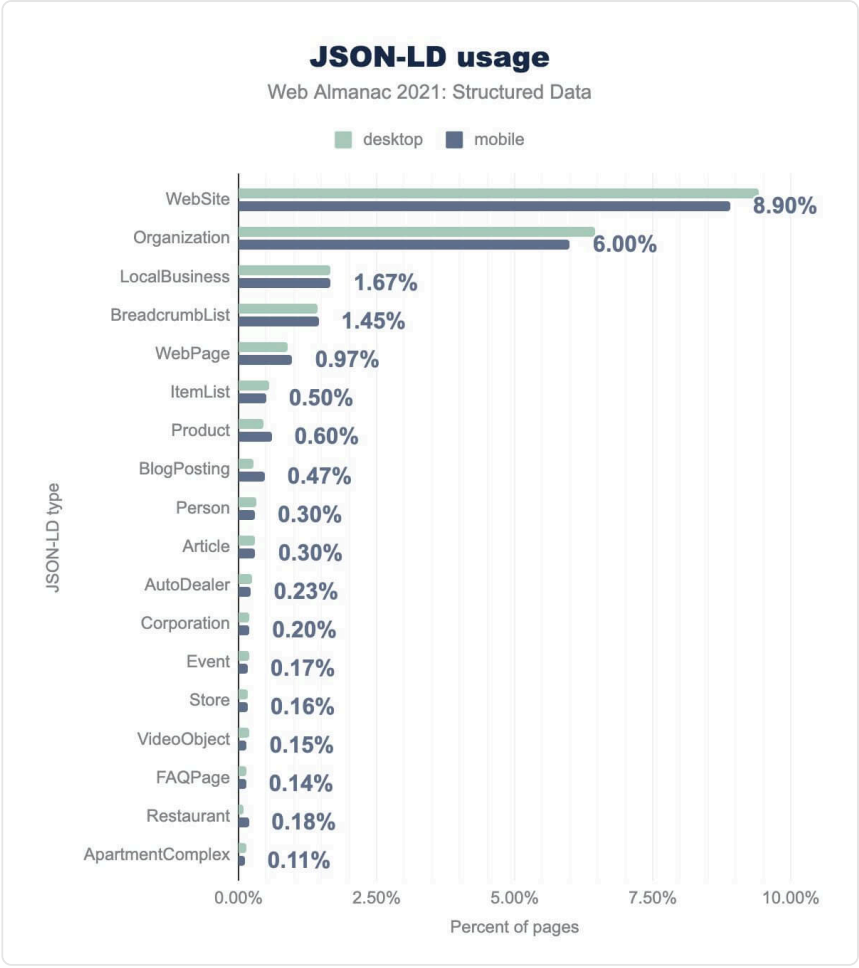


図4.14. SameAsの使用状況

`sameAs` プロパティは全JSON-LDマークアップの1.60%を占め、ページの13.03%に存在しています。

もっとも一般的な `sameAs` プロパティ（*URL からホスト名への正規化*）の値は、ソーシャルメディア プラットフォーム（例：facebook.com、instagram.com）と公式ソース（例：wikipedia.org、yelp.com）で、前者の合計が使用率の約75% を占めていることがわかります。

このプロパティは、主にウェブサイトや企業のソーシャルメディアアカウントを特定するために使用されていることは明らかです。おそらく、Googleが検索結果のナレッジパネルを管

理するための入力として、このデータに歴史的に依存していることが動機となっているのでしょう。この要件が2019年に非推奨[149]となったことを考えると、このデータセットが今後、徐々に変化していくことが予想されるかもしれません。

結論

構造化データは、ウェブ上で広く、そして多様に使用されています。その一部は間違いなく陳腐化していますが（時代遅れのフォーマットを使用したレガシーなサイトやページ）、新しい標準や新興の標準も強力に採用されています。

schema.org（とくにJSON-LD経由）のような最新の標準の採用は、ページやコンテンツに関するデータの提供に対する検索エンジンのサポート（および報酬）を利用したい組織や個人によって動機づけられているように見受けられるという逸話があります。しかし、これ以外にも、他の理由で構造化データを使ってページを充実させている人たちの豊かな風景があるのです。他のシステムと統合し、コンテンツをよりよく理解するため、あるいは*他*の人が*自分自身*のストーリーを語り、独自の製品を構築するのを容易にするためウェブサイトやコンテンツを記述するのです。

深く結びついた構造化データからなるウェブが、より統合された世界を動かすというのは、長い間SF的な夢だった。しかし、おそらく、もうそれほど長くはないでしょう。これらの規格が進化し、その普及が進むにつれ、私たちはエキサイティングな未来への道を切り開くことになります。

今後の展開

将来的には、ここで始めた分析を継続し、構造化データ利用の時間的な変遷をマップ化できるようにしたいと考えています。

さらなる探求を期待しています。

149. https://x.com/googlesearchc/status/1143558928439005184

著者



Jono Alderson

@jonoalderson jonoalderson https://www.jonoalderson.com

デジタル戦略家、マーケティングテクノロジスト、フルスタックデベロッパー。Webサイトのパフォーマンス、テクニカルSEO、schema.org、構造化データなど、あらゆることに手を出すのが好き。

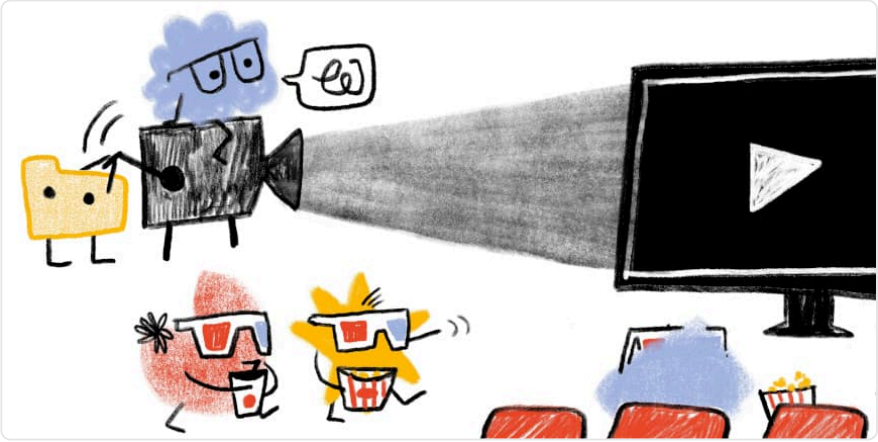


Andrea Volpini

@cyberandy cyberandy https://wordlift.io/blog/en/entity/andrea-volpini

アンドレア・ボルピーニはWordLiftのCEOで、現在はセマンティックウェブ、SEO、人工知能に注力しています。

部 I 章 5
メディア



Eric Portis と Doug Sillars によって書かれた。
Navaneeth Krishna、 Tamas Piros、 Akshay Ranganath と Addy Osmani によってレビュ。
Eric Portis、 Doug Sillars と Akshay Ranganath による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

ほぼ30年前 `<img>` タグが生まれました、そしてハイパーテキストは、ハイパーメディアになりました。それ以来、ウェブはますますビジュアル化されています。2021年、Web上のメディアはどうなっているのか？画像と動画を、順番に見ていきましょう。

画像

ウェブ上では、画像はどこにでもあるものです。ほとんどすべてのページが画像コンテンツを含んでいます。

95.9%

図5.1. 少なくとも1つのコンテンツ付き `<img>` が含まれるページの割合

そして、事実上すべてのページが何らかの画像を提供しています（それが単なる背景やファビコンであっても）。

99.9%

図5.2. 画像リソースへのリクエストが1件以上発生したページの割合

これらの映像が持つインパクトは計り知れないものがあります。ページの重さの章で強調しているように、画像は依然として他のどのリソースタイプよりもページあたり多くのバイトを担当しています。しかし、前年比では、1ページあたりの画像転送サイズは減少しています。

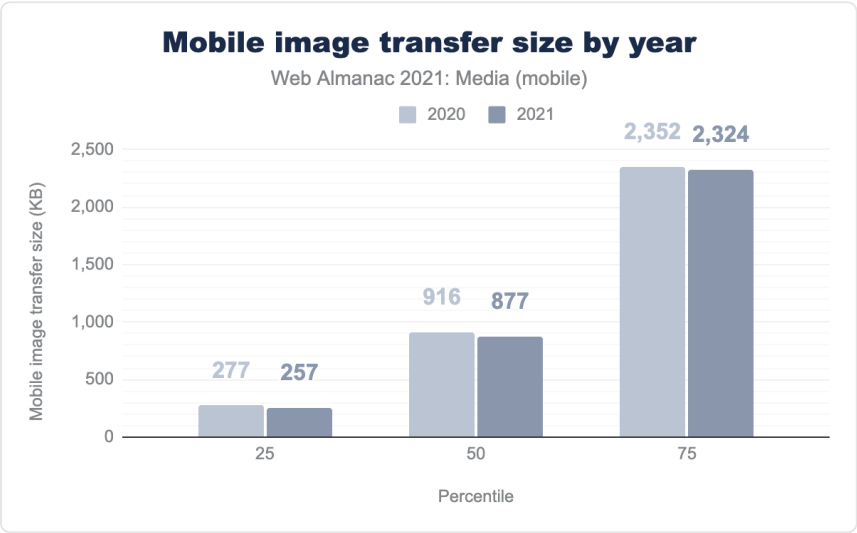


図5.3. 携帯電話の画像転送サイズの年別推移。

これは驚きです。過去10年間、HTTP Archiveの毎月の画像状態レポート[150]に掲載されてる

150. *https://httparchive.org/reports/state-of-images*

Image Bytes[151] チャートは、一見すると上昇する一方です。2021年、この流れを逆転させたのは何だったのか。後ほど詳しく説明する、ネイティブの遅延読み込みの急速な普及と関係があるのではないかと思います。

いずれにせよ、量的に見れば、画像はウェブの中で非常に多くのものを占め続けているのです。しかし、要素数、リクエスト数、バイト数だけでは、画像がユーザーの体験にどれだけ重要であるかはわかりません。それを知るために、Largest Contentful Paint[152] という指標を見てみましょう。これは、任意のページにおけるabove-the-foldコンテンツのもっとも重要な部分を識別しようとするものです。パフォーマンスの章にあるように、LCP要素は約4分の3のページで画像が表示されます。



図5.4. LCP要素に画像があるモバイルページ。デスクトップでは79.4%です！

画像は、ユーザーのウェブ体験に欠かせないものです。ここでは画像がどのようにエンコードされ、埋め込まれ、レイアウトされ、配信されるのかを詳しく見ていきましょう。

エンコーディング

ウェブ上の画像データは、ファイルにエンコードされています。このファイルや画像データについて、私たちはどのようなことが言えるのでしょうか。

まず、画素の大きさから見てみましょう。まずは小さいサイズから。

1画素の画像

多くの `<img>` 要素は実際にはコンテンツの多い画像[153] を表さず、その代わりに1画素しか含まないのです。

クライアント	1x1の画像
モバイル	7.5%
デスクトップ	7.0%

図5.5. 1画素の画像使用。

151. https://httparchive.org/reports/state-of-images#bytesImg
152. https://web.dev/i18n/ja/lcp/
153. https://www.merriam-webster.com/dictionary/image

これらの1画素 `<img>` 要素は、はっきり言ってハッキングです。レイアウトのため[154]（これはCSSでやったほうがいい）か ユーザーの追跡[155]（これは Beacon API[156] でやったほうがいい）かのいずれかで乱用されているのです。

これらの1画素の画像 `<img>` がどのような仕事をしているかは、データ URI[157] を使っているものがどれだけあるかを見ることで、基本的な内訳を知ることができます。

クライアント	データURI 1画素 <img>
モバイル	44.7%
デスクトップ	47.1%

図5.6. データ URI 1画素の画像。

データURIを含む1画素 `<img>` は、ほぼ間違いなくレイアウトに使用されています。リクエストを発生させる残りの約54%は、レイアウトのためかもしれませんし、トラッキングピクセルかもしれませんが、私たちにはわかりません。

なお、この後の解析では、1画素 `<img>` は解析結果から除外しています。このメディアの章では、トラッキングピクセルやレイアウトハックではなく、ユーザーに視覚情報を提供する `<img>` 要素に関心をもっています。

多画素の画像

`<img>` に複数のピクセルが含まれる場合、そのピクセルは何ピクセルになりますか？

154. https://en.wikipedia.org/wiki/Spacer_GIF
155. https://ja.wikipedia.org/wiki/%E3%82%A6%E3%82%A7%E3%83%96%E3%83%93%E3%83%BC%E3%82%B3%E3%83%B3
156. https://developer.mozilla.org/docs/Web/API/Beacon_API
157. https://developer.mozilla.org/docs/Web/HTTP/Basics_of_HTTP/Data_URIs

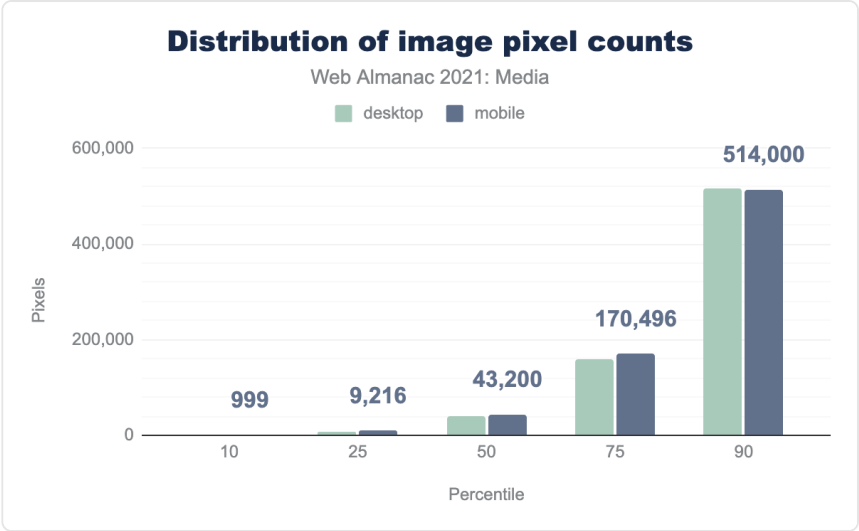


図5.7. 画像の画素数の分布。

中央の `<img>` は、モバイルでは40,000ピクセル強を読み込みます。この数字は意外と小さいと感じました。クロールされた `<img>` の半数弱（1ピクセルの画像や何も読み込まないものを除く）は、200x200の画像とほぼ同じ数のピクセルが含まれています。

しかし、1ページあたりの `<img>` 要素の数を考えると、この統計はそれほど驚くことでありません。ほとんどのページが15枚以上の画像を含んでいるため、多くの小さな画像やアイコンで構成されていることが多いのです。このように、ハーフメガピクセル以上の画像は、`<img>` 要素の10個に1個しかないとはいえ、ページ間を移動する際には決して珍しいことではありません。多くのページには、少なくとも1枚の大きな画像が含まれます。

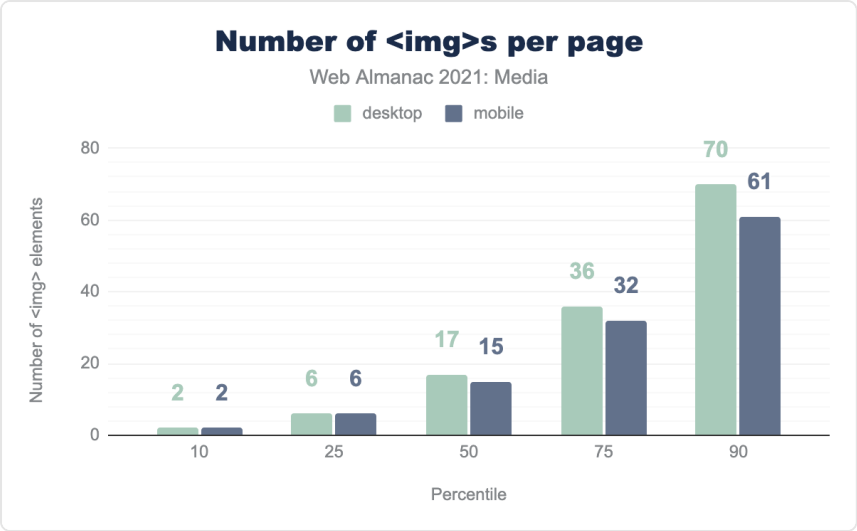


図5.8. 1ページに表示される `<img>` の数。

また、画素数分布の上端では、デスクトップとモバイルの差がほとんどないことにも驚きました。当初、これはレスポンシブ画像機能が効果的に採用されていないことを示しているように思われましたが、モバイルクローラーのビューポートが360×512px @3x（つまり1,080×1,536物理ピクセル）であるのに対し、デスクトップのビューポートは1,376×768px @1xということを考えると、実は驚くべきことではありません。クローラーのビューポートが物理ピクセルで同幅である（1,080 vs 1,376）のです。クローラー間の物理的なピクセル解像度の差が大きければ、もっと明らかになるはずです。

アスペクト比

Web上の画像は横向きのものが多く、縦向きのものは比較的少ないです。

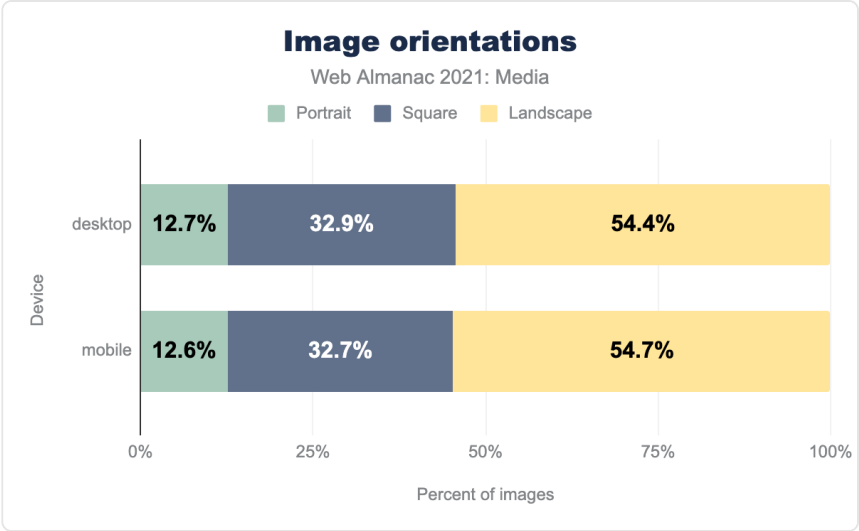


図5.9. 画像の向き

これは、モバイルでの機会を逸した感があります。「ストーリー」UIパターン[158]の成功は、縦長のモバイル画面を埋めるため調整された画像に価値があることを示しています。

画像のアスペクト比は、4：3、16：9、とくに1：1（正方形）など、「標準的」な値に集中していました。アスペクト比のトップ10は、全`<img>`の約半分を占めています。

158. *https://uxdesign.cc/the-powerful-interaction-design-of-instagram-stories-47cdeb30e5b6*

アスペクト比	デスクトップ画像	モバイル画像
1:1	32.9%	32.7%
4:3	3.7%	4.1%
3:2	2.5%	2.6%
2:1	1.6%	1.7%
16:9	1.5%	1.5%
3:4	0.9%	1.0%
2:3	0.7%	0.7%
5:3	0.6%	0.5%
6:5	0.5%	0.5%
8:5	0.5%	0.5%

図5.10. 画像のアスペクト比のトップ10をランキング形式でご紹介します。

バイト数

ここで、ファイルサイズに目を向けてみましょう。

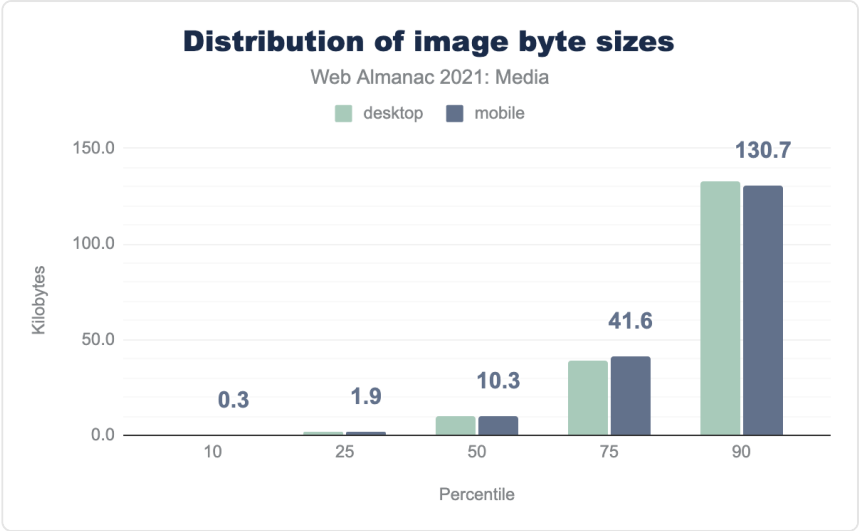


図5.11. 画像のバイトサイズの分布。

コンテンツが多い `<img>` の中央値は、10kB強です。しかし、中央のページには15個以上の `<img>` が含まれているため、ページ全体の全画像の90パーセンタイルを見ると、100kBを超える画像はまったく珍しくありません。

画素あたりのビット数

バイトと寸法はそれ自体で興味深いものですが、ウェブの画像データがどの程度圧縮されているかを知るには、バイトとピクセルを組み合わせて、*1ピクセルあたりのビット数*を計算する必要があります。これにより、解像度が異なる画像でも、その情報量を比較することができるようになりました。

一般にWeb上のビットマップは、1チャンネル、1ピクセルあたり8ビットの情報にデコードされます。つまり、透明度のないRGB画像であれば、デコードされた非圧縮画像は24ビット/ピクセル[159])になると予想されるのです。可逆圧縮の目安は、ファイルサイズを2：1の割合で小さくすることです（8ビットRGB画像では1ピクセルあたり12ビットに相当）。1990年代の非可逆圧縮方式（JPEGやMP3）は、10：1（2.4ビット/ピクセル）が目安でした。画像のコンテンツやエンコーディングの設定によって、これらの比率は広く変化し、MozJPEG[160]のような最新のJPEGエンコーダーは、デフォルト設定でこの10:1目標を上回ることが多いことに留意する必要があります。

159. *https://ja.wikipedia.org/wiki/%E8%89%B2%E6%B7%B1%E5%BA%A6#%E3%83%88%E3%82%A5%E3%83%AB%E3%83%BC%E3%82%AB%E3%83%A9%E3%83%BC%EF%BC%8824/32%E3%83%93%E3%83%83%E3%83%88*
160. *https://github.com/mozilla/mozjpeg*

このような背景のもと、Web上の画像はどのような位置づけにあるのか、ご紹介します。

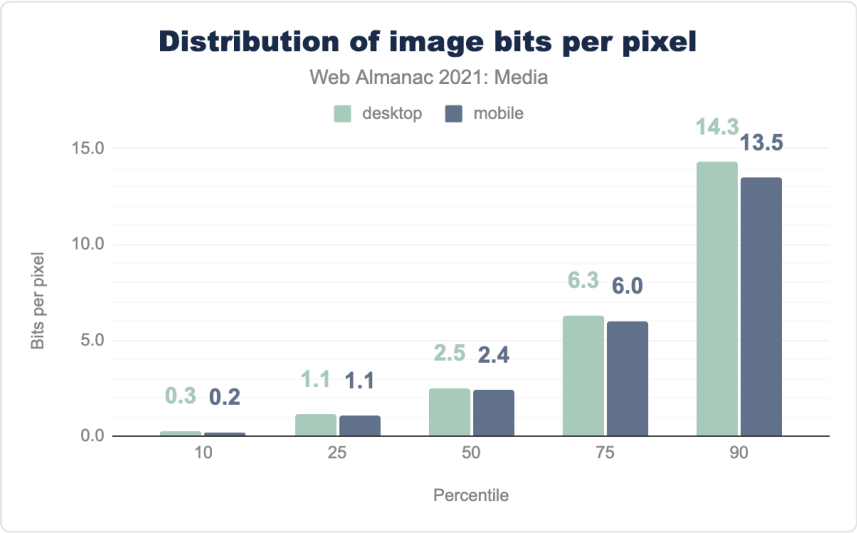


図5.12. 1画素あたりの画像ビット数の分布。

モバイルの中央の `<img>` は、10:1の圧縮率の目標にぴったりで、2.4ビット/画素です。しかし、その中央値付近では、とてつもなく大きな開きがあるのです。もう少し詳しく知るために、フォーマット別に分けて考えてみましょう。

フォーマット別、画素あたりのビット数

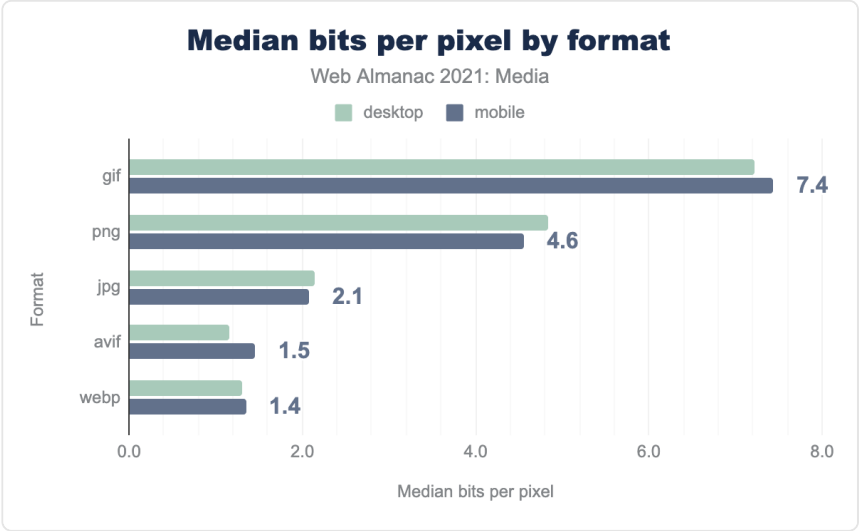


図5.13. フォーマット別の1画素あたりのビット数の中央値。

JPEGの中央値は、1ピクセルあたり2.1ビットです。このフォーマットはどこにでもあるため、他のフォーマットを測定するための最適な基準値となります。

PNGの中央値は、その2倍以上です。PNGはロスレスフォーマットと呼ばれることがありますが、1ピクセルあたり4.6ビットの中央値は、これがいかに誤ったものであるかを示しています。真のロスレス圧縮は、通常、1ピクセルあたり約12〜16ビット（アルファチャンネルを扱うかどうかによって異なる）に収まるはずです。PNGがこれほどまでに低いのは、一般的なPNGツールは通常、*非可逆圧縮*だからです。圧縮率を高めるために、ピクセルをエンコードする前にカラーパレットを減らしたり、ディザリングパターンを導入したりと、ピクセルに手を加えてしまうのです。

1ピクセルあたり7.4ビットのGIFは、ここではひどく見劣りします。これら[161]は[162]大変[163]！しかし、ウェブ上の多くのGIFはアニメーションであるため、ここでも少し不利な立場に立たされています。WebプラットフォームのAPIは、アニメーション画像のフレーム数を公開しないため、フレーム数を考慮していない。たとえば、1ピクセルあたり20ビットで計測されるGIFは、10フレームを含む場合、1ピクセルあたり2ビットと計算するのが妥当でしょう。

次世代フォーマットである2つのフォーマットについて見てみると、実に興味深いことがわかります。WebPとAVIFです。どちらも1ピクセルあたり1.3〜1.5ビットで、JPEGより40%

161. https://web.dev/efficient-animated-content/
162. https://bitsofco.de/optimising-gifs/
163. https://dougsillars.com/2019/01/15/state-of-the-web-animated-gifs/

近く軽くなっています。matched qualities[164] を使用した正式な研究では、WebPはJPEGよりも 25 ～ 34%[165] 高性能であり、実際のパフォーマンスは驚くほど *良い* と思われます。一方、AVIFの作成者は、実験室で、最新のJPEGエンコーダーJPEGを50% 以上上回ることができる[166]というデータを発表しています。ですから、ここでのAVIFの性能は良いのですが、私はもっと良い結果を期待していました。実験室のデータと実際の性能の間にあるこのような不一致について、私はいくつかの可能性を考えることができます。

第一に、ツールです。JPEGエンコーダーは、画像をうまく圧縮するのにあまり労力をかけないカメラに搭載されているハードウェアエンコーダーから、何十年も前にインストールされた `libjpeg` の古いコピー、MozJPEGのような最先端の、デフォルトで最高の機能を持つエンコーダーまで非常に幅広く存在しているのです。要するに、古くてひどく圧縮されたJPEGはたくさんありますが、すべてのWebPとAVIFは最新のツールで圧縮されているのです。

また、参考となるWebPエンコーダー (`cwebp`) は、品質と圧縮について比較的積極的で、ほとんどの一般的なJPEGツールよりも低品質でより圧縮された結果をデフォルトで返すという逸話があります。

AVIFに関して言えば、 `libavif` は、どの「速度」設定を選ぶかによって、さまざまな圧縮率を実現することが可能です。もっとも遅い速度（もっとも効率の良いファイルを生成）では、 `libavif` は1つの画像をエンコードするのに数分かかることもあります。画像レンダリングパイプラインが異なれば、その制約によって速度設定のトレードオフも異なると考えるのが妥当でしょう。その結果、圧縮性能に大きなばらつきが生じます。

もうひとつ、AVIFの実力を評価する際に気をつけたいのは、まだウェブ上にそれほど多くのAVIFが存在しないということです。このフォーマットは現在、比較的少数のサイト、限られたコンテンツで使用されているため、最終的に「どのよう」に機能するかについては、まだ完全には把握できていません。今後数年間、採用が進む（そしてツールも改善される）につれ、この点を追跡するのは興味深いことです。

絶対に明らかなのは、WebPとAVIFの両方を使用すれば、Webのレガシー形式よりも効率的にさまざまなコンテンツ（写真、 イラスト[167]、透明度の高い画像など）を配信できることです。しかし、次節で紹介するように、それほど多くのサイトが採用しているわけではありません。

164. *https://kornel.ski/en/faircomparison*
165. *https://developers.google.com/speed/webp/docs/webp_study*
166. *https://netflixtechblog.com/avif-for-next-generation-image-coding-b1d75675fe4*
167. *https://jakearchibald.com/2020/avif-has-landed/#flat-illustration*

フォーマットの採用

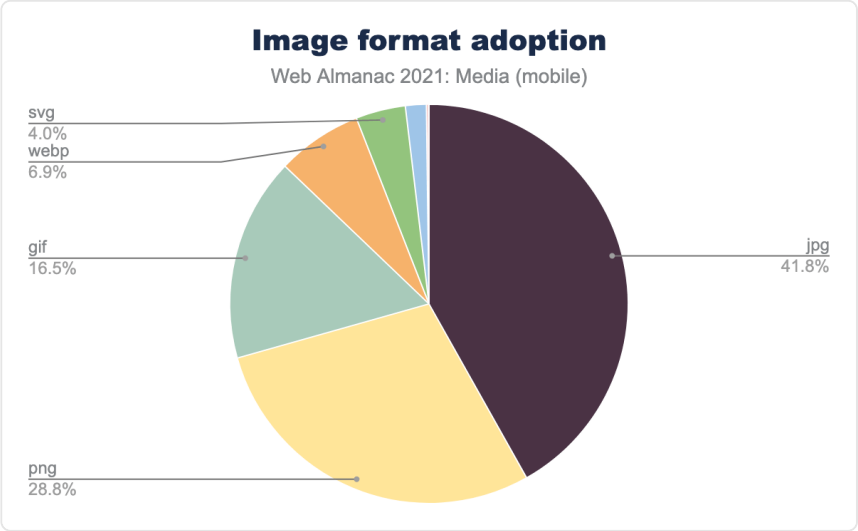


図5.14. 画像フォーマットの採用（モバイル）。

古いフォーマットが依然として君臨している。JPEGが優勢で、PNGとGIFが表彰台を独占しています。これらを合わせると、ウェブ上の画像のほぼ90%を占めることになります。WebPはもう10年以上前のものですが、昨年ユニバーサル ブラウザ サポートを実現したばかり[168]で、まだ1桁の数字にとどまっています。そして、事実上誰もAVIFを使っておらず、クロールされたリソースのわずか0.04%を占めただけでした。AVIF1枚につき、JPEG1,000枚を発見しました。

WebPとAVIFの採用が時間とともにどのように変化したか（およびその理由の推測）についての詳細な分析については、ImageReadyにおける Paul Calvano[169] の最近の素晴らしい講演（フル動画[170] および スライド13〜15[171]）が最高のリソースとなるでしょう。その中で、Safariがサポートを追加した2020年7月から2021年7月にかけて、WebPの採用率が〜34%増加したことを紹介しています。AVIFはまだ新しいフォーマットであり、比較的少数のサイトが使用していることを考えると、驚くには値しないがパーセンテージで見ると、AVIFの数字はさらに急上昇している。AVIFを採用したのは、数人の大[172]プレーヤー[173]だけだったのです。

168. *https://www.macrumors.com/2020/06/22/webp-safari-14/*
169. *https://x.com/paulcalvano*
170. *https://www.youtube.com/watch?v=tz5bpAQY43k*
171. *https://docs.google.com/presentation/d/1VS5QjNR6lh2y9jL5xaeainQ2cTAWyy7QiEjDMh4hNQA/edit#slide=id.gefc0d6ffce_0_0*
172. *https://x.com/chriscoyier/status/1465474900588646408*
173. *https://medium.com/vimeo-engineering-blog/upgrading-images-on-vimeo-620f79da8605*

埋め込み

ウェブページに画像を表示するには、`<img>`要素を用いて画像を埋め込む必要があります。この由緒ある要素は、過去数年の間にいくつかの新機能を獲得しましたが、それらの機能はどのように実践されているのでしょうか？

遅延読み込み

ウェブ上の画像に関して今年ブレイクアウトした話があるとすれば、ネイティブ遅延読み込み[174]の採用でしょう。このチャートを見てください。

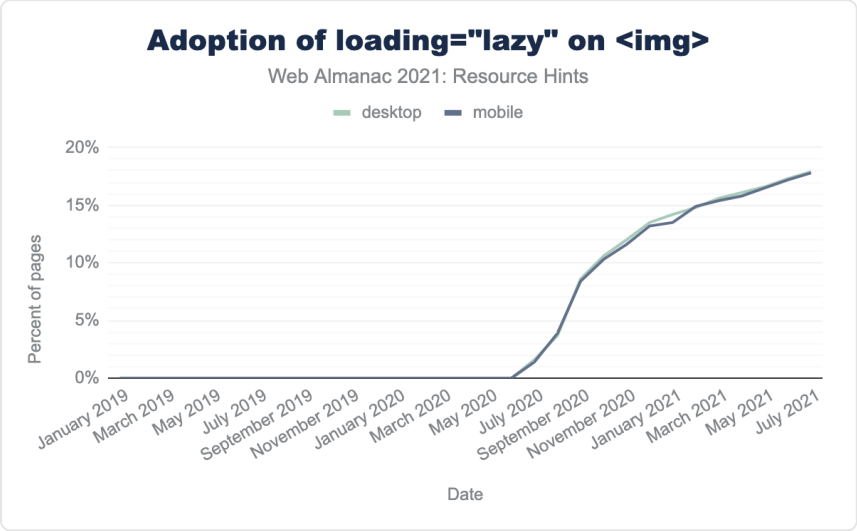


図5.15. `<img>` に `loading="lazy"` を採用する。

2020年7月、ネイティブ遅延読み込みはわずか1%のページで使用されていました。2021年7月には、その数は18%にまで爆発的に増えています。これは、毎年更新されない膨大な数のページやテンプレートを考えると、信じられないような伸び率です。

個人的には、ネイティブ遅延読み込みの急速な普及は、今年、ページあたりの画像バイトが減少したことを説明する最良の方法だと思います。

何が遅延読み込みの普及を促したのか？使いやすさ、開発者の需要の高まり、そしてWordPress 遅延読み込みをWebの広大な領域に関しデフォルトで有効にする[175]という組み合わせであることは、ある程度コンセンサスになっています。

174. *https://web.dev/browser-level-image-lazy-loading/*
175. *https://make.wordpress.org/core/2020/07/14/lazy-loading-images-in-5-5/*

もしかしたら、ネイティブの遅延読み込みが成功しすぎたのでしょうか？Resource Hintsの章では、遅延ロードされた画像の大半は初期ビューポートにあったと記しています（一方、この機能は「below the fold」の画像に使うのが理想的です）。 さらに、パフォーマンスの章では、Largest Contentful Paint 要素の 9.3% が `loading` 属性を `lazy` に設定している ことを発見しました。これはページのもっとも重要なコンテンツの読み込みを大幅に遅らせ、ユーザーの体験を損ねることになります。

デコード

`<img>` の `decoding` 属性は、ネイティブ遅延読み込みの成功を強調するための有効な対照点として機能します。2018年にはじめてサポートされた[176]ネイティブ遅延読み込みの約1年前 `decoding` 属性により、開発者は大きな画像デコード操作がメイン スレッドをブロックするのを防止することができるようになりました。すべてのウェブ開発者が必要とし、理解しているわけではない機能を提供し、それが使用データにも表れています。 `decoding` はページのわずか1%、 `<img>` 要素のわずか0.3%にしか使われていません。

アクセシビリティ

ウェブページにコンテンツのある画像を埋め込む場合、そのコンテンツは視覚的でないユーザーにも可能な限りアクセスできるようにする必要があります。この章では、ウェブ上の画像アクセシビリティを詳細に分析し、前年比でわずかな進歩が見られたものの、大部分は改善の余地があります。

レスポンシブ画像

2013年、レスポンシブWebサイトにおける画像のアダプティブ・ローディングを可能にする一連の機能が、あまりに大きな反響を呼んで上陸しました。それから8年、レスポンシブ画像機能はどのように活用されているのでしょうか？

まず、 `srcset` 属性 について考えてみましょう。この属性によって、開発者は同じ `<img>` に対して複数の可能なリソースを提供できます。

176. *https://www.chromestatus.com/feature/4897260684967936*

`x` と `w` の記述子の採用

30.9%

図5.16. モバイルページで `srcset` を使用している割合。

クロールされたページのほぼ3分の1が `srcset` を使っている、これはとても良いことだ！

そして `w` 記述子、これは、ブラウザが さまざまなレイアウト幅とさまざまな画面密度の両方に基づいて[177] リソースを選択することを可能にし、DPR適応のみを可能にする[178] `x` 記述子よりも4倍も人気があります。

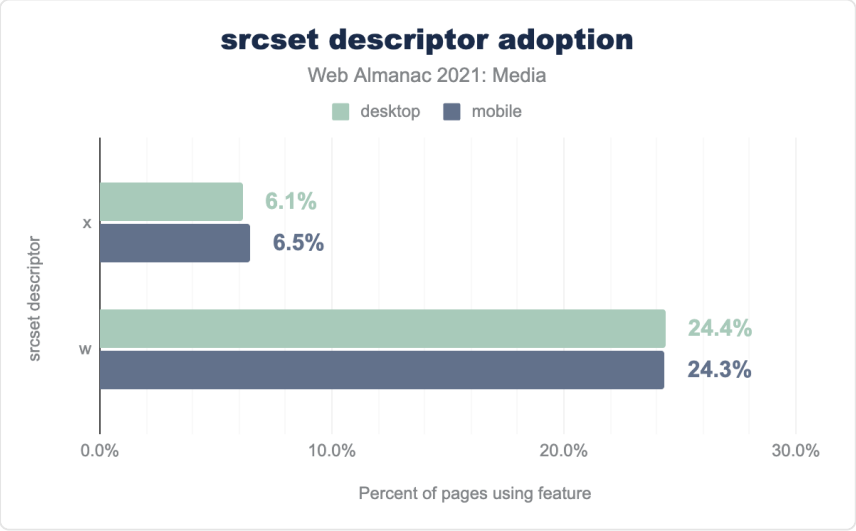


図5.17. `srcset` 記述子の採用。

開発者はどのように `srcset` にリソースを投入しているのですか？

`srcset` 候補の数

まず、開発者が入れている候補リソースの数を見てみましょう。

177. https://jakearchibald.com/2015/anatomy-of-responsive-images/#varying-size-and-density
178. https://jakearchibald.com/2015/anatomy-of-responsive-images/#fixed-size-varying-density

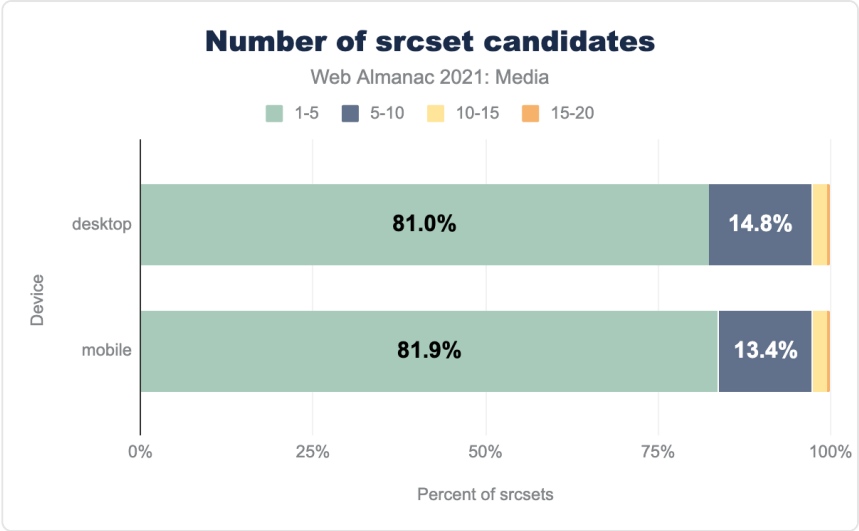


図5.18. srcset候補の数。

大半の `srcset` は5個かそれ以下のリソースで構成されています。

`srcset` の密度の範囲

開発者はブラウザに、彼らの `srcset` の中で、適切な幅広い選択肢を与えていますか？この質問に答えるためには、まず `srcset` と `sizes` の値がブラウザでどのように使用されるかを理解する必要があります。

ブラウザが `srcset` から読み込むリソースを選ぶとき、まず候補となるすべてのリソースに密度[179]を割り当てます。`x` 記述子を使用するリソースの密度を計算するのは簡単です。`2x` の密度記述子を持つリソースは、（待てよ）2xの密度を持つ。

`w` 記述子は物事を複雑にします。`1000w` のリソースの密度は？解決された `sizes` 値に依存します(ビューポートの幅に依存するかもしれません！)。`w` 記述子を使用する場合、各記述子を `sizes` 値で割って、その密度を決定します。たとえば

```
<img
  srcset="large.jpg 1000w, medium.jpg 750w, small.jpg 500w"
  sizes="100vw"
```

179. *https://html.spec.whatwg.org/multipage/images.html#current-pixel-density*

```
/>
```

500-CSS-`px` 幅のビューポートでは、これらのリソースは以下の密度で割り当てられます。

リソース	密度
large.jpg	1000w ÷ 500px = 2x
medium.jpg	750w ÷ 500px = 1.5x
small.jpg	500w ÷ 500px = 1x

しかし、1000-CSS-`px` 幅のビューポートでは、同じリソースが同じ `srcset` と `sizes` 値でマークアップされ、異なる密度を持つことになります。

リソース	密度
large.jpg	1000w ÷ 1000px = 1x
medium.jpg	750w ÷ 1000px = 0.75x
small.jpg	500w ÷ 1000px = 0.5x

これらの密度が計算された後、ブラウザは現在の閲覧状況にもっともマッチする密度のリソースを選択します。この例では、`srcset` に十分な広さのリソースが含まれていなかったと言ってよいでしょう。CSSのビューポートは1,000pxを超えるものもあり、1xを超える密度も珍しくはない。ノートパソコンでこれを読んでいる人は、今まさに、そんな状況で閲覧していることでしょう。そして、このような状況でブラウザができる最善のことは、`large.jpg` を選ぶことです。その1倍の密度は、高密度のディスプレイではまだぼやけて見えるでしょう。

だから、両方で武装する。

1. ブラウザが `x` と `w` の記述子、 `sizes` 値、およびブラウジングコンテキストをどのようにリソース密度に変換するかを理解していること。
2. ブラウジングの状況に応じて `srcset` 内のリソース密度の範囲がどのように変化し、最終的にユーザーに影響を与えるかを理解すること。

。。。ここで、 `x` 記述子、または `w` 記述子を使用した `srcset` によって提供される密度の範囲を見てみましょう。

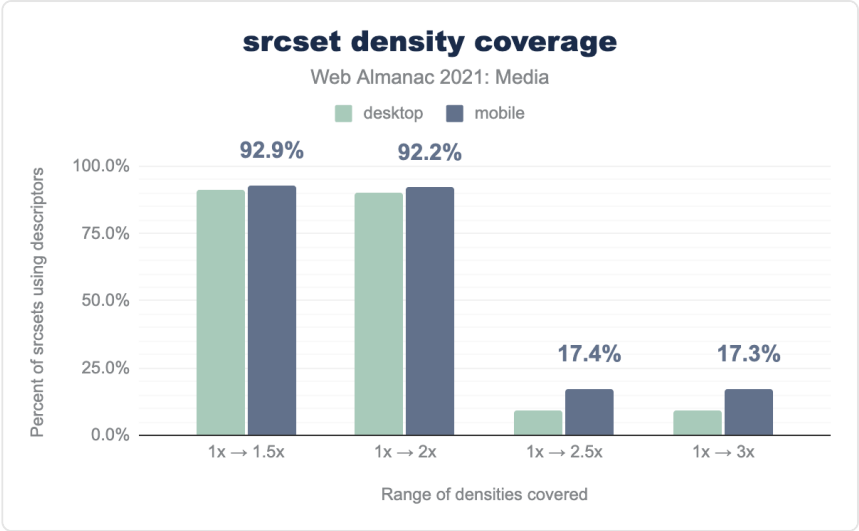


図5.19. `x` または `w` 記述子を使用する `srcset` がカバーする密度の範囲。

このデータを解釈する際には、2つの異なるクローラーのビューポートを念頭に置いてください。

- デスクトップ: 1,376 × 768px @1x
- モバイル: 360 × 512px @3x

ビューポートの幅が異なれば、解決された多くの `sizes` 値が変化し、異なる結果が得られたでしょう。

とはいえ、この結果は良さそうですね。10個の `srcset` のうち9個は、より大きなデスクトップのビューポートでも、妥当な範囲の出力ディスプレイ密度（1x-2x）をカバーするリソースの範囲を提供しているのです。指数関数的な帯域幅コストと、2xを超える密度での視覚的リターンの減少[180]を考えると、2 x以降の急降下は妥当なだけでなく、おそらく最適とさえ思われます。

サイズの精度

レスポンシブ画像は厄介なものです。適度に正確な `sizes` 属性を作成し、進化するページレイアウトやコンテンツに合わせて最新の属性を維持することは、レスポンシブ画像を正しく表示する上でもっとも困難なことかもしれません。どれだけの作成者が `sizes` を間違え

180. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2019/capping-image-fidelity-on-ultra-high-resolution-devices

ているか？そして、どの程度間違っているのでしょうか？

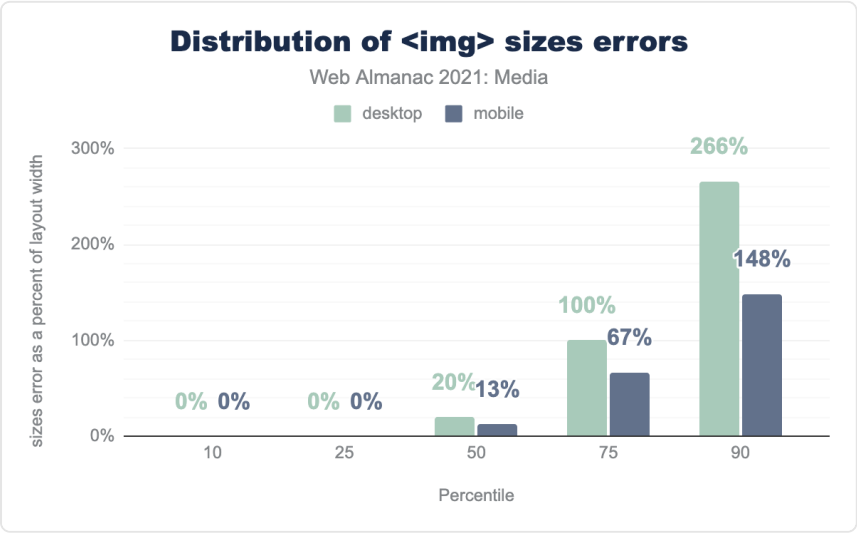


図5.20. `<img>` サイズエラーの分布。

4分の1以上の `sizes` 属性が完璧で、画像のレイアウトサイズと完全に一致します。私自身、誤った `sizes` の属性をいくつも書いてきた人間として、これは驚きであり、印象的でした。つまり、ここでの精度測定はJavaScriptの実行後に行われ、多くの `sizes` 属性は最終的にクライアントサイドのJavaScriptによって書かれていることに気づくまででした。以下は、JavaScriptを実行する前の、もっとも一般的な `sizes` の値です。

サイズ	デスクトップ	モバイル
`auto`	8.2%	9.6%
`(max-width: 300px) 100vw, 300px`	4.7%	5.9%
`(max-width: 150px) 100vw, 150px`	1.3%	1.6%
`(max-width: 600px) 100vw, 600px`	1.0%	1.2%
`(max-width: 400px) 100vw, 400px`	1.0%	1.1%
`(max-width: 800px) 100vw, 800px`	0.8%	0.9%
`(max-width: 500px) 100vw, 500px`	0.8%	0.9%
`(max-width: 1024px) 100vw, 1024px`	0.7%	0.9%
`(max-width: 320px) 100vw, 320px`	0.5%	0.8%
`(max-width: 100px) 100vw, 100px`	0.7%	0.8%
`100vw`	0.7%	0.7%

図5.21. もっとも一般的なサイズの属性値のランキングリスト（JavaScript実行前）。

モバイルの `sizes` 属性の10個に1個は、初期値が `auto` になっています。この標準外の値は、おそらくJavaScriptのライブラリ（おそらくlazysizes.js[181]）によって、画像の測定されたレイアウトサイズを使って置き換えられると思われます。

レイアウトが完了する前に、ブラウザが読み込むべき適切なリソースを選択するためのヒントを提供するため、 `sizes` の多少の誤差は許容範囲内とします。しかし、大きな誤差は、リソースの選択を誤らせることになります。これは、もっとも精度の低い4分の1の `sizes` 属性について言えることで、デスクトップでは実際の `<img>` レイアウト幅の2倍、モバイルでは1.5倍の幅が報告されます。

つまり、10個の `sizes` 属性のうち1個はJavaScriptライブラリによってクライアントで作成されており、少なくとも4個のうち1個は、そのエラーがリソースの選択に影響を与えるほど不正確なのです。これらの事実は、既存のツール[182]または 新しいWebプラットフォーム機能[183]が、より多くの著者が `sizes` を正しく理解できるようにするための大きなチャンスであることを示しています。

181. https://github.com/aFarkas/lazysizes
182. https://github.com/ausi/respimagelint
183. https://github.com/whatwg/html/issues/4654

`<picture>` の使い方

`<picture>` 要素は、いくつかのユースケースに対応しています。

1. アートディレクション、`media` 属性付き
2. MIMEタイプに基づくフォーマット切り替え、`type` 属性 による

5.9%

図5.22. モバイルページで `<picture>` が使用されている割合。

`<picture>` は `srcset` よりもはるかに使用頻度が低いです。この2つのユースケースで、どのように使い分けがされているのかを紹介します。

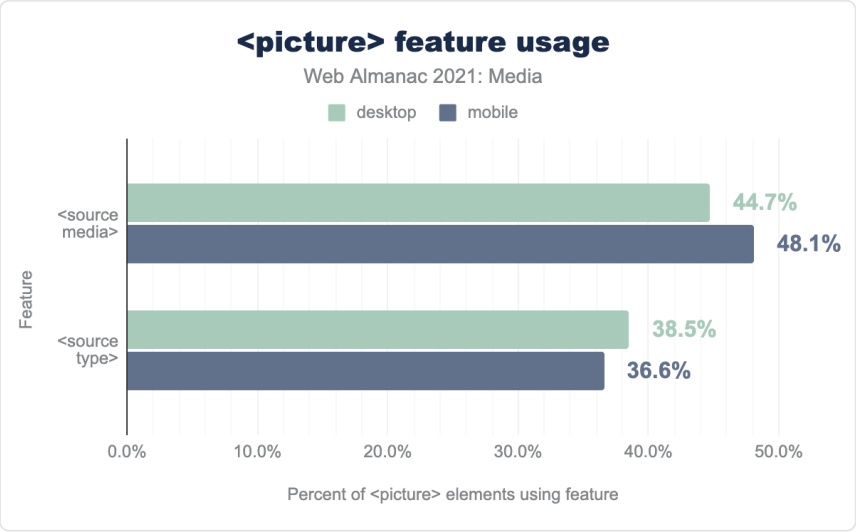


図5.23. `<picture>` 機能の使用法。

アートディレクションは、フォーマット切り替えに比べるとやや多いようですが、どちらも潜在的な有用性を考えると、あまり活用されていないように思います。これまで見てきたように、モバイル画面に合わせて画像のアスペクト比を調整しているページはほとんどなく、多くのページが次世代フォーマットを使ってより効率的に画像を配信できるはずです。これらはまさに `<picture>` が解決するために考案された問題であり、おそらく5.9%以上のページがこの機能を使って、これらの問題に対処することができたのでしょう。

フォーマット切り替えは サーバサイドコンテンツネゴシエーション[184] でも実現できるので、`<source type>` によるフォーマット切り替えは2〜3%のページでしか使われていない可能性があります。残念ながら、サーバーサイドの適応メカニズムは、クローリングされたデータから検出することが困難であり、ここでは解析していない。

なお、`<source type>` と `<source media>` は相互に排他的ではないので、ここでの使用比率を合わせても100%にはならない。このことから、少なくとも15%の `<picture>` 要素はこれらの属性のどちらも利用しておらず、それらの `<picture>` は機能的に `<span>` と同等であることがわかります。

レイアウト

ページに画像を埋め込んだら、それを他のコンテンツと一緒にレイアウトする必要があります。この方法にはさまざまなものがありますが、画像を拡大表示し、2つの大きな質問に答えることで、一般的な方法についていくつかの洞察を得ることができます。

内在サイジングと外在サイジング

置換要素[185]である画像は、自然で"内在"サイズ[186]を持っています。このサイズは、CSSルールによる「外在」レイアウト制約がない場合に、デフォルトでレンダリングされるサイズです。

内在性と外在性のサイズの画像はいくつありますか？

184. *https://developer.mozilla.org/docs/Web/HTTP/Content_negotiation*
185. *https://developer.mozilla.org/docs/Web/CSS/Replaced_element*
186. *https://developer.mozilla.org/docs/Glossary/Intrinsic_Size*

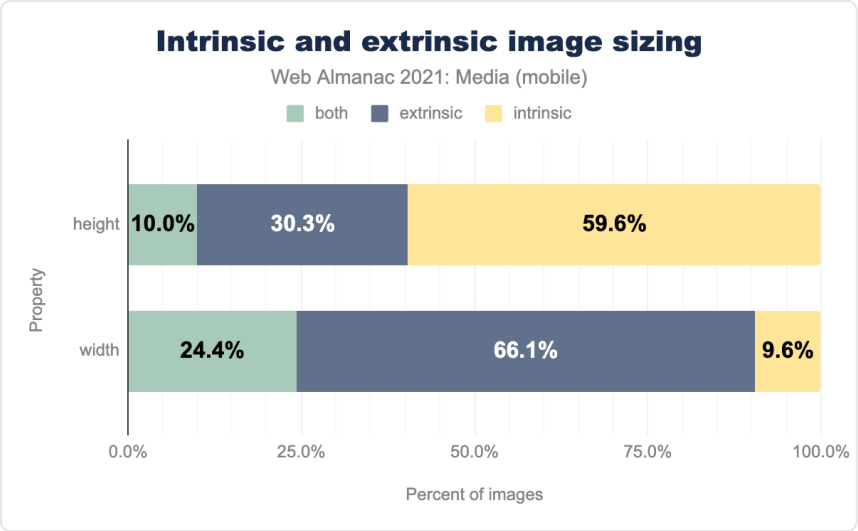


図5.24. 内在的および外在的な画像サイズ調整。

この質問は少し複雑で、画像によっては（`max-width`、`max-height`、`min-width`、`min-height`の制約があるもの）、外在サイズもあれば、内在サイズのままであることもあります。このような画像は“両方”とラベル付けしています。

いずれにせよ、当然のことながら、ほとんどの画像には幅の制約があり、高さに制約のあるサイジングはあまり一般的でありません。

`height`と`width`によるレイアウトのずれの軽減

そこで、最後に調査したいのが、ウェブプラットフォームの機能です。`height`と`width`属性を使用して、柔軟な画像のためのレイアウトスペースを確保します。

デフォルトでは、画像が読み込まれ、その固有寸法が判明するまで、内在寸法のままではスペースを取りません。その時点で、パッとページに現れて、レイアウト シフト[187]が発生するのです。これは、まさに`height`と`width`属性が解決するために発明された問題でした。1996年[188]。

残念ながら`height`と`width`は、ある次元では可変の外在サイズ（たとえば、`width: 100%;`）が割り当てられ、他の次元では内在の縦横比を満たすように放置される画像とはうまく動作しません。これはレスポンシブデザインにおける支配的なパターンです。そのた

187. https://developers.google.com/publisher-tag/guides/minimize-layout-shift
188. https://www.w3.org/TR/2018/SPSD-html32-20180315/#img

め、`width` と `height` はレスポンシブコンテキスト内で人気がなくなりましたが、2019年に ブラウザが `height` と `width` を使用する方法を調整 によってこの問題が修正されました。さて、一貫して `height` と `width` を設定することは、Cumulative Layout Shift[189] を減らすために作成者ができる最善のことの1つです。これらの属性はどれくらいの頻度でこのタスクを達成しているのでしょうか？



図5.25. モバイルの `<img>` のうち、 `height` と `width` の両方の属性を持ち、1次元のみの外形寸法を持つものの割合です。

これらの `<img>` のうち、いくつが新しいブラウザの動作を考慮して作成されたかは分かりませんが、すべてこの恩恵を受けています。そして、既存の属性を再利用することで、多くの既存コンテンツが自動的にその恩恵を受けることができる、というのがポイントでした。

デリバリー

最後に、ネットワーク上で画像がどのように配信されるかを見てみましょう。

クロスオリジン画像ホスト

埋め込まれている画像と同じオリジンでホストされている画像はどれくらいあるのでしょうか？ごくわずかな少数派

189. https://web.dev/i18n/ja/cls/

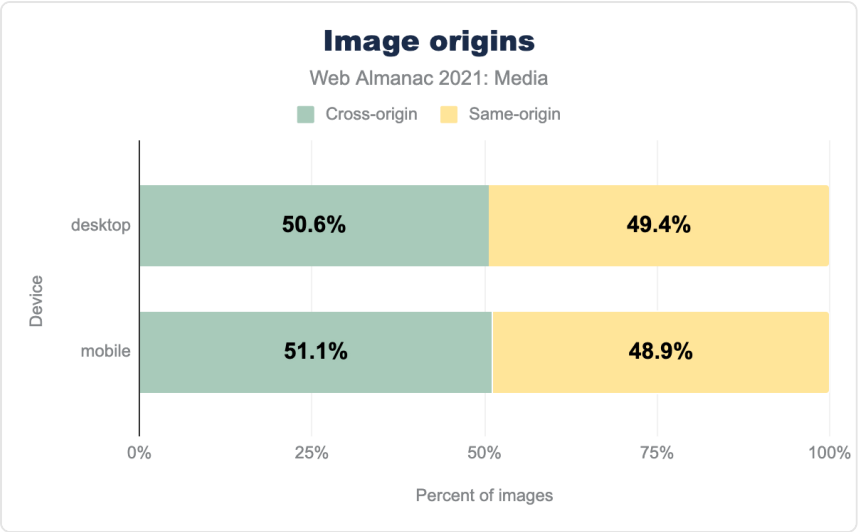


図5.26. 画像の出典。

クロスオリジン画像は、重要なセキュリティ制限[190]の対象となり、時にはパフォーマンスコスト[191]を発生することがあります。一方、静的資産を専用のCDNに移動することは、最初の1バイトまでの時間[192]を助けるためにできるもっともインパクトのあることの1つです。画像CDNは、あらゆる種類のベストプラクティスを自動化できる強力な変換および最適化[193]機能を提供します。51%のクロスオリジン画像のうち、何枚が画像CDNでホストされているか、またそのパフォーマンスを他のウェブのものと比較することは、とても興味深いことでしょう。残念ながら、これは私たちの分析の範囲外でした。

と、いうことで、そろそろ目を向けてもいいのでは。。。

動画

昨年から世の中が劇的に変化する中で、Web上での動画の利用が大きく伸びています。2020年のメディアレポートでは、ウェブサイトの1〜2%が`<video>`タグを持っていると推定されています。2021年には、その数が大幅に増え、デスクトップサイトの5%以上、モバイルサイトの4%が`<video>`タグを組み込んでいます。

190. *https://developer.mozilla.org/docs/Web/HTML/CORS_enabled_image*
191. *https://andydavies.me/blog/2019/03/22/improving-perceived-performance-with-a-link-rel-equals-preconnect-http-header/*
192. *https://developer.mozilla.org/docs/Glossary/time_to_first_byte*
193. *https://web.dev/i18n/ja/image-cdns/*

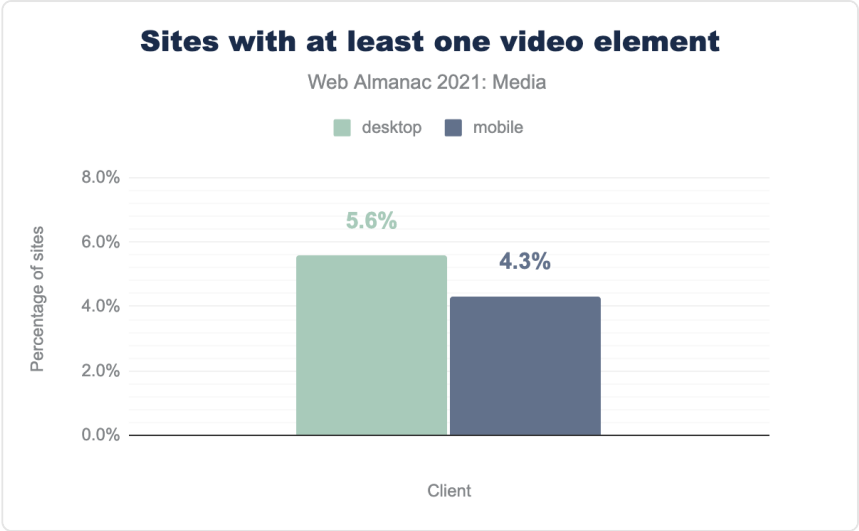


図5.27. 動画要素が1つ以上あるサイト。

このようにWeb上での動画利用が大きく伸びているのは、デバイス/ネットワークの向上に伴い、サイトに動画のような没入感を与えたいという要望があることを示しています。

動画との相互作用については、ウェブページに掲載した場合の動画の長さが興味深いです。440k本のデスクトップ動画と382k本のモバイル動画についてこの値を照会することができ、継続時間をさまざまなバケット（秒単位）に分解することができました。

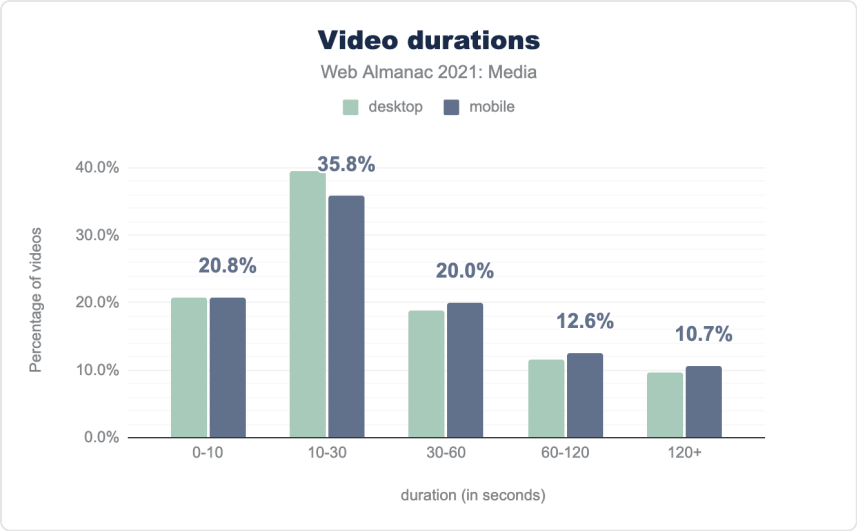


図5.28. 動画の持続時間。

モバイル、デスクトップともに、60%の動画が30秒以下です。しかし、1分以上2分未満が12〜13%、2分以上の動画が10%となっています。

動画：フォーマット

動画として配信されているのは、どのようなファイルですか？MIMEタイプに`video`を含むすべてのファイルを照会し、ファイル拡張子でソートしています。

下図は、シェア1%以上の動画拡張機能をすべて表示したものです。

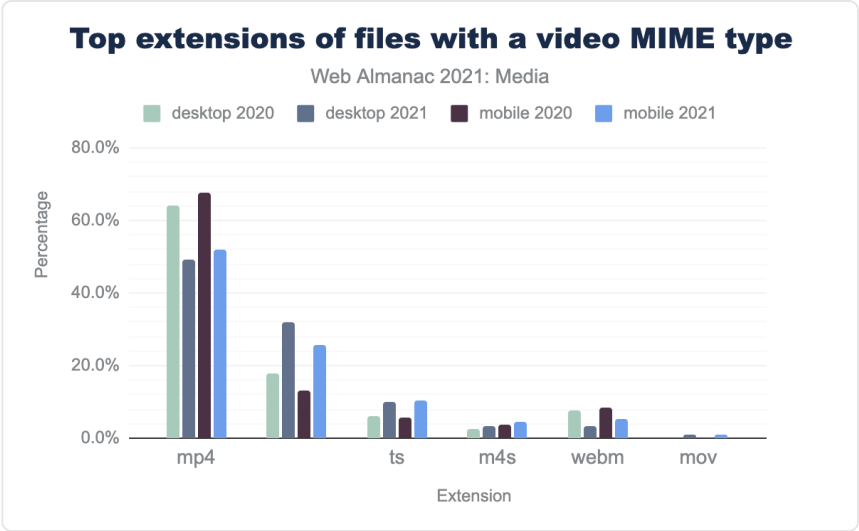


図5.29. videoのMIMEタイプを持つファイルの上位の拡張子。

Web上での動画フォーマットのナンバーワンは、圧倒的に `mp4`（またはMPEG-4）です。mp4 h264形式は、すべてのモダンブラウザで98.4% サポートされていて[194]、 `mp4` に対応していないブラウザの1.9% は動画をサポートしていないので、実質100%カバーというわけです。興味深いことに、 `mp4` の使用率はデスクトップとモバイルの両方で前年同期比 ~15% 減少しています。WebMのサポートも 前年同期比で大幅に減少[195]（モバイルとデスクトップの両方で50%減少）しています。

伸びているのは、拡張子のないファイル（YouTubeなどのストリーミングプラットフォームに多い）、そしてウェブストリーミングです。 `ts` ファイルは、HTTPライブストリーミング (HLS) で使用されるセグメントで、使用率が4%上昇しています。 `.m4s` はMPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) のビデオセグメントです。M4Sファイルは、前年同期の2.3%から3.3%へと50%増加しました。

動画CSS： `display`

まず始めに、動画がページ上にどのように表示されるかを、その動画のCSS `display` プロパティで見てみましょう。その結果、約半数の動画が `block` という表示値を使用しており、動画を独自の行に配置し、動画の高さと幅の値を設定できるようにしていることがわかりました。また、 `inline-block` の値では、高さと幅を指定することができ、すべての動画の合計で3分の2を指定できます。

194. *https://caniuse.com/mpeg4*
195. *https://almanac.httparchive.org/ja/2020/media#videos*

`display: none` 宣言は、視聴者から動画を隠します。Web上の動画の5本に1本は、この表示値の後ろに隠されているのです。データ使用量の観点からは、ブラウザで動画をダウンロードしたままなので、最適とは言えません。

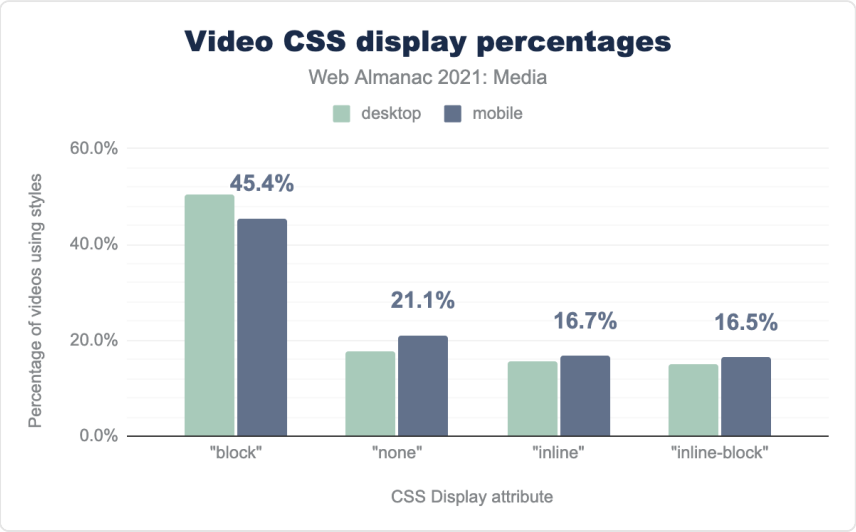


図5.30. 動画CSSの表示率。

動画属性

HTML5の `<video>` タグには、エンドユーザーに対するビデオプレーヤーの表示方法を定義するために使用できる属性が多数あります。

もっとも一般的な属性と、それらが `<video>` タグの中でどのように使われるかを見てみましょう。

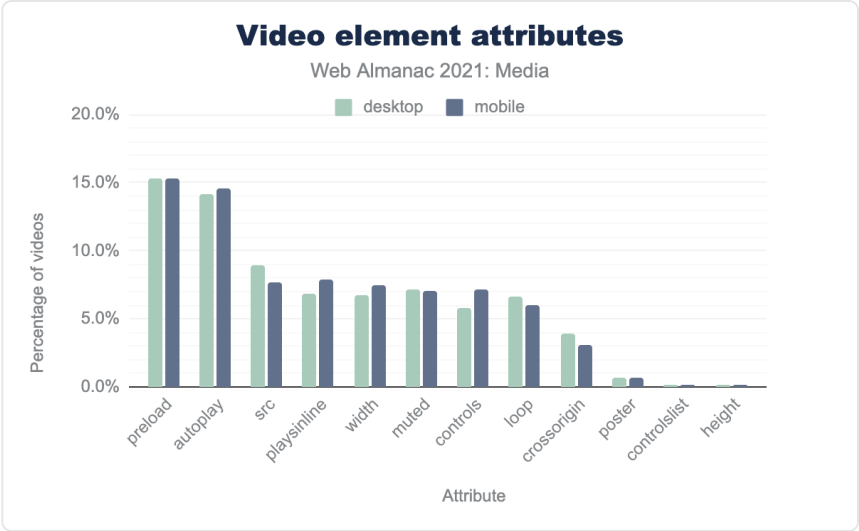


図5.31. 動画要素の属性。

`preload`

もっともよく使われる属性はpreloadです。preload属性は、動画のダウンロードを処理する最良の方法について、ブラウザにヒントを与えます。4つのオプションが可能です。`auto`, `metadata`, `none`, そして空のレスポンス（デフォルトの `auto` が使用される）です。

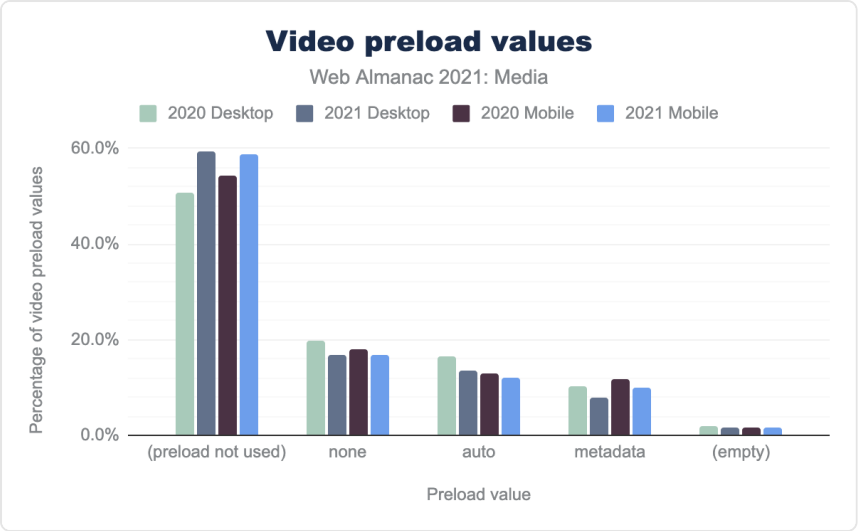


図5.32. 動画プリロードの値。

興味深いことに、モバイルとデスクトップの両方で `preload` が大幅に削減されていることがわかります。多くの動画で変更された可能性がありますが、昨年ウェブに追加された新しい動画がこの設定を利用していないだけかもしれません。ページ重量の観点からは、これはウェブにとって大きな勝利です。

`autoplay`

次によく使われる属性は `autoplay` です。これは、動画ができるだけ早く再生されるようにブラウザに指示します（このため、実際にはautoplayはpreload属性を上書きします）。

autoplay属性はブール値属性であり、デフォルトでその存在が真を意味することを意味します。したがって、`autoplay="false"` を使用している190のサイトには、申し訳ありませんが、それはうまくいきません。

`width`

また、`width` 属性は `<video>` の上位属性の1つです。これは、ビデオプレーヤーの幅をブラウザに指定します。なお、`height` が使用されることは非常に稀です。ブラウザはこれを設定できますが、デフォルトのアスペクト比は2:1[196] を使うので、`aspect-ratio` CSS stylingで明示的に上書きしないと不正確な場合があります。

196. *https://github.com/whatwg/html/issues/3090*

幅はパーセンテージ、またはピクセル単位で表示することができる。

- パーセント幅が定義されている場合、99%の確率で `100%` という値が使用されます。
- ピクセル単位で幅を定義すると、低い幅では非常に似たような数の動画が見られますが、1800と1920の幅では大きく落ち込んでいることがわかります。

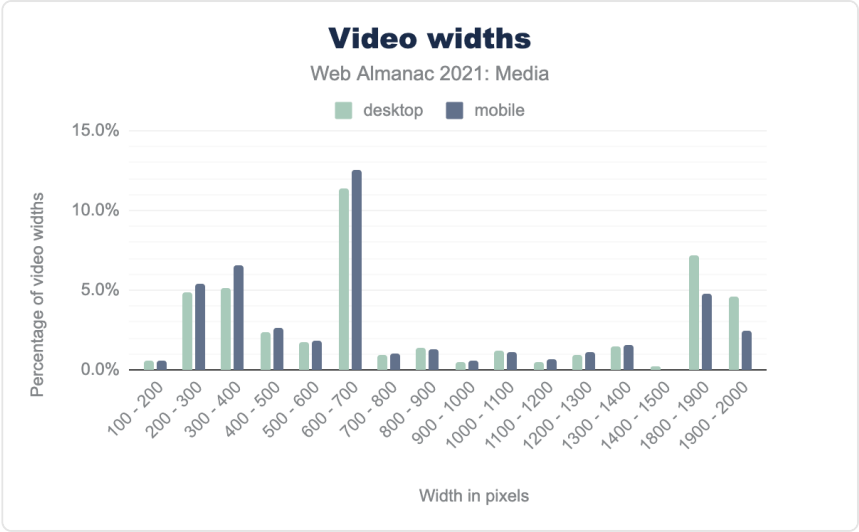


図5.33. 動画の幅

動画の幅も定義している大きな動画を持つサイトの約半数が、モバイルデバイス用に大きな動画を削除しているようです。ウェブサイトに埋め込まれた1080p（1920ワイド）のビデオを必要とするデバイスはほとんどないため、これは理にかなっていると言えるでしょう。

`src` と `<source>`

`src` 属性は、`<video>` タグ内で、再生する動画のURLを指定するために使用します。動画を参照する別の方法として、`<source>` 要素を使用することもできます。

`<source>` 要素の重要な考え方の1つは、開発者が複数の動画フォーマットをブラウザに供給し、ブラウザが理解できる最初のフォーマットを選択することです。

`<source>` の使用状況を見ると、約40%の動画が `<source>` 要素を持たず、`src` 属性を使用していることがわかります。これは、2020年に見られた比率（35%）とほぼ同じです。

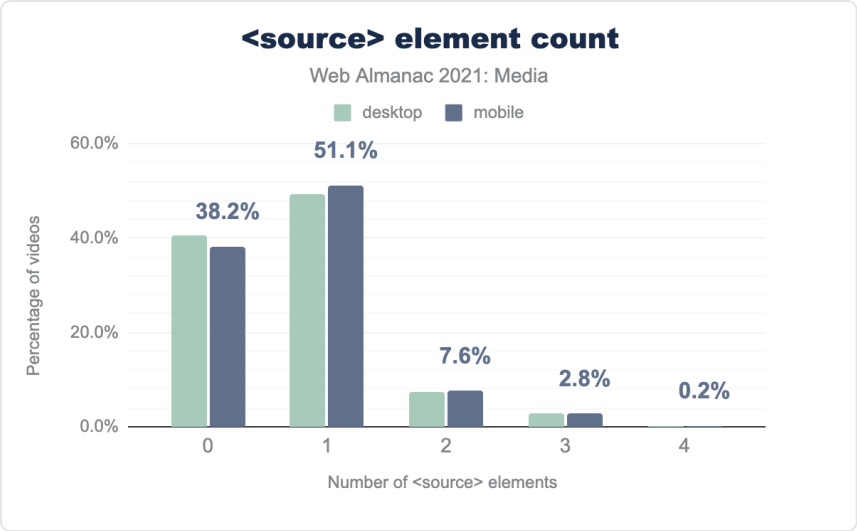


図5.34. `source` 要素の数。

また、`<source>` 要素は、1つの要素だけで使用されることが、もっとも多いことがわかります（`<video>` タグ全体の50%）。2つ以上のビデオ ソースを指定した `<video>` 要素は、わずか10% です。3:1の割合で、2つのソースが3つのソースよりも一般的で、3つ以上のソースもわずかにあります（48個のソースがある動画は1つあります！）。

2つのソースを使用している動画を見てみましょう。出現率上位10位を紹介します。

フォーマット	デスクトップ	モバイル
`["video/mp4","video/webm"]`	25.9%	26.1%
`["video/webm","video/mp4"]`	22.3%	23.3%
`["video/mp4","video/ogg"]`	20.2%	24.2%
`[null,null]`	14.1%	8.0%
`["video/mp4"]`	3.6%	3.4%
`["video/mp4","video/mp4"]`	3.5%	5.1%
`["application/x-mpegURL","video/mp4"]`	2.4%	2.1%
`[]`	2.1%	1.8%
`["video/mp4; codecs="avc1.42E01E, mp4a.40.2","video/webm; codecs="vp8, vorbis"]`	0.8%	0.3%
`["video/mp4;","video/webm;"]`	0.4%	0.3%

図5.35. `video` 要素の中に 2 つの `source` 要素がある場合に、もっとも一般的な `type` 値の順序のペアです。

上位10例のうち6例では、MP4が最初のソースとして記載されています。WebでのMP4サポート率は98.4%で[197]、MP4をサポートしていないブラウザは一般に `<video>` タグをまったくサポートしていません。つまり、これらのサイトは2つのソースを必要とせず、WebMやOggのビデオソースを削除することでウェブサーバーのストレージを節約できる、あるいはビデオの順番を逆にすればWebMをサポートするブラウザがWebMをダウンロードするようになるということを意味しています。

3つのソースを持つ `<video>` 要素についても同じ傾向があり、上位10例のうち8例がMP4で始まっています。

197. https://caniuse.com/mpeg4

フォーマット	デスクトップ	モバイル
`["video/mp4","video/webm","video/ogg"]`	30.4%	28.6%
`["video/mp4; codecs=avc1","video/mp4; codecs=avc1","video/mp4; codecs=avc1"]`	13.3%	16.4%
`["video/webm","video/mp4","video/ogg"]`	7.0%	6.3%
`["video/mp4; codecs=avc1"]`	5.8%	7.1%
`["video/mp4","video/ogg","video/webm"]`	5.0%	5.5%
`["video/mp4;","video/ogg; codecs="theora, vorbis","video/webm; codecs="vp8, vorbis"]`	3.8%	1.2%
`["video/mp4; codecs=hevc","video/webm","video/mp4"]`	3.2%	3.4%
`["video/mp4"]`	3.0%	3.0%
`["video/ogg; codecs="theora, vorbis","video/webm","video/mp4"]`	2.7%	3.3%
`["video/mp4","video/webm","video/ogv"]`	2.5%	1.7%

図5.36. `video` 要素の中に 3 つの `source` 要素がある場合、もっとも一般的な `type` 値の順序付きトリプレットです。

もちろん、これらの実装では、MP4ファイルを再生するだけで、WebMやOggファイルは無視されます。

ウェブページに占める動画の割合は、1〜2%から4〜5%へと飛躍的に伸びています。この成長は今後も続くと思われます。興味深いことに、「動画の王様」であるMP4は、依然として王様ではあるものの、（レスポンシブで順応できるビデオサイジングを特徴とする）動画ストリーミングフォーマットにシェアを侵食されつつあるのです。

`preload=auto` の使用を減らし、`preload=none` の使用を増やすなど、`<video>` タグをより効率的に使用する動きが見られます。また、`width` 属性の動作から、小さい画面用に動画が修正（または削除）されていることがわかります。

結論

冒頭で述べたようにウェブはますますビジュアル化しておりウェブの進化する機能セットを使ってメディアをエンコードし、埋め込み、レイアウトし、配信する方法は進化し続けているのです。今年は、ネイティブの遅延読み込みが、増大し続ける画像の転送サイズに歯止めをかけました。また、WebPのユニバーサルサポートとAVIFの初期サポートは、より豊かなビジュアルと効率的な未来への道を開くものです。映像面では、`<video>`要素の数が爆発的に増え、アダプティブ・ビットレート・ストリーミングのような高度な配信方式が使われるようになりました。

Web Almanacは、棚卸しや振り返りの機会です。そして、これからの道筋を描くときでもあります。2022年のウェブが、より効果的なビジュアルコミュニケーションになることを祈念しています。

著者



Eric Portis

@etportis @ericportis.com eeeps https://ericportis.com

Eric PortisはCloudinary[198]のWebプラットフォーム提唱者です。



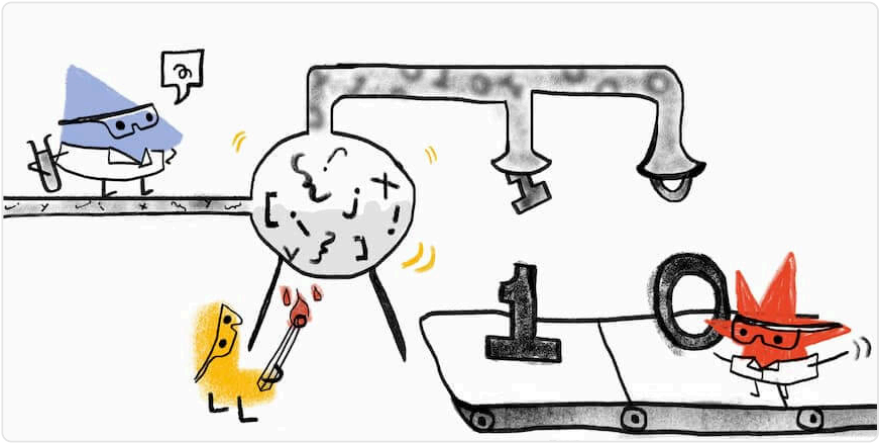
Doug Sillars

@dougsillars dougsillars https://dougsillars.com

Doug Sillarsは、開発者関係のリーダーであり、パフォーマンスとメディアの交差点で活動するデジタルノマドでもあります。彼のツイッターは @dougsillars で、ブログは dougsillars.com[199] で定期的に更新しています。

198. *https://cloudinary.com/*
199. *https://dougsillars.com*

部 I 章 6
WebAssembly



Ingvar Stepanyan によって書かれた。
Jarrod Overson、 Carlo Piovesan、 Alon Zakai、 Rick Viscomi と Barry Pollard によってレビュ。
Ingvar Stepanyan による分析。
Shaina Hantsis 編集。
Sakae Kotaro によって翻訳された。

序章

WebAssembly[200] はバイナリ命令形式で、開発者がJavaScript以外の言語で書かれたコードをコンパイルし、効率的で移植性の高いパッケージでWebに持ってくることを可能にするものです。既存のユースケースは、再利用可能なライブラリやコーデックから完全なGUIアプリケーションまで多岐にわたります。2017年から4年間、すべてのブラウザで利用できるようになり、それ以来、採用が進み、今年からWeb Almanacで利用状況を追跡する良いタイミングだと判断しました。

方法論

この分析では、2021-09-01にHTTP Archiveをクロールし、`Content-Type`

200. https://webassembly.org/

(`application/wasm`) またはファイル拡張子 (`.wasm`) にマッチするすべてのWebAssemblyレスポンスを選択しました。そして、それらすべて[201]をダウンロードし、その過程でさらにURL、レスポンス サイズ、圧縮前のサイズ、コンテンツ ハッシュをCSVファイル[202]に保存するscript[203] を使用しました。サーバーエラーで何度も応答が得られないリクエストや、コンテンツが実際にWebAssemblyらしくないものは除外しました。たとえば、いくつかのBlazor[204] ウェブサイトでは、.NET DLLs[205] に `Content-Type: application/wasm` を付けて提供していましたが、これらは実際にはフレームワークコアでパースされるDLLであって、WebAssemblyモジュールではありません。

WebAssemblyのコンテンツ解析では、BigQueryを直接利用することができませんでした。その代わりに、与えられたディレクトリ内のすべてのWebAssemblyモジュールを解析し、カテゴリごとの命令数、セクションサイズ、インポート/エクスポート数などを収集し、すべての統計情報を `stats.json` ファイルに格納する ツール[206] を作成しました。前のステップでダウンロードしたファイルのあるディレクトリで実行した後、結果のJSONファイルをBigQueryにインポート[207]し、対応する `summary_requests` と `summary_pages` テーブルと一緒に `httparchive.almanac.wasm_stats` へ結合して、それぞれのレコードが自己完結しWebAssembly要求、応答、モジュールコンテンツに関するすべての必要情報を含むようにします。この最終的なテーブルは、本章のすべての分析に使用されました。

クローラーリクエストを解析のソースとして使用することは、この記事の数字を見る際に注意すべきトレードオフがあります。

- まず、ユーザーとのインタラクションをきっかけに発生するリクエストの情報がありませんでした。ページロード中に収集されたリソースのみを対象としました。
- 次に、ウェブサイトには人気のあるものとそうでないものがありますが、正確な訪問者データがなかったため、それを考慮せず、検出されたWasmの利用状況をそれぞれ同等に扱いました。
- 最後に、サイズのようなグラフでは、ユニークなファイルだけを比較するのではなく、複数のウェブサイトで使用されている同じWebAssemblyモジュールをユニークな使用量としてカウントしています。これは、ライブラリ同士を比較するのではなく、Webページ全体におけるWebAssemblyの使用状況の全体像にもっとも興味があるためです。

これらのトレードオフは他の章で行われた分析ともっとも一致していますが、もし他の統計を集めることに興味があるなら、`httparchive.almanac.wasm_stats` のテーブルに対し

201. https://github.com/RReverser/wasm-stats/blob/master/downloader/wasms.csv
202. https://github.com/RReverser/wasm-stats/blob/master/downloader/results.csv
203. https://github.com/RReverser/wasm-stats/blob/master/downloader/index.mjs
204. https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor
205. https://docs.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library#the-net-framework-assembly
206. https://github.com/RReverser/wasm-stats
207. https://cloud.google.com/bigquery/docs/batch-loading-data

て独自のクエリを実行することを歓迎します。

モジュール数は？

デスクトップで3854、モバイルで3173のWebAssemblyリクエストが確認されました。これらのWasmモジュールはデスクトップでは2724ドメイン、モバイルでは2300ドメインで使用されており、デスクトップとモバイルの全ドメインのそれぞれ0.06%と0.04%に相当します。

興味深いことに、もっとも人気のあるMIMEタイプを見ると、`Content-Type: application/wasm`が圧倒的に人気がある一方で、すべてのWasmレスポンスをカバーしているわけではないことがわかります。`.wasm`の拡張子を持つ他のURLも含めてよかったです。

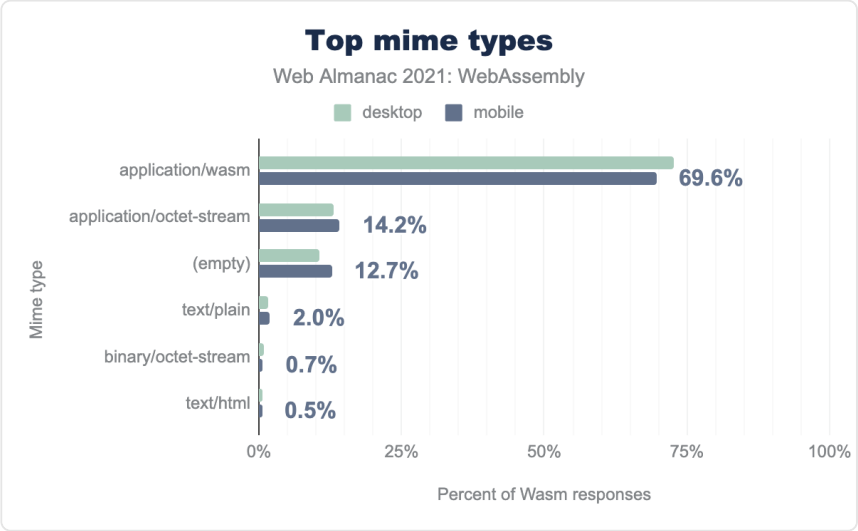


図6.1. 上位のMIMEタイプ。

そのうちのいくつかは`application/octet-stream`という任意のバイナリデータの一般的な型を使い、いくつかは`Content-Type`ヘッダーを持たず、他のものはplainやHTMLなどのテキスト型、あるいは`binary/octet-stream`のように不正な型を使っています。

WebAssemblyの場合、正しい`Content-Type`ヘッダーを提供することは、セキュリティ上の理由だけでなく、`WebAssembly.compileStreaming`や`WebAssembly.instantiateStreaming`によるストリーミング、コンパイルやインスタンス化の高速化も可能になるので、重要です。

Wasmのライブラリはどのくらいの頻度で再利用されているのでしょうか？

また、これらの応答をダウンロードする際にその内容をハッシュ化し、そのハッシュをディスク上のファイル名として使用することで、重複排除を行いました。その後、デスクトップでは656個、モバイルでは534個の一意のWebAssemblyファイルが残りました。

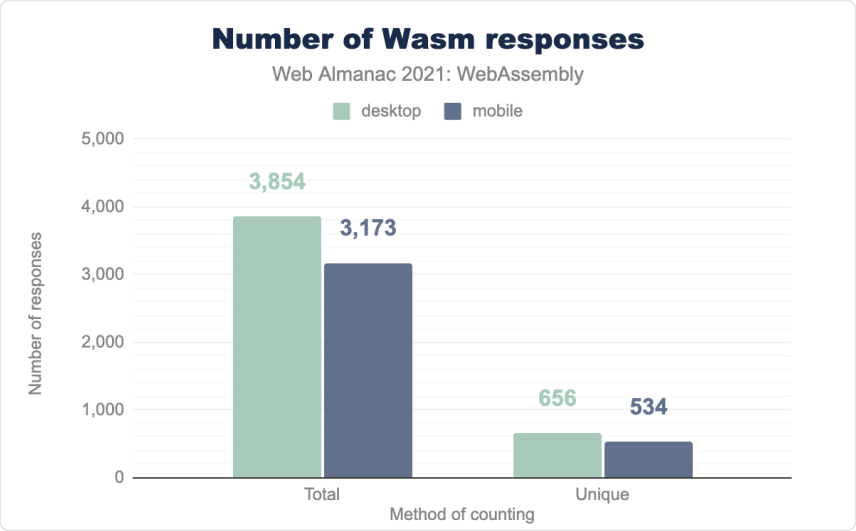


図6.2. Wasmのレスポンス数

ユニークなファイルの数と総レスポンス数の差が激しいことから、さまざまなWebサイトでWebAssemblyライブラリが高度に再利用されていることがすでに示唆されています。クロスオリジン/同一オリジンのWebAssemblyリクエストの分布を見ると、さらに確認できます。

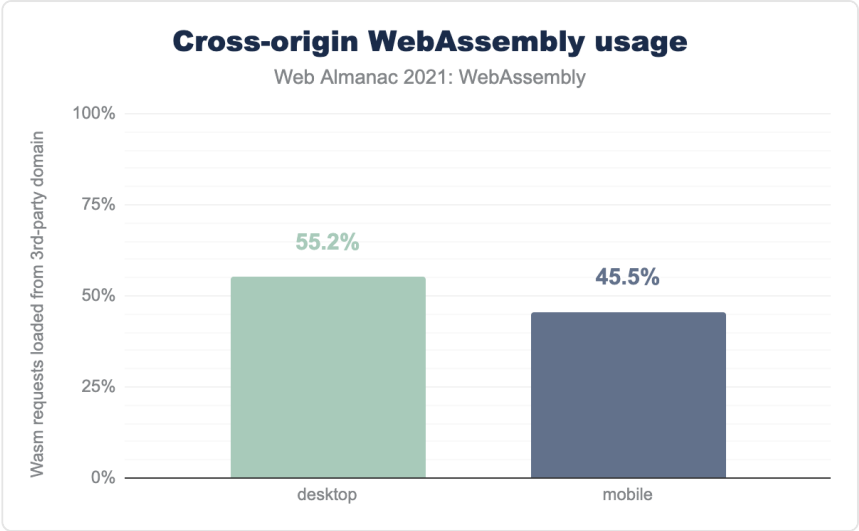


図6.3. クロスオリジンのWebAssemblyの使用。

では、その再利用されるライブラリは何なのか、もっと掘り下げて考えてみましょう。まず、コンテンツ・ハッシュだけでライブラリの重複排除を試みましたが、残ったものの多くが、ライブラリのバージョンだけが異なる重複ライブラリであることがすぐに判明しました。

そこで、URLからライブラリ名を抽出することにしました。名前の衝突の可能性があるため、理論的にはより問題がありますが、実際にはトップライブラリのためのより信頼性の高いオプションであることが判明しました。URLからファイル名を抽出し、拡張子、マイナーバージョン、コンテンツハッシュのように見えるサフィックスを削除し、結果を繰り返し回数でソートして、各クライアントの上位10個のモジュールを抽出しました。残ったモジュールについては、どのライブラリから来たものなのか、手作業で調べました。

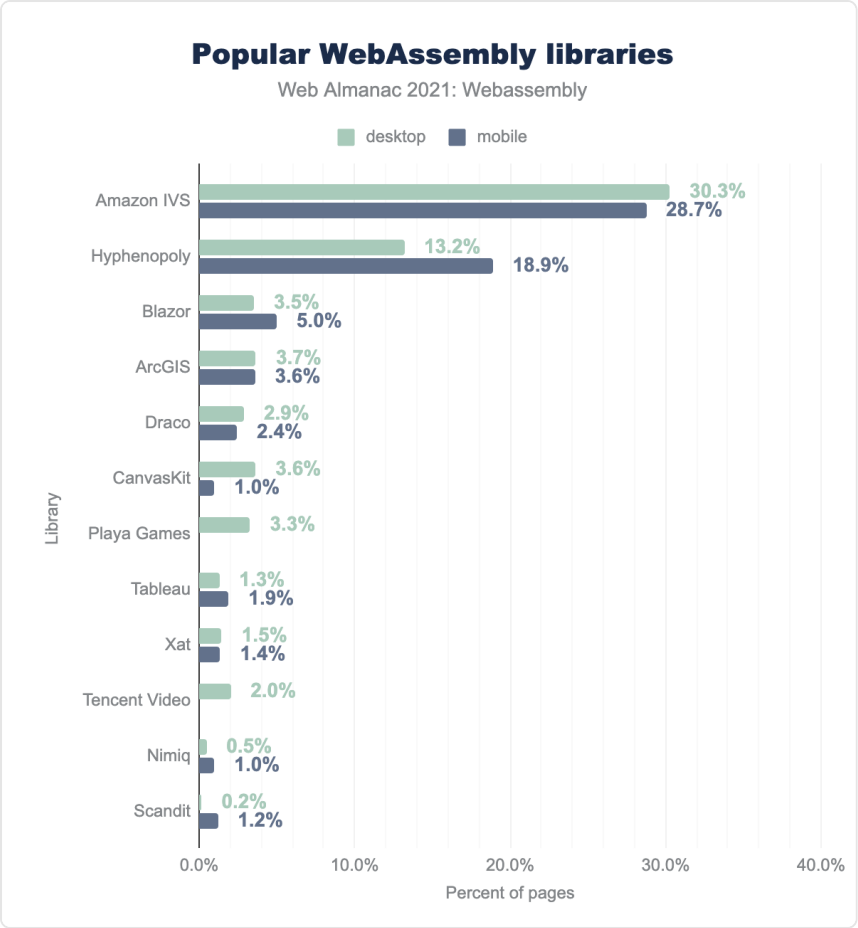


図6.4. WebAssemblyの人気ライブラリ。

デスクトップとモバイルの両方で使用されるWebAssemblyのほぼ3分の1は、Amazon Interactive Video Service[208] プレーヤー ライブラリに属しています。オープンソースではありませんが、関連するJavaScriptのグルーコードを調べると、Emscripten[209] でビルドされていることがわかります。

次に、Hyphenopoly[210]は、さまざまな言語のテキストをハイフンでつなぐためのライブラリで、デスクトップとモバイルでそれぞれ13%と19%のWasmリクエストを占めています。JavaScriptとAssemblyScript[211]で構築されています。

208. *https://aws.amazon.com/ivs/*
209. *https://emscripten.org/*
210. *https://github.com/mnater/Hyphenopoly*
211. *https://www.assemblyscript.org/*

デスクトップとモバイルのトップ10リストにある他のライブラリは、それぞれWebAssemblyリクエストの最大5%を占めています。上記のライブラリの完全なリストと、推測されるツールチェーン、および詳細情報を含む対応するホームページへのリンクはこちらです。

- Amazon IVS[212] (Emscripten)
- Hyphenopoly[213] (AssemblyScript)
- Blazor[214] (.NET)
- ArcGIS[215] (Emscripten)
- Draco[216] (Emscripten)
- CanvasKit[217] (Emscripten)
- Playa Games[218] (Unity via Emscripten)
- Tableau[219] (Emscripten)
- Xat[220] (Emscripten)
- Tencent Video[221] (Emscripten)
- Nimiq[222] (Emscripten)
- Scandit[223] (Emscripten)

方法論と結果について、もう少しだけ注意点があります。

1. Hyphenopolyはさまざまな言語の辞書を小さなWebAssemblyファイルとしてロードしますが、これらは技術的に別のライブラリではなく、Hyphenopoly自体のユニークな使用法でもないため、上のグラフからは除外しています。
2. Playa GamesのWebAssemblyファイルは、似たようなドメインでホストされている同じゲームで使用されているようです。私たちは、クエリでそれらを個々の使用として数えますが、リストの他の項目とは異なり、再利用可能なライブラリとして数えるべきかどうかは明らかではありません。

212. *https://aws.amazon.com/ivs/*
213. *https://mnater.github.io/Hyphenopoly/*
214. *https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor*
215. *https://developers.arcgis.com/javascript/latest/*
216. *https://google.github.io/draco/*
217. *https://skia.org/docs/user/modules/canvaskit/*
218. *https://www.playa-games.com/en/*
219. *https://help.tableau.com/current/api/js_api/en-us/JavaScriptAPI/js_api.htm*
220. *https://xat.com/*
221. *https://intl.cloud.tencent.com/products/vod*
222. *https://www.npmjs.com/package/@nimiq/core-web*
223. *https://www.scandit.com/developers/*

送るサイズはどのくらいですか？

WebAssemblyにコンパイルされた言語は、通常、独自の標準ライブラリを持っています。言語によってAPIや値の型が大きく異なるため、JavaScriptのビルトインを再利用することができないのです。その代わりに、独自のコードだけでなく、標準ライブラリのAPIもコンパイルして、単一のバイナリとしてまとめてユーザーに提供する必要があります。その結果、ファイルサイズはどうなるのだろうか？見てみよう。

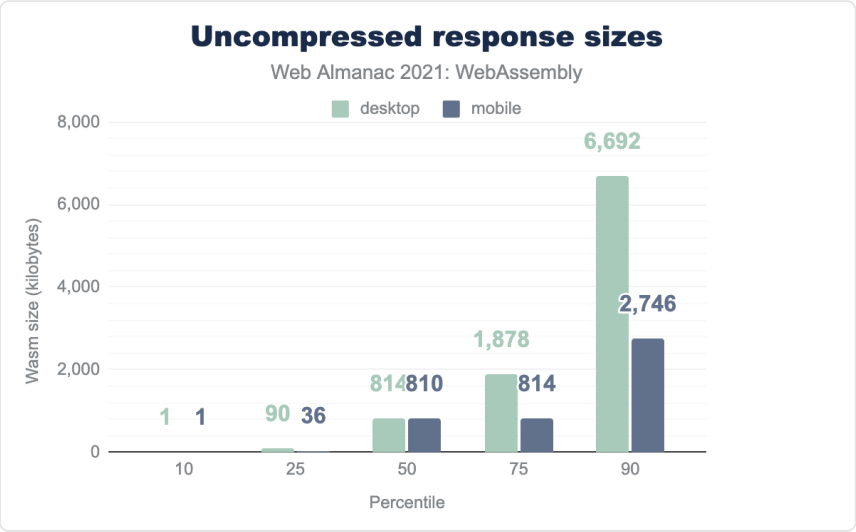


図6.5. 非圧縮のレスポンスサイズ。

サイズもさまざまで、単純なヘルパーライブラリからWebAssemblyでコンパイルされた完全なアプリケーションまで、さまざまな種類のコンテンツをきちんとカバーしていることが分かります。

最大で81MBのサイズが確認され、かなり気になるところですが、これは非圧縮のレスポンスであることを念頭に置いてください。RAMの使用量や起動時のパフォーマンスも重要ですが、Wasmバイトコードの利点の1つは高圧縮であり、ダウンロード速度や課金上の理由から、通信上のサイズが重要になります。

代わりに、サーバーから送信された生のレスポンスボディのサイズを確認してみましょう。

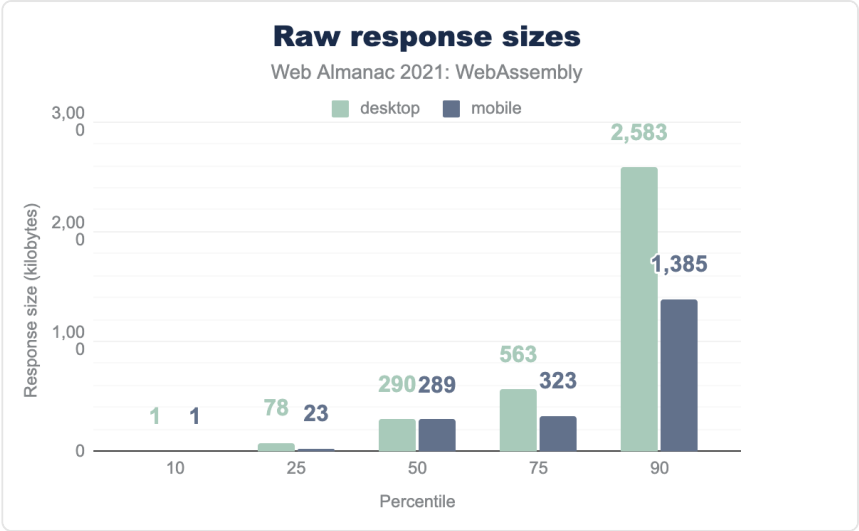


図6.6. 生のレスポンスサイズ。

中央値は約290KBで、半分が290KB以下、半分がそれ以上のダウンロード量ということになります。Wasmの全レスポンスの90%は、デスクトップで2.6MB未満、モバイルで1.4MB未満に留まっています。

44 MB

図6.7. デスクトップでダウンロードした最大のWasmレスポンス。

HTTP Archiveの最大レスポンスでは、デスクトップで約44MB、モバイルで約28MBのWasmがダウンロードされます。

圧縮しても、世界の多くの地域ではまだ高速インターネット接続ができないことを考えると、この数字はかなり極端です。アプリケーションやライブラリの範囲を狭める以外に、Webサイトがこの数字を改善するためにできることはあるのでしょうか？

Wasmは、実際どのように圧縮されているのですか？

まず、`Content-Encoding`ヘッダーに基づいて、これらの生のレスポンスで使用される圧縮方法について見てみましょう。モバイルでは帯域幅がさらに重要なので、ここではモバイルのデータセットを表示しますが、デスクトップの数値もかなり似ています。

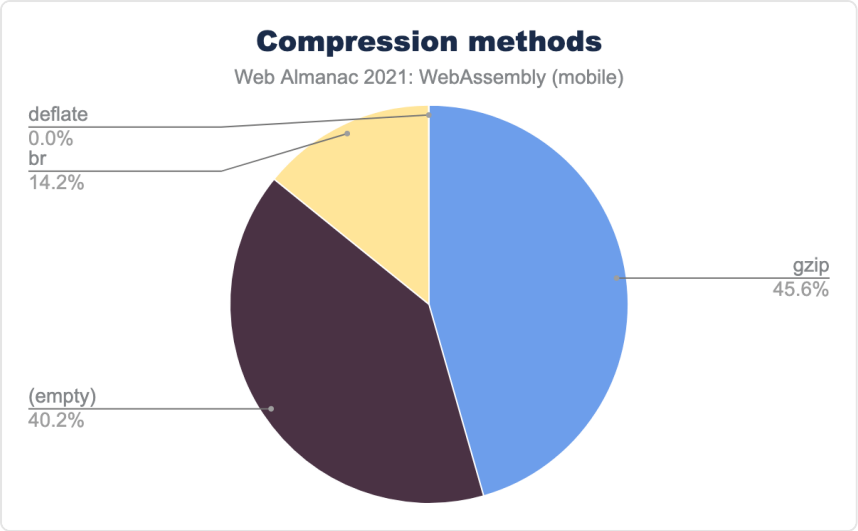


図6.8. Compression methods.

残念ながら、モバイルにおけるWebAssemblyのレスポンスの40%は、圧縮されずに送信されていることがわかります。

40.2%

図6.9. モバイルでの非圧縮WebAssemblyレスポンスの割合。

また、46%はgzipを使用しています。gzipは長い間、ウェブにおける事実上の標準的な方法であり、今でもきちんとした圧縮率を提供していますが、今日のベストアルゴリズムとは言えません。最後に、14% のみがBrotliを使用しています。この最新の圧縮形式はさらに優れた比率を提供し、 すべてのモダン ブラウザ[224]でサポートされています。実際、BrotliはWebAssemblyをサポートしているすべてのブラウザでサポートされているので、これらを一緒に使用しない理由はありません。

圧縮率を改善できないか？

違いがあったのでしょうか？我々は、それを解明するために、すべてのWebAssemblyファイルをBrotli（圧縮レベル9）で再圧縮することにしました。各ファイルで使用したコマンドは

224. https://caniuse.com/brotli

```
brotli -k9f some.wasm -o some.wasm.br
```

でき上がったサイズはこちら。

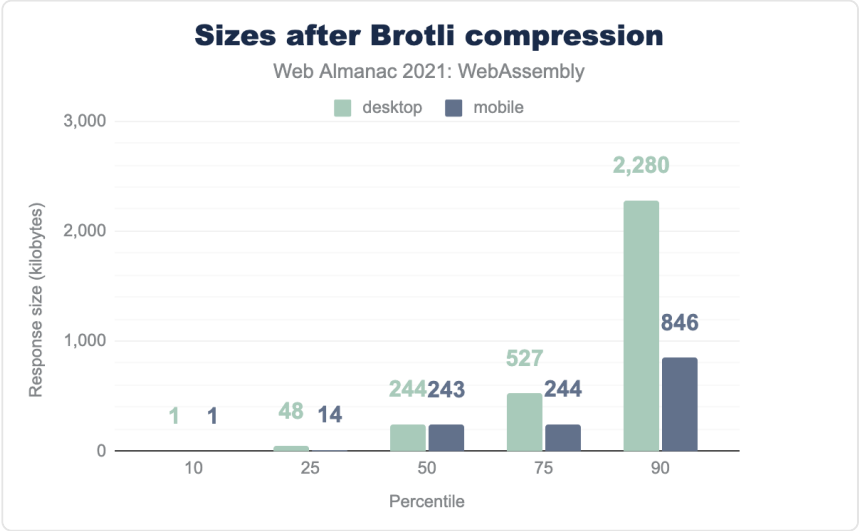


図6.10. Brotli圧縮後のサイズ。

中央値はほぼ290KBからほぼ240KBに下がり、これはもうかなり良い兆候です。上位10パーセンタイルは、2.5MB / 1.4MBから2.2MB / 0.8MBに減少しています。その他のパーセンタイルでも、大幅な改善が見られます。

パーセンタイルはその性質上、データセット間で必ずしも同じファイルに収まるとは限らないので、グラフ間で直接数値を比較し、サイズの節約を理解するのは難しいかもしれません。その代わり、これからは各最適化によってもたらされる節約額そのものを、一歩ずつ見ていくことにしましょう。

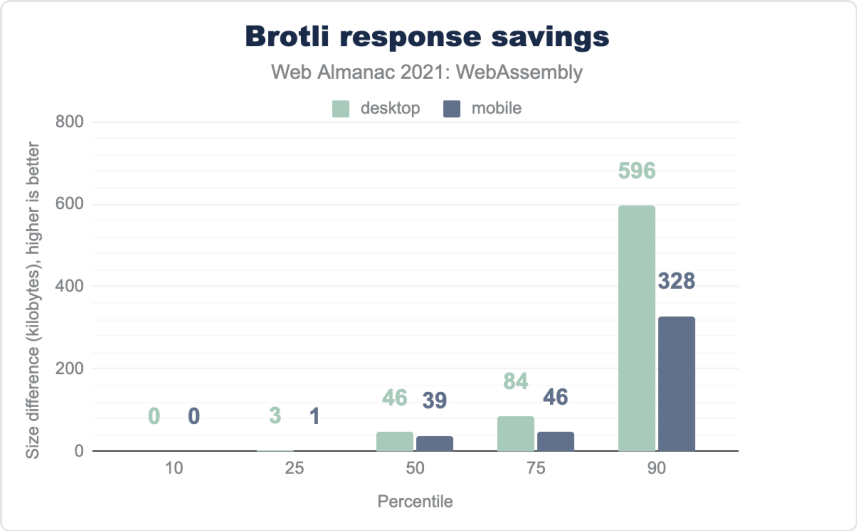


図6.11. Brotli対応節約術。

中央値は約40KBの節約となった。上位10%は、デスクトップで600KB弱、モバイルで330KBを節約しています。最大の削減量は、35MB/21MBに達します。これらの違いは、少なくともWebAssemblyコンテンツについては、可能な限りBrotli圧縮を有効にすることへ有利に働きます。

さらに興味深いことに、グラフの反対側では、もっとも節約できるはずの1.4MBまで後退していることがわかりました。何があったのでしょうか。Brotliの再圧縮が、一部のモジュールで状況を悪化させたというのは、どういうことなのでしょうか？

前述したように今回は圧縮レベル9のBrotliを使用したが、正直、この記事を書くまですっかり忘れていたのだが、レベル10と11もあるです。たとえば、Squashのベンチマーク[225]で見られるように、これらのレベルでは、パフォーマンスが急激に低下する代わりに、さらに良い結果が得られます。このようなトレードオフの関係から、一般的なオンザフライ圧縮の候補としては不利になるため、この記事では使用せず、より穏やかなレベル9を採用しました。しかしウェブサイトの作者は、静的リソースを先に圧縮したり圧縮結果をキャッシュしたりすることを選択でき、CPU時間を犠牲にすることなく、さらに帯域を節約できます。このようなケースは、分析の結果、リグレッションとして表示されます。つまりリソースは、この記事で行ったよりもさらに最適化することが可能であり、場合によっては、すでに最適化されていることもあるのです。

225. *https://quixdb.github.io/squash-benchmark/#results-table*

どの部分が一番スペースを取っているのでしょうか？

圧縮は別として、WebAssemblyバイナリのハイレベルな構造を分析することで、最適化の機会を探すことも可能でしょう。どのセクションがもっとも大きなスペースを占めているか？これを確認するために、すべてのWasmモジュールのセクションサイズを合計し、バイナリサイズ合計で割りました。ここでもモバイルデータセットの数字を使っていますが、デスクトップの数字もそれほど大きくはずれていません。

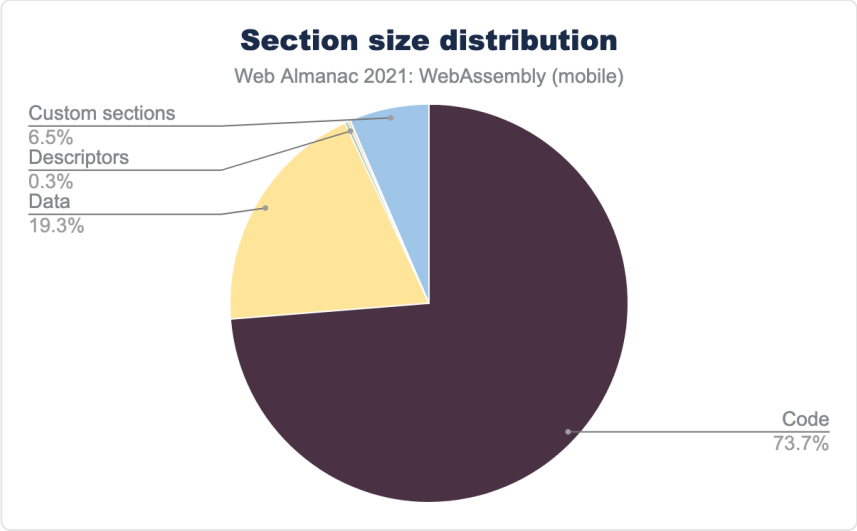


図6.12. セクションのサイズ分布。

当然のことながら、バイナリサイズの大部分（~74%）はコンパイルされたコード自身によるもので、その次に埋め込まれた静的データが~19%となっています。関数型、インポート/エクスポート記述子などは、総サイズのごく一部を構成しているに過ぎません。カスタムセクションは、モバイルデータセットにおける総サイズの約6.5%を占めています。



図6.13. モバイルデータセットのバイナリサイズに占めるカスタムセクションの割合。

カスタムセクションは、主にWebAssemblyのサードパーティツールのために使用されます。型結合システム、リンカー、DevToolsなどのための情報を含むかもしれません。どれも正当なユースケースですが、プロダクションコードではほとんど必要ないので、このような大きな比率は疑わしいと言えます。それでは、カスタムセクションがもっとも多いファイル

トップ10にあるものを見てみましょう。

URL	カスタムセクションのサイズ	カスタムセクション
.../dotnet.wasm[226]	15,053,733	`name`
.../unity.wasm.br?v=1.0.8874[227]	9,705,643	`name`
.../nanoleq-HTML5-Shipping.wasmgz[228]	8,531,376	`name`
.../export.wasm[229]	7,306,371	`name`
.../c0c43115a4de5de0/.../northstar_api.wasm[230]	6,470,360	`name`, `external_debug_info`
.../9982942a9e080158/.../northstar_api.wasm[231]	6,435,469	`name`, `external_debug_info`
.../ReactGodot.wasm[232]	4,672,588	`name`
.../v18.0-591dd9336/trace_processor.wasm[233]	2,079,991	`name`
.../v18.0-615704773/trace_processor.wasm[234]	2,079,991	`name`
.../canvaskit.wasm[235]	1,491,602	`name`

図6.14. 最大のカスタムセクション。

それらはすべて、基本的なデバッグのための関数名を含む `name` セクションにほぼ限定されています。実際、データセットに目を通し続けると、それらのカスタムセクションのほとんどすべてがデバッグ情報のみを含んでいることがわかります。

デバッグ情報を削除することで、どの程度節約できるのか？

デバッグ情報はローカルでの開発には有用ですが、これらのセクションは非常に大きく、上の表では圧縮前に14MB以上あります。もしユーザが体験している問題をデバッグしたいのであれば、送信前に `llvm-strip` や `wasm-strip` または `wasm-opt --strip-debug` を

226. https://gallery.platform.uno/package_85a43e09d7152711f12894936a8986e20694304a/dotnet.wasm
227. https://cdn.decentraland.org/@dcl/unity-renderer/1.0.12536-20210902152600.commit-86fe4be/unity.wasm.br?v=1.0.8874
228. https://nanoleq.com/nanoleq-HTML5-Shipping.wasmgz
229. https://convertmodel.com/export.wasm
230. https://webasset-akm.imvu.com/asset/c0c43115a4de5de0/build/northstar/js/northstar_api.wasm
231. https://webasset-akm.imvu.com/asset/9982942a9e080158/build/northstar/js/northstar_api.wasm
232. https://superctf.com/ReactGodot.wasm
233. https://ui.perfetto.dev/v18.0-591dd9336/trace_processor.wasm
234. https://ui.perfetto.dev/v18.0-615704773/trace_processor.wasm
235. https://unpkg.com/canvaskit-wasm@0.25.1/bin/profiling/canvaskit.wasm

使ってデバッグ情報を取り除き、生のスタックトレースを収集し、オリジナルのバイナリを使ってローカルでソースロケーションにマッチさせるというアプローチがよいかもしれません。

このデバッグ情報を削除することで、前のステップのBrotliだけと比較して、Brotliとの組み合わせでどの程度節約できるかを確認するのは興味深いことです。しかし、データセットに含まれるほとんどのモジュールにはカスタムセクションがないので、90パーセンタイル以下は役に立たないだろう。

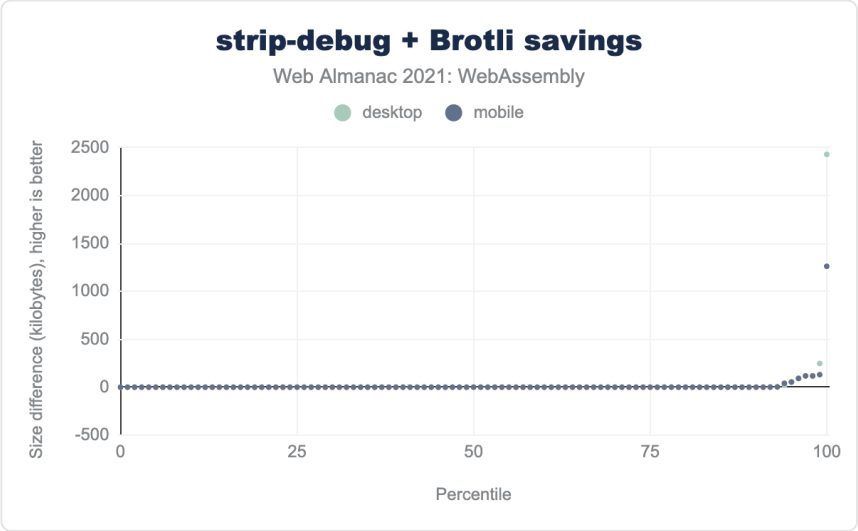


図6.15. strip-debug + Brotliの節約。

代わりに、カスタムセクションを持つファイルにのみ、貯蓄の分配を見てみましょう。

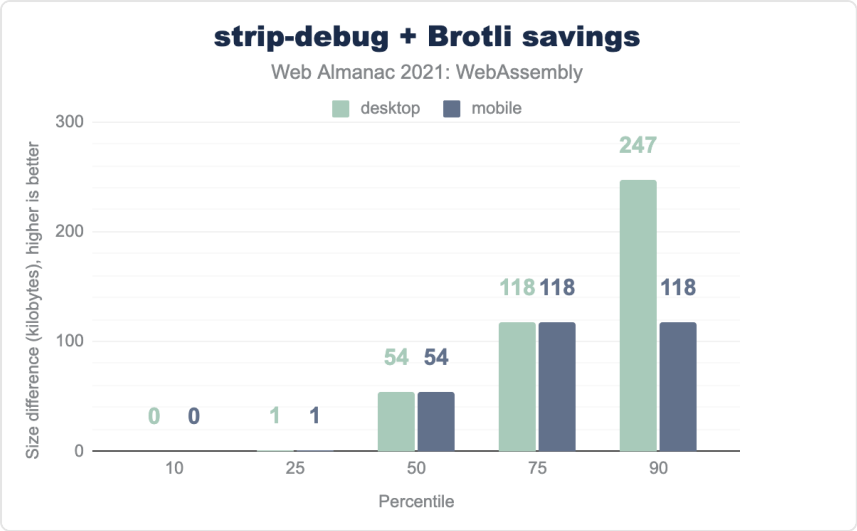


図6.16. strip-debug + Brotliの節約。

グラフからわかるように、いくつかのファイルのカスタムセクションは無視できるほど小さいですが、中央値は54KBで、90パーセンタイルはデスクトップで247KB、モバイルで118KBです。デスクトップとモバイルで最大のWasmバイナリで2.4MB/1.3MBの節約となり、とくに低速接続ではかなり顕著な改善となりました。

上の表から、カスタムセクションの生のサイズと比べると、その差がかなり小さいことにお気づきだろうか。その理由は、`name`セクションは、その名前が示すように、ほとんどが関数名で構成されているからです。関数名は、繰り返しの多いASCII文字列であり、そのため非常に圧縮されやすいのです。

カスタムセクションを`llvm-strip`で削除する過程で、WebAssemblyモジュールに変更が加えられ、圧縮前は小さくても圧縮後はわずかに大きくなるという異常事態がいくつか発生しています。しかし、このようなケースはまれで、圧縮後のモジュールの合計サイズと比較すると、サイズの差は重要でありません。

`wasm-opt`を使うとどのくらい節約できるのでしょうか？

Binaryen[236]スイートの`wasm-opt`は、結果のバイナリのサイズとパフォーマンスの両方を改善することができる強力な最適化ツールです。Emscripten、wasm-pack、AssemblyScriptなどの主要なWebAssemblyツールチェーンで使用され、基礎となるコンパイラが生成するバイナリを最適化するために使用されます。

236. *https://github.com/WebAssembly/binaryen*

非圧縮と圧縮の両方の実世界ベンチマークにおいて、大幅なサイズ削減を実現します。

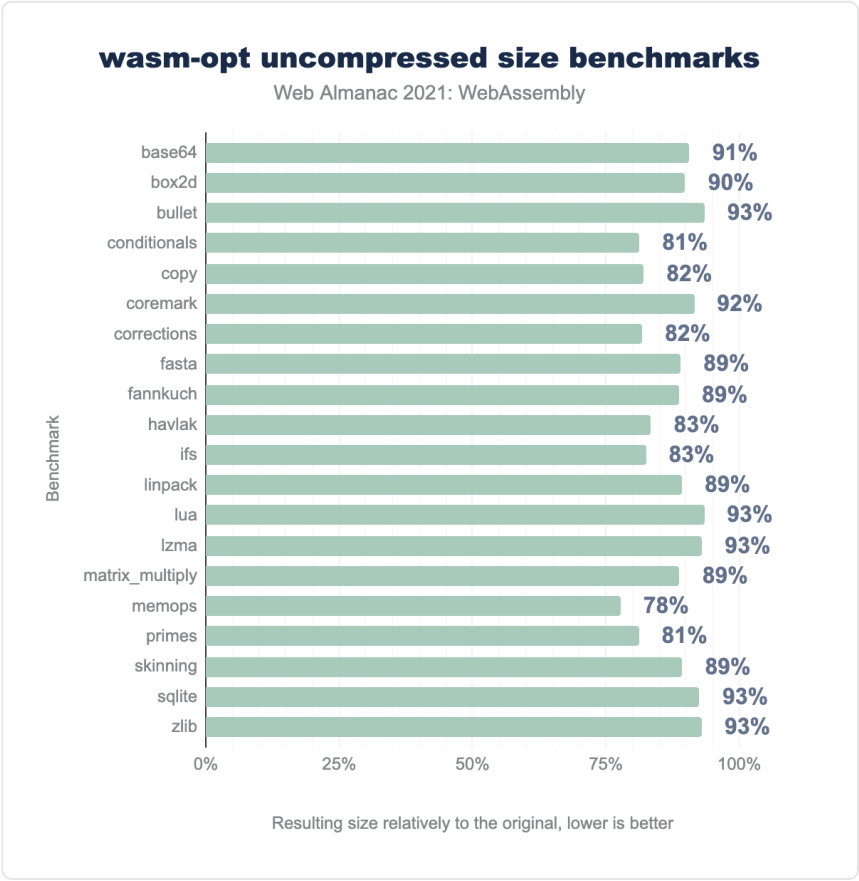


図6.17. `wasm-opt` の非圧縮サイズベンチマーク。

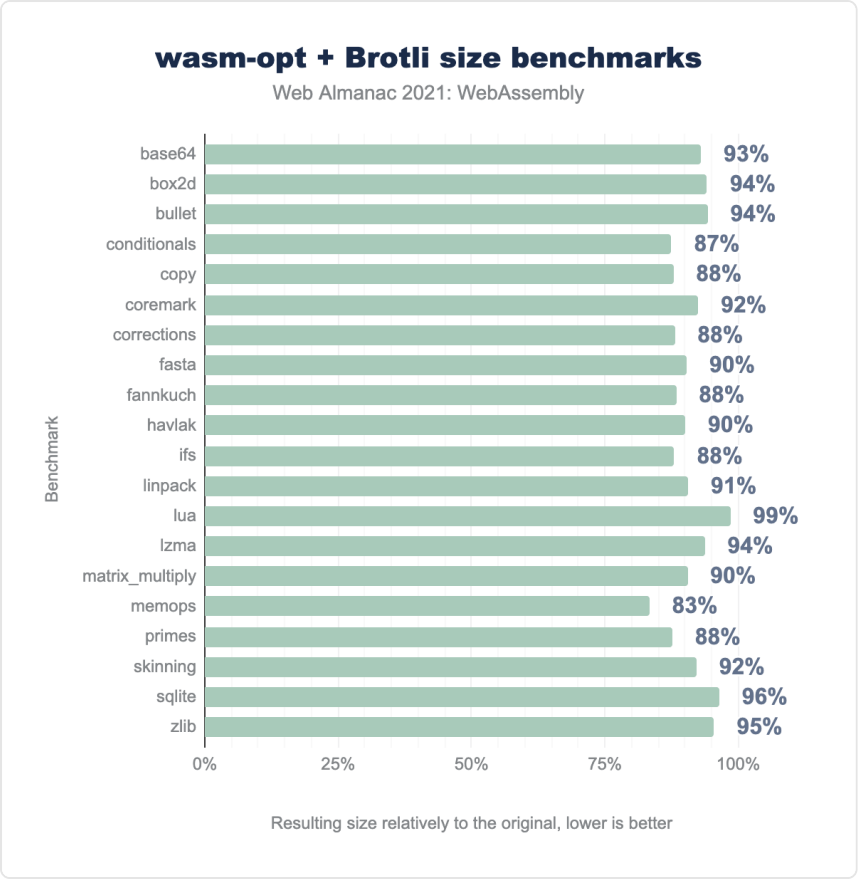


図6.18. `wasm-opt` *+ Brotliサイズベンチマーク。*

収集したHTTPアーカイブのデータセットでも `wasm-opt` の性能を確認することにしましたが、引っ掛かりがあります。

前述の通り、 `wasm-opt` はすでにほとんどのコンパイラツールチェーンで使用されているので、データセット内のモジュールのほとんどはすでにその結果の成果物となっています。上記の圧縮解析とは異なり、既存の最適化を逆にしてオリジナルで `wasm-opt` を実行する方法はありません。その代わりに、最適化前のバイナリに対して `wasm-opt` を再実行することで、結果に歪みを与えています。これは、strip-debugステップの後に生成されたバイナリに対して使用したコマンドです。

```
wasm-opt -O -all some.wasm -o some.opt.wasm
```

そして、その結果をBrotliに圧縮して、いつものように前のステップと比較しました。

このデータは実際の使用状況を代表するものではなく、一般消費者は `wasm-opt` を通常通り使用すべきですが、CDNのように最適化を大規模に実行したい消費者や、バイナリエンのチーム自身には有用かもしれません。

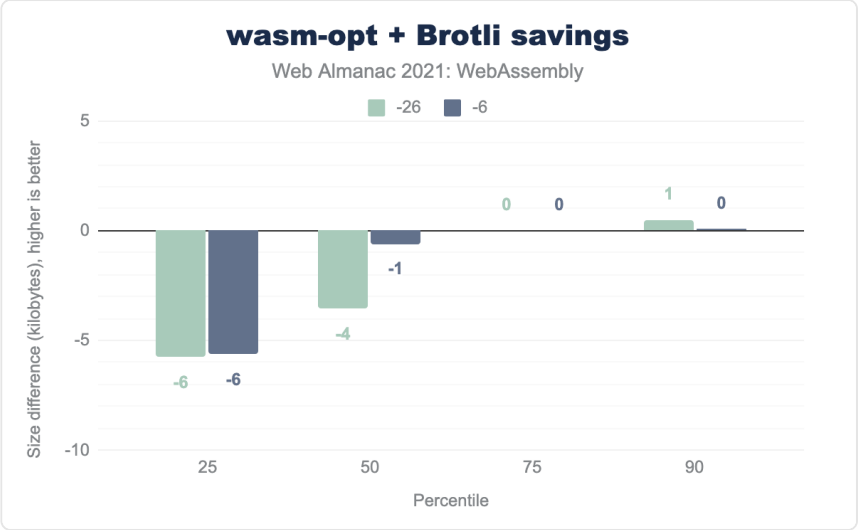


図6.19. `wasm-opt` + Brotliの節約。

グラフの結果はまちまちですが、どの変化も26KBまでと比較的小さいものです。もし、外れ値（0と100のパーセンタイル）を含めると、最良側でデスクトップが最大1MB、モバイルが240KB、最悪デスクトップが255KB、モバイルが175KBと、より大幅に改善されることが分かります。

ごく一部のファイルで大幅な節約ができたということは、ウェブで公開する前に最適化されていなかった可能性が高いということです。しかし、なぜ他の結果はこれほどまでにまちまちなのでしょうか？

圧縮しない場合の節約サイズを見てみると、今回のデータセットでも、`wasm-opt` は一貫してファイルサイズをほぼ同じにするか、あるいはさらにサイズをわずかに向上させるケースが多く、最適化されていないファイルでは大きな節約となることがより明らかになります。

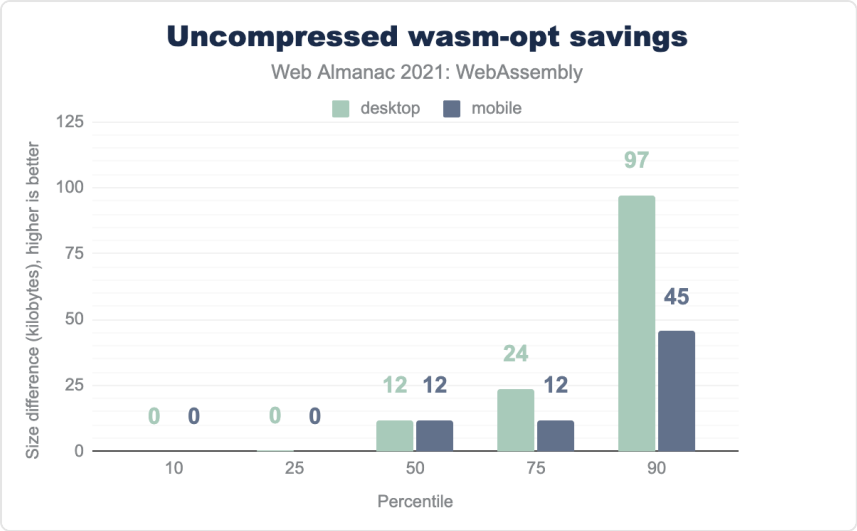


図6.20. 圧縮されていない `wasm-opt` の節約。

このことから、圧縮後のグラフに驚くべき分布が見られる理由はいくつか考えられます。

1. 前述の通り、我々のデータセットは実際の `wasm-opt` の使用状況とは異なっており、大半のファイルは `wasm-opt` によってすでに最適化されています。さらに命令の並べ替えを行い非圧縮サイズを少し改善すると、特定のパターンが他のパターンよりも圧縮されやすくなったり、されにくくなったりすることになり、その結果、統計的なノイズが発生することになります。
2. 我々はデフォルトの `wasm-opt` パラメーターを使用していますが、ユーザによっては `wasm-opt` フラグを微調整して、特定のモジュールに対してさらに良い節約を実現しているかもしれません。
3. 前述したように、ネットワーク（圧縮）サイズがすべてではありません。WebAssemblyのバイナリが小さいと、VMでのコンパイルが速く、コンパイル中のメモリ消費量が少なく、コンパイルしたコードを保持するメモリも少なくなる傾向があります。`wasm-opt` はここでバランスを取る必要があり、圧縮されたサイズが、より良い生のサイズを優先して後退することもあります。
4. 最後に、いくつかの回帰は、そのバランスを研究し改善するための貴重な例となる可能性がありそうです。私たちはBinaryenチームに それらを報告し[237]、最適化の可能性についてより深く検討できるようにしました。

237. https://github.com/WebAssembly/binaryen/issues/4322

人気のあるインストラクションは何ですか？

ここまでで、Wasmをセクションごとに切り分けたときの中身を垣間見ることができました。ここでは、WebAssemblyモジュールの中でもっとも大きく、もっとも重要な部分であるコードセクションの内容をより深く見ていきましょう。

インストラクションをさまざまなカテゴリーに分け、すべてのモジュールでまとめてカウントしています。

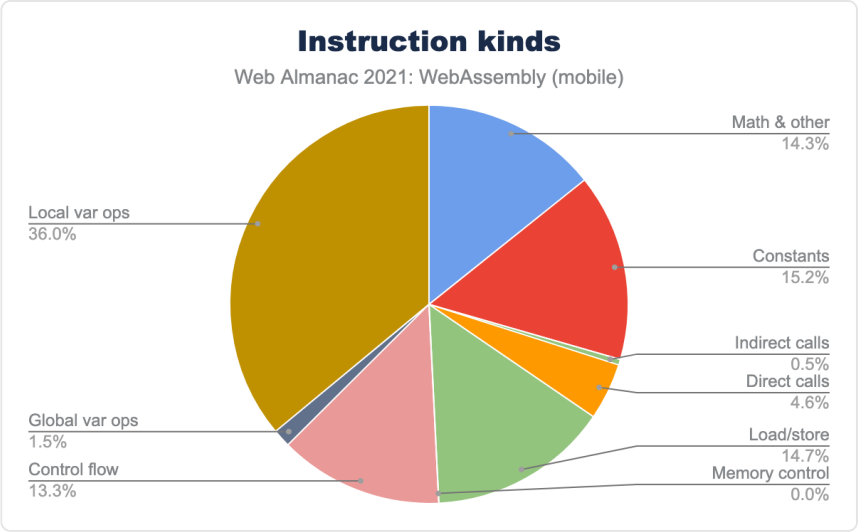


図6.21. インストラクションの種類。

この分布から得られる1つの驚くべきことは、ローカル変数の操作、つまり `local.get`, `local.set`, `local.tee` が最大のカテゴリーで36%を占め、次のカテゴリーであるライン定数 (15.2%), ロード/ストア操作 (14.7%) とすべての数学および論理演算 (14.3%) をはるかに上回っていることです。ローカル変数演算は、通常、コンパイラの最適化パスの結果として生成されます。これは、高価なメモリアクセス操作を可能な限りローカル変数にダウングレードし、その後、エンジンがこれらのローカル変数をCPUレジスタに置くことで、より安価にアクセスできるようにするためです。

Wasmにコンパイルする開発者にとって実用的な情報ではありませんが、エンジンやツールの開発者にとっては、さらなるサイズ最適化の可能性がある領域として興味深いものでしょう。

MVP後の拡張機能の使い方は？

もう1つ興味深い指標は、MVP後のWasmの拡張性です。WebAssembly 1.0は数年前にリリースされましたが、今でも活発に開発されており、時間の経過とともに新しい機能が追加されて成長しています。これらの中には、共通の操作をエンジンに移すことでコードサイズを改善するもの、より強力なパフォーマンスプリミティブを提供するもの、開発者の体験やウェブとの統合を改善するものなどがあります。公式の 機能ロードマップ[238] では、人気のあるすべてのエンジンの最新バージョンで、これらの提案のサポートを追跡しています。

Almanacのデータセットでも、その採用状況を見てみよう。

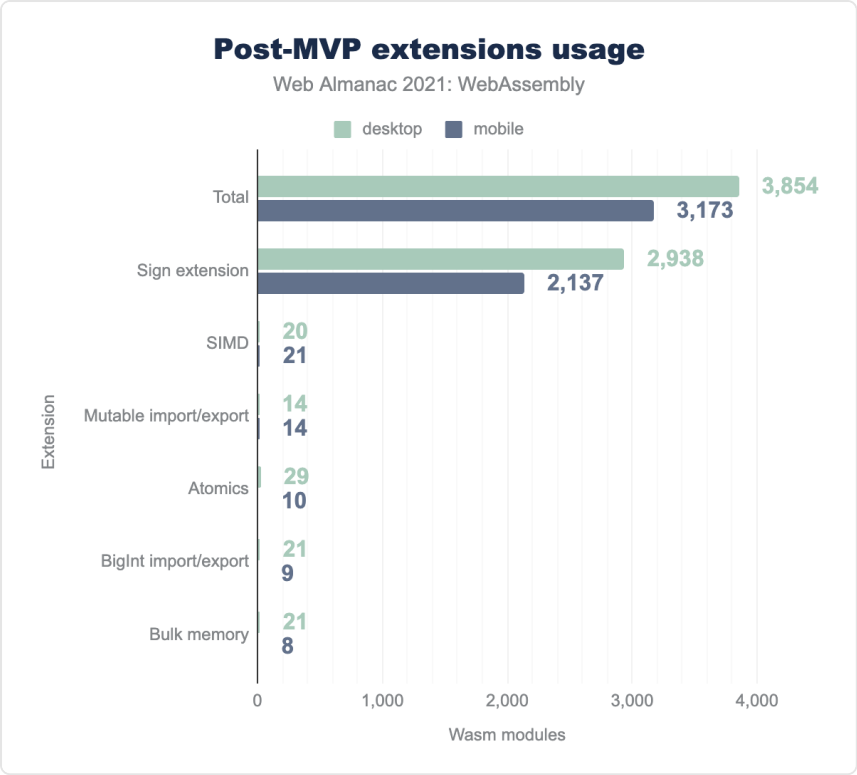


図6.22. MVP後の拡張機能の使用状況。

それは、sign-extension operators proposal[239] という機能です。MVPからそれほど時間が経たないうちにすべてのブラウザに提供され、LLVM（Clang / EmscriptenやRustが使用するコン

238. https://webassembly.org/roadmap/
239. https://github.com/WebAssembly/sign-extension-ops/blob/master/proposals/sign-extension-ops/Overview.md

パイラバックエンド）でもデフォルトで有効になったので、その採用率の高さが理解できます。それ以外の機能は、現在のところ開発者がコンパイル時に明示的に有効にする必要があります。

たとえば、non-trapping float-to-int conversions[240] はsign-extension operatorsと非常によく似た精神を持っています。コード サイズを多少節約するために数値型の組み込み変換も提供していますが、つい最近Safari15のリリースで統一的にサポートされました。そのため、この機能はまだデフォルトでは有効になっていません。ほとんどの開発者は、非常に説得力のある理由なしに、異なるブラウザに異なるバージョンのWebAssemblyモジュールを構築して出荷する複雑さを望んでいません。その結果、データセット内のWasmモジュールのどれもが、これらの変換を使用していませんでした。

複数値、参照型、末尾呼び出しなど、使用実績がゼロの機能も同様の状況です。これらはほとんどのWebAssemblyユースケースに恩恵をもたらす可能性がありますが、コンパイラやエンジンのサポートが不完全なためです。

残り、使用されている機能の中で、とくに興味深いのはSIMDとatomicです。どちらも、さまざまなレベルで実行を並列化し、高速化するための命令を提供します。SIMD[241] により、一度に複数の値に対して数学演算を行うことができ、atomicにより Wasmにおけるマルチスレッド[242]の基礎が提供されます。それらの機能はデフォルトでは有効ではなく、特定のユースケースを必要とし、とくにマルチスレッドはソースコードで特別なAPIを使用する必要があり、さらにWebサイトで使用する前 cross-origin isolated[243] にするための追加設定も必要です。そのため、比較的低い利用レベルであることは当然ですが、時間の経過とともに成長していくことが期待されます。

結論

WebAssemblyは比較的新しく、Web上ではややニッチな参加者ですが、単純なライブラリから大規模なアプリケーションまで、さまざまなWebサイトやユースケースで採用されているのは素晴らしいことです。

実際、WebAssemblyはWebのエコシステムに非常によく統合されており、多くのWebサイトのオーナーは、WebAssemblyをすでに使っていることにさえ気づかないかもしれませんし、彼らには他のサードパーティのJavaScript依存のように見えます。

提供サイズに改善の余地があり、さらなる分析により、コンパイラやサーバーの設定を変更することで達成可能なようです。また、エンジン、ツール、CDN開発者がWebAssemblyの利用を理解し、規模に応じて最適化するのに役立つかもしれない、いくつかの興味深い統計

240. *https://github.com/WebAssembly/nontrapping-float-to-int-conversions/blob/master/proposals/nontrapping-float-to-int-conversion/Overview.md*
241. *https://v8.dev/features/simd*
242. *https://web.dev/webassembly-threads/*
243. *https://web.dev/i18n/ja/coop-coep/*

や事例を発見しています。

Web Almanacの次版では、これらの統計を長期にわたって追跡し、最新情報をお届けする予定です。

著者

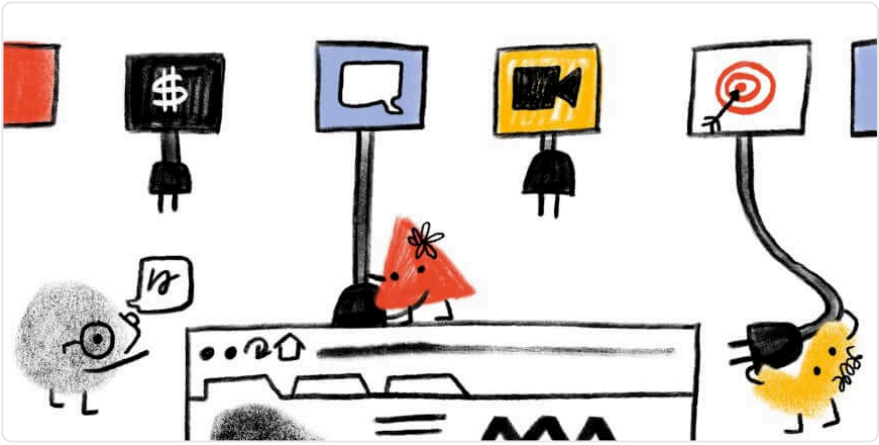


Ingvar Stepanyan

@RReverser RReverser https://rreverser.com/

Ingvarは、より良いツール、仕様、ドキュメントを通じて開発者の体験を向上させることに常に取り組んでいる情熱的なD2D（開発者間）プログラマーです。現在、Google Chrome チームで WebAssembly Developer Advocate として働いています。

部 I 章 7
サードパーティ



Barry Pollard によって書かれた。
Patrick Hulce、 Andy Davies、 Simon Hearne と Harry Roberts によってレビュ。
Barry Pollard による分析。
Rick Viscomi 編集。
Sakae Kotaro によって翻訳された。

序章

ああ、サードパーティはウェブ上の多くの問題の解決策であり。。。他の多くの問題の原因でもあります。基本的に、ウェブは常に相互接続と共有のためにあります。サードパーティのコンテンツをウェブサイトで使用することは、その自然な延長であり、HTML 2.0の `<img>` 要素の導入ではじめて動き出したものである。それ以来、外部のコンテンツを直接文書にハイパーリンクすることができるようになりました。これは、CSS や JavaScript の導入により、一見単純な `<link>` や `<script>` 要素を含むだけでページの一部（または全部！）を完全に変更することができるようになり、さらに発展しています。

サードパーティは、画像、ビデオ、フォント、ツール、ライブラリ、ウィジェット、トラッカー、広告、その他あなたが想像できるあらゆるものを、私たちのウェブページへ埋め込むために、尽きることのないコレクションを提供しています。これにより、技術者でない人でもコンテンツを作成し、ウェブに公開することができるようになりました。サードパーティがいなければ、ウェブは今日の私たちの生活に欠かせないリッチで没入感のある複雑なプラ

ットフォームではなく、非常に退屈な、テキストベースの学術的メディアになる可能性が高いでしょう。

しかし、ウェブ上でサードパーティーのコンテンツを利用することには、暗黒面があります。画像や便利なライブラリを無邪気に取り入れると、多くの開発者が十分に考慮しない、あらゆる種類のパフォーマンス、プライバシー、セキュリティへの門が開かれます。そのような業界のプロフェッショナルに話を聞くと、彼らはサードパーティのコンテンツの使用が生活をより困難にしていることを嘆きます。GoogleのCore Web Vitals initiative[244] によってパフォーマンスがとくに注目され、政府や個人によるプライバシーへの注目が高まり、Webに固有の悪意のある脆弱性の脅威が増え続ける中、精査は確実に進むと思われます。

この章では、ウェブにおけるサードパーティの状況について見ていきます。サードパーティをどれくらい使っているのか、何のために使っているのか、そしてとくに上記3つの懸念事項を考慮すると、昨年から使い方に変化はあったのでしょうか。これらの疑問に答えたいと思います。

定義

何をもって「サードパーティ」とするか、「サードパーティのコンテンツを利用」とするかについては、それぞれ考え方が異なるでしょうから、この章ではまず「サードパーティとは何か」という定義から説明します。

“サードパーティ”

サードパーティの定義は、2019[245]、2020[246]編と同じものを使用していますが、少し異なる解釈をすると、次節で述べるように今年はあるカテゴリーを除外することになります。

サードパーティとは、主たるサイト利用者関係の外にある存在、すなわちサイト所有者の直接の管理下にないが、その承認を得て存在する側面のことである。たとえば、Google Analyticsのスクリプトは、一般的なサードパーティリソースの一例です。

サードパーティのリソースは

- 共有・公開オリジンでのホスティング
- さまざまなサイトで広く利用されている
- 個々のサイトオーナーに影響されない

244. *https://web.dev/i18n/ja/vitals/*
245. *https://almanac.httparchive.org/ja/2019/third-parties*
246. *https://almanac.httparchive.org/ja/2020/third-parties*

これらの目標にできるだけ近づけるため、この章で使用するサードパーティリソースの正式な定義は、HTTP Archiveデータセット内の少なくとも50のユニークなページでそのリソースが見つかるドメインから発信されるものです。

これらの定義を使用すると、ファーストパーティのドメインから提供されるサードパーティのコンテンツは、ファーストパーティのコンテンツとしてカウントされることに留意してください。たとえば、セルフホスティングのGoogle Fontsや `bootstrap.css` は、*ファーストパーティのコンテンツ*としてカウントされます。

同様にサードパーティのドメインから提供されたファーストパーティコンテンツは、たとえリソース自体がそのウェブサイトに固有のものであっても、「50ページ以上の基準」をクリアしていれば、サードパーティのコンテンツとしてカウントされます（ドメインによってはそうなる可能性があります）。たとえば、サードパーティのドメインでCDNを介して提供されるファーストパーティの画像は、*サードパーティ*とみなされます。

サードパーティーのカテゴリー

今年も、サードパーティを特定し分類するために、Patrick Hulce[247] による third-party-web[248] リポジトリを大いに利用する予定です。このリポジトリでは、よく使われるサードパーティのURLを以下のように分類しています。

- **Ad** - これらのスクリプトは、広告ネットワークの一部であり、配信または測定を行っています。
- **Analytics** - ユーザーとその行動を測定または追跡するスクリプトです。何を追跡するかによって、さまざまな影響があります。
- **CDN** - これらは、一般にホストされているオープンソースライブラリ（例：jQuery）が、さまざまなパブリックCDNやプライベートCDNで提供されているものです。
- **Content** - これらのスクリプトは、コンテンツプロバイダーや出版社特有のアフィリエイトトラッキングによるものです。
- **Customer Success** - これらのスクリプトは、チャットやコンタクトソリューションを提供するカスタマーサポート/マーケティングプロバイダーからのものです。これらのスクリプトは、一般的に重いスクリプトです。
- **Hosting** - これらのスクリプトは、ウェブホスティングプラットフォーム（WordPress、Wix、Squarespaceなど）のものです。

247. *https://x.com/patrickhulce*
248. *https://github.com/patrickhulce/third-party-web/blob/master/data/entities.js*

- **Marketing** - これらのスクリプトは、ポップアップ/ニュースレターなどを追加するマーケティングツールのものです。
- **Social** - ソーシャル機能を実現するスクリプトです。
- **Tag Manager** - これらのスクリプトは、他の多くのスクリプトをロードし、多くのタスクを開始させる傾向があります。
- **Utility** - これらのスクリプトは、開発者向けのユーティリティ（APIクライアント、サイト監視、不正検出など）です。
- **Video** - ビデオプレーヤーやストリーミング機能を実現するスクリプトです。
- **Other** - これらは共有のオリジンを介して配信される雑多なスクリプトで、正確なカテゴリや属性はありません。

注：ここでいうCDNのカテゴリには、公開CDNドメイン（bootstrapcdn.com、cdnjs.cloudflare.comなど）でリソースを提供するプロバイダが含まれ、単にCDN経由で提供されるリソースは含まれません。たとえば、Cloudflareをページの前に置いても、私たちの基準によるファーストパーティの指定には影響しません。

今年、私たちが行った方法の変更の1つは、ホスティングカテゴリを分析から除外することです。WordPress.comをブログに、Shopifyをeコマースプラットフォームに使用している場合、そのサイトによるこれらのドメインに対する他のリクエストは多くの点でこれらのプラットフォームのホスティングの一部であり、真の「サードパーティ」ではないとして無視することにしています。上記の注記と同様に、ページの前にあるCDNは「サードパーティ」とは見なしません。実際には、これは数字にほとんど影響を与えませんでしたが、上記の定義で「サードパーティ」と見なすべきものをより正確に反映し、また他の章がこの用語をどのように使用しているかにより近いと私たちは考えています。

注意事項

- ここで紹介するデータはすべて、非インタラクティブ、コールドロードに基づくものです。これらの値は、ユーザーとのインタラクションの後では、かなり違って見えるようになる可能性があります。
- ページはCookieが設定されていない米国内のサーバーからテストされているため、オプトイン後に要求されたサードパーティは含まれません。これはとくに、一般データ保護規則[249]やその他の類似の法律の適用範囲にある国でホストされ、主に提供されるページに影響します。

249. *https://ja.wikipedia.org/wiki/EU%E4%B8%80%E8%88%AC%E3%83%87%E3%83%BC%E3%82%BF%E4%BF%9D%E8%AD%B7%E8%A6%8F%E5%89%87*

- トップページのみテストしています。その他のページでは、サードパーティーの要件が、異なる場合があります。
- あまり使われていないサードパーティードメインの一部は、*unknown*カテゴリに分類されています。この分析の一環として、サードパーティウェブデータセットを改善するために、使用率の高いドメインのカテゴリを増やして提出しました。

方法論の詳細はこちらです。

普及率

では、サードパーティはどの程度利用されているのでしょうか。その答えは、「たくさん」です。



図7.1. 少なくとも1つのサードパーティリソースを使用しているモバイルサイトの割合。

なんと、モバイルサイトの94.4%、デスクトップサイトの94.1%が、少なくとも1つのサードパーティリソースを利用していることが判明しました。サードパーティの定義を新しくしたとはいえ、これはWeb Almanacがスタートした2019[250]当時から継続的に成長していることを表しています。

250. *https://almanac.httparchive.org/ja/2019/third-parties*

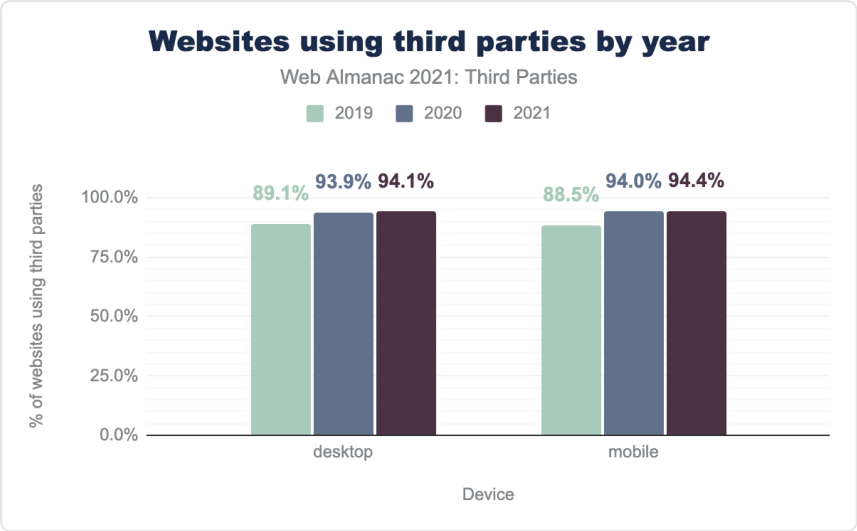


図7.2. サードパーティを利用しているウェブサイト（年別）

過去3回のWeb Almanacのデータセットを、より厳格な新しい定義で再実行したところ、上のグラフのように、当社のウェブサイトにおけるサードパーティの利用は、デスクトップで0.2%、モバイルでは0.4%と昨年よりわずかに増加していることがわかりました。

45.9%

図7.3. サードパーティからのリクエストの割合。

モバイルで45.9%、デスクトップでは45.1%がサードパーティからのリクエストで、これは昨年の結果[251]とほぼ同じです。

GDPR[252] や CCPA[253] のようなプライバシー保護のための規制は、第三者の使用に対する私たちの意欲を減退させていないように思われます。ただし、私たちの方法論は、米国のデータセンターからウェブサイトをテストしているので、そのために異なるコンテンツが提供されるかもしれないことを忘れてはなりません。

では、ほぼすべてのサイトがサードパーティを利用していることはわかったが、どれくらいの数を利用しているのだろうか。

251. *https://almanac.httparchive.org/ja/2020/third-parties*
252. *https://ja.wikipedia.org/wiki/EU%E4%B8%80%E8%88%AC%E3%83%87%E3%83%BC%E3%82%BF%E4%BF%9D%E8%AD%B7%E8%A6%8F%E5%89%87*
253. *https://en.wikipedia.org/wiki/California_Consumer_Privacy_Act*

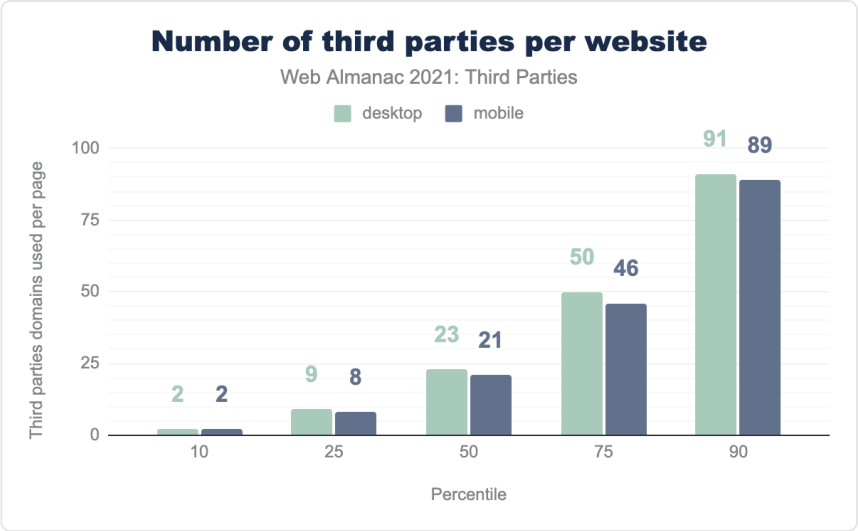


図7.4. ウェブサイト毎のサードパーティの数。

サードパーティーのホストネームの数で示されるサードパーティーを2つしか使用していないウェブサイトが10パーセンタイルにあり、90パーセンタイルでは89または91までと、大きなばらつきがあります。

なお90パーセンタイルは、デスクトップとモバイルでそれぞれ104と106だった昨年の分析[254]から少し下がっていますが、これは昨年の統計で行わなかった、今年50ウェブサイト以上で使用されている資産にドメインを限定したことに起因しているようです。

中央値のウェブサイトは、モバイルで21、デスクトップで23のサードパーティを使用しており、これでもかなり多いようです。

254. *https://almanac.httparchive.org/ja/2020/third-parties#fig-2*

ランク別サードパーティ普及率

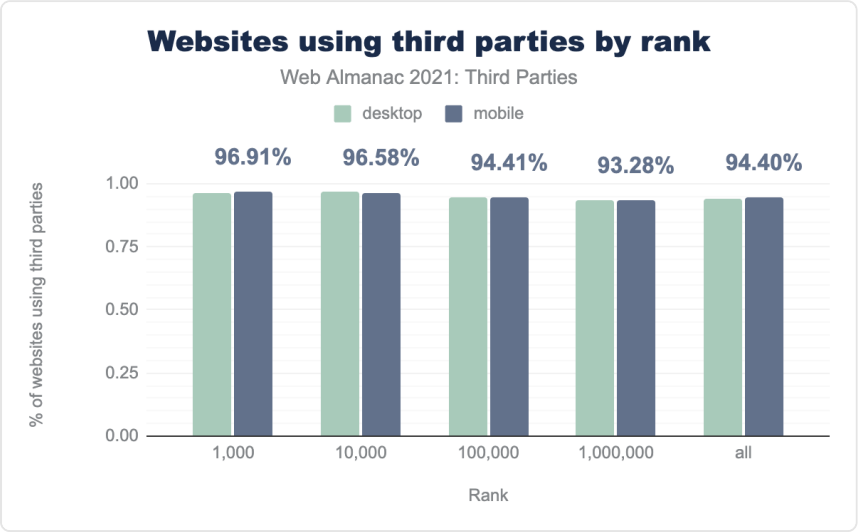


図7.5. ランク別にサードパーティを利用したウェブサイトを紹介。

今年は、各ウェブサイトのChrome UX Report (CrUX) “rank”[255] へアクセスできるようになりました。これは各サイトの人気度を割り出すもので、これにより、もっとも利用されている上位1,000サイト（ページビューベース）、上位10,000サイトなどにデータを分類できます。この人気順位でデータをスライスすると、人気のないウェブサイトではサードパーティの利用率がわずかに減少していますが、93.3%を下回ることはなく、かなり多くのウェブサイトが少なくとも1つのサードパーティを含めることが好きであることを再確認できます。

しかし、変わるのは、ウェブサイトが利用するサードパーティの数です。

255. https://developers.google.com/web/updates/2021/03/crux-rank-magnitude

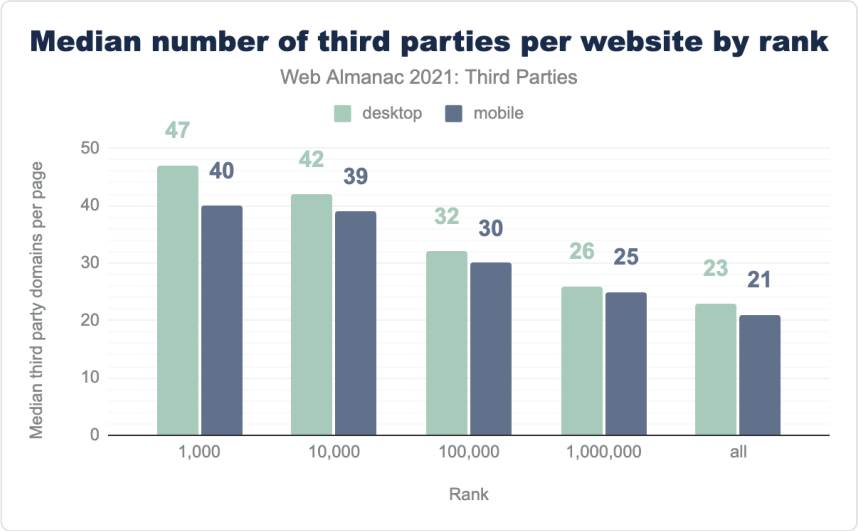


図7.6. ウェブサイトあたりのサードパーティ数の順位別中央値。

中央値（50パーセンタイル）の統計を見ると、ランキングが上がるにつれて著しく低下しており、もっとも人気のあるウェブサイトは、データセット全体の2倍の数のサードパーティを使用していることがわかります。これは、ほぼ完全に広告によってもたらされていることがおわかりいただけると思います。これは、収益化する目玉が増える、より人気のあるウェブサイトでより一般的であることは、おそらく驚くべきことではないでしょう。

サードパーティーの種類

私たちの分析によると、私たちはサードパーティーをたくさん使っているようですが、何のために使っているのでしょうか？各サードパーティーのリクエストのカテゴリーを見てみると、以下のようになります。

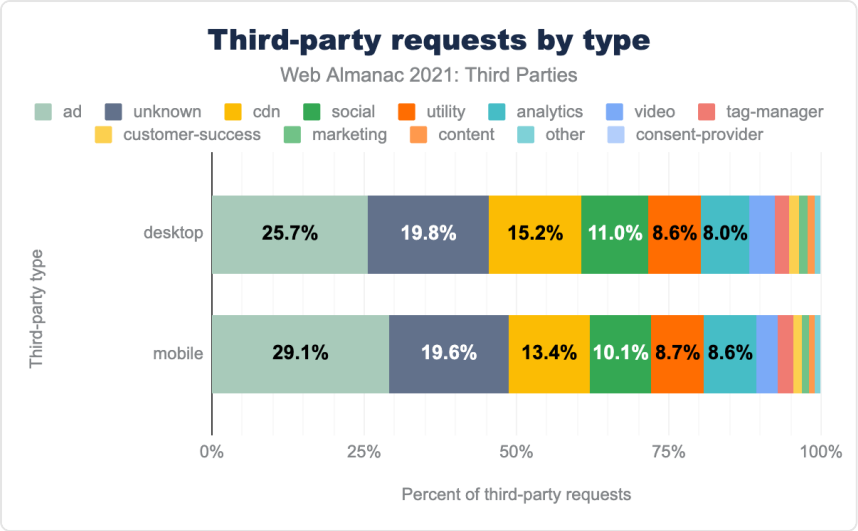


図7.7. サードパーティリクエストの種類別内訳

adがもっとも多く、次いで「unknown」（分類されていない、またはあまり利用されていないさまざまなサイトの集合体）、そしてCDN、social、utility、analyticsという順になっています。このように、あるカテゴリが他のカテゴリよりも人気がある一方で、サードパーティの利用がいかに多様であるかということが、おそらくここから得られる大きな収穫です。1つや2つのユースケースがすべてを支配するのではなく、本当にさまざまな理由で使われているのです。

サードパーティリクエストの種類と順位

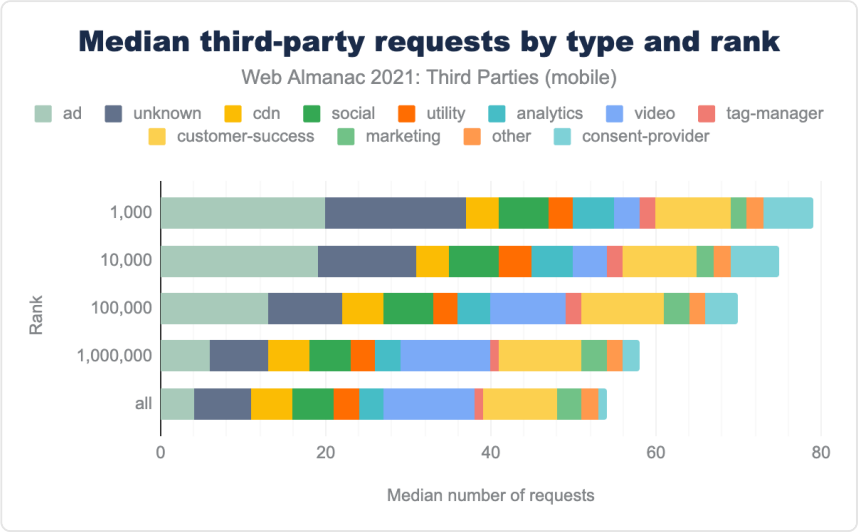


図7.8. サードパーティーのリクエストの種類と順位別の中央値。

順位別、カテゴリー別に分けてみると、先に述べたリクエスト数の多さの理由が見えてきます。人気のあるサイトでは、広告がより多く利用されてます。

このグラフは、各カテゴリーの順位別リクエスト数の中央値を示していますが、すべてのページですべてのカテゴリーが使用されているわけではありません。そのため、順位ごとの合計が、前のグラフの順位別リクエスト数の中央値よりもずっと高くなっているのです。

コンテンツの種類

このデータをもとに、サードパーティからのリクエストからどのようなコンテンツが返ってくるかを見てみましょう。

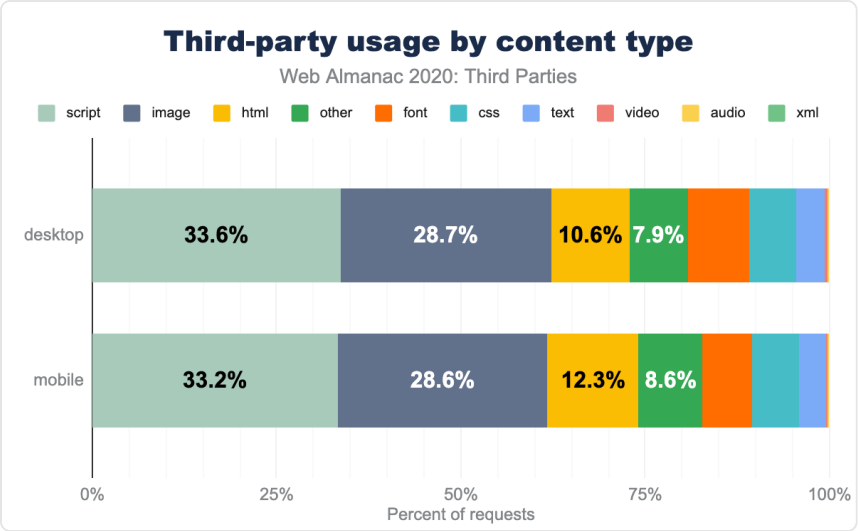


図7.9. コンテンツタイプ別のサードパーティ利用状況。

当然のことながら、JavaScript、images、HTMLがサードパーティからのリクエストの大部分を占めています。JavaScriptは、広告、トラッカー、ライブラリなど、機能を追加するためにほとんどのサードパーティによって使用されます。同様に、トラッキングソリューションで好まれる1ピクセルの空白画像も含まれるため、画像の使用頻度が高いことも予想されます。

HTMLの使用率が高いことは、最初は意外に思われるかもしれません（確かにドキュメントはHTMLの一般的な形式であり、それらはファーストパーティのリクエストでしょう）。しかし調査の結果、ほとんどiframeであることがわかりました。これは、広告やその他のウィジェットを格納するために使用されることが多いので、より理にかなっています（たとえば、YouTubeは動画そのものではなくプレーヤーを含むiframeでHTMLドキュメントを提供します）。

だから、純粋にリクエストの数で判断します。サードパーティは、コンテンツよりも機能を追加しているように見えますが、それは少し誤解があります。YouTubeの例のように、一部のサードパーティはコンテンツを有効にするため機能を追加しています。

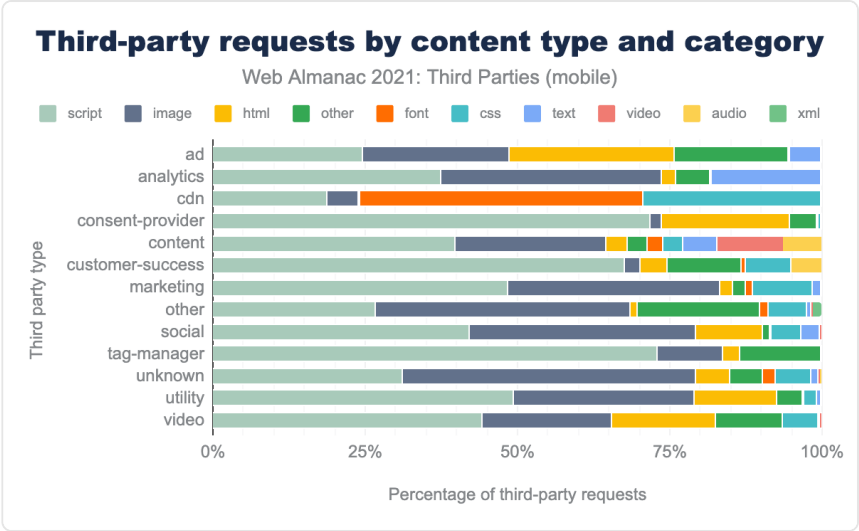


図7.10. コンテンツの種類およびカテゴリー別サードパーティからのリクエスト（モバイル）。

リクエストされたコンテンツの種類をサードパーティーの種類で分けると、ほとんどの種類でスクリプト、画像、HTMLの3種類が多く、JavaScriptの多いことがわかります（動画タイプでも！）。上の図はモバイルのものですが、デスクトップの様子もよく似ています。

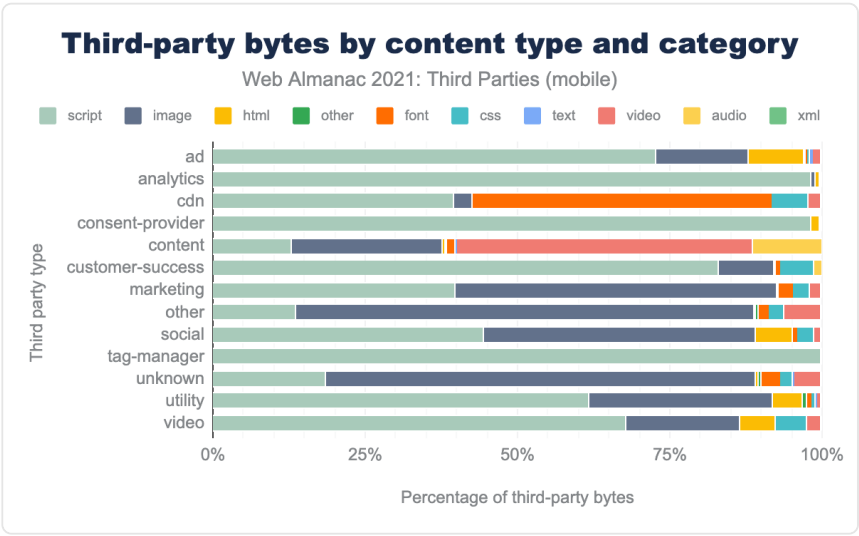


図7.11. コンテンツの種類とカテゴリー別のサードパーティリクエスト。

リクエスト数ではなく、バイト数で見ると、JavaScriptの多さがさらに気になります。ここでもモバイルを示しましたが、デスクトップの場合に大きな違いはありません。

Addy Osmani[256]（同じ文章で2回！）の“Cost of JavaScript”[257] 投稿を引用すると、「バイト単位でみるとJavaScriptは今でも私たちが送信するもっとも高価なリソース」、そして「200KBのスクリプトと200KBの画像にはまったく異なるコスト」だそうです。Analytics、Consent Provider、Tag Managerなどのカテゴリはかなり JavaScriptが多く、AdやCustomer Successなどのカテゴリも遠くおよびません。サードパーティーリソースの使用によるパフォーマンスへの影響については、JavaScriptの使用によるコスト高が、原因であることが多いので、また後日ご紹介します。

サードパーティードメイン

サードパーティからの要望はどこから来るのでしょうか？これらの名前のほとんどは驚くようなものでありませんが、ある名前の普及は、その会社が多くの異なるカテゴリーで優位に立っていることを改めて示すものです。

256. *https://x.com/addyosmani*
257. *https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4*

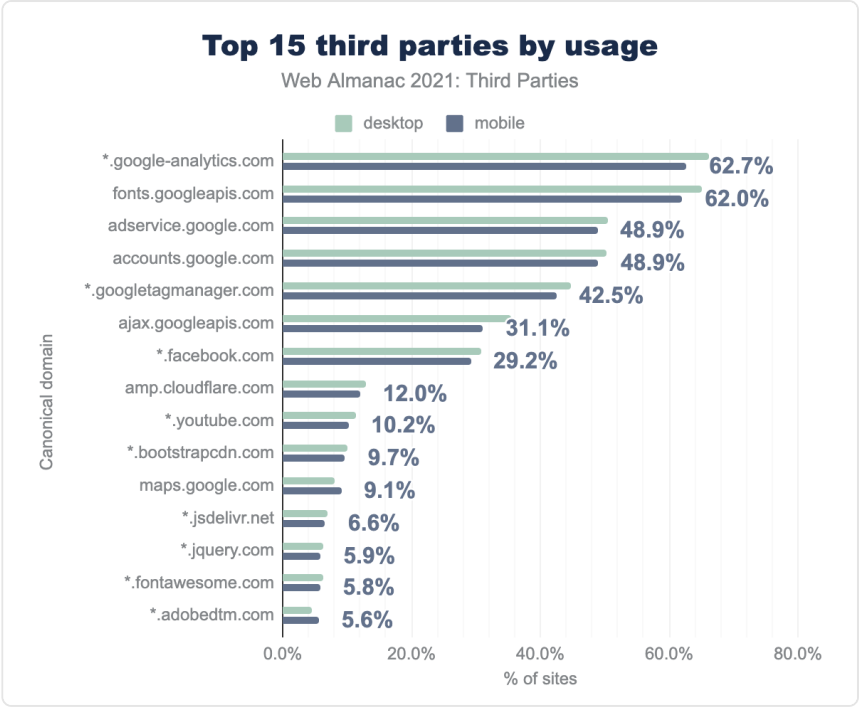


図7.12. サードパーティーの利用率上位15位。

もっとも利用されているサードパーティーの上位15位のうち8位（うち上位6位）をGoogleが占めており、他の追随を許さない状況です。Googleは、Analytics、フォント、広告、アカウント、タグマネージャー、ビデオなどの分野で市場をリードしています。モバイルサイトの62.7%がGoogle Analyticsを、ほぼ同数がGoogle Fontsを利用しており、広告、アカウント、タグマネージャーも42%から49%の範囲で利用されています。

非Googleの1位はFacebookで、29.2%と比較的低い利用率になっています。続いて、人気の高いライブラリなどを前面に押し出したCloudflareのCDNが続きます。amp.cloudflare.comと表示されていますが、より大きなcdnjs.cloudflare.comも含まれており、来年はよりよく使われるドメインを表示するように更新されました。

この後、YouTubeでGoogleに戻り、Mapsはその2つ後のスポットとなります。残りのスポットは、他の一般的なライブラリやツールのCDNで満たされています。

サードパーティによるパフォーマンスへの影響

サードパーティを使用すると、パフォーマンスに顕著な影響を与えることがあります。それ

は、必ずしもサードパーティーであること自体が原因ではありません。サイトオーナーがファーストパーティーリソースとして実装したのと同じ機能でも、サードパーティーが持つ特定の分野の専門知識を考慮すると、パフォーマンスが低下することはよくあるのです。

ですから必ずしもリソースが、サードパーティであることがパフォーマンスに影響するわけではなく、むしろそのリソースが何をしているかが問題なのです。また、サードパーティーの利用は、単にサービスを提供する場所としてではなく、サードパーティーのサービスに依存することがほとんどです。

しかし、第三者のビジネスは、そのコンテンツやサービスが多くのウェブサイトでホストされることを許可することにあります。サードパーティは、その依存による悪影響を最小限に抑えることを保証する義務があるのです。サイト所有者は、サードパーティを利用するかしないか以外に、サードパーティのパフォーマンスへの影響に対するコントロールや影響力が限られていることが多いため、これはとくに重要な義務であると言えます。

サードパーティードメインの使用とセルフホスティングの比較

ほとんどのサードパーティがグローバルに分散した高性能のCDNを使用しているにもかかわらず、他のドメインに接続するには明確なコストがかかります。多くのウェブパフォーマンス支持者（この著者を含む！）は、このペナルティを避けるために可能な限りセルフホスティングを推奨しています。これは、すべての主要なブラウザがオリジン間でキャッシュを共有しないようになった今、とくに重要なことです。したがって、あるサイトがそのリソースをダウンロードしたら、訪問した他のサイトもその恩恵を受けることができるという主張は、もはや真実ではありません。とはいえ、ライブラリのバージョン数やHTTPキャッシュの制限を考えると、昔もこの主張には疑問があったのですが。

とはいえ、思うようにいかないのが現実で、場合によっては、セルフホストはかえってパフォーマンスを低下させることもあります。筆者は以前、Googleフォントをセルフホスティングするかどうか[258]という問題は、見た目ほど明確ではなく、パフォーマンス面でGoogleフォントが行っていることをすべて再現するためには、ある程度の専門知識が必要だということを書きました。その手間を省くには、ホストされたバージョンを使用すればよく、Harry Roberts[259] が The Fastest Google Fonts[260] という投稿で述べたように、パフォーマンスの影響をできる限り軽減していることを確認できます。

同様に画像CDNはほとんどのファーストパーティよりもメディアを最適化することができ、さらに重要なことは必然的に省略されたり、時に間違って行われたりする手動ステップを必要とせずに、自動的にこれを行うことができます。

258. *https://www.tunetheweb.com/blog/should-you-self-host-google-fonts/*
259. *https://x.com/csswizardry*
260. *https://csswizardry.com/2020/05/the-fastest-google-fonts/*

人気のサードパーティの埋め込みとそのパフォーマンスへの影響

サードパーティーのパフォーマンスへの影響を理解するために、もっとも人気のあるサードパーティーの埋め込みをいくつか見ていきます。これらの中には、Webパフォーマンス界で悪名高いものもあるので、その悪評が本当にふさわしいかどうか見てみましょう。そのために、2つのLighthouse監査を活用することにします。レンダーブロッキングリソースの排除[261]と サードパーティーコードの影響を軽減[262]を基にしたHoussein Djirdeh[263]による同様の研究[264]があります。

人気のあるサードパーティとレンダーへの影響

サードパーティがレンダリングに与える影響を理解するために、Lighthouseのレンダーブロッキングリソース監査でサイトリソースのパフォーマンスを分析し、third-party-web データセットと相互参照することでサードパーティを識別しています。

261. *https://web.dev/render-blocking-resources/*
262. *https://web.dev/third-party-summary/*
263. *https://x.com/hdjirdeh*
264. *https://docs.google.com/spreadsheets/d/1Td-4qFjuBzxp8af_if5iBC0Lkqm_OROb7_2OcbxrU_g/edit?usp=sharing&resourcekey=0-ZCfve5cngWxF0-sv5pLRzg*

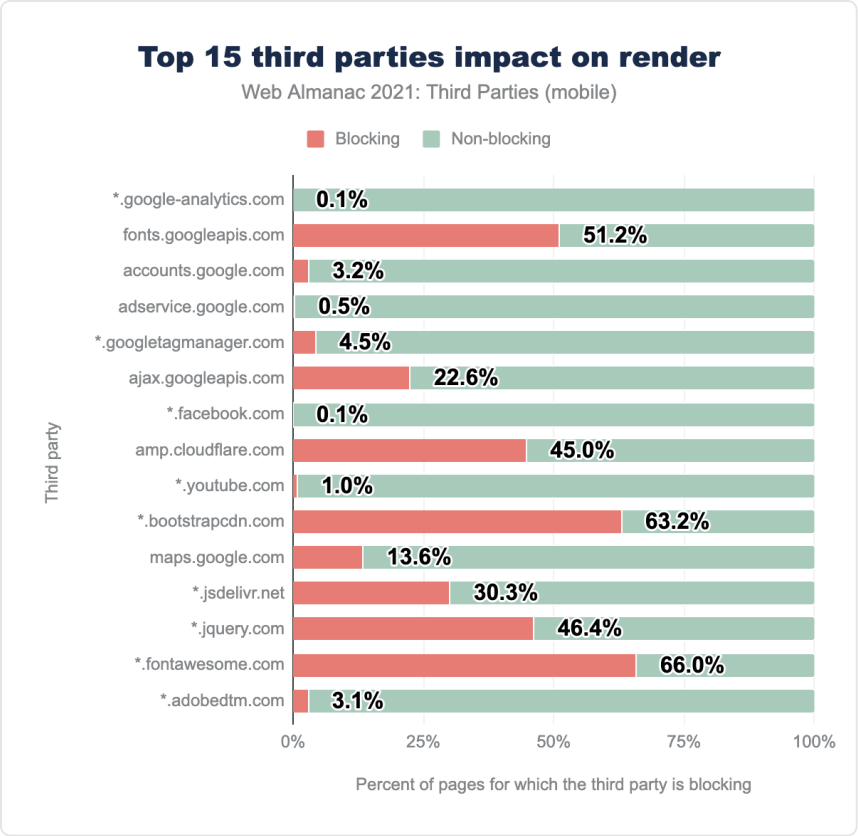


図7.13. レンダーに影響を与えるサードパーティーの上位15位。

もっとも人気のあるサードパーティーの上位15位は、ページの初期レンダリング時にブロックされるリソースの割合とともに、上に表示されています。

全体として、これは肯定的な話です。ほとんどはレンダリングをブロックせず、ブロックするのはレイアウトに関連する一般的なライブラリ (例: bootstrap) や、おそらく最初のレンダリングをブロックすべきフォント (この作者は `font-display: swap` や `optional` を使うことが良いことだとは思っていない) のためです。

多くの場合、サードパーティの組み込みはレンダリングのブロックを避けるために `async` や `defer` を使用するように助言していますし、その多くがそうであるように見えます。

人気サードパーティと本スレッドへの影響

Lighthouseには サードパーティコードの影響軽減[265] 監査があり、すべてのサード パーティ製リソースのメイン スレッドの時間が一覧表示されます。では、もっとも人気のあるものはどれくらいの時間メイン スレッドをブロックしているのでしょうか？

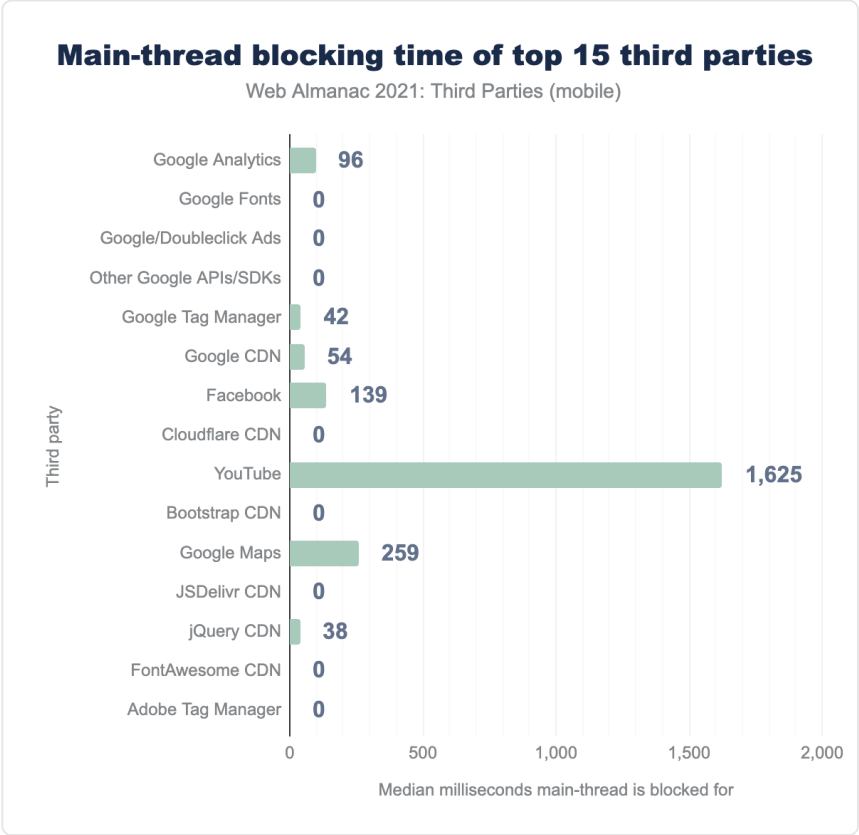


図7.14. サードパーティーの上位15位のメインスレッドブロック時間。

ここでは、YouTubeが突出しているので、もう少し掘り下げてみましょう。

265. *https://web.dev/third-party-summary/*

YouTube

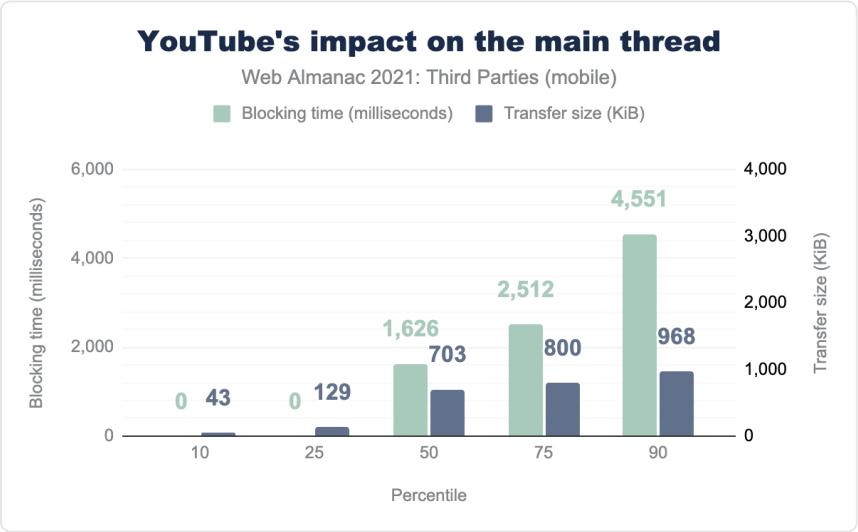


図7.15. YouTubeが本スレッドに与える影響。

中央値（50パーセンタイル）では、メインスレッド活動が1.6秒、90パーセンタイルでは4.6秒と、大きな影響があります（それでも10%のウェブサイトは、これよりも悪い影響を受けているということです！）。しかし、これはスロットルで調整された、実験室でシミュレートされたタイミングであるため、実際のユーザーの多くはこのレベルの影響を経験していないかもしれませんが、それでもかなりのものであることは覚えておく必要があります。

また、転送サイズは大きくなるほど影響が大きくなることも明らかで、処理量が多いので当然かもしれません。また、クロールはこれらのビデオとインタラクションを行わないため、自動再生されるビデオか、YouTubeプレーヤー自体がこのような使用を引き起こしていることがわかります。

ここで、リストにある他のサードパーティの埋め込みについて、もう少し掘り下げてみましょう。

Google Analytics

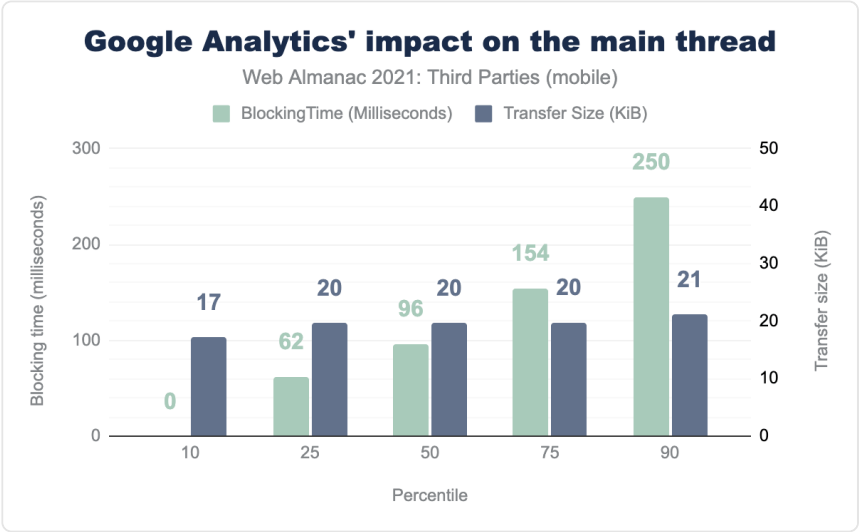


図7.16. Google Analyticsが本スレッドに与える影響。

Google Analyticsはかなり優秀なので、トラッキングのすべてを考慮すると、明らかに最適化に多くの労力が費やされています。

Google/Doubleclick広告

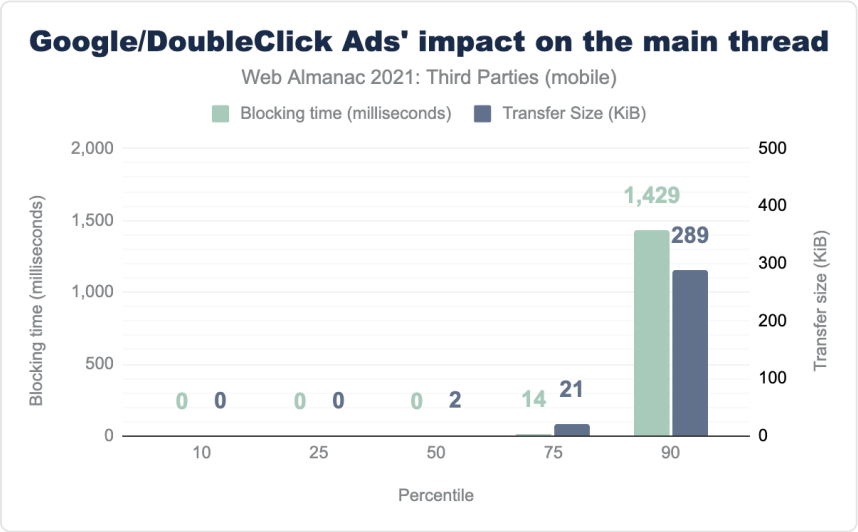


図7.17. Google/Doubleclick広告の本スレッドへの影響。

Google広告は、90パーセンタイルに達するまでは、とてもうまくいっていたのですが、チャートから吹き飛ばされてしまったのです。繰り返しますが、これは10%のウェブサイトがこれらより悪い数字であることを意味します。

Googleタグマネージャー

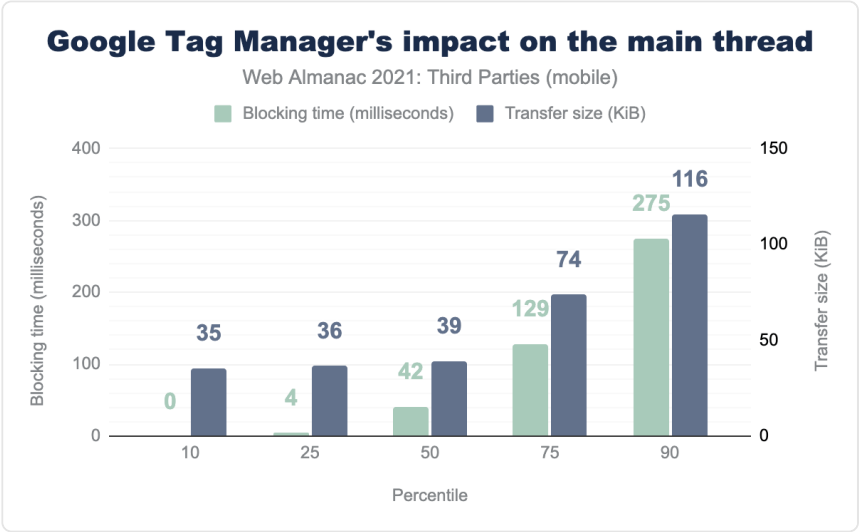


図7.18. Google Tag Managerが本スレッドに与える影響。

Googleタグマネージャーは、正直なところ、予想以上に良い結果を出しています。筆者は、もう使われない古いタグやトリガーで溢れかえった、恐ろしいGTMの実装をいくつか見てきました。しかし、GTMはテストページのロードでメインスレッドを長時間ブロックすることがなく、うまくいっているようです。

Facebook

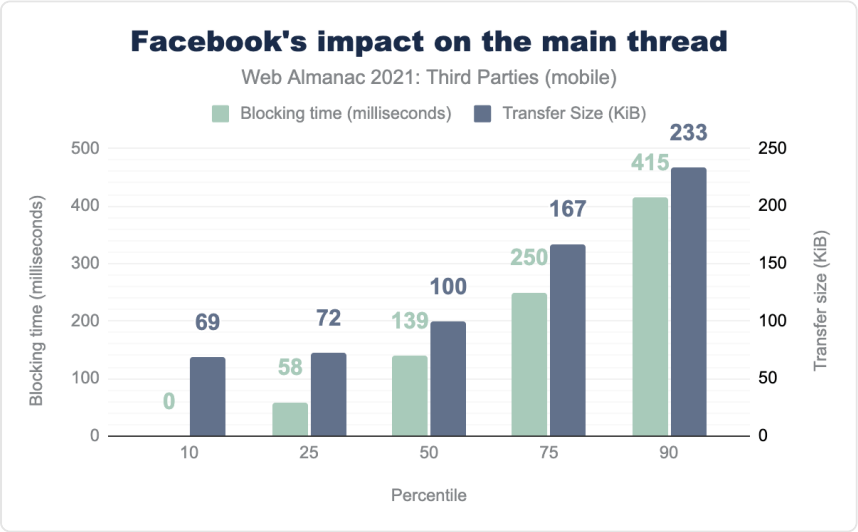


図7.19. Facebookが本スレッドに与える影響

Facebookも、思ったほどリソースを消費しません。Facebookの投稿の埋め込みは、Twitterの埋め込みよりも人気がないようなので、これらはFacebookのリターゲティング・トラッカーになる可能性が高いでしょう。これらのトラッカーはバックグラウンドで静かに動作し、メインスレッドにまったく影響を与えないはずです。したがって、Facebookがここで行うべきことはまだあることが明らかです。私は、Facebook JavaScript APIを使用せず、Googleタグマネージャー[266]を通じてピクセルトラッキングを使用しても、機能を失うことなく良い結果を得ているので、このオプションを検討することをオススメします。

266. https://www.tunetheweb.com/blog/adding-controls-to-google-tag-manager/#pixels

Googleマップ

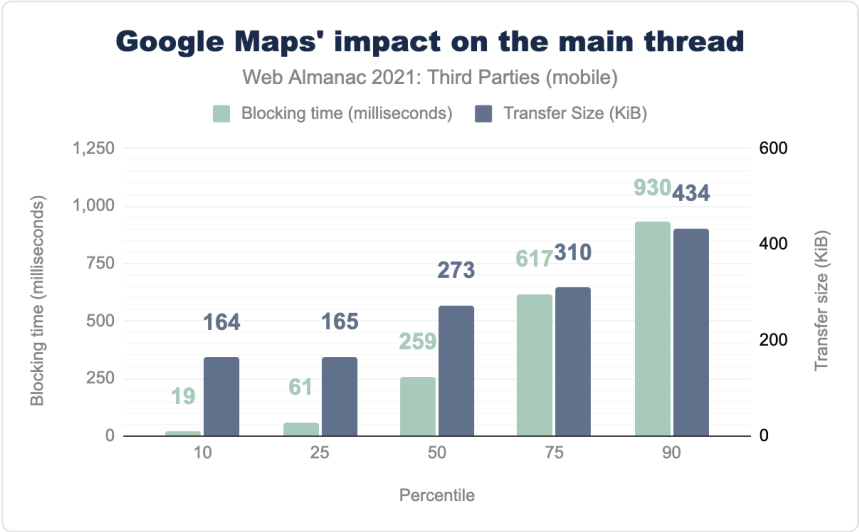


図7.20. Googleマップが本スレッドに与える影響。

Googleマップは、間違いなくいくつかの改善が必要です。とくに、メインコンテンツではなく、ページ上の小さな余分な部分として存在することが多いからです。ウェブサイトの所有者として、Googleマップのコードを必要なページにのみ含めることの重要性が浮き彫りになっています。

Twitter

そして最後に、さらに1つ下のリストを見てみましょう。Twitterです。

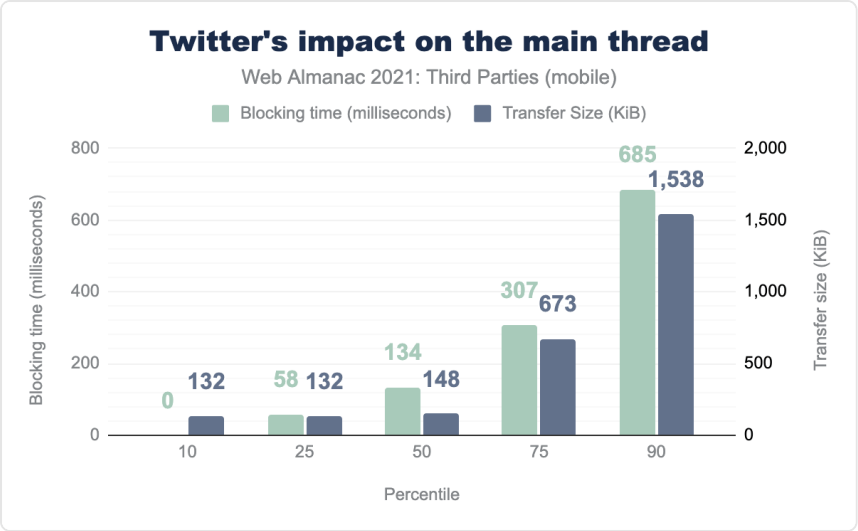


図7.21. Twitterが本スレッドに与える影響

サードパーティとしてのTwitterは、リターゲティング広告のトラッカーとして、またツイートを埋め込む方法として、2つの方法で利用できます。ページへのツイートの埋め込みは、他のソーシャルネットワークよりも人気があります。しかし、Matt Hobbs[267] の PuppeteerとSquooshを使って埋め込みツイートのWebパフォーマンスを修正する[268] という投稿を含め、ウェブパフォーマンスのコミュニティの多くからページに過度の影響を与えるとして指摘されています。とくに、上記のグラフにあるようなトラッキングのユースケース（おそらくより軽量なもの）は希薄になるため、我々の分析はそれを裏付けるものです。

上記の例の中には、良くも悪くもなるものがありますが、ウェブサイトのパフォーマンスに本当に影響を与えるのは、これらの累積効果であることを忘れてはなりません。地図と埋め込みTweetのあるページにGoogle Analytics、Facebook、Twitterのトラッキングをロードするを追加すると、膨大な量になります。スマホがときどき熱くて手に負えなくなったり、ネットサーフィンをしているだけでPCのファンが回りだしたりするのは、当然といえば当然ですよね。

これらのことから、Googleがドキュメントの順序付け、遅延ローディング、ファサード、その他のテクニックを使用して、埋め込みの影響を軽減する[269]（皮肉にもほとんどが自分たちのものです！）ことを推奨している理由がわかります。しかし、これらのうちのいくつかはデフォルトではなく、これらのような高度なテクニックはウェブサイト所有者の責任に帰する必要があることは、実に腹立たしいことです。ここで取り上げたサードパーティは、本当

267. https://x.com/TheRealNooshu
268. https://nooshu.com/blog/2021/02/06/using-puppeteer-and-squoosh-to-fix-twitter-embeds/
269. https://web.dev/embed-best-practices/

はリソースや技術的なノウハウを持っていて、デフォルトですべての人の製品使用による影響を減らすことができるのに、そうしないことを選択することが多いのです。このパフォーマンス編は、サードパーティを使うことが必ずしもパフォーマンスに悪いわけではない、というところから始まりましたが、これらの例を見ると確かにこの分野ではもっとできることがあるようです。

このような有名な例を取り上げることで、読者が自分のサイトにおけるサードパーティーの埋め込みの影響を調査し、本当にすべて価値があるのか自問するきっかけになればと思います。おそらく、このテーマをサードパーティにもっと重要視してもらえば、パフォーマンスを優先してくれるでしょう。

Timing-Allow-Originヘッダーの普及率

昨年は、Resource Timing API[270]をサードパーティのリクエストで使用できるようにする `timing-allow-origin` header の普及について調べました。このHTTPヘッダーがない場合、サードパーティからのリクエストに対してオンページ・パフォーマンス・モニタリング・ツールが利用できる情報は、セキュリティとプライバシー上の理由から制限されます。しかし、静的なリクエストについては、このヘッダーを許可するサードパーティは、そのリソースの読み込み性能についてより高い透明性を実現します。

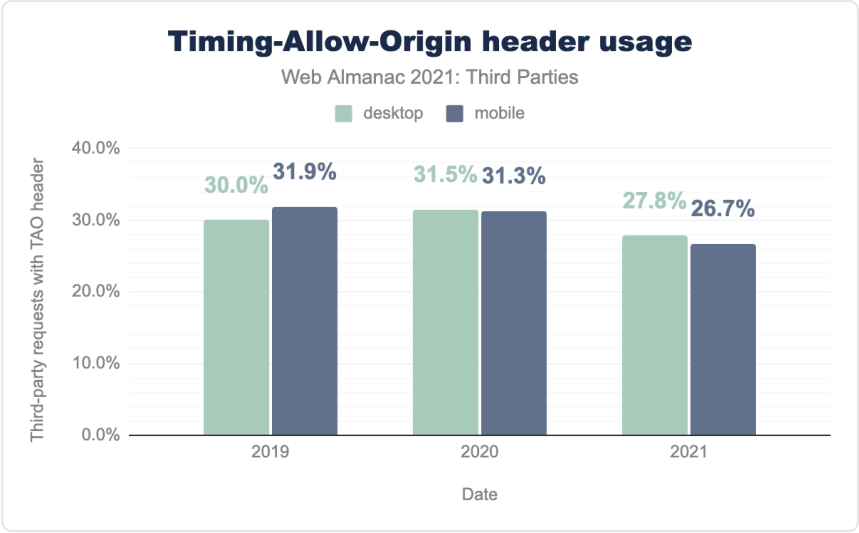


図7.22. Timing-Allow-Originヘッダーの使用状況。

過去3年間のWeb Almanacの利用状況を見ると、今年は利用状況がかなり落ち込んでいま

270. https://developer.mozilla.org/docs/Web/API/Resource_Timing_API/Using_the_Resource_Timing_API

す。データを深く掘り下げると、Facebookのリクエストは33%減少していることがわかりました。このヘッダーをサポートし、広く使用されていることを考えると、この減少の大部分はこれで説明できます。おもしろいことに、Facebookが使われているページの数は実際に増えています。しかし、Facebookは昨年、リクエストを少なくするように埋め込みを変更したようで、その普及率を考えると、`timing-allow-origin`ヘッダーの使用にかなりの打撃を与えているようです。それは、Core Web Vitalsのランキングへの影響[271]によりパフォーマンスに焦点が当てられていることを考えると、少し残念なことです。

セキュリティとプライバシー

サードパーティを利用することによるセキュリティやプライバシーへの影響を測定することは、より困難です。サードパーティにアクセスすることは、セキュリティとプライバシーの両面でリスクを増大させることは間違いなく、もっとも一般的であることが分かっているスクリプトを実行するためのアクセスを与えることは、実質的にウェブサイトへのフルアクセスを意味します。しかしサードパーティーのリソースは、サイト上でシームレスに使用できるようすることが目的であり、これを制限することは、サードパーティーの機能そのものを制限することになります。

セキュリティ

サイト自身は、さまざまな方法でサードパーティを利用するリスクを減らすことができます。`HttpOnly`属性でクッキーへのアクセスを制限し[272]、JavaScriptからアクセスできないようにし、`SameSite`属性を適切に使用します。これらについては、セキュリティの章で詳しく説明されていますので、ここではこれ以上掘り下げません。

サードパーティのリソースをより安全にするもう1つのセキュリティ機能は、Subresource Integrity[273] (SRI) の使用です。これは、リソースを読み込む`<link>`または`<script>`要素にリソースの暗号ハッシュを追加して有効にするものです。このハッシュをブラウザで確認することにより、ダウンロードされたコンテンツが期待通りのものであることを確認します。しかし、サードパーティーのリソースはさまざまな性質を持っているため、サードパーティーが意図的にリソースを更新した場合、サイトが壊れるなど、解決策よりもリスクが大きくなる可能性もあります。もし、本当に静的なコンテンツであれば、セルフホスティングが可能であり、SRIの必要性はない。このように多くの人がSRIを推奨していますが、筆者はSRIが、本当に推進派が主張するようなセキュリティ上の利点を提供しているのか、まだ確信が持てません。

サードパーティのリソースの使用や、ユーザーが作成したコンテンツなど、サードパーティ

271. *https://developers.google.com/search/blog/2020/11/timing-for-page-experience*
272. *https://developer.mozilla.org/docs/Web/HTTP/Cookies#restrict_access_to_cookies*
273. *https://developer.mozilla.org/docs/Web/Security/Subresource_Integrity*

のコンテンツがサイトに入ってくる際のセキュリティ リスクを減らすには、堅牢な Content Security Policy[274] (CSP) が最適な方法の1つでしょう。これは、オリジナルのウェブサイトとともに送信されるHTTPヘッダーで、どのようなリソースを誰が読み込むめるのか、またできないかを正確にブラウザに伝えます。セキュリティの章によると、使っているサイトは少ないという、より高度な技術でありCSPの使用状況の分析は彼らに任せるとして、ここで取り上げる価値があるのは普及しない理由の1つがサードパーティにあるのではないかということです。筆者の経験では、サードパーティを問題なく利用するためにサイトがポリシーに追加しなければならない要件を正確に記載したCSP情報を公開しているサードパーティはごくわずかです。さらに悪いことに、安全なCSPと互換性のないものもあります。一部のサードパーティは、インラインのスクリプト要素を使用したり、通知なしにドメインを変更したりするため、CSPを使用しているサイトでは、ポリシーを更新するまでその機能が壊れてしまいます。Google広告は、国ごとに異なるドメインの使用[275]により、CSPを本当にロックダウンすることを難しくしているもう1つの例です。

そもそも自分の管理下にあるサイトの部分についてCSPを設定することだけでも十分難しいのに、サードパーティーの複雑さが加わって、自分の管理外のことがさらに難しくなっているのです。サードパーティは、サイトがCSPを使用するリスクを軽減するために、CSPのサポートをより良くする必要があります。

プライバシー

サードパーティを利用することによるプライバシーへの影響については、改めてこのトピックに特化したプライバシーの章に譲るとして、上記の分析からすでに明らかなのは、Webユーザーのプライバシーに大きな影響を与える次の2点である。

- Webサイトにおけるサードパーティーの利用率は95%弱。
- GoogleやFacebookなど、プライバシーの味方とは言い難い特定のサードパーティが優勢であること。

もちろん、あなたのサイトでサードパーティを利用する主な理由の1つは、広告目的のトラッキングであり、その性質上、あなたの訪問者の最高のプライバシー利益になるとは思えません。基本的にサードパーティの利用によってのみ可能となるこの広範なトラッキングに対する代替案は、Googleのプライバシー サンドボックスやFLoCイニシアチブ[276]など提案されていますが、これまでのところ幅広いエコシステムに十分な牽引力を与えるには至っていません。

さらに問題なのは、ウェブサイトの利用者や所有者が気づかないうちに、トラッキングが行われていることでしょう。製品やサービスに対してお金を払っていないなら、あなたが製品

274. https://developer.mozilla.org/docs/Web/HTTP/CSP
275. https://stackoverflow.com/questions/34361383/google-adwords-csp-content-security-policy-img-src
276. https://blog.google/products/ads-commerce/2021-01-privacy-sandbox/

であるという古い格言があります。多くのサードパーティは、製品を「無料」で提供していますが、これはほとんどの場合、他の方法で収益化していることを意味します。通常、訪問者のプライバシーを犠牲にしています

`feature-policy` や `permission-policy` といった新しい技術を採用すると、マイクやビデオカメラといったブラウザの特定の機能の使用を制限できます。これらの多くは、通常、ブラウザのプロンプトの背後に保護されているため、黙って起動されることはありませんが、プライバシーとセキュリティのリスクを軽減できます。Googleはまた、ウェブブラウザのプライバシーへの影響を制限するためのプライバシー予算案[277]に取り組んでいますが、他の人々は この分野での彼らの仕事に対して懐疑的なまま[278]なのです。全体として、多くのサードパーティリソースの意図を考えると、プライバシーコントロールを追加することは、流れに逆らっているように見えます。

結論

サードパーティはウェブに不可欠な存在です。サードパーティの普及がなければ、ウェブサイトを構築するのは難しく、機能も充実していないでしょう。冒頭で述べたように、ウェブの核心は相互接続性であり、サードパーティはその自然な延長線上にあるのです。私たちの分析によると、サードパーティはかつてないほど普及しており、サードパーティのないサイトは非常に例外的なのです。

しかし、サードパーティーの利用はリスクがないわけではありません。本章では、サードパーティーがパフォーマンスに与える影響を調査し、サードパーティーをサイトで利用する際に起こりうるセキュリティとプライバシーのリスクについて説明しました。

サードパーティのツール、ウィジェット、トラッカーなど、思いつく限りのものでウェブサイトを不必要に満載することには、結果が伴います。サイトの所有者は、すべてのサードパーティコンテンツの影響を調べ、その潜在的な影響に見合う機能性があるかどうかを判断する責任があります。

しかし、ネガティブなことばかりに目を奪われがちなので、この章の最後にポジティブなことを振り返ってみましょう。サードパーティがこれほど普及しているのには理由があり、彼らは（たいてい！）選択の余地なく使っているのです。共有することがウェブの本質なので、サードパーティはウェブの精神にとても合っているのです。私たちウェブデベロッパーが自由に使える機能、そしてそれをサイトに追加することがいかに簡単であるかは、驚くべきことです。この章を読んで、あなたがそのような取引をするときに、十分に理解しているかどうか、もう少し考えるようになることを願っています。

277. *https://github.com/bslassey/privacy-budget*
278. *https://blog.mozilla.org/en/mozilla/google-privacy-budget-analysis/*

著者



Barry Pollard

@tunetheweb @https://webperf.social/@tunetheweb @tunetheweb.com tunetheweb

tunetheweb https://www.tunetheweb.com

Barry Pollard はソフトウェア開発者であり、Munning の本 HTTP/2 in Action[279] の著者です。彼は、ウェブは素晴らしいが、それをさらに良くしたいと思っている。ツイッターは@tunetheweb 、ブログはwww.tunetheweb.com でご覧いただけます。

279. *https://www.manning.com/books/http2-in-action*

部 II 章 8
SEO



Patrick Stox、 Tomek Rudzki と Ian Lurie によって書かれた。
Fili Wiese、 Rob Teitelman と Jamie Indigo によってレビュ。
JR Oakes と Ruth Everett による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

SEO (Search Engine Optimization) は、検索エンジンのオーガニック検索結果からのトラフィックの量と質を高めるために、ウェブサイトやウェブページを最適化することです。

SEOはかつてないほど人気があり、企業が顧客にアプローチする新しい方法を模索する中で、ここ数年大きな成長を遂げています。SEOの人気は、他のデジタルチャネルをはるかにしのいでいます。

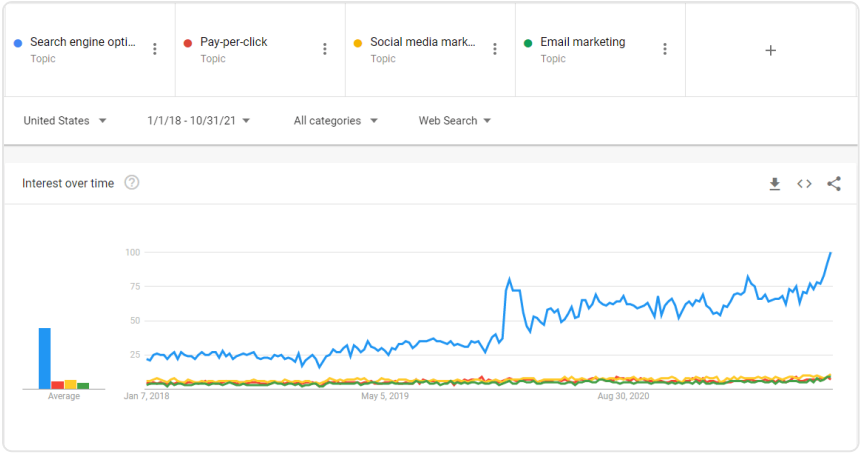


図8.1. Google TrendsによるSEOとペイパークリック、ソーシャルメディアマーケティング、Eメールマーケティングの比較。

Web AlmanacのSEOの章の目的は、Webサイトの最適化に関連するさまざまな要素を分析することです。この章では、Webサイトがユーザーと検索エンジンに優れた体験を提供しているかどうかをチェックします。

分析には、Lighthouse[280]、Chrome User Experience Report (CrUX)[281]、モバイルとデスクトップのHTTP Archive[282]から得られる生とレンダリングのHTML要素を含む多くのデータが使用されました。HTTP ArchiveとLighthouseの場合、サイト全体のクロールではなく、ウェブサイトのホームページのみから特定されるデータに限定されます。このことを念頭に置いて、私たちの結果から結論を出してください。分析の詳細については、方法論のページでご覧いただけます。

Webの現状と検索エンジンの利便性については、こちらをご覧ください。

クロール性とインデックス性

このようなユーザーの問い合わせに対して適切な結果を返すために、検索エンジンはウェブの索引を作成する必要がある。そのためのプロセスには

1. **クローリング** - 検索エンジンは、ウェブクローラー（スパイダー）を使ってインターネット上のページを巡回しています。スパイダーは、サイトマップやページ間のリンクなどの情報源から新しいページを見つけます。

280. https://developers.google.com/web/tools/lighthouse/
281. https://developers.google.com/web/tools/chrome-user-experience-report
282. https://httparchive.org/

2. **処理** - このステップでは、検索エンジンがページのコンテンツをレンダリングすることがあります。検索エンジンはインデックスを作成・更新し、ページをランク付けし、新しいコンテンツを発見するために使用するコンテンツやリンクなど、必要な情報を抽出します。
3. **インデックス作成** - コンテンツの品質と独自性に関する一定のインデックス作成要件を満たしたページは、通常、インデックスに登録されます。インデックスされたページは、ユーザーのクエリに対して返される資格があります。

ここでは、クローラビリティとインデックス作成に影響を与える可能性のある問題をいくつか見てみましょう。

`robots.txt`

`robots.txt` とは、ウェブサイトの各サブドメインのルートフォルダーに置かれるファイルで、検索エンジンのクローラーなどのロボットへどこに行ってよいか、どこに行ってはいけないかを伝えるためのものです。

81.9%のWebサイトがrobots.txtファイル（モバイル）を活用している。前回（2019年72.2%、2020年80.5%）と比較すると、やや改善されたことになります。

robots.txtの作成は必須でありません。404 not foundを返している場合、Googleはウェブサイトのすべてのページがクロールできると仮定しています。他の検索エンジンでは、これとは異なる扱いを受ける可能性があります。

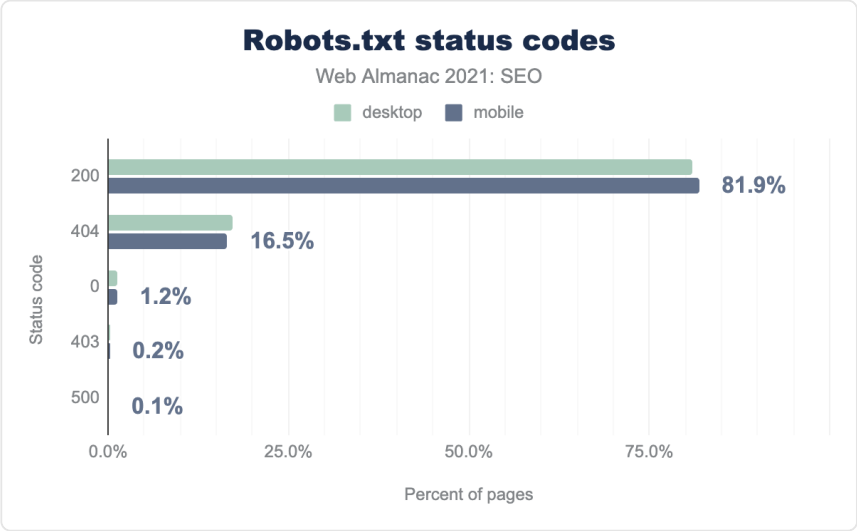


図8.2. robots.txtのステータスコードの内訳。

`robots.txt`を使用することで、ウェブサイトのオーナーは検索エンジンのロボットを制御できます。しかし、16.5%ものウェブサイトが`robots.txt`ファイルを持っていないというデータも出ています。

ウェブサイトが`robots.txt`ファイルを誤って設定している可能性があります。たとえば、人気のあるウェブサイトの中には、（おそらく間違って）検索エンジンをブロックしているものがありました。Googleはこれらのウェブサイトのインデックスを一定期間維持するかもしれませんが、最終的には検索結果での可視性は低下します。

`robots.txt`に関連するエラーのもう1つのカテゴリーは、アクセシビリティやネットワークエラーです。つまり、`robots.txt`は存在するがアクセスできない状態です。Googleが`robots.txt`ファイルを要求してこのようなエラーが発生した場合、ボットはしばらくの間、ページの要求を停止することがあります。これは、検索エンジンがあるページをクロールできるかできないか分からないので、`robots.txt`がアクセスできるようになるまで待つというロジックです。

データセットに含まれるウェブサイトの~0.3%が403 Forbiddenまたは5xxを返しました。ボットによってこれらのエラーの扱いが、異なる可能性があるため、Googlebotが何を見たのかは正確にはわかりません。

Googleが公開している2019年からの[283]最新情報では、5%ものウェブサイトがrobots.txtで一時的に5xxを返し、26%ものウェブサイトが到達不能になっていたそうです。

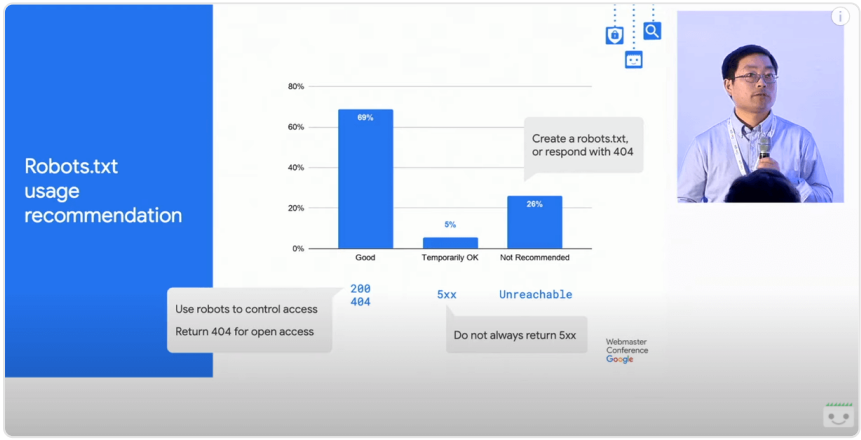


図8.3. Googlebotが遭遇したrobots.txtのステータスコードの内訳。

HTTP ArchiveとGoogleのデータの不一致の原因として、2つのことが考えられます。

283. https://www.youtube.com/watch?v=JvYh1oe5Zx0&t=315s

1. Googleは2年前のデータを提示し、HTTP Archiveは最近の情報に基づいている、あるいは

2. HTTP Archiveは、CrUXのデータに含まれるほど人気のあるWebサイトに焦点を当て、Googleは既知のWebサイトをすべて訪問しようとします。

`robots.txt` のサイズ

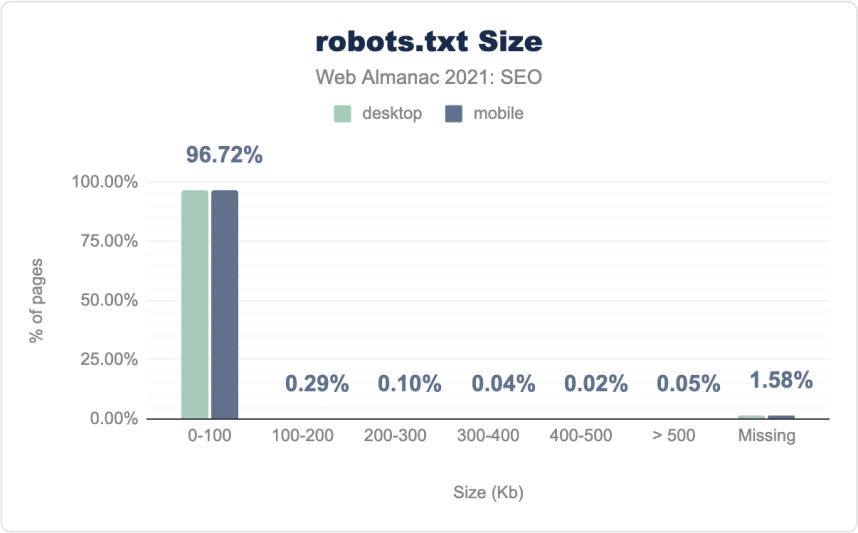


図8.4. `robots.txt` のサイズ分布。

ほとんどのrobots.txtファイルは0〜100kbの間でかなり小さいです。しかし、Googleの最大制限を超える500キロバイトを超えるrobots.txtファイルを持っている3,000以上のドメインを発見しました。このサイズ制限を超えたルールは無視されます。

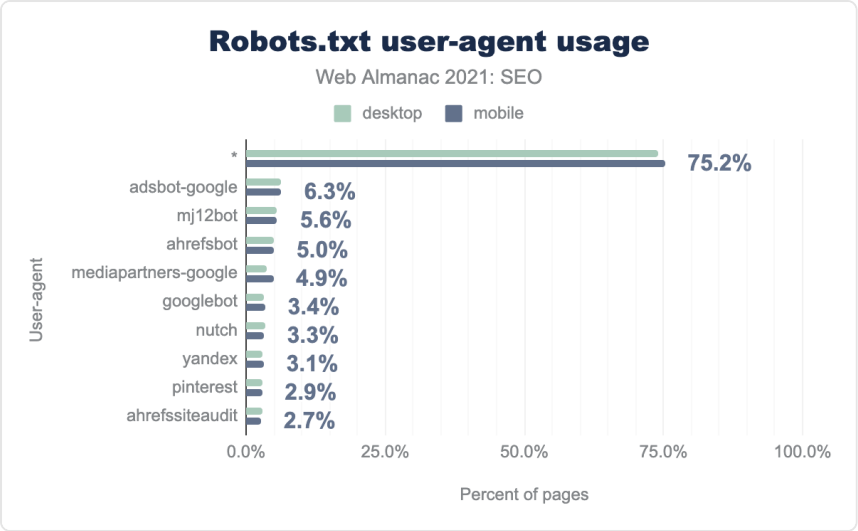


図8.5. `robots.txt` のユーザーエージェント使用状況。

全ロボットに対するルールを宣言することも、特定のロボットに対するルールを指定することも可能です。ボットは通常、ユーザーエージェントのもっとも具体的なルールに従おうとします。`User-agent: Googlebot` はGooglebotのみを参照し、`User-agent: *` はより具体的なルールを持たないすべてのボットを指します。

もっとも指定されたユーザーエージェントのトップ5に、`mj12bot` (Majestic) と `ahrefsbot` (Ahrefs) というSEO関連の2つの人気ロボットが、あることがわかりました。

`robots.txt` 検索エンジンの内訳

ユーザーエージェント	デスクトップ	モバイル
Googlebot	*3.3%*	*3.4%*
Bingbot	*2.5%*	*3.4%*
Baiduspider	*1.9%*	*1.9%*
Yandexbot	*0.5%*	*0.5%*

図8.6. `robots.txt` 検索エンジンの内訳。

特定の検索エンジンに適用されるルールを見ると、Googlebotがもっとも参照され、クロー

ルされたWebサイトの3.3%に出現しています。

Bing、Baidu、Yandexなど他の検索エンジンに関連するロボットルールはあまり人気がない（それぞれ2.5%、1.9%、0.5%）。これらのボットにどのようなルールが適用されているかは調べていません。

Canonicalタグ

ウェブは膨大なドキュメントの集合体であり、その中には重複しているものもあります。重複コンテンツの問題を防ぐために、ウェブマスターはcanonicalタグを使って、どのバージョンをインデックスされるのが好ましいかを検索エンジンに伝えることができます。また、canonicalは、ランキングページへのリンクなどのシグナルを統合するのに役立ちます。

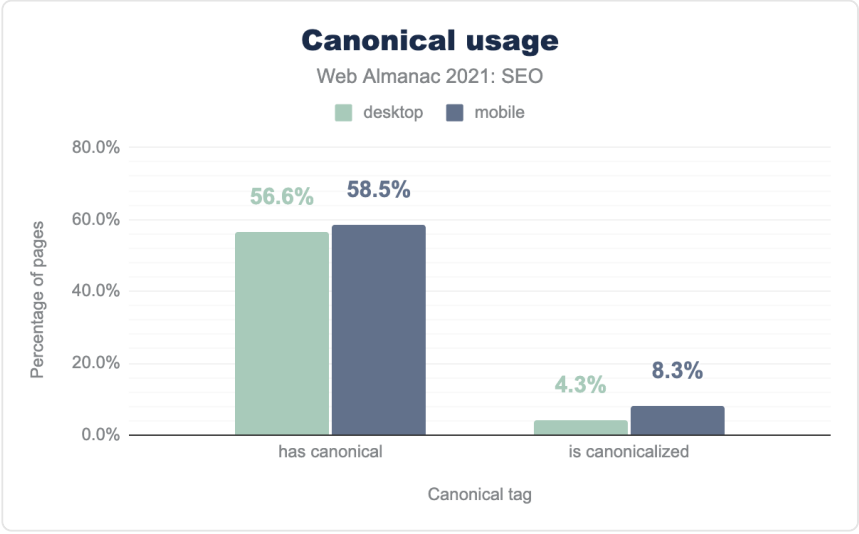


図8.7. Canonicalタグの使用方法。

このデータでは、年々canonicalタグの導入が進んでいることがわかります。たとえば、2019年版では、48.3%のモバイルページがcanonicalタグを使用していたことがわかります。2020年版では53.6%に増え、2021年版では58.5%となっています。

モバイルページにはデスクトップページよりも多くのcanonicalが設定されています。また、モバイルページの8.3%、デスクトップページの4.3%が別のページに正規化されており、Googleやその他の検索エンジンにcanonicalタグで示されたページがインデックスされるべきページであることを明確に示唆するようになっています。

モバイルで正規化されたページの数が多いのは、個別のモバイルURL[284]を使用しているWebサイトに関連していると思われます。このような場合、Googleは対応するデスクトップのURLを指す`rel="canonical"`タグを設置することを推奨しています。

今回のデータセットと分析は、ウェブサイトのホームページに限定したものであり、テスト対象ウェブサイトのすべてのURLを考慮すると、データは異なる可能性があります。

canonicalタグを実装する2つの方法

canonicalを実装する場合、指定する方法は2つあります。

1. ページのHTMLの`<head>`セクションで
2. HTTPヘッダーの中で（`Link` HTTPヘッダーを経由して）

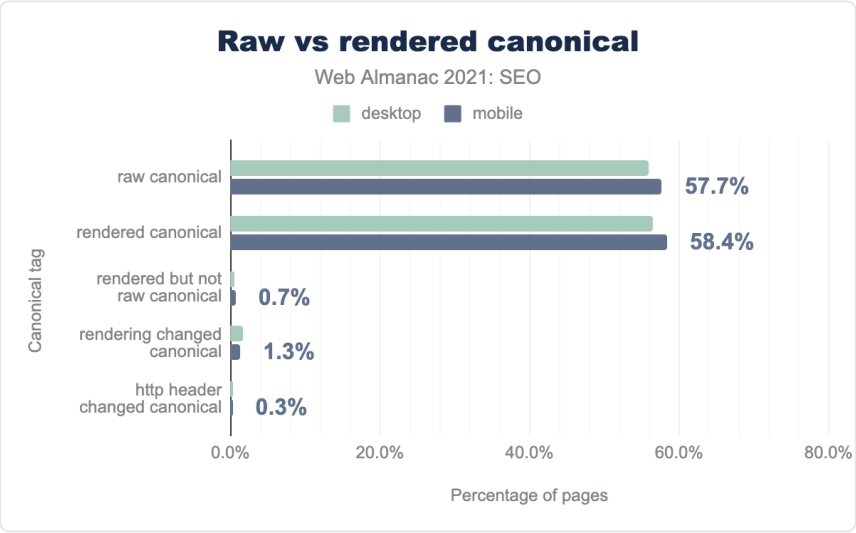


図8.8. 未加工のCanonicalとレンダリングの使い分けの目安。

canonicalタグをHTMLページの`<head>`に実装することは、`Link`ヘッダーを使う方法よりもはるかに一般的です。一般に、headセクションにタグを実装する方が簡単だと考えられており、そのため、使用率が非常に高くなっています。

また、配信された未加工のHTMLと、JavaScriptを適用した後のレンダリングHTMLの間で、canonicalにわずかな変化（1%未満）が見られました。

284. *https://developers.google.com/search/mobile-sites/mobile-seo/separate-urls*

canonicalタグの矛盾

ページには、複数のcanonicalタグを含むことがあります。このように相反するシグナルがある場合、検索エンジンはそれを把握しなければなりません。GoogleのSearch Advocateの一人であるMartin Splitt[285]は、かつてGoogle側で未定義の動作を引き起こす[286]と発言しています。

先ほどの図では、1.3%ものモバイルページが、最初のHTMLとレンダリングバージョンで異なるcanonicalタグを持っていることが示されています。

昨年の章では次のように指摘しています[287] “実装方法の違いでも同様の衝突が見られ、モバイルページの0.15%、デスクトップページの0.17%が、HTTPヘッダーとHTMLヘッドを介して実装したcanonicalタグの衝突を示しました”。

その衝突に関する今年のデータは、さらに心配なものです。デスクトップで0.4%、モバイルでは0.3%のケースでページが矛盾する信号を送っています。

Web Almanacのデータはホームページのみを対象としているため、アーキテクチャの奥深くに位置するページにはさらに問題のある可能性があり、これらは正規化シグナルを必要なページである可能性が高くなります。

ページ経験

2021年は、ユーザー体験への注目が高まりました。Googleはページ体験の更新[288]を開始し、HTTPSやモバイルフレンドリーなどの既存のシグナルと、Core Web Vitalsという新しいスピード指標を盛り込みました。

285. *https://x.com/g33konaut*
286. *https://www.youtube.com/watch?v=bAE3L1E1Fmk&t=772s*
287. *https://almanac.httparchive.org/ja/2020/seo#正規化*
288. *https://developers.google.com/search/blog/2020/11/timing-for-page-experience*

HTTPS

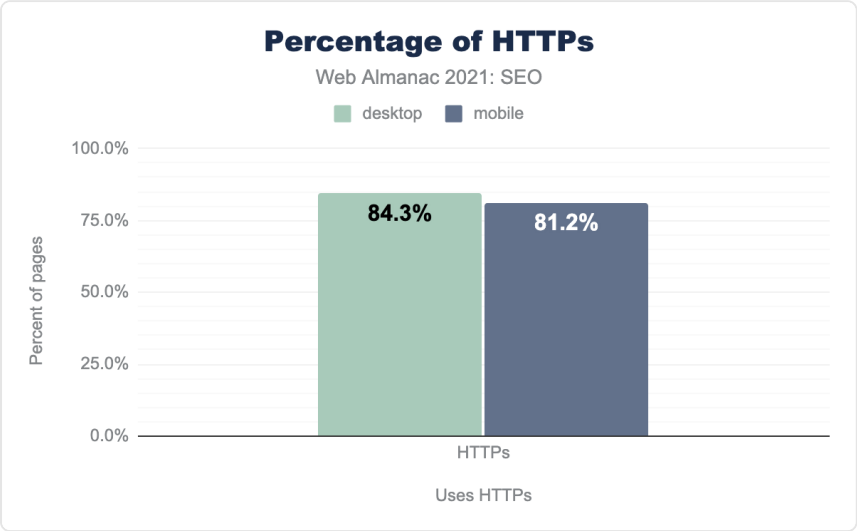


図8.9. HTTPSで提供されるデスクトップおよびモバイルページの割合。

HTTPSの採用は依然として増加している。モバイルページの81.2%、デスクトップページの84.3%でHTTPSがデフォルトとして採用されました。これは、前年比でモバイルサイトでは約8%、デスクトップサイトでは7%増加しています。

モバイルフレンドリー

今年はモバイルの利便性が若干向上しています。レスポンシブデザインの導入は増加し、ダイナミックサーブは比較的横ばいで推移しています。

レスポンシブデザインは同じコードを送信し、画面サイズに応じてウェブサイトの表示方法を調整するのに対し、ダイナミックサービングはデバイスに応じて異なるコードを送信します。レスポンシブWebサイトの識別には`viewport`メタタグが、`Vary: User-Agent header`は、ダイナミックサービングを使用しているウェブサイトを識別するために使用されます。

91.1%

図8.10. モバイルフレンドリーのシグナルである`viewport`メタタグを使用したモバイルページの割合。

モバイルページの91.1%が`viewport`メタタグを含んでおり、2020年の89.2%から増加した。また、デスクトップページの86.4%がviewport metaタグを含んでおり、2020年の83.8%から増加しています。

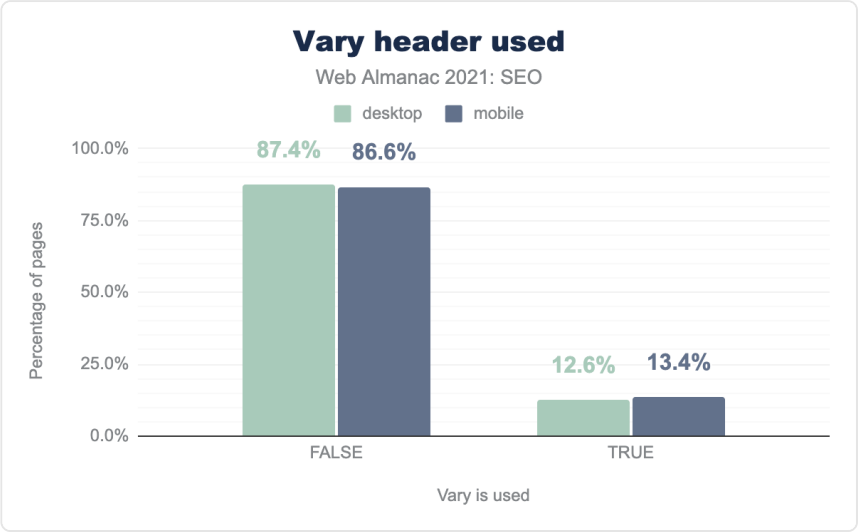


図8.11. `Vary: User-Agent` *ヘッダーの使用状況。*

また、`Vary: User-Agent`ヘッダーについては、このフットプリントのデスクトップページの12.6%、モバイルページの13.4%がこのヘッダーを使用しており、数値はほとんど変わりません。

13.5%

図8.12. モバイルページで読みやすいフォントサイズを使用していない割合。

モバイルフレンドリーに失敗した最大の理由のひとつは、13.5%のページが読みやすいフォントサイズを使用していないことでした。つまり、60%以上のテキストが、モバイルで読みにくい12pxより小さいフォントサイズだった[289]ということです。

Core Web Vitals

Core Web Vitalsは、Googleのページ体験シグナルの一部である新しい速度指標です。この指

289. https://web.dev/font-size/

標は、最大コンテンツペイント（LCP）による視覚的負荷、累積レイアウトシフト（CLS）による視覚的安定性、最初の入力遅延（FID）を使用した対話性を測定します。

このデータは、オプトインしたChromeユーザーの実測データを記録した「Chromeユーザーエクスペリエンスレポート（CrUX）」から得られたものです。

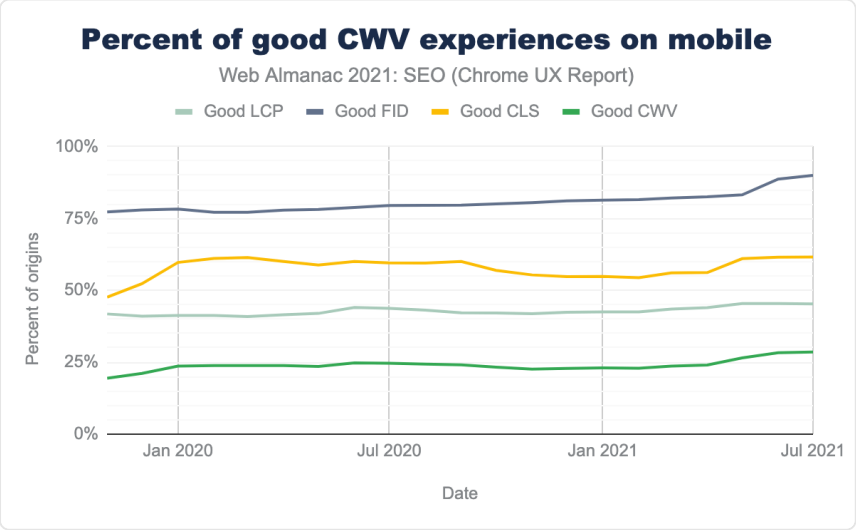


図8.13. Core web vitalsの指標推移。

モバイルサイトの29%がコアウェブバイタルの基準値をクリアしており、昨年の20%から増加しています。ほとんどのWebサイトがFIDをクリアしていますが、WebサイトのオーナーはCLSとLCPの改善に苦戦しているようです。このトピックについては、パフォーマンスの章を参照してください。

On-Page

検索エンジンは、あなたのページのコンテンツを見て、それが検索クエリに関連する結果であるかどうかを判断します。その他のページ上の要素も、検索エンジンでの順位や見た目に影響を与える場合があります。

メタデータ

メタデータには、`<title>` 要素と `<meta name="description">` タグが含まれます。メタデータは直接的、間接的にSEOのパフォーマンスに影響を与えることがあります。

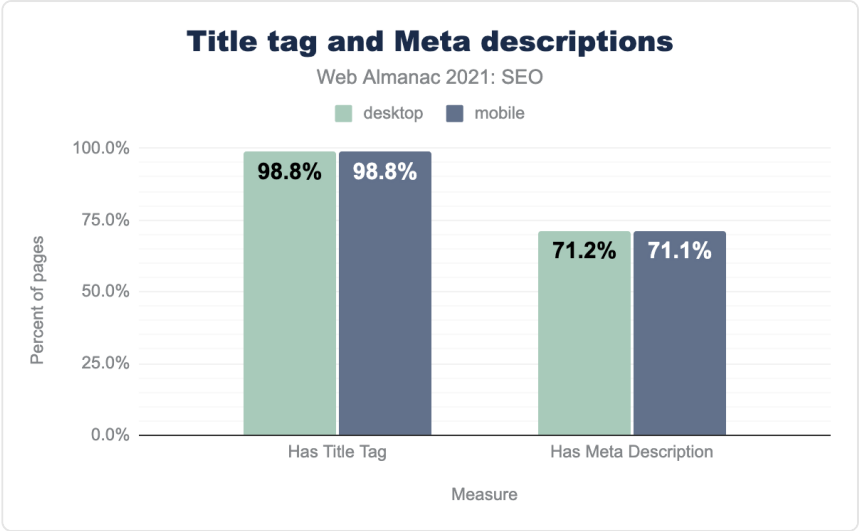


図8.14. タイトルとメタ記述の使用状況の内訳。

2021年、デスクトップとモバイルのページの98.8%が `<title>` 要素を持っていた。デスクトップとモバイルのホームページの71.1%が `<meta name="description">` タグを持っていました。

`<title>` 要素

`<title>` 要素は、ページの関連性に関する強いヒントを提供するページ上のランキング要因であり、検索エンジンの結果ページ（SERP）に表示されることがあります。2021年8月、Googleは検索結果のタイトルをより多く書き換えるようになりました[290]。

290. *https://developers.google.com/search/blog/2021/08/update-to-generating-page-titles*

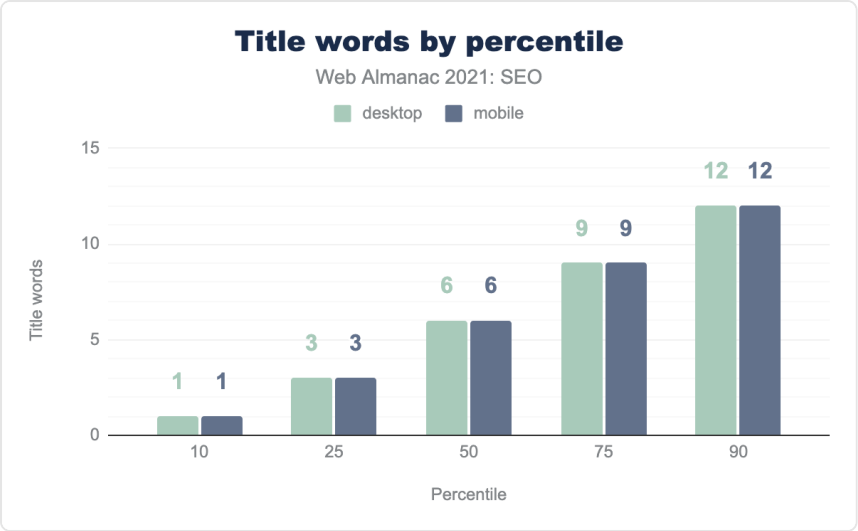


図8.15. title要素に使用されている単語数。

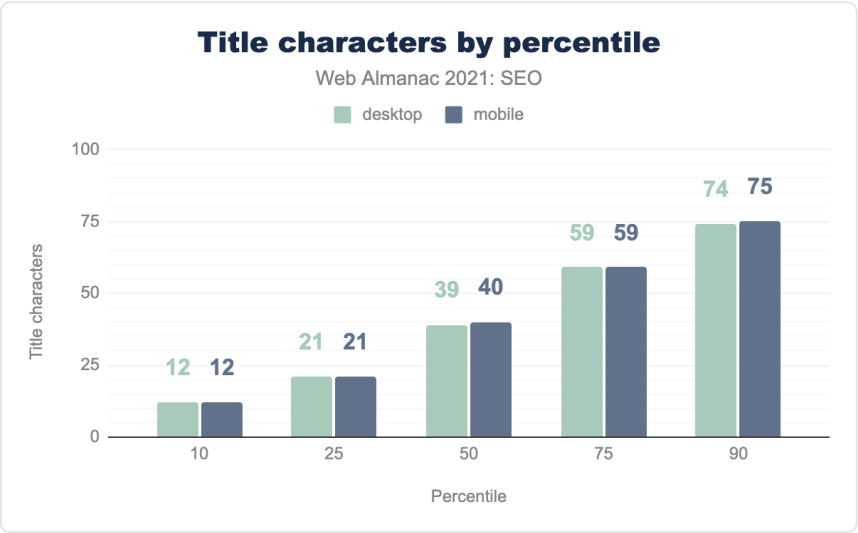


図8.16. title要素で使用される文字数。

2021年に

- 中央のページ `<title>` には6つの単語が含まれていた。
- ページの `<title>` の中央値は、デスクトップとモバイルでそれぞれ39文字と40

文字でした。

- 10%のページで、12語を含む `<title>` 要素があった。
- デスクトップとモバイルのページの10%に、それぞれ74文字と75文字を含む `<title>` 要素がありました。

これらの統計のほとんどは、昨年から比較的変化していません。これらはホームページのタイトルで、より深いページで使用されるものより短い傾向があることに注意してください。

メタ記述タグ

`<meta name="description">` タグは、ランキングに直接影響を与えません。しかし、SERP上でページの説明文として表示されることがあります。

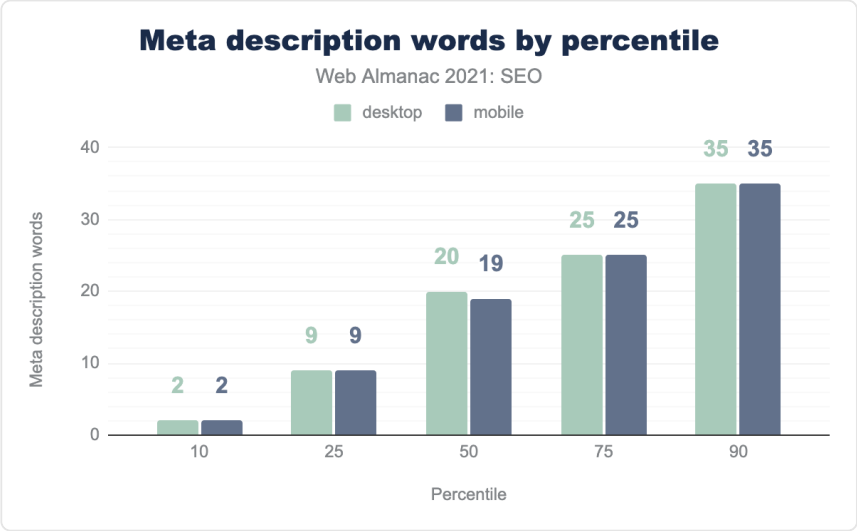


図8.17. メタ記述に使用されている単語数。

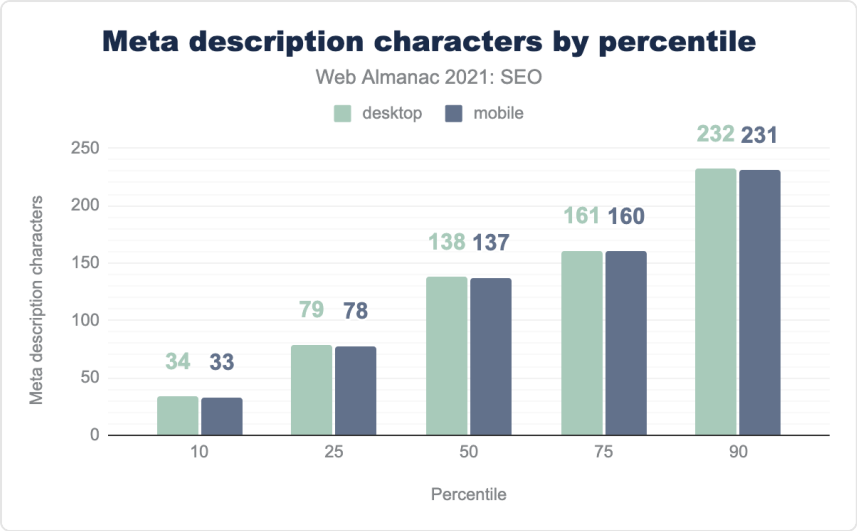


図8.18. メタ記述に使用される文字数。

2021年に

- デスクトップとモバイルのページの `<meta name="description">` タグには、それぞれ20語と19語が含まれています（中央値）。
- デスクトップとモバイルページの `<meta name="description">` タグ文字数の中央値は、それぞれ138文字と127文字でした。
- デスクトップとモバイルのページの10%に、35語を含む `<meta name="description">` タグがありました。
- デスクトップとモバイルのページの10%に、それぞれ232文字と231文字を含む `<meta name="description">` タグがありました。

この数値は、昨年と比較すると、ほぼ横ばいです。

画像

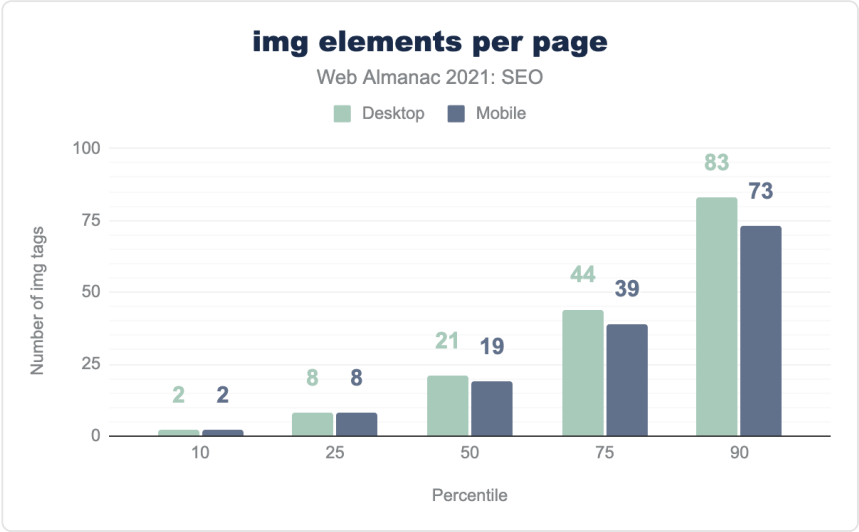


図8.19. 各ページの画像枚数。

画像は、画像検索順位やページパフォーマンスに影響を与えるため、直接的・間接的にSEOに影響を与える可能性があります。

- 10%のページでは、`<img>`タグが2つ以下になっています。これは、デスクトップとモバイルの両方に当てはまります。
- デスクトップ用ページの中央値には21個の`<img>`タグがあり、モバイル用ページの中央値には19個の`<img>`タグがあります。
- デスクトップページの10%は、83個以上の`<img>`タグを持っています。モバイルページの10%は、73個以上の`<img>`タグを持っています。

この数字は2020年以降、ほとんど変化していません。

画像の `alt` 属性

`<img>`要素の`alt`属性は、画像の内容を説明するのに役立ち、アクセシビリティ[291]へ影響を及ぼします。

291. *https://almanac.httparchive.org/en/2021/accessibility*

なお、`alt`属性の欠落は問題を示さない場合があります。ページには極端に小さい画像や空白の画像が含まれていることがありますが、SEO（あるいはアクセシビリティ）上の理由から`alt`属性は必要でありません。

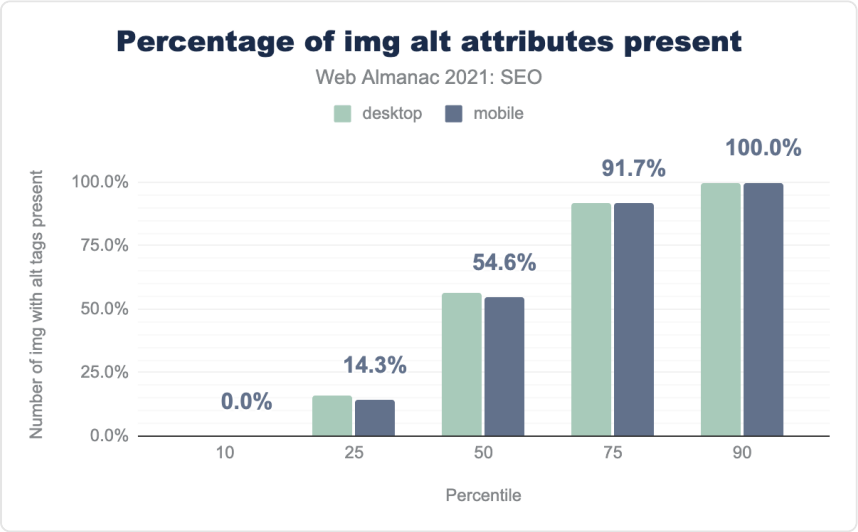


図8.20. `alt`属性を含む画像の割合。

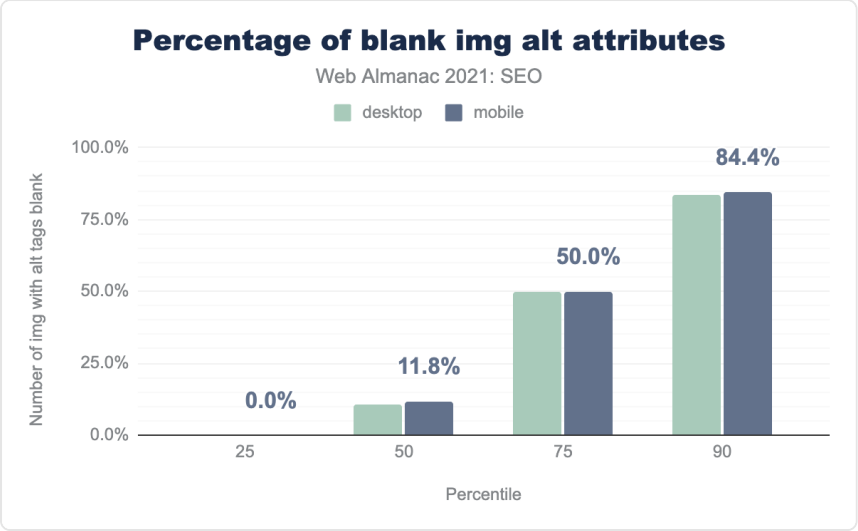


図8.21. `alt`属性が空白の割合。

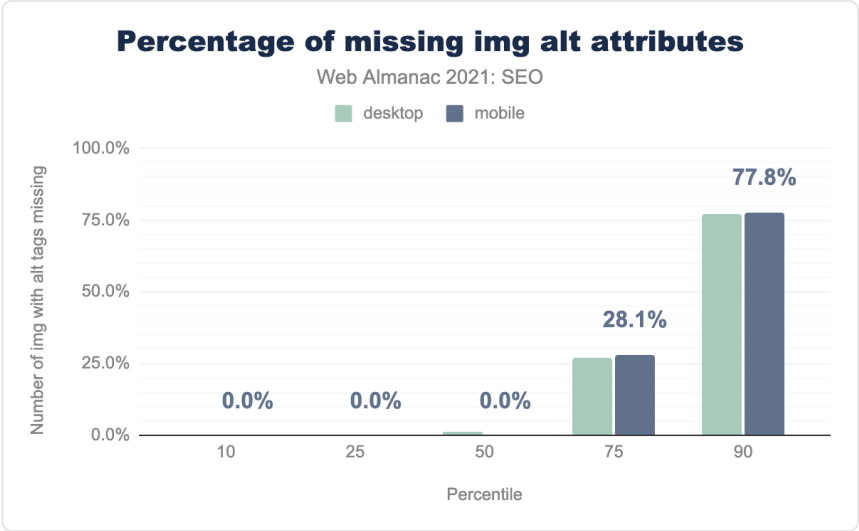


図8.22. `alt` 属性がない画像の割合。

私たちはそれがわかりました

- デスクトップ用ページの中央値では、56.5%の `<img>` タグに `alt` 属性が設定されています。これは、2020年に比べてわずかに増加しています。
- モバイルページの中央値では、54.6%の `<img>` タグに `alt` 属性が設定されています。これは、2020年と比較して若干の増加です。
- しかし、デスクトップとモバイルページの中央値では、10.5%と11.8%の `<img>` タグが `alt` 属性を空白にしています（それぞれ）。これは事実上、2020年と同じです。
- 中央のデスクトップとモバイルのページでは、`<img>` タグに `alt` 属性なしがゼロか、それと近い値になっています。これは、中央ページにある `<img>` タグの2〜3%に `alt` 属性がなかった2020年に比べて改善されています。

画像 `loading` 属性

`<img>` 要素の `loading` 属性は、ユーザーエージェントがページ上の画像のレンダリングと表示をどのように優先させるかに影響します。ユーザー体験やページの読み込み性能に影響を与える可能性があり、これらはいずれもSEOの成功に影響を与えます。

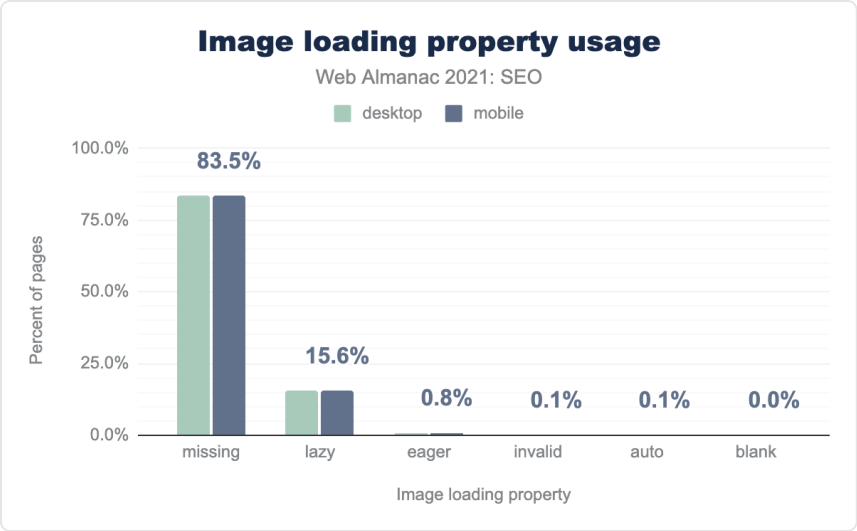


図8.23. 画像読み込みのプロパティの使用状況。

私たちはそれを見たのです。

- 85.5%のページでは、画像の `loading` プロパティが使用されていません。
- 15.6%のページが `loading="lazy"` を使用しており、ビューポートに入る寸前まで画像の読み込みを遅らせています。
- 0.8%のページでは、ブラウザがコードを読み込むと同時に画像を読み込む `loading="eager"` が使用されています。
- 0.1%のページで無効なローディングプロパティが使用されています。
- 0.1%のページでは、ブラウザのデフォルトの読み込み方法を使用する `loading="auto"` が使用されています。

単語数

ページ内の文字数はランキング要因ではありませんが、ページが文字を配信する方法はランキングに大きな影響を与えます。言葉は、未加工のページコードに含まれることも、レンダリングされたコンテンツに含まれることもあります。

レンダリング文字数

まず、レンダリングされたページの内容を見ます。*レンダリング*とは、ブラウザがすべてのJavaScriptとDOMまたはCSSOMを変更する他のコードを実行した後のページのコンテンツです。

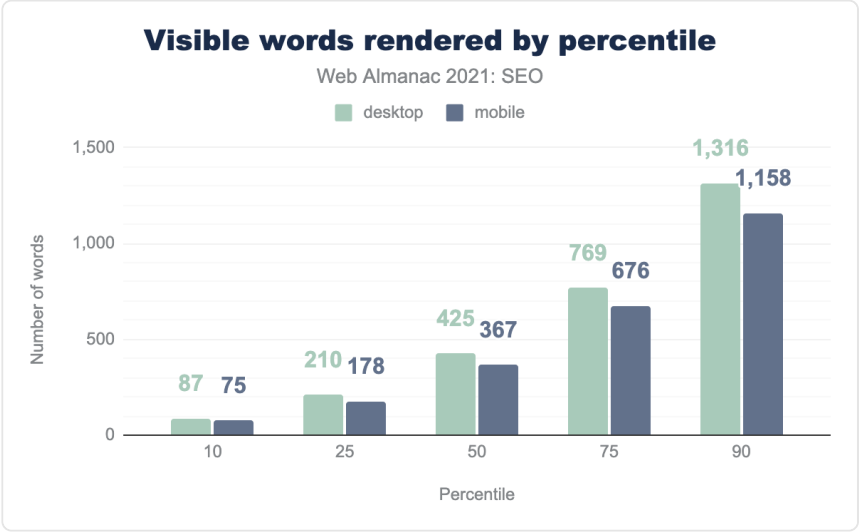


図8.24. レンダリングされた可視ワードの割合。

- デスクトップページのレンダリング文字数の中央値は425語で、2020年には402語になっています。
- モバイルページのレンダリングの中央値は367語であるのに対し、2020年には348語となっています。
- モバイルページのレンダリングは、デスクトップページのレンダリングより13.6%少ない単語数です。Googleはモバイル専用のインデックスであることに注意してください。モバイル版にないコンテンツはインデックスされない可能性があります。

未加工のワード数

次に、未加工のページコンテンツについて見ていきます。*未加工*とは、ブラウザがJavaScriptやDOMやCSSOMを修正する他のコードを実行する前のページのコンテンツのことです。ソースコードで配信され、目に見える「未加工」のコンテンツです。

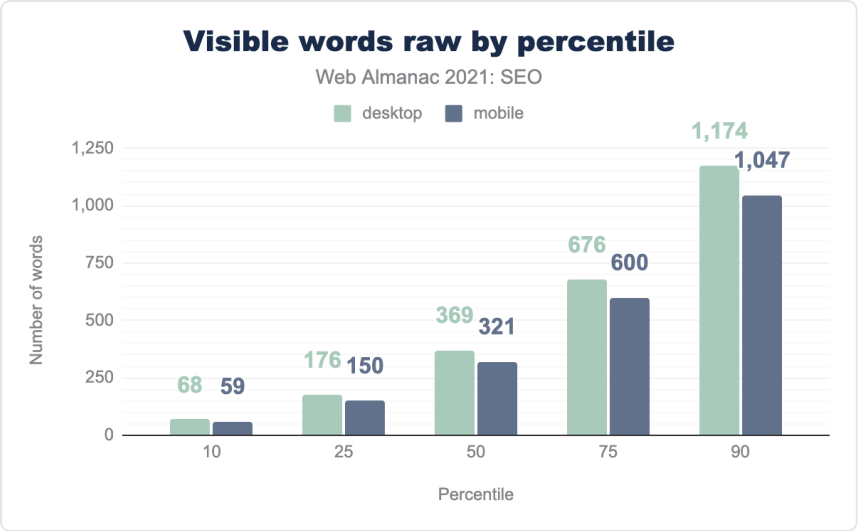


図8.25. 未加工の可視化単語の割合。

- デスクトップページの中央値は369語であるのに対し、2020年には360語になっています。
- モバイルページの中央値は321語であるのに対し、2020年には312語となっています。
- モバイルの未加工ページは、デスクトップの未加工ページよりも13.1%少ない単語しか含まれていません。Googleはモバイル専用のインデックスであることに注意してください。モバイルHTML版にないコンテンツは、インデックスされない可能性があります。

全体として、デスクトップ端末で書かれたコンテンツの15%が、モバイル版では14.3%がJavaScriptによって生成されています。

構造化データ

これまで検索エンジンは、非構造化データ、つまりページ上のテキストを構成する単語、段落、その他のコンテンツの山を扱ってきた。

スキーママークアップやその他の構造化データは、検索エンジンがコンテンツを解析し、整理するための別の方法を提供します。構造化データは、Googleの検索機能[292]の多くに力を与

292. https://developers.google.com/search/docs/advanced/structured-data/search-gallery

えています。

構造化データは、ページ上の単語と同様に、JavaScriptで変更できます。

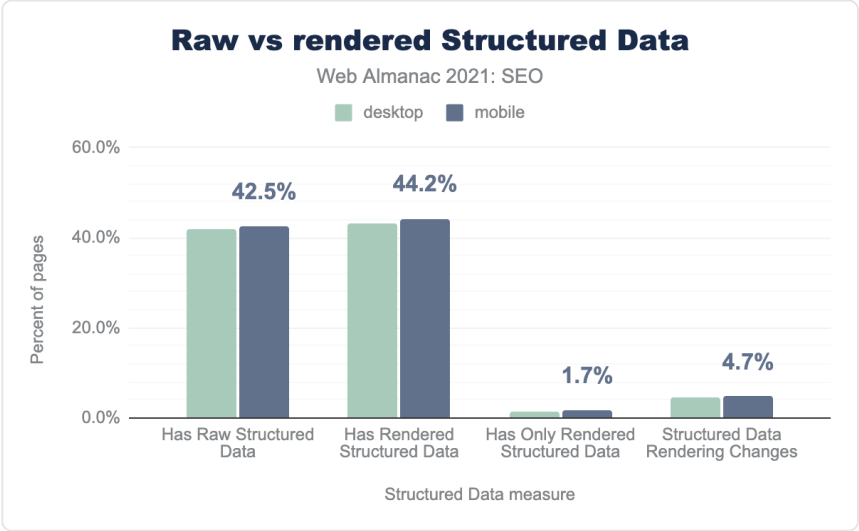


図8.26. 構造化データの割合。

モバイルページの42.5%、デスクトップページの41.8%が、HTML内に構造化データを有しています。モバイルページの4.7%、デスクトップページの4.5%で、JavaScriptが構造化データを修正しています。

モバイルページの1.7%、デスクトップページの1.4%で、最初のHTMLレスポンスに存在しない構造化データがJavaScriptによって追加されています。

もっとも一般的な構造化データ形式

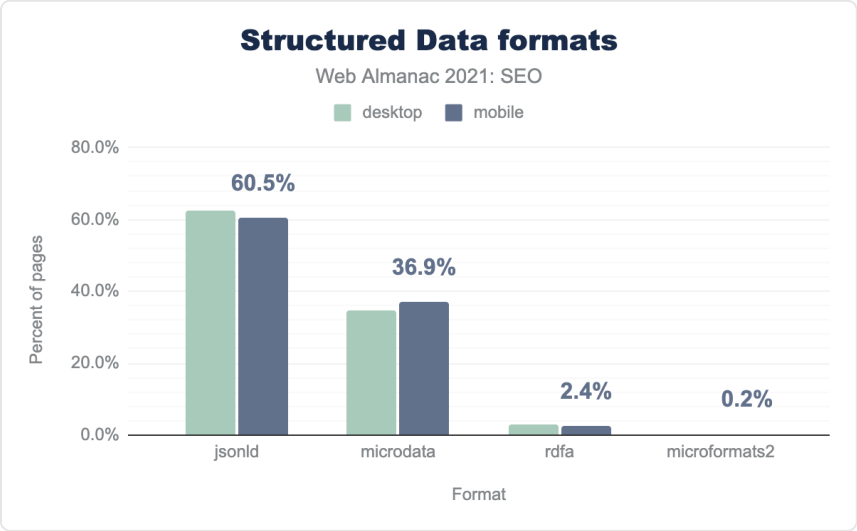


図8.27. 構造化データ形式の内訳

構造化データをページに含めるには、いくつかの方法があります。JSON-LD、microdata、RDFa、microformats2などです。JSON-LDは、もっとも一般的な実装方法です。構造化データを持つデスクトップとモバイルのページの60%以上が、JSON-LDで実装しています。

構造化データを実装しているWebサイトのうち、デスクトップおよびモバイルページの36%以上がmicrodataを使用しており、RDFaまたはmicroformats2を使用しているページは3%未満です。

構造化データの導入は昨年より少し増えている。2020年の30.6%に対し、2021年は33.2%のページで使用されています。

もっとも一般的なスキーマの種類

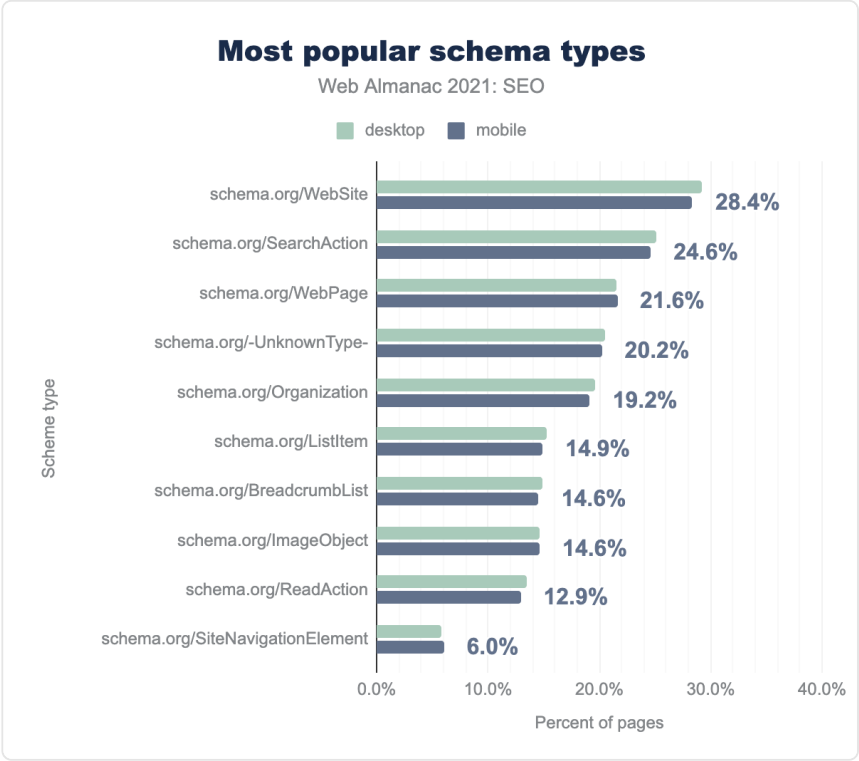


図8.28. もっとも一般的なスキーマの種類。

ホームページでもっともよく見られるスキーマタイプは `WebSite`、`SearchAction`、`WebPage` です。`SearchAction` は、サイトリンク検索ボックス[293] を動かすもので、Google が検索結果ページで表示することを選択できます。

`<h>` 要素（見出し）

見出し要素（`<h1>`、`<h2>` など）は、重要な構造要素です。ランキングに直接影響を与えるものではありませんが、Googleがページのコンテンツをより理解するのに役立ちます。

293. *https://developers.google.com/search/docs/advanced/structured-data/sitelinks-searchbox*

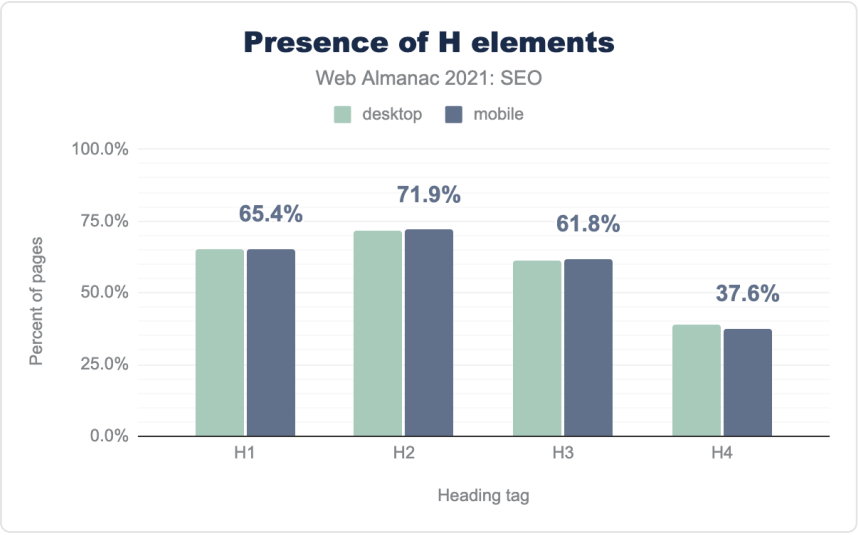


図8.29. 見出し要素の割合。

主な見出しについては、`h1`（65.4%）よりも多くのページ（71.9%）に`h2`があります。この差に明確な説明はありません。デスクトップとモバイルのページの61.4%は`h3`を使い、`h4`は39%以下です。

デスクトップとモバイルの見出しの使い方にほとんど差はなく、2020年に対して大きな変化もありませんでした。

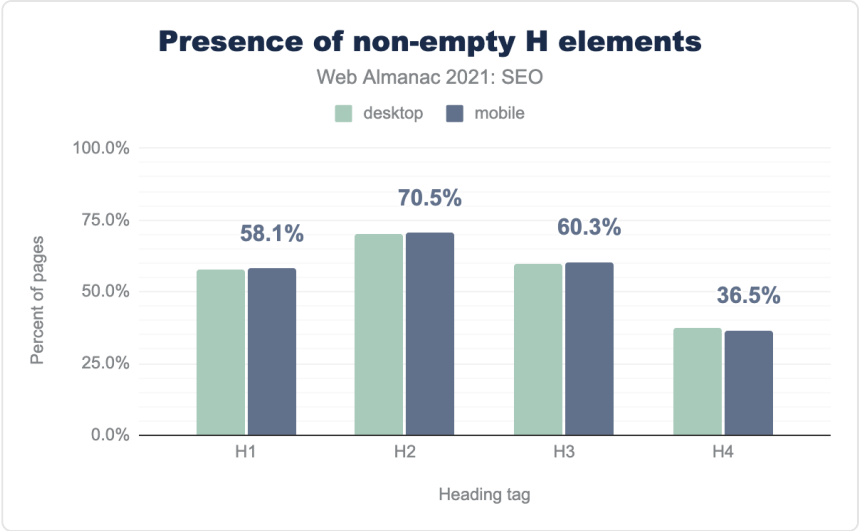


図8.30. 空でない見出し要素の使用状況。

しかし、*空ではない* `<h>` 要素、とくに `h1` を含むページの割合は低くなっています。ホームページでは、ロゴ画像を `<h1>` 要素で囲むことが多いので、このような結果になったのでしょう。

リンク

検索エンジンは、リンクを利用して新しいページを発見し、ページの重要性を判断するのに役立つページランクを渡すために使用しています。

16.0%

図8.31. 説明的でないリンクテキストを使用しているページ。

ページランクに加え、リンクアンカーとして使用されるテキストは、検索エンジンがリンク先のページが何であるかを理解するのに役立ちます。Lighthouseでは、使用されているアンカーテキストが有用なテキストであるか、それとも「詳しくはこちら」や「ここをクリック」といったあまり説明的でない一般的なアンカーテキストであるかをチェックするテストが用意されています。テストしたリンクの16%は、説明的なアンカーテキストを持っていませんでした。これは、SEOの観点からは機会を逃し、またアクセシビリティの観点からも良くありません。

内部および外部リンク

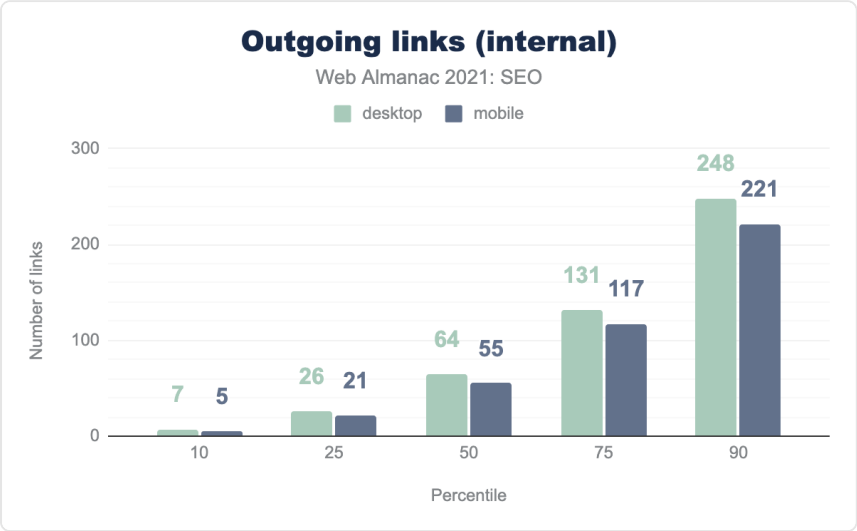


図8.32. ホームページからの内部リンク

内部リンクとは、同じサイト内の他のページへのリンクのことです。デスクトップ版に比べ、モバイル版ではページのリンクが少なかった。

このデータによると、デスクトップの内部リンク数の中央値はモバイルよりも16%多く、それぞれ64対55となっています。これは、開発者が小さい画面でも使いやすいように、モバイルではナビゲーションメニューやフッターを最小限にする傾向があるためと思われます。

もっとも人気のあるウェブサイト（CrUXのデータによると上位1,000）は、人気のないウェブサイトよりも多くの発信内部リンクを持っています。デスクトップでは144、モバイルでは110で、中央値より2倍以上多いのです。これは、一般的にページ数の多い大規模なサイトでメガメニューが使用されているためと思われます。

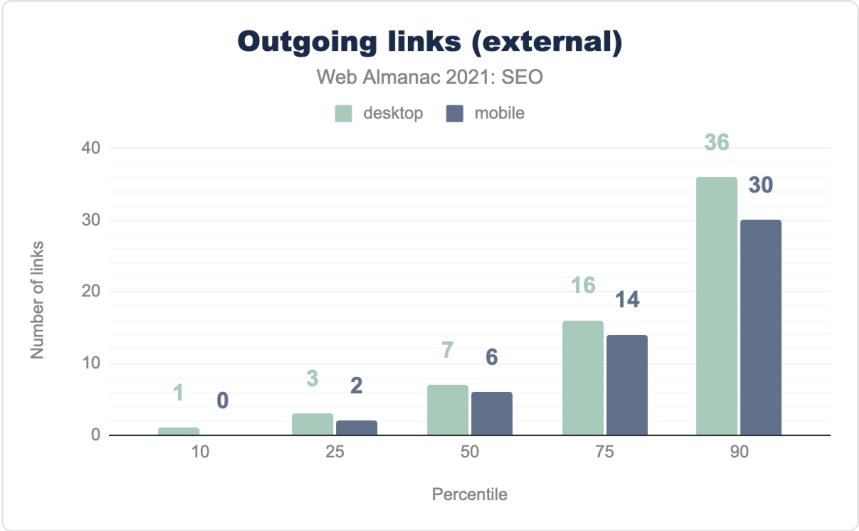


図8.33. ホームページからの外部リンク

外部リンクとは、あるウェブサイトから別のサイトへのリンクのことです。このデータでも、モバイル版のページで外部リンクは少ないことがわかります。

2020年とほぼ同じ数字になっています。Googleは今年、モバイルファーストインデックスを展開したにもかかわらず、ウェブサイトはモバイル版をデスクトップ版と同等にすることができていないのです。

テキストと画像のリンク

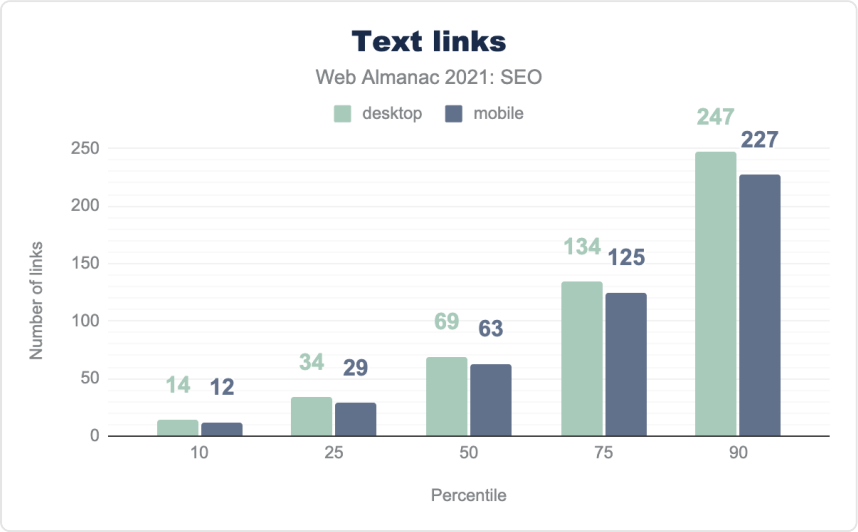


図8.34. ホームページからのテキストリンク

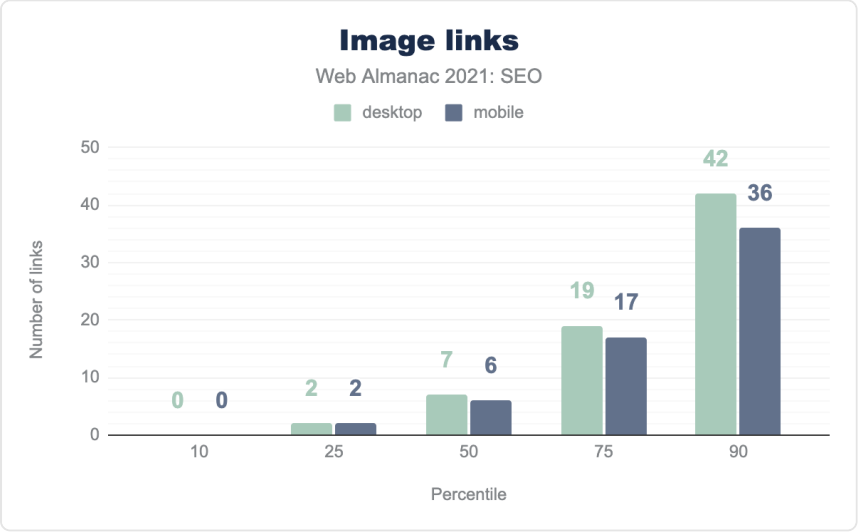


図8.35. ホームページからの画像リンク

ウェブ上のリンクの大部分はテキストベースですが、画像から他のページにリンクしているものもあります。デスクトップページのリンクの9.2%、モバイルページのリンクの8.7%が画像リンクです。画像リンクでは、画像に設定された `alt` 属性がアンカーテキストとして

機能し、そのページが何についてのページなのか、さらに詳しい情報を提供します。

リンク属性

2019年9月、Googleはリンクを*スポンサー*または*ユーザーが作成した*コンテンツに分類できる属性を導入しました。これらの属性は、以前2005年に導入された`rel=nofollow`を追加したものです。新しい属性である`rel=ugc`と`rel=sponsored`は、リンクに追加情報を与えます。

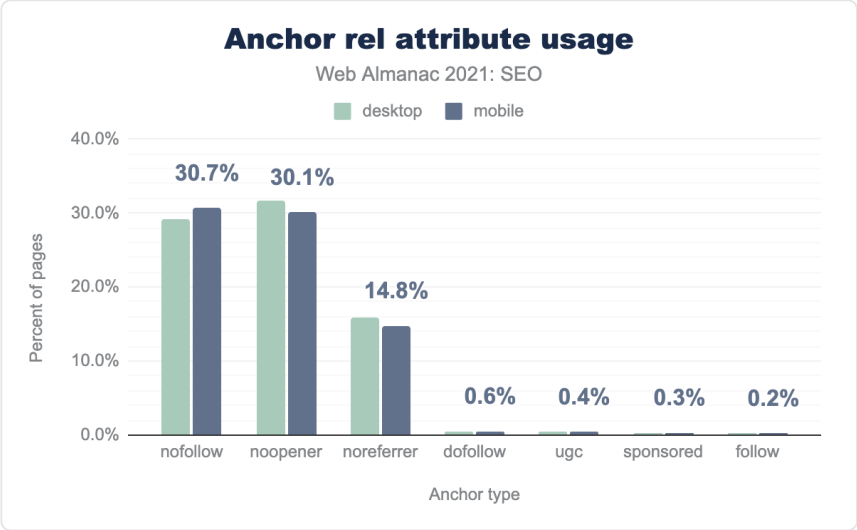


図8.36. Rel属性の使用状況。

新しい属性は、少なくともホームページではまだかなり稀で、`rel="ugc"`はモバイルページの0.4%、`rel="sponsored"`はモバイルページの0.3%に表示されます。これらの属性は、ホームページ以外のページでより多く採用されている可能性があります。

`rel="follow"`と`rel=dofollow`は`rel="ugc"`や`rel="sponsored"`よりも多くのページで表示されます。これは問題ではありませんが、Googleは`rel="follow"`と`rel="dofollow"`を公式の属性ではないので無視します。

`rel="nofollow"`は30.7%のモバイルページで見つかり、昨年と同様でした。この属性は非常に多く使われているため、Googleが`nofollow`をヒントに変更したのは当然のことで、つまり、それを尊重するかどうかを選択できるようになったのです。

アクセラレイテッド・モバイル・ページ（AMP）

2021年、アクセラレイテッド・モバイル・ページ (AMP)のエコシステムに大きな変化がありました。AMPは トップページカルーセルには必要なくなり、Google Newsアプリにも必要なくなり、GoogleはSERPのAMP結果の横にAMPロゴを表示しなくなりました[294]。

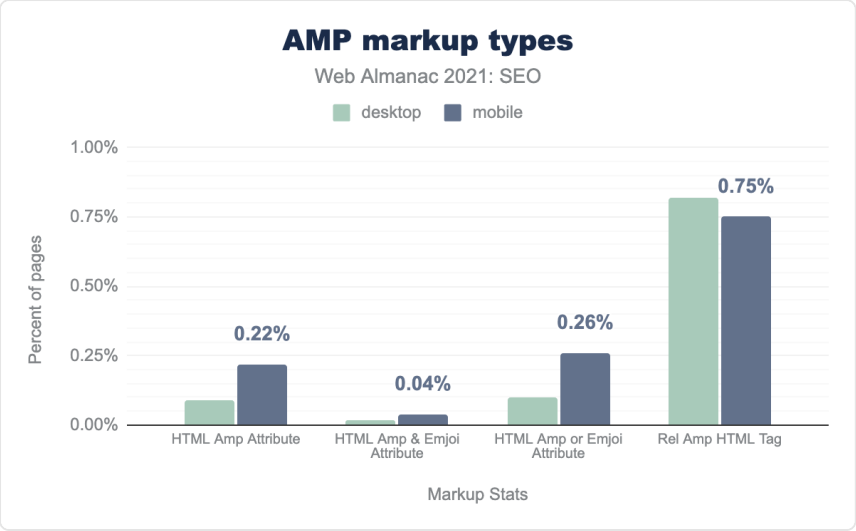


図8.37. AMP属性の使用状況。

しかし、2021年になってもAMPの採用は増え続けている。現在、デスクトップページの0.09%にAMP属性が含まれているのに対し、モバイルページは0.22%となっています。これは、2020年のデスクトップページの0.06%、モバイルページの0.15%から上昇したものです。

294. https://developers.google.com/search/blog/2021/04/more-details-page-experience#details

国際化

> *言語や地域によって複数のバージョンのページがある場合は、Google にそのことを伝えてください。そうすることで、Google検索がユーザーに言語や地域ごとにもっとも適切なバージョンのページを提供することができます。*
>
> *– Google SEOのドキュメント*[295]

検索エンジンにローカライズされたページを知らせるには、`hreflang`タグを使用します。`hreflang`属性は、Yandex[296]やBing（ある程度[297]）でも使用されています。

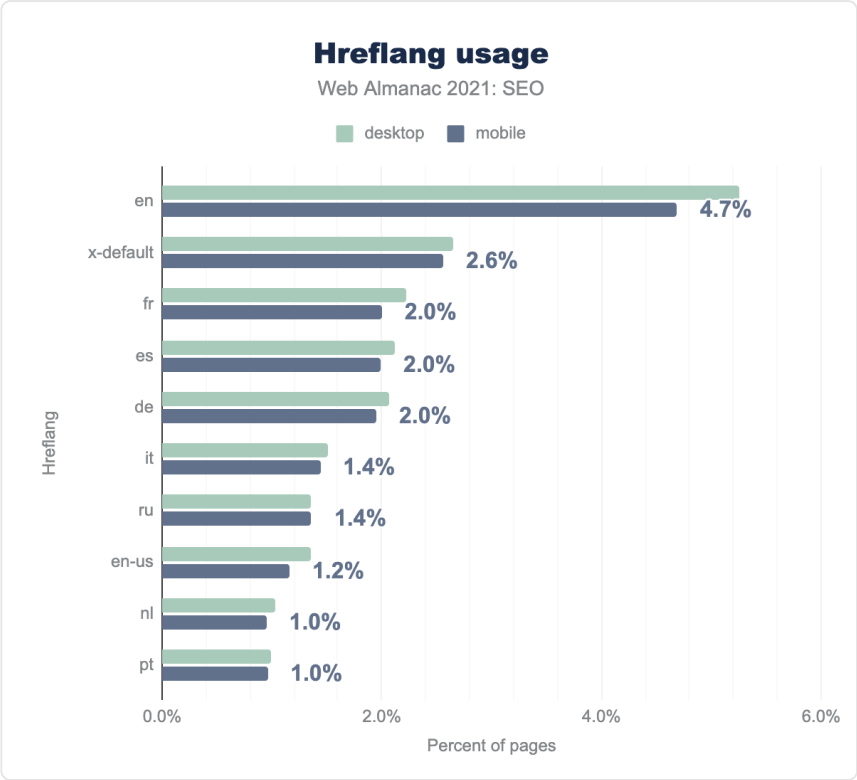


図8.38. hreflangタグのトップ属性表。

295. https://developers.google.com/search/docs/advanced/crawling/localized-versions
296. https://yandex.com/support/webmaster/yandex-indexing/locale-pages.html
297. https://x.com/facan/status/1304120691172601856

デスクトップ用ページの9.0%、モバイル用ページの8.4%がhreflang属性を使用しています。

`hreflang` 情報の実装方法には、HTMLの `<head>` 要素、 `Link` ヘッダー、XMLサイトマップの3つがあります。このデータには、XMLサイトマップのデータは含まれていない。

hreflang属性でもっとも利用されているのは、 `"en"` （英語版）です。モバイル用ホームページの4.75%、デスクトップ用ホームページの5.32%が使用しています。

`x-default` （フォールバック版とも呼ばれる）は、モバイルでは2.56%のケースで使用されています。 `hreflang` 属性で指定されるその他の一般的な言語は、フランス語とスペイン語です。

Bingにとって、 `hreflang` は `content-language` ヘッダーよりも「はるかに弱い信号」です。

他の多くのSEOパラメーターと同様に、 `content-language` には、以下のような複数の実装方法があります。

1. HTTPサーバーレスポンス
2. HTMLタグ

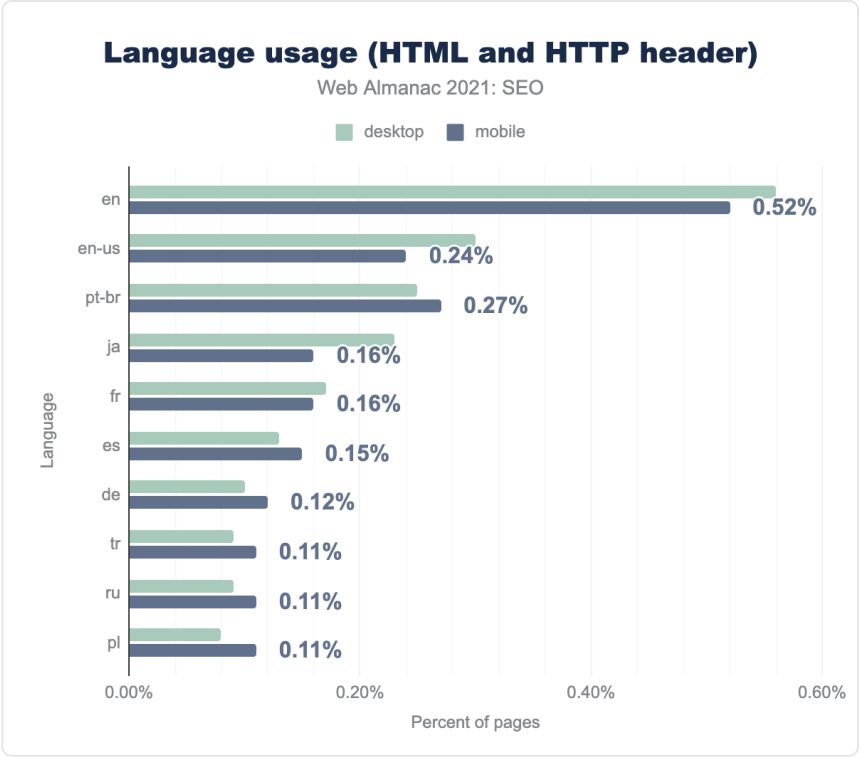


図8.39. 言語使用状況（HTML、HTTPヘッダー）。

`content-language` を実装するには、HTTPサーバーレスポンスを使用するのがもっとも一般的な方法です。デスクトップでは8.7%、モバイルでは9.3%のウェブサイトがこの方法を採用しています。

HTMLタグの使用はあまり一般的でなく、content-languageが表示されるのはモバイルサイトのわずか3.3%です。

結論

ウェブサイトは、SEOの観点から徐々に改善されてきています。おそらく、ウェブサイトがSEOを改善し、ウェブサイトをホストするプラットフォームも改善していることが複合的に作用しているのだろう。ウェブは広くて厄介な場所なので、まだまだやるべきことはたくさんありますが、一貫した進歩が見られるのは嬉しいことです。

著者



Patrick Stox

@patrickstox patrickstox https://patrickstox.com

Ahrefs[298] のプロダクトアドバイザー、テクニカルSEO、ブランドアンバサダーを務める。Raleigh SEOミートアップ[299]（米国で最も成功したSEO Meetup）、ビールとSEOミートアップ[300]、Raleigh SEOカンファレンス[301]の主催者でもある。また、Technical SEO Slack グループを運営し、/r/TechSEO on Reddit[302] のモデレーターを務めています。また、PatrickはTwitterの「Uncommon SEO Knowledge」というスレッドで、ランダムにSEOの知識を共有するのが好きです。彼は有名なカンファレンススピーカーであり、業界ブロガー（最近は主にAhrefsのブログ[303]で）、検索賞の審査員、そして米国労働省の検索マーケティング戦略家の役割の定義に貢献しました。



Tomek Rudzki

@TomekRudzki Tomek3c https://tomekseo.com/

Tomekは、Onely[304]の研究開発責任者です。また、ウェブサイト所有者がより多くのコンテンツをGoogleにインデックスさせることを目的とした製品、ZipTie[305]を開発中です。余暇には、ハイキングとポーカーを楽しんでいます。



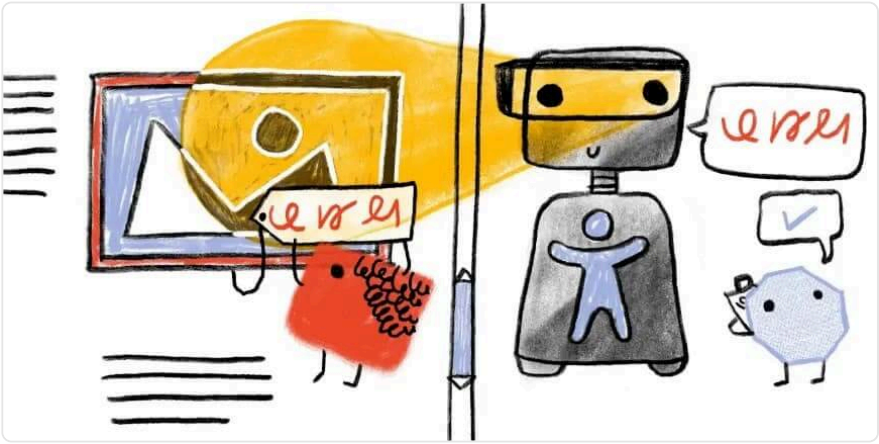
Ian Lurie

@ianlurie wrttnwrd https://www.ianlurie.com

Ianは、マーケティングコンサルタント、SEO、講演者、回復代理店の創業者です。1995年にデジタル・マーケティング・エージェンシーであるPortentを設立し、2017年にClearlinkに売却しました。今は独立して、大好きなブランドのコンサルティング[306]をしたり、ダイエットコークを提供する会議[307]で話したりしています。ダンジョンズ＆ドラゴンズのプロプレイヤーにも挑戦しているが、なかなかうまくいかない。シアトルの坂を自転車で登るのが日課。

298. https://ahrefs.com/
299. https://www.meetup.com/RaleighSEO/
300. https://www.meetup.com/beerandseo/
301. https://raleighseomeetup.org/conference/
302. https://www.reddit.com/r/TechSEO
303. https://ahrefs.com/blog/
304. http://onely.com/
305. https://www.ziptie.dev/
306. https://www.ianlurie.com/digital-marketing-consulting/
307. https://www.ianlurie.com/speaking/

部 II 章 9
アクセシビリティ



Alex Tait、 *Scott Davis*、 *Olu Niyi-Awosusi*、 *Gary Wilhelm* と *Katriel Paige* によって書かれた。
Eric Bailey、 *Cassey Lottman*、 *Shaina Hantsis*、 *Estelle Weyl*、 *Gigi Rajani* と *Carlie Dixon* によってレビュ。
David Fox による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

インターネットは年々拡大し、2021年1月現在、46億6000万人のアクティブインターネットユーザー[308]が存在すると言われています。しかし残念ながら、この章を通して見るように、アクセシビリティはこの成長とともに実質的に改善されていません。インターネットソリューションへの依存度が高まるにつれ、ウェブに平等にアクセスできない人々の疎外感も高まっています。

2021年は、現在進行中のCOVID-19のパンデミックの2年目にあたります。COVID -19の長期的な影響により、障害者人口が増加していることは明らかです[309]。COVID-19の長期的な健康への影響と同時に、パンデミックの結果、社会全体がデジタルサービスへの依存度を高めています。誰もがより多くの時間をオンラインで過ごし、より多くの必要な活動もオンラ

308. https://www.statista.com/statistics/617136/digital-population-worldwide/
309. https://www.scientificamerican.com/article/a-tsunami-of-disability-is-coming-as-a-result-of-lsquo-long-covid-rsquo/

インで完了するようになったのです。カナダ統計局インターネット利用調査[310]によると、「15歳以上のカナダ人の75%がパンデミックの発生以来、さまざまなインターネット関連の活動に従事する頻度が増えた」という。

また、パンデミックによって商品やサービスが急速にオンラインに移行しています。このマッキンゼーのレポート[311]によると、「おそらくもっと驚くべきことは、デジタルまたはデジタル的に強化された提供物の作成のスピードが上がっていることです。地域別に見ると、企業がこうした[オンライン]製品やサービスを開発する割合は、平均で7年増加していることが示唆された。

ウェブ・アクセシビリティとは、機能と情報の平等性を実現することで、障害者がインターフェイスのあらゆる側面に完全にアクセスできるようにすることです。デジタル製品やウェブサイトは、誰にでも使えるものでなければ、完全なものとは言えません。デジタル製品が特定の障害者を排除している場合、これは差別であり、罰金や訴訟の対象となる可能性があります。昨年は、米国障害者法に関連する訴訟が20%増加[312]しました。

悲しいことに、年々、私たちや、WebAIM Million[313] のような分析を行う他のチームは、これらの指標にほとんど改善が見られないことを発見しています。WebAIMの調査では、ホームページの97.4%がアクセシビリティの障害を自動的に検出しており、これは2020年の監査より1%も低いものでした。

すべてのLighthouse Accessibility[314]監査データのサイト総合スコアの中央値は、2020年の80%から2021年の82%に上昇しました。この2%の増加は、正しい方向へのシフトを意味するものであることを期待しています。しかし、これらは自動チェックであり、開発者がルール エンジンをよりうまく破壊していることを潜在的に意味する可能性もあります。

私たちの分析は、自動化された測定基準のみに基づいているため、自動化されたテストは、インターフェイスに存在し得るアクセシビリティ・バリアのほんの一部しか捉えていないことを忘れてはいけません。アクセシブルなWebサイトやアプリケーションを実現するためには、障がい者によるマニュアルテストやユーザビリティテストなどの定性的な分析が必要です。

私たちは、もっとも興味深い洞察を6つのカテゴリーに分けました。

- 読みやすさ
- ページナビゲーションのしやすさ
- フォーム

310. *https://www150.statcan.gc.ca/n1/pub/45-28-0001/2021001/article/00027-eng.htm*
311. *https://www.mckinsey.com/business-functions/strategy-and-corporate-finance/our-insights/how-covid-19-has-pushed-companies-over-the-technology-tipping-point-and-transformed-business-forever*
312. *https://info.usablenet.com/2020-report-on-digital-accessibility-lawsuits*
313. *https://webaim.org/projects/million/*
314. *https://web.dev/i18n/ja/lighthouse-accessibility/*

- Web上のメディア
- ARIAによる支援技術のサポート
- アクセシビリティオーバーレイ

この章ではウェブにおける深刻な測定基準と実証されたアクセシビリティの過失を満載し、読者がこの作業を優先し、より包括的なインターネットへとシフトしていくよう、その実践を変えるきっかけになることを願っています。

本章では、人物優先の用語として「障害者」を使用することにしました。私たちは、多くの人がアイデンティティを優先する「障害者」という用語を好むことを認めます。私たちが選んだ用語は、どの用語が適切かを規定するものでは決してありません。

読みやすさ

Webアクセシビリティの重要な要素として、コンテンツを可能な限りシンプルかつクリアに読めるようすることが挙げられます。ページのコンテンツが読めない場合、その情報にアクセスできないだけでなく、アカウント登録や購入などのタスクを完了することができなくなります。

色のコントラスト、ページの拡大・縮小、言語の識別など、ウェブページには読みやすさや読みにくさを左右するさまざまな要素があります。

カラーコントラスト

色のコントラスト[315]とは、テキストやその他のページの成果物が周囲の背景に対してどの程度容易に目立つかを示すものです。コントラストが高いほど、人は内容を識別しやすくなります。Web Content Accessibility Guidelines[316] (WCAG) には、テキストとテキスト以外のコンテンツに対する最低限のコントラスト要件が定められています。

低コントラストのコンテンツは、色覚異常の方、軽度から中等度の視力低下の方、明るい場所で画面がまぶしいなど状況的に視聴が困難な方など、視聴困難な場合があります。

315. *https://www.a11yproject.com/posts/2015-01-05-what-is-color-contrast/*
316. *https://www.w3.org/WAI/standards-guidelines/wcag/*

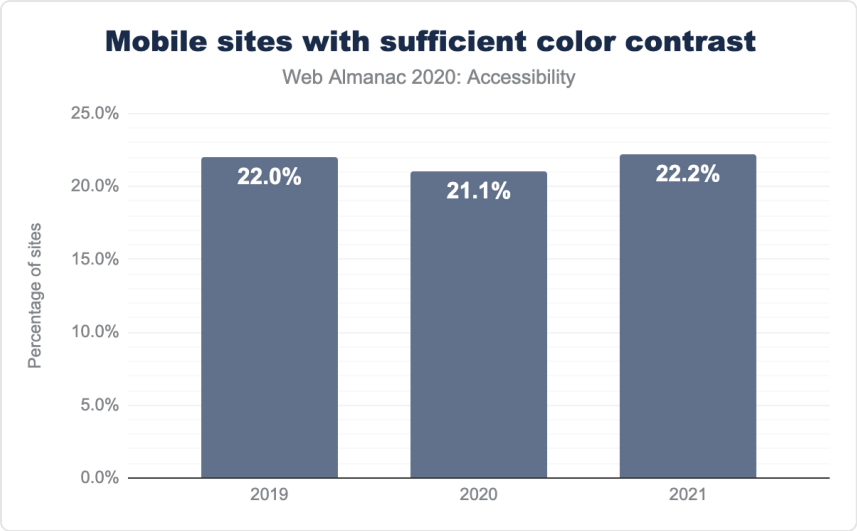


図9.1. 十分なカラーコントラストが確保されたモバイルサイト

今年、Lighthouseでカラーコントラストのスコアが合格だったサイトは、わずか22%であることがわかりました。非テキストコンテンツは非常に多様であるため、これらのスキャンはテキストベースのコントラストの問題しか捉えることができないことは注目に値します。このスコアは前年とほぼ同じで、2020年は21%、2019年は22%でした。テキストベースのコントラストの問題をキャッチすることは、一般的なさまざまな自動化ツールで可能であるため、この指標はややがっかりさせられるものです。

ズームとスケーリング

弱視の方は、ページの内容（とくに文字）を見るために、システム設定や画面拡大・縮小ソフトに頼ることがあります。Web Content Accessibility Guidelinesでは、とくにテキストは少なくとも200%[317]までリサイズできることが求められています。

Adrian Roselli[318] は、ユーザーに対してズームが有効でない場合に生じるさまざまな害について、包括的な記事[319]を書きました。現在、多くのブラウザは開発者がズームコントロールを上書きすることを防いでいますが、世界規模でブラウザやOSの使用範囲が広いことを考慮すると、すべてのブラウザがこの動作を上書きしているとは言い切れないため、コードレベルで回避する必要があります。

317. *https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-scale.html*
318. *https://x.com/aardrian*
319. *https://adrianroselli.com/2015/10/dont-disable-zoom.html*

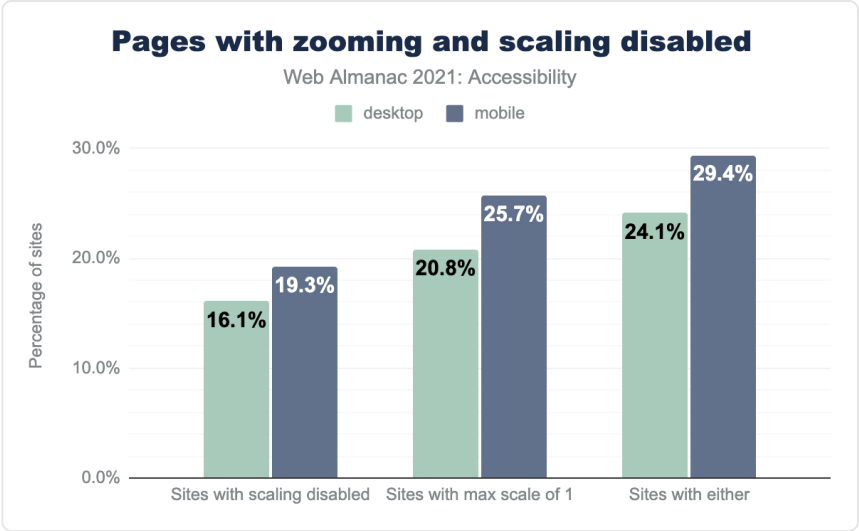


図9.2. 拡大・縮小が無効になっているページ。

デスクトップ用ホームページの24%とモバイル用ホームページの29%が、`maximum-scale`を1以下に設定するか、`user-scalable`を`0`または`none`に設定して、スケーリングを無効にしようとしていることがわかりました。

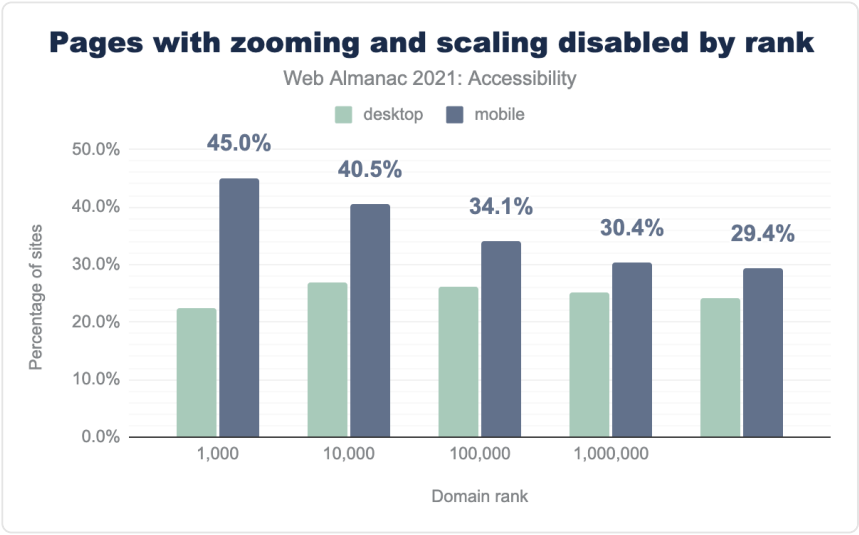


図9.3. ランクによってズームや拡大縮小が無効になっているページ。

とくに人気のあるサイトについて考えてみると、モバイルの数字が気になります。トラフィックの多い上位1,000サイトのうち、デスクトップサイトの22%、モバイルサイトの45%に、ユーザーのスケーリングを無効化しようとするコードがあります。これは、Webアプリケーションの普及から来る傾向かもしれません。人々は、コンテンツがWebサイトかWebアプリケーションかにかかわらず、Webブラウジングの体験（ズームや拡大縮小など）をカスタマイズできるようにする必要があるのです。

言語の識別



図9.4. モバイルサイトでは、有効な `lang` 属性があります。

HTMLの `lang` 属性を設定することで、ページの翻訳が容易になり、スクリーン・リーダーのサポートが向上し、一部のスクリーン・リーダーが読み上げるテキストに適切なアクセントや抑揚を適用することができるようになります。今年は、`lang` 属性が存在するサイトの割合が81%（2020年の78%から増加）となり、属性が存在するサイトのうち、99.7%が有効な `lang` 属性を持っていました。

文字サイズと行の高さ

最小フォントサイズや行の高さに関しては、WCAGからの具体的な要求はありません。しかし、基本のフォントサイズを16px[320]以上にすれば、誰もが読みやすくなるというのが一般的な見解です。とくに弱視の方。 ただし、テキストを200%まで拡大・縮小できるという要件があります。また、ユーザーはブラウザレベルで独自の最小フォントサイズを設定することができ、これらのカスタマイズ設定にも対応する必要があります。

320. *https://accessibility.digital.gov/visual-design/typography/*

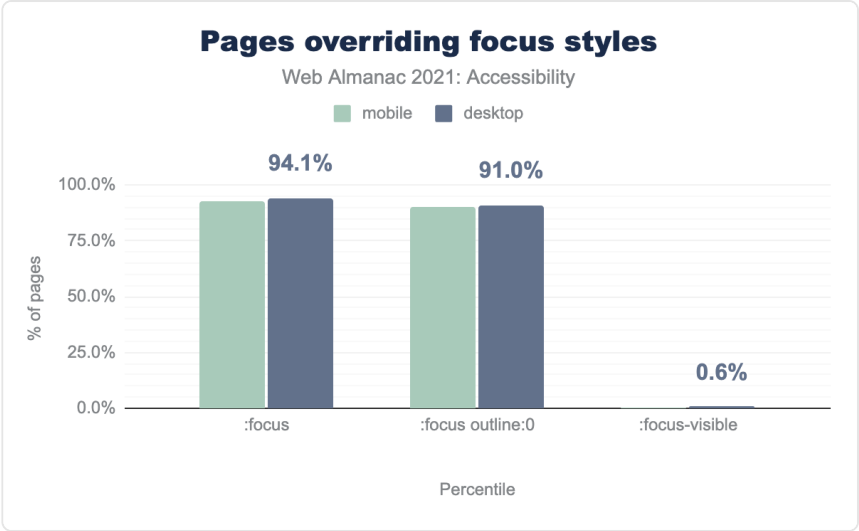


図9.5. フォント単位の使用状況。

フォントが `px` 単位で宣言されている場合、それらは静的なサイズである。ブラウザを拡大したときにフォントが適切に拡大縮小されるようにするには、`em` や `rem` などの相対単位を使用するのがもっともよい方法です。デスクトップのフォントサイズ宣言の68%は `px` 単位で、17%は `em` 単位で、5%は `rem` 単位で設定されていることがわかった。

フォーカススタイル

ビジブルフォーカススタイルは誰にとっても便利ですが、その存在に依存してナビゲートする目の見えるキーボードユーザーにとっては必要なものです。WCAGでは、すべての対話型コンテンツに 視認性の良いフォーカスインジケーター[321] を要求しています。

321. *https://www.w3.org/TR/UNDERSTANDING-WCAG20/navigation-mechanisms-focus-visible.html*

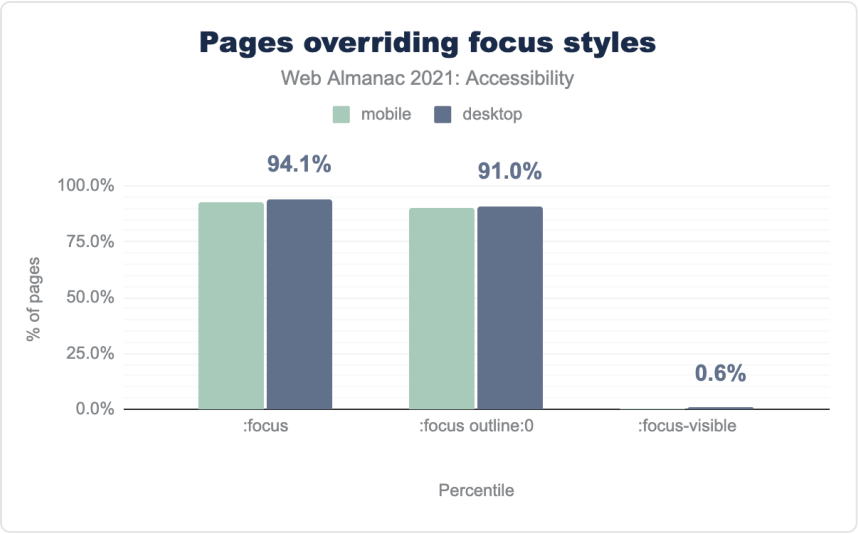


図9.6. フォーカススタイルをオーバーライドするページ。

ボタンやフォームコントロール、リンクなどのインタラクティブなコンテンツでは、CSSプロパティ `:focus { outline: none; }` や `:focus { outline: 0; }` を用いて、時には `:focus-within` や `:focus-visible` と共に、デフォルトのフォーカス表示を解除することがよくあります。デスクトップページの91%に `:focus { outline: 0; }` が宣言されていることがわかりました。場合によっては、より効果的なカスタムスタイルを適用できるように、この宣言は削除されます。残念ながら、多くの場合、単に削除されただけで置き換えられることはなく、キーボード・ユーザーにとって使い勝手の悪いページになってしまいます。

ブラウザのデフォルトのフォーカススタイルのいくつかの制限を含む、アクセシブルなフォーカス表示を実現する方法の詳細については、Sara Soueidan[322] の記事、"A guide to designing accessible, WCAG-compliant focus indicators"[323] がオススメです。

ユーザー嗜好のメディアクエリ、ハイコントラストへの対応

2020年に公開されたCSSメディアクエリ レベル5仕様[324]では、ユーザーがウェブサイト自体の外で設定したアクセシビリティ機能をウェブサイトが検出できるようにする*User Preference Media Features*というコレクションが導入されました。これらの機能は、通常、オペレーティングシステムやプラットフォームの環境設定によって構成されます。

322. *https://x.com/SaraSoueidan*
323. *https://www.sarasoueidan.com/blog/focus-indicators/*
324. *https://www.w3.org/TR/mediaqueries-5*

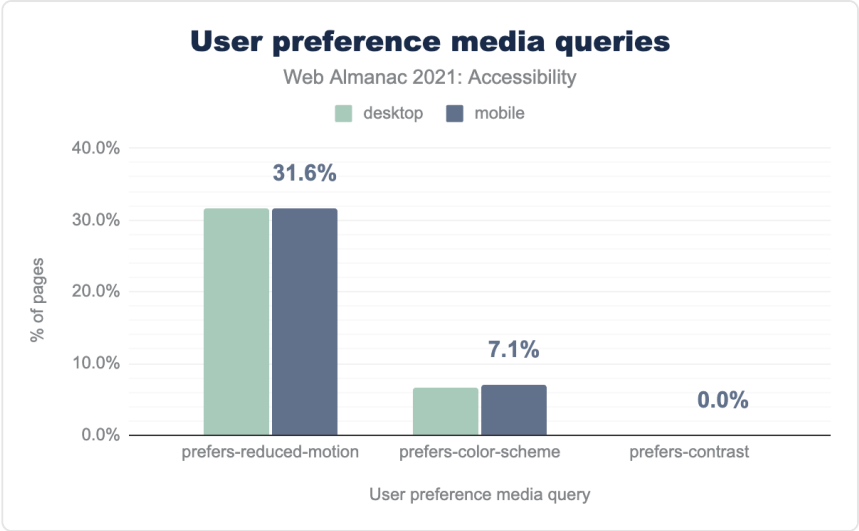


図9.7. ユーザー嗜好のメディアクエリ。

`prefers-reduced-motion` は、ウェブ制作者がウェブページ上のアニメーションやその他の動きのあるソースを通常はコンテンツを削除したり、置き換えたりして、より静的な体験へ置き換えるために使用します。 これは、画面上の速い動きによって注意力が散漫になったり、その他の誘因を受けたりするさまざまな人々を助けることができます。32%のウェブサイトが `prefers-reduced-motion` メディアクエリを使用していることがわかりました。

`prefers-reduced-transparency` は、エンドユーザーがオペレーティングシステムに半透明や透明効果を最小化または除去するように依頼したことを示します。このアフォーダンスは、エンドユーザーが読解を助けるため、あるいは視覚障害のあるユーザーに悪影響を与える可能性のある一般的な「ハロー効果」を避けるためオンにされるかもしれません。この比較的新しいメディアクエリの使用に関するデータはありません。

`prefers-contrast` (`high` または `low`) は、エンドユーザーが ハイコントラストかローコントラスト のコントラストテーマを好むことを示唆します。読解力や眼精疲労に効果が期待できます。この比較的新しいメディアクエリの使用に関するデータはありませんが、25%のウェブサイトが `ms-high-contrast` を使用しており、Windows特有のコントラスト優先の処理方法であることがわかっています。

`prefers-color-scheme` (`light` または `dark`) は、ユーザーが暗い背景の体験に明るい色を要求したり、その逆を行うことができるようにします。User Preference Media Queriesの中でもっとも早く導入されました。この機能は通称「ダークモード」対応と呼ばれ、2019年にAppleがiOS 13とiPadOSで標準化[325]したことにより一躍有名になりましたが、それ

325. *https://en.wikipedia.org/wiki/Light-on-dark_color_scheme#History*

以前からアクセシビリティ機能としては一般的な機能となっていました。

ダークモードはアクセシビリティのアフォーダンスとして多くの開発者やデザイナーに認識されていますが、実際には特定のユーザーにとってアクセシビリティを低下させる可能性があることに注意することが重要です。失読症や乱視の人の中には、暗い背景に明るいテキストは読みにくく[326]、ハロー効果を悪化させると感じる人がいるかもしれません。ここで重要なことは、ユーザーに最適なものを選んでもらうことです。7%のウェブサイトが `prefers-color-scheme` メディアクエリを使用していることがわかりました。

ページ移動のしやすさ

ウェブコンテンツを閲覧することは、私たちがオンラインに関わる基本的な方法の1つであり、これを実現する方法は数多くあります。人によっては、マウスでスクロールしながら、ページを視覚的にスキャンすることもあります。また、スクリーンリーダーを使ってページの見出しを操作することから始める人もいるかもしれません。ユーザーが迷ったり、探しているコンテンツを見つけられないということがないように、ウェブサイトは簡単に操作できる必要があります。

ランドマークとページ構成

ランドマークとは、HTMLの指定要素、または他のHTML要素に適用できるARIAの役割で、支援技術を必要とする障害者がページ全体の構造やナビゲーションをすばやく理解できるようにするためのものです。たとえば、ローターメニュー[327]は、ページの異なるランドマークへ移動するために使うことができ、またskip link は `<main>` ランドマークへ移動するために使うことができます。

HTML5導入以前は、これを実現するためにARIAのランドマークロールが必要でした。しかし、現在では、ランドマークページの構造の大部分を達成するために、ネイティブのHTML要素が利用できるようになっています。ネイティブのHTMLランドマーク要素を活用することは、ARIAロールを適用するよりも望ましいことです。ARIAの最初の規則[328]に従います。詳しくは、この章のARIAロールの節を参照してください。

326. *https://www.boia.org/blog/dark-mode-can-improve-text-readability-but-not-for-everyone*
327. *https://webaim.org/articles/voiceover/mobile#rotor*
328. *https://www.w3.org/TR/using-aria/#rule1*

HTML5 要素	ARIA ロールと同じ	要素のあるページ	ロールのあるページ	要素またはロールのあるページ
`<main>`	`role="main"`	27.68%	16.90%	35.00%
`<header>`	`role="banner"`	62.13%	14.34%	63.49%
`<nav>`	`role="navigation"`	61.69%	22.79%	65.53%
`<footer>`	`role="contentinfo"`	63.35%	12.21%	64.52%

図9.8. ランドマーク要素とロールの使い方（デスクトップ）。

ウェブページの大半が持つべきランドマークとして、もっとも一般的に期待されているのは、`<main>`、`<header>`、`<nav>`、`<footer>`です。デスクトップページの28%だけがネイティブなHTMLの`<main>`要素を持ち、17%が`role="main"`の要素を持ち、35%がどちらかの要素を持つことがわかりました。

たとえば、プライマリーサイトナビゲーションとパンくずセカンダリーナビゲーションなど、1つのページに同じランドマークが複数ある場合、それぞれに一意のアクセス可能な名前を付けることが重要です。これにより、支援技術を必要とする障害者は、どのナビゲーションランドマークに遭遇したかをよりよく理解できるようになります。これを実現するためのテクニックは、さまざまなランドマークと、異なるスクリーン リーダーのナビゲーション方法[329]に関する Scott O'Hara[330] の総合記事に網羅されています。

ドキュメントタイトル

説明的なページタイトルは、コンテキストが変更されると通知されるため、支援技術を使ってページ、タブ、ウィンドウ間を移動する際のコンテキストに役立ちます。

329. https://www.scottohara.me/blog/2018/03/03/landmarks.html
330. https://x.com/scottohara

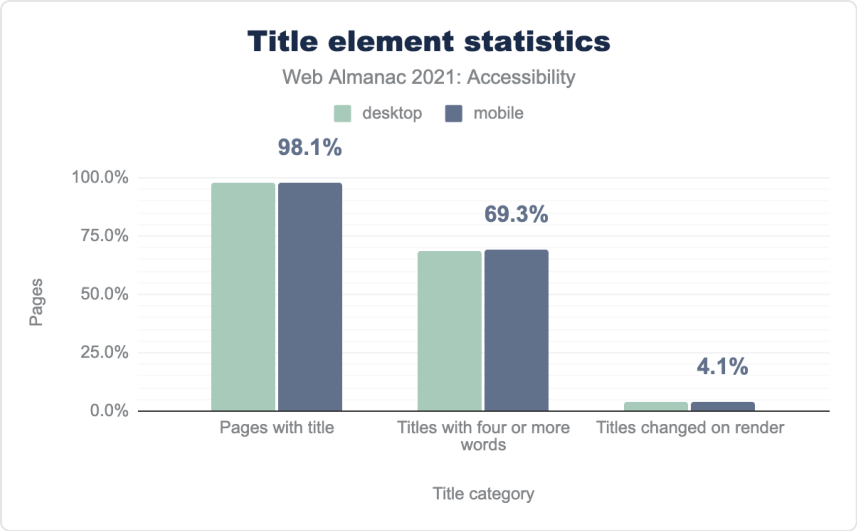


図9.9. Title要素の統計

当社のデータでは、98%のウェブページにタイトルがあります。しかし、そのうち4語以上のタイトルを持つページは68%に過ぎず、かなりの割合のウェブページが、ページの内容に関する十分な情報を提供するユニークで意味のあるタイトルを持っていない可能性があります。

セカンダリーナビゲーション

多くのユーザーは、Webサイトで探しているコンテンツを見つけるために、二次的なナビゲーションの方法を利用しています。WCAGは、複雑なウェブサイトには二次的なナビゲーション手段を設けることを要求しています[331]。もっとも一般的で有用なセカンダリーナビゲーションの方法のひとつに、検索メカニズムがあります。全サイトの24%が検索入力を使っていることがわかりました。

また、セカンダリーナビゲーションの方法として、サイトマップというWebサイト上のすべてのリンクを明確に整理したコレクションを実装する方法もあります。サイトマップの存在に関するデータはありませんが、このW3Cによるテクニックガイド[332]では、サイトマップとは何か、どのようにすれば効果的に実装できるかについて詳しく説明されています。

331. *https://www.w3.org/TR/UNDERSTANDING-WCAG20/navigation-mechanisms-mult-loc.html*
332. *https://www.w3.org/TR/WCAG20-TECHS/G63.html*

タブインデックス

`tabindex` は、要素に追加することで、その要素にフォーカスを当てることができるかどうかを制御することができる属性です。その値によって、その要素はキーボードフォーカス、つまり「タブ」順になることもできます。

`tabindex` の値が `0` の場合、その要素はプログラム的にフォーカス可能で、キーボードフォーカスの順番になることができます。ボタン、リンク、フォームコントロールなどのインタラクティブコンテンツは `tabindex` 値が `0` と同等であり、キーボードフォーカスの順番がネイティブであることを意味します。

インタラクティブでキーボードフォーカスの順番を意図したカスタム要素やウィジェットは、明示的に `tabindex="0"` を割り当てる必要があり、そうしないとキーボードで使用できなくなります。

もしある要素がフォーカス可能であるべきだが、キーボードフォーカスの順番にない場合、`tabindex` 値の `-1`（あるいは任意の負の整数）をフックとして使うことで、キーボードフォーカスの順番に加えずJavaScriptでプログラム的にその要素にフォーカスできるようにできます。これは、フォーカスを割り当てたい場合に便利です。たとえば、Marcy Sutton[333] が post on accessible client-side routing techniques[334] で取り上げた、単一ページのアプリケーションで新しいページに移動するとき見出しへフォーカスを当てるような場合などです。キーボードのフォーカス順に非インタラクティブ要素を配置すると、視覚障害者が混乱するため、避けるべきです。

ページのフォーカス順序は、常にドキュメントフロー、つまりドキュメント内のHTML要素の順序で決定されるべきです。`tabindex` に正の整数を設定すると、ページの自然な順序が上書きされ、しばしばWCAG 2.4.3 - Focus Order[335]の失敗を招きます。一般に、ページの自然なフォーカス順序を尊重することは、キーボードのフォーカス順序を過剰に操作するよりも、よりアクセシブルな体験につながります。

その結果、デスクトップサイトの58%、モバイルサイトの56%が、`tabindex` 属性を何らかの形で利用していることがわかりました。

333. *https://x.com/marcysutton*
334. *https://www.gatsbyjs.com/blog/2019-07-11-user-testing-accessible-client-routing/*
335. *https://www.w3.org/TR/UNDERSTANDING-WCAG20/navigation-mechanisms-focus-order.html*

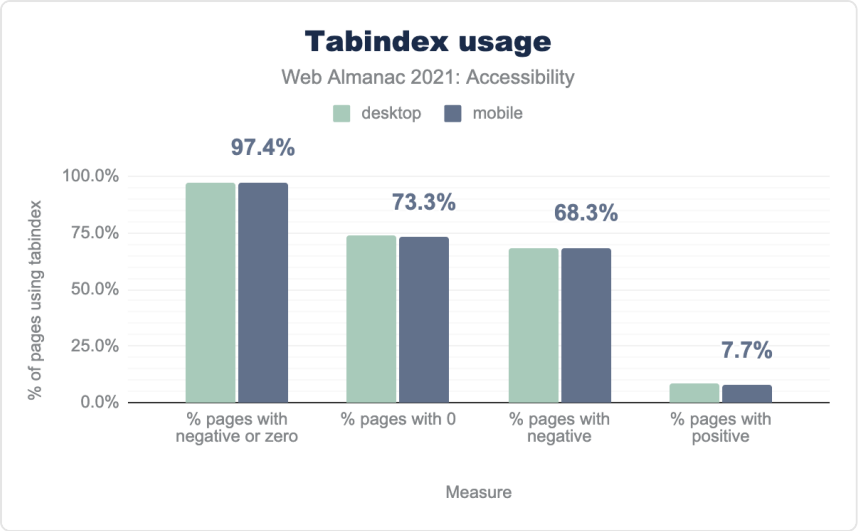


図9.10. `tabindex` の使用状況

少なくとも1つの `tabindex` 属性のインスタンスを持つデスクトップページを見ると

- 96.9%は `0` の値を使用しており、これは要素がフォーカス可能で、キーボードフォーカスの順番に追加されることを意味します。
- 68.2%が負の整数を使用、つまりキーボードのフォーカス順から明示的に要素を削除
- 8.7%は正の整数値であり、ウェブ制作者がDOM構造に任せるのではなく、フォーカスの順番をコントロールしようとしていることを意味します。

`tabindex` 属性には有効な宣言がありますが、これらのテクニックに誤って手を出すと、多くのキーボードや支援技術を必要とする障害者にとって共通のアクセシビリティの障害になります。正の整数を `tabindex` に使うことの落とし穴については、 Karl Groves[336] の記事、“0以上の `tabindex` 値を使用することが悪い理由” がオススメです。

リンクをスキップする

キーボードに頼った操作をする人のために、スキップリンクを用意しました。複数のページで繰り返されるコンテンツのセクションやナビゲーションセクションをスキップして、別の目的地、通常はページの `<main>` 要素に移動できます。スキップリンクは通常、ページの

336. *https://x.com/karlgroves*

最初の要素であり、UIで永続的に表示するか、キーボードフォーカスがあるまで目に見えるように隠すことができます。たとえば、多くのインタラクティブなコンテンツ（リンクでいっぱいの堅牢なナビゲーションシステムなど）は、画面のメインコンテンツへ到達する前にタブで移動するのが非常に面倒で、とくにこれらが複数のページにわたって繰り返される傾向があります。

情報量の多いサイトでは、ユーザーがよくアクセスする場所にジャンプできるように、スキップリンクをいくつか用意しているところもあります。たとえば、カナダ政府のウェブサイト[337]には、「メインコンテンツへスキップ」「政府についてへスキップ」「基本HTMLバージョンへ切り替え」の3つがあります。

スキップリンクは、ブロックのバイパス[338]とみなされます。しかし、デスクトップとモバイルのサイトの20%*近く*がスキップリンクを使用している可能性が、あることがわかりました。これは、ページの最初の3つのリンクのうち1つに `href="#main"` 属性があるかどうかを調べることで判断しました。これは、スキップリンクの一般的な実装です。

見出しの階層

見出しは、スクリーンリーダーがページを適切に操作できるよう、目次のような階層を提供します。

58%

図9.11. モバイルサイトは、見出しの順序が適切かどうかのLighthouse監査に合格しています。

私たちの監査では、チェックしたサイトの58%が、見出しテスト[339]で、レベルを飛ばさないテストに合格していることがわかりました。 WebAIMが2021年に調査したスクリーン リーダーのユーザーの[340] 85% 以上が、ウェブのナビゲーションに見出しが役に立つと回答しています。見出しを正しい順序で表示すること、つまり、レベルを飛ばすことなく昇順に表示することは、支援技術を必要とする障害者が最高の体験をすることを意味します。

テーブル

テーブルは、データを2軸の関係で効率的に表示することができるため、比較検討などに有効です。支援技術を必要とする障害者は、ユーザーが効果的にナビゲートし、理解し、対話

337. https://www.canada.ca/
338. https://www.w3.org/WAI/WCAG21/Understanding/bypass-blocks.html
339. https://web.dev/i18n/ja/heading-order/
340. https://webaim.org/projects/screenreadersurvey9/#heading

できるように機械可読構造を提供する特定のテーブルマークアップに依存しています。

テーブルは、キャプション、適切なヘッダーとフッター、すべてのデータセルに対応するヘッダーセルを含む、適切な要素と定義された関係を持つ整形された構造である必要があります。スクリーンリーダーのユーザーは発表された内容を通じて、このような明確に定義された関係に依存しているため、不完全または不正確に宣言された構造は、誤解を招いたり、情報が欠落したりする可能性があります。

	表サイト		*全サイト*	
	デスクトップ	*モバイル*	*デスクトップ*	*モバイル*
キャプション付きテーブル	*5.4%*	*4.6%*	*1.2%*	*1.0%*
プレゼン用テーブル	*1.2%*	*0.9%*	*0.5%*	*0.4%*

図9.12. アクセシブルなテーブルの使用状況。

テーブルキャプション

テーブルのキャプションは、テーブル全体の見出しとして機能し、その情報の要約を提供します。テーブルにラベルを付ける場合、`<caption>` 要素はスクリーン・リーダーのユーザーにもっとも多くのコンテキストを提供する正しい意味での選択ですが、テーブルのための他の代替キャプション技術[341] もあることに注意すべきです。

テーブル全体の見出し要素は、`<caption>` 要素が適切に実装されていれば、しばしば不要となり、`<caption>` 要素は見出しに似たスタイルと視覚的な位置づけが可能です。テーブル要素が存在するデスクトップとモバイルのサイトのうち、`<caption>` を使用しているのはわずか5%で、2020年からわずかに増加しました。

レイアウト用テーブル

Flexbox[342] や Grid[343] などのCSSメソッドの導入は、ウェブ開発者が簡単に流体対応のレイアウトを作成できるようにする機能を提供したのです。Flexbox[344] や Grid[345] などのCSSメソッドの導入は、ウェブ開発者が簡単に流動的なレスポンシブレイアウトを作成できる機能を提供したのです。それ以前は、テーブルをデータの表示ではなく、レイアウトのために使うことが多かった。残念ながら、レガシーなWebサイトやレガシーな開発手法の組み合わせにより、レイアウトにテーブルを使用しているWebサイトがまだ存在しています。このレガシーな開

341. *https://www.w3.org/WAI/tutorials/tables/caption-summary/*
342. *https://www.w3schools.com/css/css3_flexbox.asp*
343. *https://www.w3schools.com/css/css_grid.asp*
344. *https://www.w3schools.com/css/css3_flexbox.asp*
345. *https://www.w3schools.com/css/css_grid.asp*

発手法が今もどの程度使われているのか、判断するのは難しい。

この技術に手を伸ばす絶対的な必要がある場合、支援技術がテーブルのセマンティクスを無視するように、`プレゼンテーション`の役割をテーブルに適用する必要があります。デスクトップとモバイルのページの1%が、プレゼンテーションの役割を持つテーブルを含んでいることがわかりました。これが良いことなのか悪いことなのか、判断がつきません。これは、プレゼンテーション用のテーブルが少ないということもありますが、レイアウト用のテーブルが不足しているだけである可能性が高いのです。

タブ

タブは非常に一般的なインターフェイスウィジェットですが、それをアクセシブルにすることは多くの開発者にとって課題となっています。アクセシブルな実装のための一般的なパターンは、WAI-ARIAオーサリングプラクティスのデザインパターン[346]に由来しています。ARIAオーサリングプラクティス文書は仕様書ではなく、一般的なウィジェットに対するARIAの理想的な使い方を示すことを意図していることに注意してください。これらは、ユーザーによるテストなしに実運用で使用すべきではありません。

オーサリングプラクティスのガイドラインでは、常に`tabpanel`ロールと`role="tab"`を組み合わせて使用することを推奨しています。デスクトップページの8%が`role="tablist"`を持つ要素を持ち、7%が`role="tab"`を持つ要素を持ち、6%が`role="tabpanel"`を持つ要素を持つことが分かっています。詳しくは、以下のARIAロールの項をご覧ください。

キャプチャ

一般公開されているWebサイトには、人間とWebを巡回するコンピューターの2種類の訪問者が定期的に訪れます。人間の訪問者を惹きつけるために、ウェブサイトは検索エンジンに大きく取り上げられることを望みます。検索エンジンはクローラーと呼ばれる自動化されたプログラムを送り込み、ウェブサイトを訪問し、見て回り、その結果を検索エンジンに報告し、コンテンツを分類・整理している。

たとえばWeb Almanacは、毎年、同様のウェブクローラーを派遣して、約800万種類のウェブサイトの情報を収集し、作成しています。そして、その結果を著者が要約し、読者に提供します。

Webサイトが、訪問者が人間であることを確認したい場合、Web制作者は、人間が理論的にパスでき、コンピューターがパスできないテストを設置するテクニックを用いることがあります。このような「人間限定」のテストは、キャプチャと呼ばれます。「コンピューターと

346. *https://www.w3.org/TR/wai-aria-practices-1.1/#tabpanel*

人間を区別するための完全に自動化された公開チューリングテスト」です。

10.2%

図9.13. キャプチャを使用したデスクトップサイト

デスクトップとモバイルの両方で、訪問したウェブサイトのおよそ10%にキャプチャが、あることがわかりました。

キャプチャには、アクセシビリティを阻害する可能性のあるものが多数存在します。たとえば、もっとも一般的なキャプチャの1つは、波打った歪んだテキストの画像を表示し、そのテキストを解読して入力するようユーザーに要求します。この種のテストは、誰にとっても解きにくいものですが、弱視やその他の視覚・読書関連の障害を持つ人にとっては、より困難なものになると思われます。あるユーザビリティ調査では、およそ3人に1人のユーザーがキャプチャの解読に失敗していることが分かっています[347]。

もしキャプチャがaltテキスト を含んでいれば、答えはプレーンテキストとして提供されるので、コンピューターがテストをパスすることは些細なことでしょう。しかし、altテキストを含めないことで、キャプチャはスクリーンリーダーやそれを使用する視覚障害者を排除していることになります。

キャプチャがもたらすアクセシビリティの障壁については、W3Cの論文をオススメします。「キャプチャにアクセスできない：Webでのビジュアルチューリングテストの代替案」[348]。

論文より「セキュリティソリューションとしてキャプチャを使うことは、ますます効果がなくなってきていることを認めることが重要である。2段階認証やマルチデバイス認証などの代替セキュリティ手法や、高い信頼性で人間のユーザーを識別する新しいプロトコルも、セキュリティとアクセス性の両方の理由から、従来の画像ベースのキャプチャ手法よりも優先して慎重に検討すべきである。」

フォーム

フォームは、ウェブへのアクセスを左右するものであり、それはますます社会参加や必要不可欠なサービスへのアクセスを意味するようになっています。多くの人が銀行、食料品の買い物、飛行機の予約、予約のスケジュール、仕事など、さまざまな活動をオンラインで行っています。

COVID-19の大流行の影響により、2021年には数百万人の子どもたちがオンラインで学校へ

347. https://baymard.com/blog/captchas-in-checkout
348. https://www.w3.org/TR/turingtest/

通うようになりました。これらのサービスはすべて、最低でも登録とサインインのためのフォームを必要とし、多くは財務情報など他の機密情報を必要とする、より複雑なフォームを持っています。アクセスできないフォームは差別的であり、深刻な被害をもたらす可能性があります。

2019年に英国で行われたクリック・アウェイ・パウンドの調査は、「障害者のオンラインショッピング体験を探り、障害者の買い物客を無視した場合のビジネスへのコストを検証する」ことを目的としています。その結果、英国企業はウェブサイトのアクセシビリティの障壁のために放棄されたショッピングカートで170億ポンド以上の売上を逃すことがわかりました。利益が障害者の権利を尊重する第一の理由であってはなりませんが、ビジネスケースは非常に充実しています。

`<label>` 要素

HTMLフォームをアクセシブルにするもっとも重要な方法の1つは、`<label>`要素を使って、フォームコントロール[349]を説明する短い記述テキストをプログラムでリンクさせることです。これは通常、`<label>`要素の`for`属性とフォームコントロール要素の`id`属性をマッチングさせることで行われます。たとえば

```
<label for="first-name">First Name</label>
<input type="text" id="first-name">
```

ウェブ開発者が`<label>`要素を入力に関連付けない場合、そうでなければ無料で手に入るはずの多くの重要な機能を逃していることになるのです。たとえば、`<label>`が`<input>`フィールドに適切に関連付けられている場合、`<label>`をタップまたはクリックすると自動的に`<input>`フィールドにフォーカスが移ります。 これは、ユーザビリティの大きなメリットであるだけでなく、Web上で期待される動作でもあります。

349. *https://developer.mozilla.org/docs/Learn/Forms/Basic_native_form_controls*

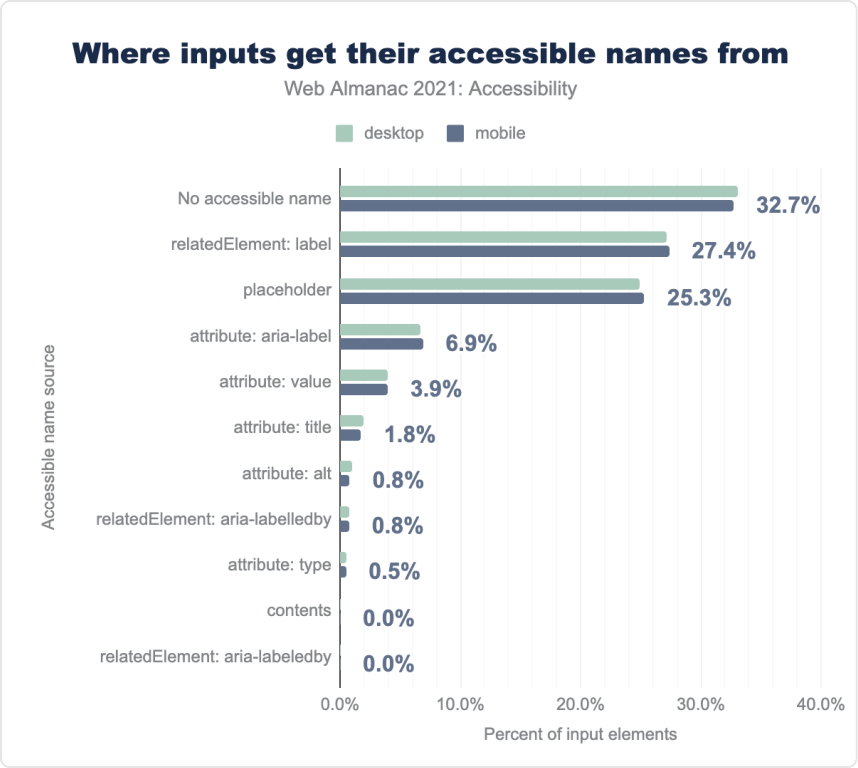


図9.14. 入力のアクセス可能な名称はどこから来るのか

`<label>` 要素は、1999年のHTML4で導入されました。過去20年以上にわたってすべてのモダンブラウザで利用可能であるにもかかわらず、すべての `<input>` 要素のうち、プログラムによって関連付けられたラベルからアクセス可能な名前を得るのは27%のみで、32%の入力要素には アクセス可能な名前 がまったくありません。

もっとも重要なことは適切なアクセス可能な名前がないと、スクリーン・リーダーや音声合成の利用者が、フォーム・フィールドの目的を狙ったり、特定したりできない可能性があることです。入力に関連する `<label>` 要素は、これを行うためのもっとも強固で期待できる方法です。

これは、エンドユーザーがはじめてフォームに入力するときだけでなく、フォームの検証で特定のフィールドにエラーが見つかり、ユーザーがフォームを送信する前に修正しなければならない場合にも同様に重要なことです。たとえば、クレジットカードの有効期限を記入し忘れた場合、ユーザーは購入手続きを行うことができません。また、入力漏れのあるフィールドを見つけ、入力の目的とエラーを修正するために必要な手順の両方を理解できなければ、購入を完了することはできません。

入力のラベル付けに `placeholder` 属性の不適切な使用について

2014年にHTML5で導入された`placeholder`属性。その使用目的は、ユーザーから提供されることが予想されるデータの例を提供することです。たとえば、`<input type="text" id="credit-card" placeholder="1234-5678-9999-0000">`は入力フィールドにプレースホルダーをかすかなテキストとして表示し、ユーザーがフィールドに入力を始めた瞬間に消えます。

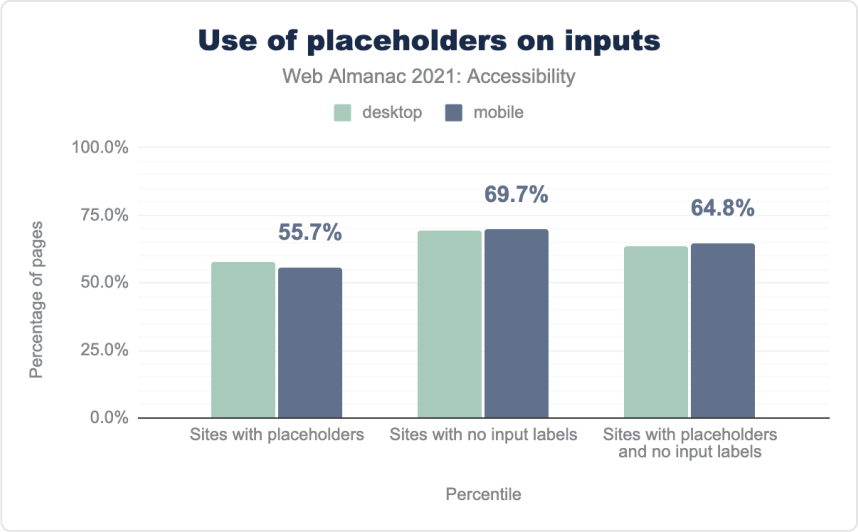


図9.15. 入力にプレースホルダーを使用。

`<label>`要素の代わりにプレースホルダーを使うという不適切な使い方は、驚くほど多く見受けられます。今年の調査では、デスクトップおよびモバイルWebサイトの約58%が`placeholder`属性を使用していました。 これらのサイトのうち、65%近くが`placeholder`属性を含み、プログラムで関連付けられた`<label>`要素を含めることができませんでした。

プレースホルダーのテキストがもたらす多くのアクセシビリティの問題[350]が存在します。たとえば、ユーザーが入力を始めると消えてしまうため、認知障害のある人は、フォーム要素の目的に対する文脈を見失う可能性があります。

HTML5仕様[351]には、”placeholder属性はラベルの代わりとして使用してはならない“と明確に記載されています。

350. *https://www.smashingmagazine.com/2018/06/placeholder-attribute/*
351. *https://html.spec.whatwg.org/#the-placeholder-attribute*

W3Cのプレースホルダー調査[352]は、意味的に正しい `<label>` 要素の代わりにプレースホルダーを使うという欠陥のある設計手法に反対する、26の異なる記事をリストアップしています。そして、こう言っています。

> *ラベルの代わりにplaceholder属性を使用すると、高齢者や認知、移動、運動、視覚に障害のあるユーザーを含む様々なユーザーにとって、コントロールのアクセシビリティとユーザビリティが低下する可能性があります。*
>
> *– W3Cのプレースホルダーに関する研究*[353]

情報提供の必要性

ウェブ開発者がエンドユーザーから情報を収集する場合、どの情報がオプションで、どの情報が必要なのかを明確に示す必要があります。たとえば、エンドユーザーがオンラインで何かをダウンロード購入する場合、配送先の住所は任意です。しかし、支払い方法は、販売を完了するために必要である可能性が高いです。

2014年にHTML5が `<input>` フィールドに `required` 属性を導入する以前は、Web開発者はこの問題をアドホックにケースバイケースで解決することを余儀なくされました。一般的な慣例は、必須入力フィールドのラベルにアスタリスク（ `*` ）を付けることです。これは純粋に視覚的、様式的な慣習であり、アスタリスクの付いたラベルはいかなる種類のフィールド検証を強制するものではありません。さらに、スクリーン・リーダーは、この文字が明示的に支援技術から隠されていない限り、通常「星」とアナウンスするので、混乱することがあります。

フォームフィールドの必要な状態を支援技術に伝えるため使用できる属性は2つあります。 `required` 属性は、ほとんどのスクリーンリーダーによって通知され、必須フィールドが適切に記入されていない場合、実際にはフォームの送信を阻止します。 `aria-required` 属性は、支援技術に必須フィールドを示すために使用できますが、フォーム送信を妨害するような関連した動作は付属していません。

352. https://www.w3.org/WAI/GL/low-vision-a11y-tf/wiki/Placeholder_Research
353. https://www.w3.org/WAI/GL/low-vision-a11y-tf/wiki/Placeholder_Research

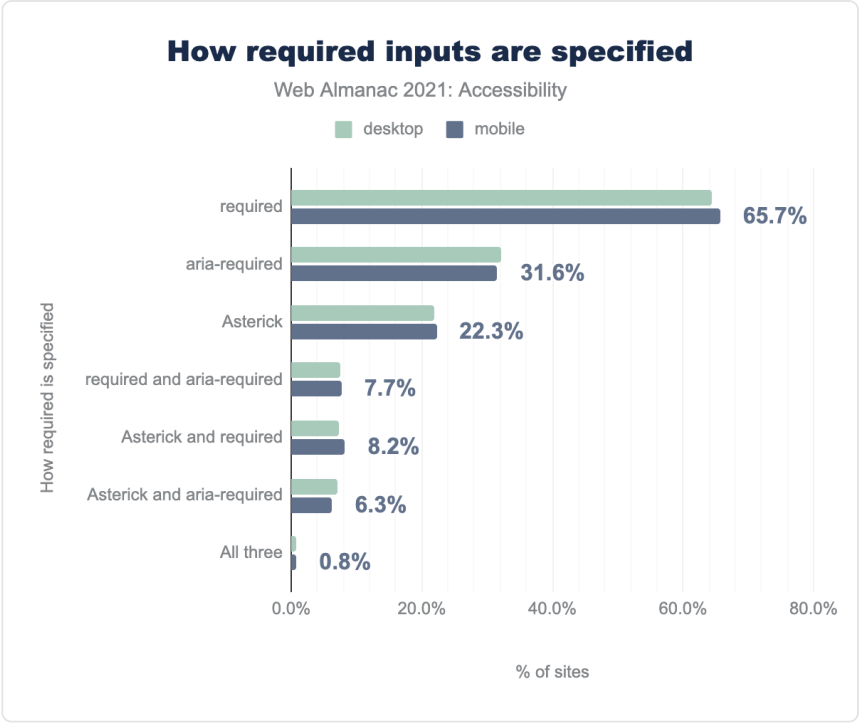


図9.16. 必要な入力の指定方法

デスクトップ型ウェブサイトの21%に、ラベルにアスタリスク（`*`）、`required`属性、`aria-required`属性のいずれか、またはこれらの手法を組み合わせたフォーム要素がありました。これらのフォーム要素の3分の2は`required`属性を使用していました。必須入力の約3分の1は`aria-required`属性を使用しています。およそ22%はラベルにアスタリスクを使用していました。

ウェブ上のメディア

アクセシビリティは、ウェブ上のあらゆるメディア消費において、ますます重要な役割を果たすようになっています。聴覚障害者のために、キャプションはビデオへのアクセスを提供します。目が不自由な人や視覚に障害がある人は、音声ガイドで情景を説明できます。メディアコンテンツへのアクセスの障壁を取り除かない限り、私たちはウェブ上で訪問されるものの大半から人々を排除していることになります。

このStreaming Mediaの調査[354]によると、「2022年までに、動画視聴はすべてのインターネットトラフィックの82%を占めるようになる」そうです。ウェブコンテンツで画像、音声、動画などのメディアを使用する場合は、必ずすべての人がアクセスできるようにする必要があります。

代替テキストの概要

すべてのHTMLメディア要素では、テキストの代替を提供できますが、すべての作者がこのアクセシビリティ機能を利用しているわけではありません。

1995年のHTML 2.0仕様で、画像を表示するための `<img>` 要素が導入された。同時に導入された `alt` 属性は、ウェブ開発者が画像に代わるテキストを提供するための明確なメカニズムを提供します。

この画像の代替説明文は、スクリーンリーダーが画像を見ることができない人のために画像を説明するために使用されます。また、画像をダウンロードまたは表示できない場合に、すべての人に画像を説明するために使用されます。画像を見ることのできないユーザーとは、検索エンジンのことです。検索エンジン最適化（SEO）では、画像を表示したウェブページをテキスト検索で発見できるよう、優れた `alt` テキストが重要な役割を担います。

HTML5仕様では、2014年に `<video>` と `<audio>` 要素が導入され、サードパーティのブラウザープラグインを必要としない、リッチメディアをウェブサイトに組み込むための標準ベースの方法が提供されました。`<video>` と `<audio>` の両方の要素に `<track>` を含めることができ、クローズドキャプション、字幕、音声ディスクリプションは、リッチメディアを楽しむためのテキストベースの代替手段を提供できます。

このトラックは、画像に対する `alt` テキストと同様のSEO効果をもたらしますが、2021年の調査では、 `<track>` 要素を提供しているウェブサイトは1%未満でした。

画像

`alt` 属性は、ウェブ制作者が画像で伝達される視覚情報の代替テキストを提供することを可能にします。スクリーン・リーダーは、画像の代替テキストをアナウンスすることで、その視覚的な意味を音声で伝えることができます。また、画像を読み込むことができない場合は、説明文の代替テキストが表示されます。

画像は適切に説明する必要があります。短い説明文が役立つ場合もあれば、画像の意味や意図を理解するために長い説明文が必要な場合もあります。

2021年のLighthouse監査データによると、57%のサイトが `alt` テキスト付き画像のテスト

354. *https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=144177*

をパスし、前年の54%からわずかに上昇しました。このテストは `img` 要素に `alt`、`aria-label` または `aria-labelledby` 属性のうち少なくとも1つが存在するかどうかを調べます。ほとんどの場合、`alt` 属性を使用するのがもっとも良い選択です。

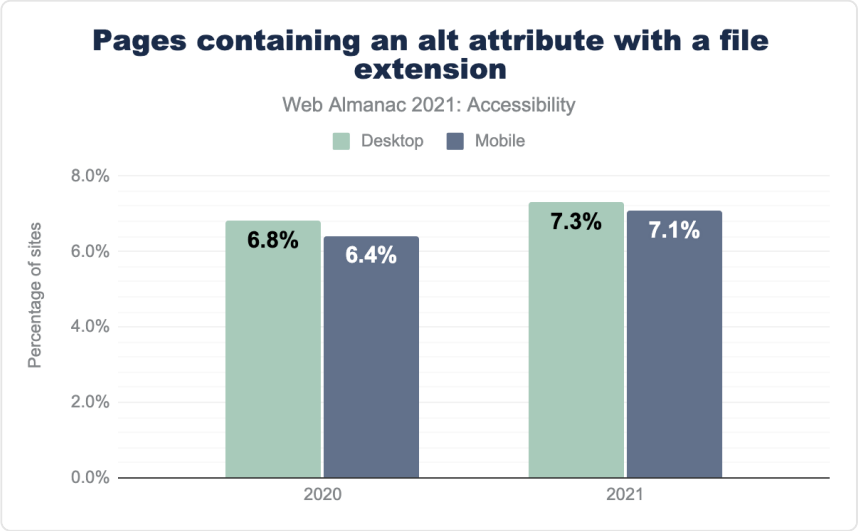


図9.17. ファイル拡張子を持つ `alt` 属性を含むページ。

代替テキストの有無の自動チェックは、通常、このテキストの品質を評価するものではありません。役に立たないパターンの1つは、画像のファイル拡張子名での説明です。私たちは、デスクトップ・サイト（`alt` 属性のインスタンスを少なくとも1つ持つ）の7.1%が、少なくとも1つの `img` 要素の `alt` 属性の値にファイル拡張子を持つことを発見しました（前年は7.3%でした）。

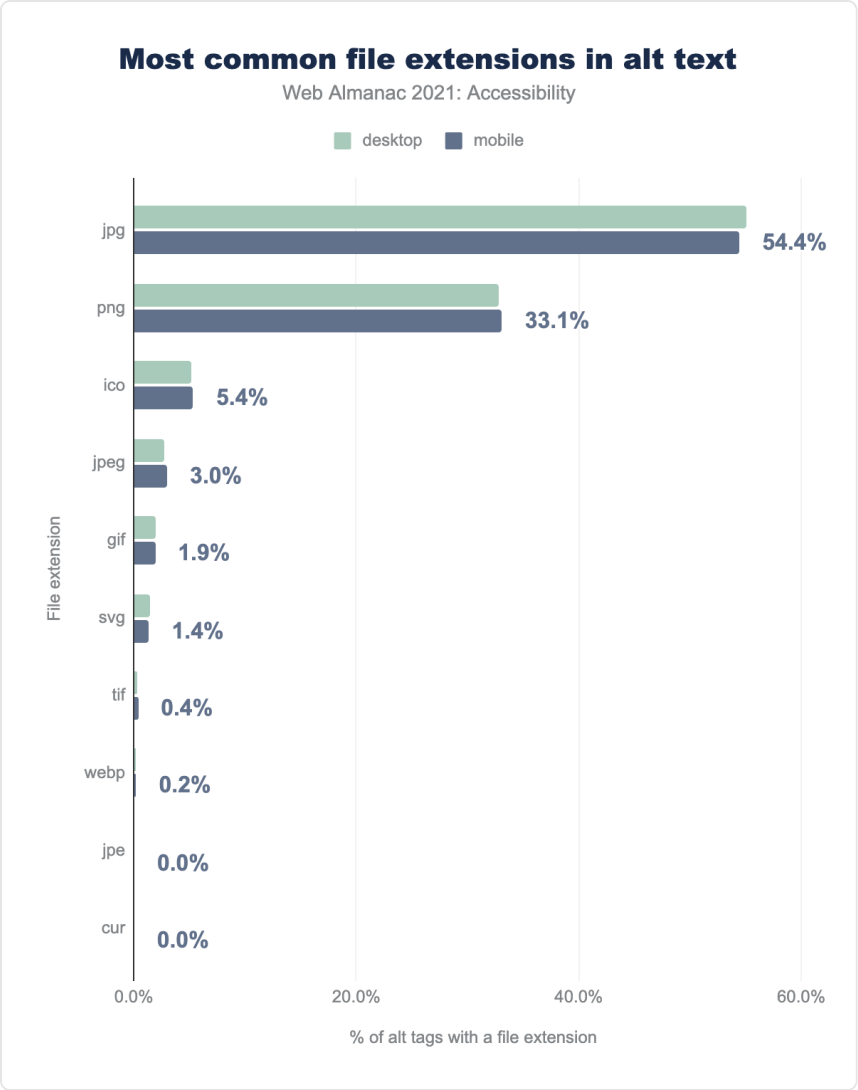


図9.18. もっとも一般的なファイル拡張子を `alt` テキストで表示します。

テキスト値 `alt` に明示的に含まれるファイル拡張子のトップ5（空ではない `alt` 値を持つ画像を持つサイトでは）は、`jpg`、`png`、`ico`、`gif`、`jpeg` となっています。これは、CMSやその他の自動生成された代替テキストの仕組みに由来していると思われます。これらの `alt` 属性値は、どのように実装されているかにかかわらず、意味のある値であることが不可欠です。

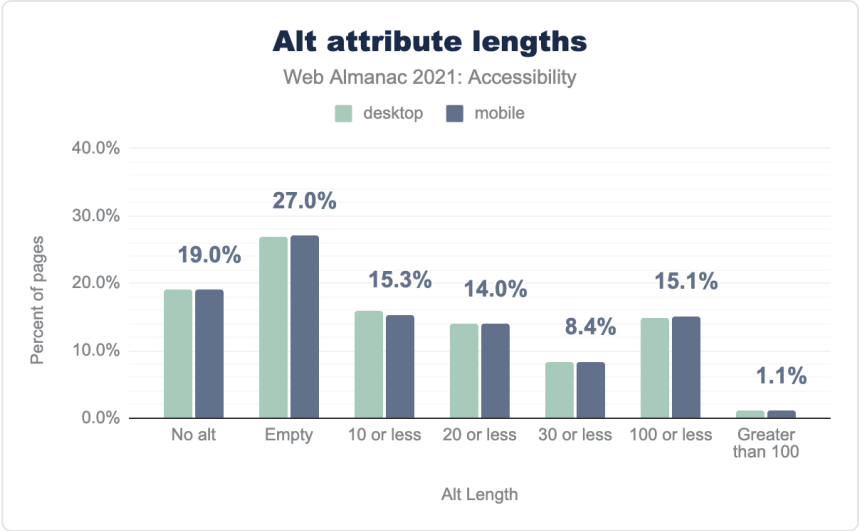


図9.19. `alt` 属性の長さ。

その結果、`alt` のテキスト属性の27%は空であることが判明した。理想的な世界では、これに関連する画像は装飾的[355]であり、支援技術で記述されるべきものではありません。しかし、大半の画像はインターフェイスに付加価値を与えるものであり、そのような画像は説明されるべきものです。その結果、10文字以下の画像が15%あり、ほとんどの画像に対して妙に短い説明文になっており、情報の同等性が達成されていないことがわかりました。

音声

`<track>` は、`<audio>` と `<video>` 要素の音声と同等のテキストの方法を提供します。これにより、永久的または一時的な難聴の方でも、音声コンテンツを理解できます。



図9.20. デスクトップの `<audio>` 要素には、少なくとも1つの `<track>` 要素が付随しています。

`<track>` は1つまたは複数のWebVTTファイルを読み込み、テキストコンテンツと説明する音声を同期させることができます。デスクトップでは全ページの0.02%、モバイルでは全ペ

355. *https://www.w3.org/WAI/tutorials/images/decorative/#:~:text=For%20example%2C%20the%20information%20provided,technologies%2C%20such%20as%20screen%20readers.*

ージの0.05%が、検出可能な `<audio>` 要素に少なくとも1つの `<track>` 要素が付随していることがわかりました。

これらのデータポイントには、 `<iframe>` 要素で埋め込まれた音声は含まれていません。これは、録音をホストしてリスト化するためにサードパーティのサービスを使用するポッドキャストのようなコンテンツによくあることです。

動画

2021年版Web Almanacに掲載されたWebサイトのうち、 `<video>` 要素は約5%にしか存在しませんでした。



図9.21. `<video>` 要素を持つデスクトップWebサイトには、少なくとも1つの `<track>` 要素が付随しています。

`<audio>` の調査結果と同様に、 `<track>` 要素が対応する `<video>` 要素と共に含まれているのは、デスクトップサイトでは1%未満、モバイルサイトでは0.6%でした。実際の数字では、630万件のデスクトップサイトのうち、 `<video>` 要素があるところに `<track>` 要素が含まれていたのは2,836件だけでした。750万件のモバイルサイトのうち、 `<video>` 要素で読み込んだコンテンツに対応する `<track>` 要素を含むことでビデオへアクセスできるようにしたサイトは、わずか2,502件でした。

この図は `<audio>` 要素と同様に、サードパーティの `<iframe>` によって読み込まれたビデオコンテンツ、たとえば埋め込まれたYouTubeビデオを考慮していないかもしれません。また、一般的なサードパーティの音声やビデオの埋め込みサービスには、同期されたテキストを追加する機能があることに留意する必要があります。

ARIAで支援技術をサポートする

Accessible Rich Internet Applications[356]ARIAは、2014年にWeb Accessibility Initiativeがはじめて発表したWeb標準のスイートである。ARIAは、支援技術のユーザーの体験を向上させるためHTMLマークアップに追加できる一連の属性を提供します。

ARIAの使用には多くのニュアンスと複雑さがあり、また支援技術のサポートの程度もさまざまです。一般的なルールとして、ARIAは控えめに使用すべきであり、同等のネイティブ

356. *https://www.w3.org/WAI/standards-guidelines/aria/*

HTMLソリューションがある場合には、決して使用しないでください。ARIAは、支援技術に役立つ情報を提供できますが、キーボードの操作性などの関連する動作はありません。

ARIAの5つのルール[357]は、ARIAの使用に関する有用な指針をいくつか述べています。2021年9月、W3ワーキンググループは、ARIAをどのように、いつ使うことができるかについて非常に詳細な情報を持つ仕様案、ARIA in HTML[358]を発表しました。

ARIAロール

支援技術が要素に遭遇したとき、その要素の役割は、誰かがそのコンテンツをどのように操作する可能性があるかについての情報を伝えます。たとえば、`<a>`要素はリンクの役割を支援技術に公開します。これは通常、要素がアクティブになるとどこかに移動することを伝えるものです。

HTML5では多くの新しいネイティブ要素が導入されましたが、それらはすべて 暗黙的意味論[359] を持ち、ロールも含まれます。たとえば、`<nav>`要素は暗黙のうちに`role="navigation"`を持っており、支援技術に目的情報を伝えるために、ARIAを介してこの役割を明示的に追加する必要はないのです。

ARIAを使用すると、適合するネイティブなHTMLロールを持たないコンテンツに、明示的にロールを追加できます。たとえば、`tablist`ウィジェットを作成する場合、コンテナー要素に`tablist`というロールを割り当てることができます。

357. https://www.w3.org/TR/using-aria/
358. https://www.w3.org/TR/2021/PR-html-aria-20210930/#priv-sec
359. https://www.w3.org/TR/wai-aria-1.1/#implicit_semantics

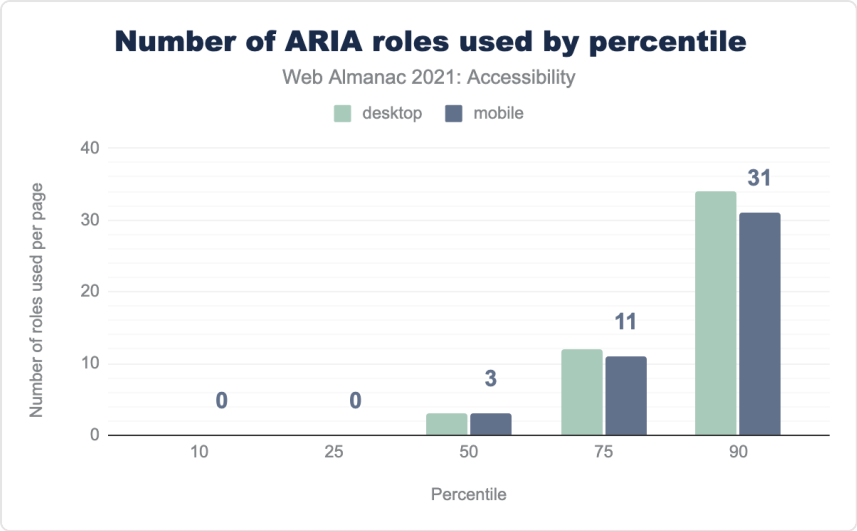


図9.22. パーセンタイルごとのARIAロールの使用数。

現在、デスクトップページの69%（2020年の65%から増加）が、ARIA `role` 属性のインスタンスを少なくとも1つ持っています。中央値では、3つの `role` 属性があります（2020年の2つから増加）。もっともよく使われるロールを以下に示します。

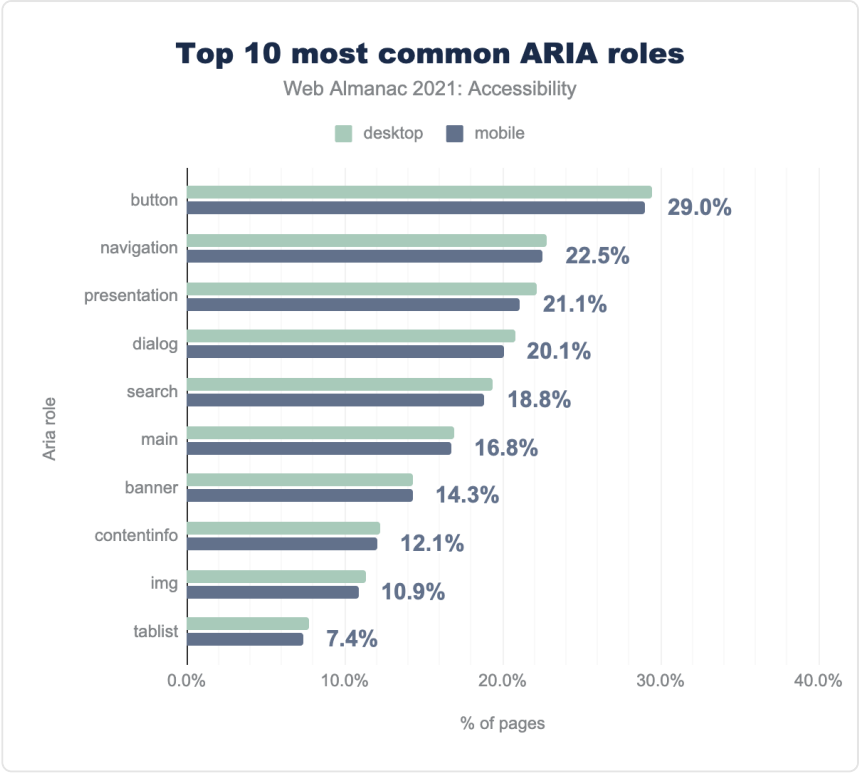


図9.23. ARIAの代表的な役割トップ10。

ボタンを使うだけ！

ARIAロールのもっとも一般的な誤用の1つは、 `<div>` や `<span>` などの非対話型要素または `<a>` 要素にボタンロールを追加することです。ネイティブのHTMLの `<button>` 要素は、暗黙のうちにボタンの役割を果たし、期待されるキーボード操作性と動作を備えており、ARIAに手を伸ばす前の最初のアプローチであるべきです。

デスクトップサイトの29%（2020年の25%から増加）とモバイルサイトの29%（2020年の25%から増加）が、明示的に割り当てられた `role="button"` の要素を少なくとも1つ持つホームページを持っていることがわかりました。このことから3分の1近くのウェブサイトが、明示的にボタンの役割を割り当てられたボタンを除いて、セマンティクスを変更するため要素に `button` の役割を使用しており、これは冗長であることがわかります。

もし、 `<div>` や `<span>` などの非インタラクティブ要素がボタンの役割を与えられていた場合、期待されるキーボードのフォーカス順や操作性が適用されず、WCAG 2.1.1

Keyboard[360]や2.4.3 Focus order[361]に問題の発生する可能性が大きいです。さらに、Windows High Contrast ModeはARIAを尊重しないため[362]、ネイティブHTMLボタン要素でない要素は、このモードでは対話可能に見えないことがあります。デスクトップとモバイルのサイトの11%に、`button`の役割を明示した`<div>`または`<span>`がありました。

ボタンの役割を`<a>`要素に適用すると、アンカー要素が持つ暗黙のリンクの役割が上書きされます。なぜなら、ボタンに期待される動作はページ内のアクションを引き起こすことであり、一方リンクは通常どこかに移動することだからです。また、正しいキーボードの動作が実装されていない場合、WCAG 2.1.1, Keyboard[363] に違反します（リンクはスペースキーで有効になりませんが、ボタンは有効になります）。さらに、ボタンの役割が期待される動作なしにスクリーンリーダーによって告知されると、支援技術ユーザーにとって混乱と見当違いを引き起こす可能性があります。



図9.24. デスクトップのWebサイトには、少なくとも1つのボタンの役割を持つリンクがあります。

デスクトップページの19%（2020年の16%から増加）、モバイルページの18%（2020年の15%から増加）に`role="button"`のアンカー要素が少なくとも1つ含まれていることがわかりました。ネイティブの`<button>`要素の方が、ARIAの最初の規則[364]にしたがって、より良い選択でしょう。

このARIAロールを追加する行為、つまり“role-up”[365]は、通常、正しいネイティブHTML要素を使用するより理想的ではありません。繰り返しますが、これらのケースの大部分では、問題の要素に明示的に`role="button"`を定義するよりも、ネイティブのHTML`<button>`要素を活用する方が良いパターンでしょう。

プレゼンテーションの役割の活用

要素に`role="presentation"`が宣言されると、その子要素と同様にそのセマンティクスが取り除かれます。たとえば、親であるテーブルやリスト要素に`role="presentation"`を宣言すると、そのロールは子要素にカスケードされます。この場合、セマンティクスも削除されます。

要素のセマンティクスを削除するということは、その要素の動作や支援技術による理解の観

360. https://www.w3.org/TR/UNDERSTANDING-WCAG20/keyboard-operation-keyboard-operable.html
361. https://www.w3.org/TR/UNDERSTANDING-WCAG20/navigation-mechanisms-focus-order.html
362. https://ericwbailey.design/writing/truths-about-digital-accessibility/#windows-high-contrast-mode-ignores-aria
363. https://www.w3.org/TR/UNDERSTANDING-WCAG20/keyboard-operation-keyboard-operable.html
364. https://www.w3.org/TR/using-aria/#rule1
365. https://adrianroselli.com/2020/02/role-up.html

点から、その要素はもはやその要素ではなく、その視覚的な外観だけが残るということを意味します。たとえば、`role="presentation"`を持つリストは、リストの構造に関する情報をスクリーン・リーダーに伝えることができなくなります。デスクトップ・ページの22%とモバイル・ページの21%に、少なくとも1つの要素が`role="presentation"`を持つことがわかりました。このロールが支援技術利用者にとってとくに役立つユースケースは非常に少ないので、控えめに、そして慎重に使用してください。

ARIAによる要素のラベリングと説明

DOMと平行して、アクセシビリティツリー[366]と呼ばれる同様のブラウザ構造があります。これには、アクセシブルな名前、説明、役割、状態など、HTML要素に関する情報が含まれています。この情報は、アクセシビリティAPIを通じて支援技術に伝達されます。

アクセシビリティツリーには、コントロール、ウィジェット、グループ、ランドマークに対して、アクセシブルネーム（ある場合）を割り当て、支援技術によって告知またはターゲット化できるようにする計算システムがあります。

アクセス可能な名前は、要素のコンテンツ（ボタンのテキストなど）、属性（画像の`alt`テキスト値など）、関連付けられた要素（プログラム的に関連付けられたフォームコントロールの`label`など）から得ることができます。複数の候補がある場合、アクセス可能な名前にどの値を割り当てるかを決定するために起こる特異性の順位付けがあります。

アクセシブルネームについての詳細は、Léonie Watson[367]の記事、アクセシブルネームとは何ですか？[368]をご覧ください。

また、ARIAを利用して、要素にアクセシブルな名前を付けることも可能です。これを実現するARIA属性は、`aria-label`と`aria-labelledby`の2つがあります。これらの属性のいずれかが、アクセス可能な名前の計算に「勝ち」、ネイティブに派生したアクセス可能な名前を上書きします。この2つの属性は注意して使用することが重要で、必ずスクリーンリーダーでテストするか、アクセシビリティツリーを見て、アクセシブルな名前がユーザーの期待するものであることを確認するようにしてください。ARIAを使って要素に名前を付けるとき、WCAG 2.5.3、名前にラベルをつける[369]基準に違反していないことを確認することが重要で、可視ラベルは少なくともそのアクセス可能な名前の一部であると期待します。

366. *https://developer.mozilla.org/docs/Glossary/Accessibility_tree*
367. *https://x.com/LeonieWatson*
368. *https://developer.paciellogroup.com/blog/2017/04/what-is-an-accessible-name/*
369. *https://www.w3.org/WAI/WCAG21/Understanding/label-in-name.html*

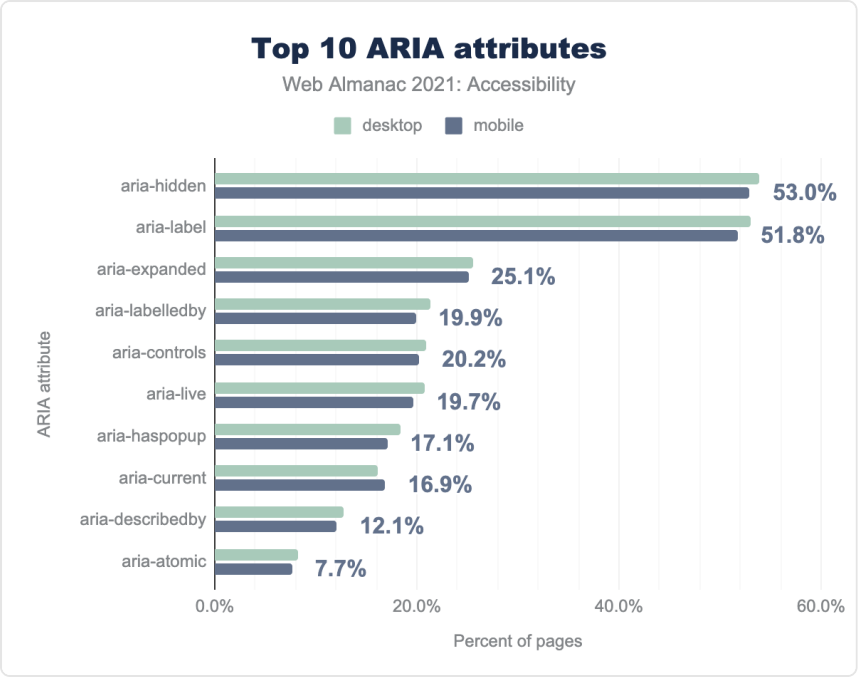


図9.25. ARIAの属性トップ10。

`aria-label` 属性は、開発者が文字列の値を提供することを可能にし、これが要素のアクセス可能な名前として使用されることになります。音声読み上げのユーザーは、目に見えるテキストを参照にしないと、名前が付いたコントロールに照準を合わせるのが難しいかもしれないことは、注目に値します。認知障害のある人も目に見えるテキストが、役に立つことがよくあります。不可視のアクセス可能な名前は、アクセス可能な名前がないよりはましですが、ほとんどの場合、可視ラベルはアクセス可能な名前を提供するか、最低でも要素のアクセス可能な名前に含まれる必要があります。

デスクトップページの53%（2020年の40%から増加）、モバイルホームページの52%（2020年の39%から増加）に少なくとも1つの要素に `aria-label` 属性があり、アクセシブルな名前を提供するARIA属性としてもっとも人気があり、1年間で使用率が非常に大きく増加していることがわかりました。これは、これまでアクセシブルな名称がなかった要素に、より多くの名称が付けられるようになったことを示すポジティブな兆候かもしれません。しかし、目に見えるラベルのない要素が増えたことを意味する場合もあり、認知障害のある人や音声読み上げの利用者に悪影響を与える可能性があります。

`aria-labelledby` 属性は、その値として `id` 参照を受け付けます。この参照は、そのアクセス可能な名前を提供するために、インターフェイスの他の要素に関連付けます。その要素は、アクセス可能な名前を提供する他の要素によって「ラベル付け」されるようになりま

す。デスクトップページの21%（2020年の18%から増加）とモバイルページの20%（2020年の16%から増加）が、少なくとも1つの要素に `aria-labelledby` 属性を持つことがわかりました。

`aria-describedby` 属性は、要素に対してより堅牢な記述が必要な場合に使用できます。また、インターフェイスの他の場所に存在する記述的なテキストと接続するために、その値としてid参照を受け入れることができます。アクセシブルネームは提供されません。アクセシブルネームは、代替ではなく、補足として併用されるべきです。デスクトップページの13%、モバイルページの12%に `aria-describedby` 属性の要素を1つ以上持つことがわかりました。

楽しい事実！1,886のウェブサイトに `aria-lavel` という属性があることがわかりました！これは `aria-label` 属性のスペルミスです。このような簡単に回避できるエラーは、必ず自動チェックで拾ってください。

ボタンのアクセス可能な名称はどこから来ているのでしょうか？

ボタンは通常、そのコンテンツまたはARIA属性からアクセス可能な名前を取得します。ARIAの最初の規則[370]により、もし要素がARIAを使わずにそのアクセス可能な名前を導き出せるなら、これは望ましいことです。したがって、 `<button>` は、可能であればARIA属性ではなく、そのテキストコンテンツからアクセス可能な名前を取得する必要があります。

ボタンが画像やアイコンを用いたグラフィカルなコントロールであるため、アクセス可能な名前を提供するためにテキストコンテンツを使用しない一般的な実装があります。これは、テキストを表示せずに制御を行う必要のある音声読み上げユーザにとって問題となる可能性があり、テキストを表示するオプションがある場合には、使用すべきではありません。

370. *https://www.w3.org/TR/using-aria/#rule1*

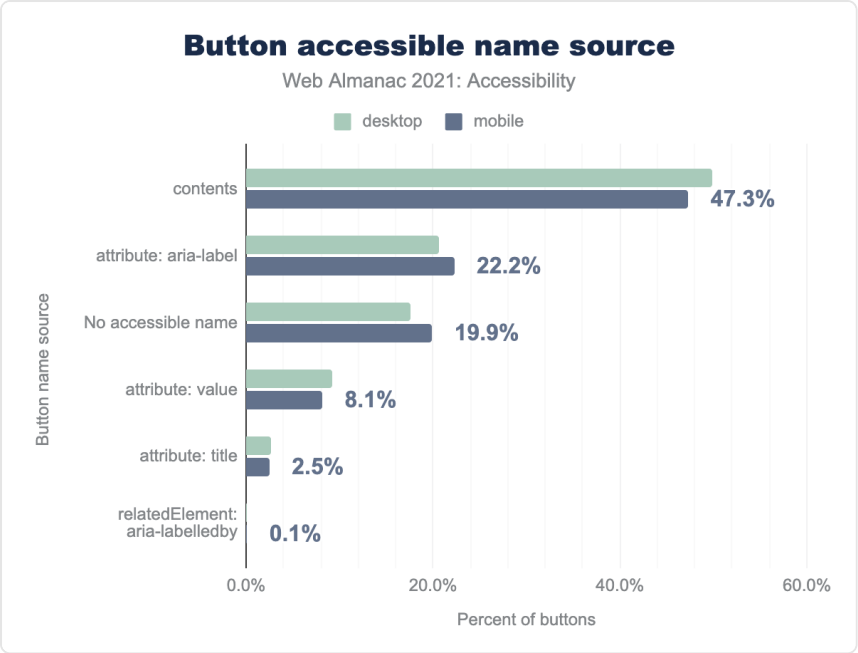


図9.26. ボタンアクセス可能な名前のソースです。

その結果、デスクトップとモバイルの両サイトにおいて、57%のボタンがコンテンツからアクセシブルネームを取得していることがわかりました。また、デスクトップサイトでは29%、モバイルサイトで27%のボタンが `aria-label` 属性からアクセシブルネームを取得していることがわかりました。

コンテンツの非表示

支援技術がコンテンツを発見しないようにするには、いくつかの方法があります。CSSの `display: none;` を利用して、アクセシビリティツリーから要素を省くことができます。スクリーン・リーダーからコンテンツを隠したい場合は、 `aria-hidden="true"` を使用できます。 `display: none;` とは異なり、 `aria-hidden="true"` の宣言は、要素とその子要素を目に見える形で削除しないことに注意してください。

53.8%

図9.27. デスクトップのウェブサイトには、少なくとも1つの `aria-hidden` 属性のインスタンスがあります。

デスクトップページの54%（2020年の48%から増加）とモバイルページの53%（2020年の49%から増加）に、少なくとも1つの `aria-hidden` 属性付き要素のインスタンスが、あることがわかりました。

これらのテクニックは、ビジュアルインターフェイスの何かが冗長であったり、支援技術のユーザーにとって役に立たない場合に、もっとも役に立ちます。アクセシブルにするのが難しいコンテンツをスキップするために、支援技術からコンテンツを隠すことは、決して行ってはなりません。

コンテンツを隠したり、見せたりすることは、現代のインターフェイスでは一般的なパターンであり、誰にとっても隠れたUIを断捨離するのに役立つことがある。Hide/showウィジェットでは、 `aria-expanded` 属性を使用して、コントロールをアクティブにすると何かが現れ、再びアクティブにすると隠されることを支援技術に示す必要があります。デスクトップページの26%（2020年の21%から増加）、モバイルページの25%（2020年の21%から増加）に、少なくとも1つの要素で `aria-expanded` 属性が、あることがわかりました。

スクリーンリーダー専用テキスト

スクリーン・リーダーの利用者に追加情報を提供するために開発者が採用する一般的な手法は、CSSを使用してテキストの一節を視覚的に隠し、スクリーン・リーダーがそれを発見できるようにすることです。 `display: none;` はアクセシビリティツリーにコンテンツが存在しないようにするため、CSSコードの特定の宣言セットを含む共通のパターンが存在します。



図9.28. `sr-only` または `visually-hidden` クラスを持つデスクトップWebサイト

この code snippet[371] のもっとも一般的なCSSクラス名は、（慣習的にもBootstrapなどのライブラリ全体でも） `sr-only` と `visually-hidden` です。デスクトップ用ページの14%、モバイル用ページの13%に、これらのCSSクラス名のいずれか、または両方が含まれていることがわかりました。スクリーン・リーダーの利用者の中には、ある程度の視力を持つ人もいるため、視覚的に隠されたテキストに過度に依存すると、一部の人は混乱する可能性があることは留意しておく必要があります。

371. *https://css-tricks.com/inclusively-hidden/*

ダイナミックレンダリングコンテンツ

DOMに新しいコンテンツや更新されたコンテンツがある場合、スクリーン・リーダーに伝える必要があります。フラストレーションを避けるために、どの更新を伝える必要があるかについて、ある程度考慮する必要があります。たとえば、フォームの検証エラーは伝える必要がありますが、遅延ロードされた画像は伝える必要がないかもしれません。DOMの更新は、混乱を招かない方法で行われる必要があります。

*ARIA*ライブリージョンは、DOMの変更を監視し、スクリーンリーダーで更新されたコンテンツを知らせることができます。デスクトップページの21%（2020年の17%から増加）、モバイルページの20%（2020年の16%から増加）がライブリージョンを持っていることがわかりました。ライブリージョンバリアントや使い方についての詳しい情報は ARIAライブリージョン[372] をチェックするか、この Dequeによるライブデモ[373]で遊んでみてください。

アクセシビリティオーバーレイ

アクセシビリティオーバーレイは、アクセシビリティプラグインやオーバーレイウィジェットと呼ばれることもあり、ウェブサイトのアクセシビリティ問題を簡単に解決するツールとして販売されているデジタル製品です。Overlay Fact Sheet[374] では、「ウェブサイトのアクセシビリティを向上させることを目的とした技術の広義の用語と定義されています。それらは、ウェブサイトのフロントエンドのコードの改善を自動化するために、サードパーティのソースコード（通常はJavaScript）を適用します。」

これらの製品の多くは、1行のコードでウェブサイトがアクセシブルになる、あるいは少なくともアクセシビリティの観点からは合法的に準拠することを示唆する、欺瞞的なマーケティング材料を使用しています。

たとえば、この分野でもっとも積極的な製品の1つである accessiBe[375] は、そのプロセスを JavaScriptのインストール コードを本番コードに貼り付けるだけで、48時間以内にサイトをアクセス可能かつコンプライアントにすることが可能であると説明しています。

残念ながら、ウェブアクセシビリティは、このようなすぐに使えるソリューションでは実現できないのです。もしそうであれば、この章にあるような悲惨な統計データを見ることはないでしょう。

372. *https://developer.mozilla.org/docs/Web/Accessibility/ARIA/ARIA_Live_Regions*
373. *https://dequeuniversity.com/library/aria/liveregion-playground*
374. *https://overlayfactsheet.com/#what-is-a-web-accessibility-overlay*
375. *https://en.wikipedia.org/wiki/AccessiBe*

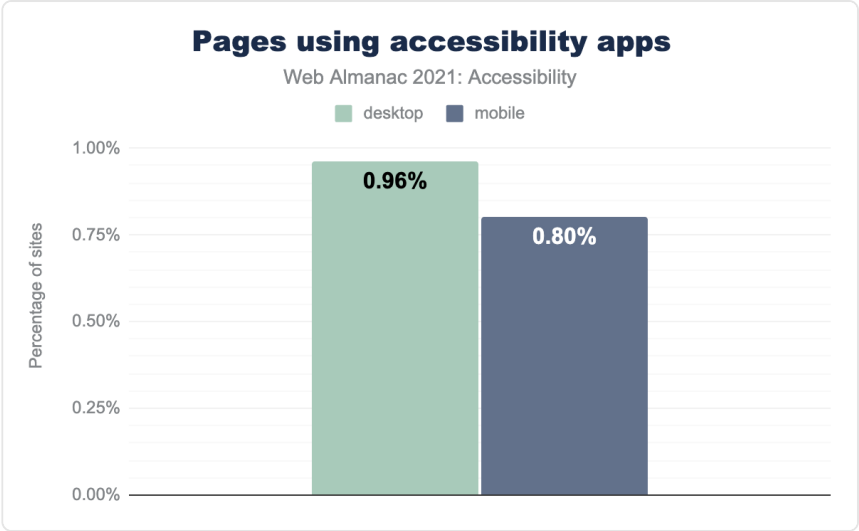


図9.29. アクセシビリティアプリを使用したページ。

その結果、デスクトップWebサイトの0.96%（6万件以上）で、これらのアクセシビリティ・オーバーレイが使用されていることがわかりました。この分野の有名な製品のリストを照会したことは、注目に値します。しかし、このリストは網羅的なものでないので、この指標は実際にはもっと高い可能性があります。

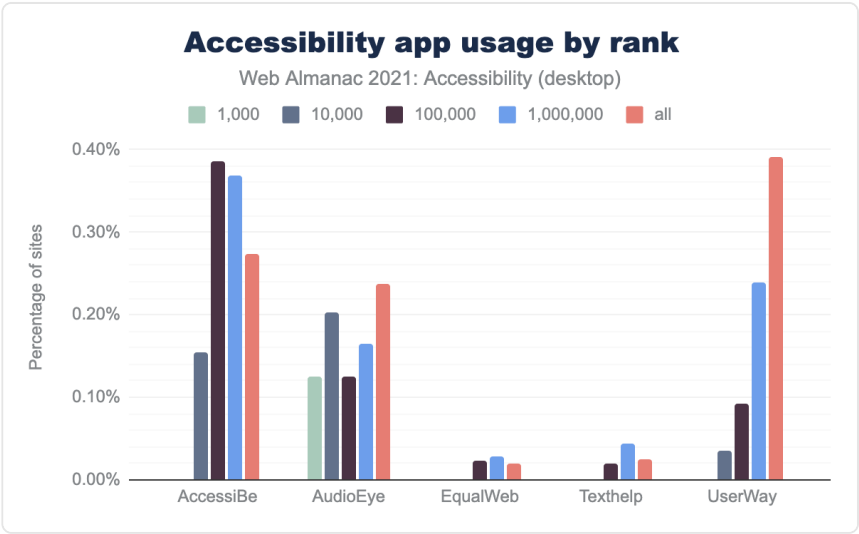


図9.30. アクセシビリティ・アプリのランク別利用状況。

ドメインランクを考慮すると、上位1,000サイトのオーバーレイ使用率は0.1%と低くなっています。しかし、これらの上位サイトのリーチを考慮すると、これだけのトラフィックを持つ1つのWebサイトがオーバーレイを使用した場合の潜在的な影響は非常に大きなものです。

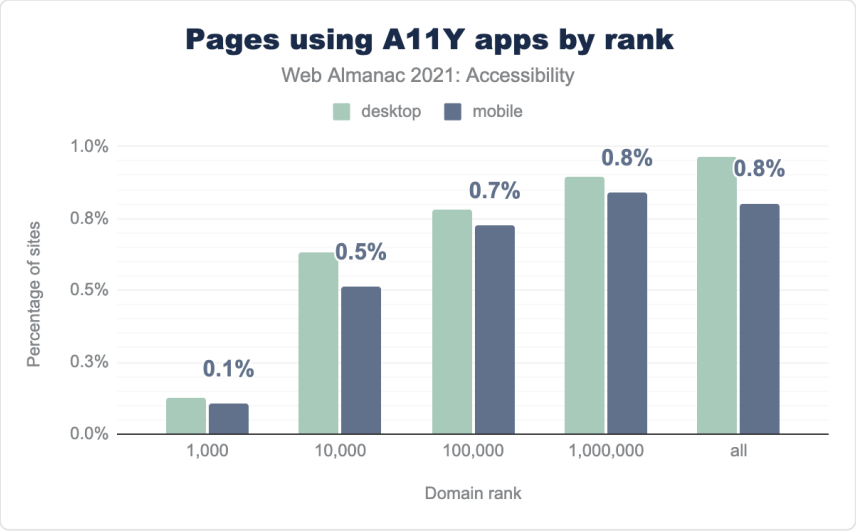


図9.31. アクセシビリティ・アプリを使用しているページのランク別推移。

オーバーレイのもたらすもの

これらのツールはしばしば支援技術を妨害し、実際に多くの人にとってウェブサイトをアクセスしにくくします。これに関して、「障害者はこのAIツールがウェブを悪化させていると言う」という適切なタイトルのViceの記事で調査されています[376]。accessiByeBye[377] というオープンソースの拡張機能もあります。これは、支援技術ユーザーがサードパーティのオーバーレイ製品を使用しているWebサイトの使用を妨げられないよう、オーバーレイをブロックするために特別に開発されたものです。

市民権弁護士のHaben Girma[378]がアクセシビリティオーバーレイに関するこの動画[379]で説明しているように、「AIはツールであり、今のところアクセシビリティに対してできることが極めて限られている」のです。さらに、自動生成された自分の名前のキャプションが、「Haben Girma」を「happen grandma」と誤訳したこと、このような情報の誤伝達が聴覚障害者のユーザーに与える影響について説明します。

376. https://www.vice.com/en/article/m7az74/people-with-disabilities-say-this-ai-tool-is-making-the-web-worse-for-them
377. https://www.accessibyebye.org/
378. https://x.com/HabenGirma
379. https://www.youtube.com/watch?v=R12Z1Sp-u4U

このようなオーバーレイ会社と、彼らが奉仕すると称している障害者コミュニティとの間には、緊張関係が存在する。たとえば、全米盲人連合は、アクセシービーをその全国大会から追放し[380]、同社による被害についてこのように発表しています[381]。

> *accessiBeは、何がアクセシブルで何がアクセシブルでないかを、盲目の専門家やスクリーンリーダーの常用者が知っていることを認めていないようです。この国の視覚障害者は、なだめたり、いじめたり、買収されたりすることはありません。*
>
> *– 全米盲人連合*[382]

個人情報保護への配慮

これらのツールの中には、支援技術の利用を検知する技術を持つものもあります。これは、本人の同意なしに、その人の障害に関する個人データを収集される可能性があることを意味します。

オーバーレイファクトシート[383]より。

> *オーバーレイの中には、同じオーバーレイを使用しているサイト間でユーザーの設定を持続させるものがあることが分かっています。これは、ユーザーのコンピューターにクッキーを設定することで行われます。ユーザーがあるサイトでオーバーレイ機能の設定を有効にすると、他のサイトでも自動的にその機能が有効になります。大きなプライバシー問題は、ユーザーが追跡されることを決してオプトインしておらず、オプトアウトする能力もないことです。このようにオプトアウト（明示的にその設定をオフにすること以外）がないため、オーバーレイのお客様には一般データ保護規則（GDPR）とカリフォルニア消費者プライバシー法（CCPA）のリスクが発生します。*
>
> *– オーバーレイファクトシート*[384]

このLéonie Watsonによる記事[385]は、アクセシビリティ オーバーレイにおけるこの種のデータ追跡のプライバシーに関する懸念を探ったものです。

380. *https://www.forbes.com/sites/gusalexiou/2021/06/26/largest-us-blind-advocacy-group-bans-web-accessibility-overlay-giant-accessibe/?sh=16621ec55a15*
381. *https://nfb.org/about-us/press-room/national-convention-sponsorship-statement-regarding-accessibe*
382. *https://nfb.org/about-us/press-room/national-convention-sponsorship-statement-regarding-accessibe*
383. *https://overlayfactsheet.com/#privacy*
384. *https://overlayfactsheet.com/*
385. *https://tink.uk/accessibe-and-data-protection/*

オーバーレイと訴訟

これらのウィジェットは、それを使用する企業に対して多くのアクセシビリティ訴訟の一部として名指しされています。UsableNetの2020年デジタルアクセシビリティ訴訟に関するレポート[386]によると、「訴えられた250社以上がアクセシビリティウィジェットやオーバーレイに投資していた」そうです。アクセシビリティ専門家のSherri Byrne-Haberは[387]、「2020年末に起こされるアクセシビリティ訴訟の10%は、プラグイン、オーバーレイ、ウィジェットをインストールした企業に対して、それらがADA訴訟の防弾になると考えてのものだ」と引用しています。アクセシビリティに関する法律は、「障害を持つアメリカ人法」に限らず、WCAGを指した法律を持つ国々が世界中に存在することは注目に値します[388]。

これらのオーバーレイを使用する際の法的な影響については、こちらをご覧ください。Lainey Feingold[389]の記事Honor the ADAを参照してください。Avoid Web Accessibility Quick-Fix Overlays[390]とAdrian Roselliの記事#accessiBe Will Get You Suedを参照してください[391]。

なぜオーバーレイを使う企業があるのですか？

根本的には、ableism[392] に後押しされ、オーバーレイは、ほとんどの組織が苦労している問題を解決するものとして位置づけられています。この章を通して、インターネットはほとんどアクセスできない、というデータは明らかです。

これらの製品は、組織のアクセシビリティに関する知識のギャップを利用したものです。問題領域の枠組みは、障害者のアクセスに対する障壁を本質的に取り除くのではなく、解決策を自動化することによって、訴訟を回避することを目的としています。なぜこのような訴訟が起こるかというと、オンラインにアクセスする権利が侵害されることで、実際に公民権侵害が起こっているからです。たとえばAIツールが画像のアクセシブルでない説明を提供しても、自動化ツールのチェックには合格するかもしれませんが、目の見えない人にとっての障壁は取り除けませんし、情報の平等性を提供するものではありません。

一部のオーバーレイ企業は、1行のコードと月々の数ドルでアクセシビリティと完全なコンプライアンスを実現すると約束し、欺瞞的なマーケティングに振り回されることがあります。しかし、残念なことに、これらのツールは障害者にとって新たな障壁となり、組織は予期しない法的問題に直面する可能性があります。

即効性のある解決策はありません。組織やデジタル事業者は、ウェブコンテンツのアクセシビリティの問題を実際に解決することを優先する必要があります。障害者のコミュニティでよく言われるのは、「私たちなしでは、私たちのことは何もできない」ということです。オ

386. *https://info.usablenet.com/2020-report-on-digital-accessibility-lawsuits*
387. *https://sheribyrnehaber.com/technology-doesnt-make-accessibility-hard-people-who-dont-care-do/*
388. *https://www.3playmedia.com/blog/countries-that-have-adopted-wcag-standards-map/*
389. *https://x.com/LFLegal*
390. *https://www.lflegal.com/2020/08/quick-fix/*
391. *https://adrianroselli.com/2020/06/accessibe-will-get-you-sued.html*
392. *https://www.forbes.com/sites/andrewpulrang/2020/10/25/words-matter-and-its-time-to-explore-the-meaning-of-ableism/?sh=7ab349837162*

ーバーレイは、障害者コミュニティがあまり関与しないまま作成され、これらの企業の中には、このことについて発言した障害者をさらに遠ざけてしまったところもあります[393]。これらの製品では、障がい者の方々のウェブへの平等なアクセスを実現することはできません。

オーバーレイに関するその他の資料

- Connor Scott-Gardener氏のオーバーレイ使用体験談[394]
- オーバーレイをめぐるADA訴訟の事例[395]
- A11yプロジェクト - アクセシビリティ・オーバーレイを使うべきですか？[396]
- 完全に自動化されたウェブアクセシビリティは存在しない[397]
- 自動化されたツールだけでは、ウェブサイトにアクセシビリティと法令遵守を実現できない理由[398]
- アクセシビリティオーバーレイを使うべきですか？[399]

結論

アクセシビリティの提唱者であるBilly Gregoryがかつて言ったように[400]、「UXがすべてのユーザーを考慮しないとき、それはSOMEユーザーエクスペリエンス、またはSUXとして知られるべきでは」ないでしょうか。アクセシビリティの作業は、追加事項、エッジケース、あるいは技術的負債と同等とみなされ、ウェブサイトや製品の成功の中核となるべきものではないことがあまりに多くあります。

成功するためには、製品チームと組織全体が、アクセシビリティを責務の一部として、C-suiteに至るまで優先させる必要があります。アクセシビリティの作業は、製品サイクルの左側にシフトする必要があります[401]。つまり、開発前の研究、構想、設計段階に焼き付ける必要があるのです。そして、もっとも重要なことは、障害者がこのプロセスに参加する必要があることです。

技術業界は、インクルージョン主導の開発に移行する必要があります。これにはある程度の先行投資が必要ですが、アクセシビリティを考慮せずに構築されたサイトやアプリを後付けしようとするよりも、製品に組み込むことができるようにサイクル全体に組み込む方がはる

393. https://www.nbcnews.com/tech/innovation/blind-people-advocates-slam-company-claiming-make-websites-ada-compliant-n1266720
394. https://catchthesewords.com/do-automated-solutions-like-accessibe-make-the-web-more-accessible/
395. https://uxdesign.cc/important-settlement-in-an-ada-lawsuit-involving-an-accessibility-overlay-748a82850249
396. https://www.a11yproject.com/posts/2021-03-08-should-i-use-an-accessibility-overlay/
397. https://uxdesign.cc/theres-no-such-thing-as-fully-automated-web-accessibility-260d6f4632a8
398. https://www.forbes.com/sites/gusalexiou/2021/10/28/why-automated-tools-alone-cant-make-your-website-accessible-and-legally-compliant/?sh=2e538b62364e
399. https://shouldiuseanaccessibilityoverlay.com/
400. https://x.com/thebillygregory/status/552466012713783297?s=20
401. https://feather.ca/shift-left/

かに簡単で、長期的に見てもコストがかからないと思われます。

業界として、この章の数字が物語るように、私たちは障害者の期待を裏切っていることを認める時が来たのです。2021年からの数字は、2020年から大きく動いていない。これは、トップダウンのリーダーシップと投資（ブラウザからの継続的な参加を含む）、そして我々の実践を前進させ、ウェブを利用する障害者のニーズ、安全、インクルージョンを擁護するボトムアップの努力の組み合わせによってもたらされなければならないのです。

著者



Alex Tait

@at_fresh_dev　alextait1　https://atfreshsolutions.com

Alex Tait はアクセシビリティのスペシャリストで、インターフェイスアーキテクチャとデザインシステムにおけるアクセシビリティとモダンな JavaScript の交差に情熱を注いでいます。開発者として、アクセシビリティを前面に押し出したインクルージョン主導の開発手法が、すべての人にとってより良い製品につながると信じています。コンサルタントおよびストラテジストとして、「より少なく」を信条とし、障害者のためのコア機能の平等性よりも新機能の範囲拡大を優先させることはできないと考えている。また、教育者として、技術産業がより多様で公平、かつ包括的な産業となるよう、情報に対する障壁を取り除くことを信条としている。



Scott Davis

@scottdavis99　scottdavis99　http://thirstyhead.com

Scott Davisは、著者であり、Thoughtworks[402]のデジタル・アクセシビリティ提唱者として、ウェブ開発の最先端/革新/新興/非伝統的な側面に焦点を当てています。”デジタル・アクセシビリティは、コンプライアンスのチェックボックスを超えるものであり、アクセシビリティはイノベーションの出発点です。”



Olu Niyi-Awosusi

@oluoluoxenfree　oluoluoxenfree　https://olu.online/

Olu Niyi-Awosusi はOddbird[403] の JavaScript エンジニアで、リスト、新しいことを学ぶこと、Bee and Puppycat、社会正義、アクセシビリティ[404] を愛し、日々努力する人です。

402.　*https://www.thoughtworks.com/*



Gary Wilhelm

gwilhelm

Gary Wilhelmは、UNC-Chapel Hill[405]のDivision of Finance and Operationsのデジタルソリューションマネージャーで、WebサイトやWebアプリケーションの開発に従事しています。2013年から仕様書を勉強してWebサイトのアクセシビリティ化に取り組み始め、数千枚のPDF文書の改善を通じてPDFアクセシビリティの学習に多大な時間を費やすなど、それ以来アクセシビリティに関心を持ち続けています。余暇には、旅行、庭仕事、ランニング、スポーツ観戦、妻と2人のティーンエイジャーを困らせること、そして愛犬がリスやウサギ探しを手伝うことが好きだそうです。



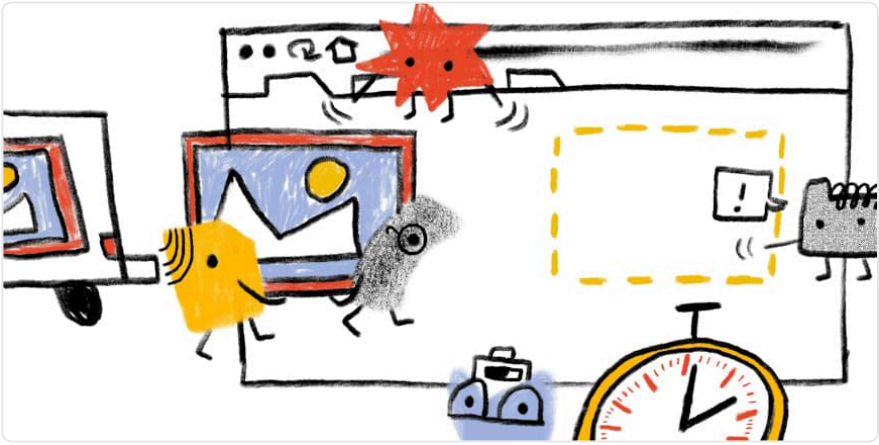
Katriel Paige

kachiden https://www.flowerstorm.tech/

QA、UX、フロントエンド開発、CSSとの愛憎関係、そして計り知れない量のコーヒーなど、技術者としての長く曲がりくねった道を歩んできたアクセシビリティエンジニアで猫好き。

403. *https://www.oddbird.net/*
404. *https://alistapart.com/article/building-the-woke-web/*
405. *https://www.unc.edu/*

部 II 章 10
パフォーマンス



Sia Karamalegos によって書かれた。
Rick Viscomi、 Kevin Farrugia、 Estelle Weyl、 Ziemek Bućko、 Julia Yang、 Fili Wiese、 Barry Pollard、 Samar Panda と Edmond W. W. Chan によってレビュ。
Sia Karamalegos、 Rick Viscomi と Nitin Pasumarthy による分析。
Julia Yang 編集。
Sakae Kotaro によって翻訳された。

序章

ユーザー体験にとって、パフォーマンスは重要です。読み込みが遅く、反応が遅いWebサイトは、ユーザーをイライラさせ、コンバージョンの低下を招きます。Core Web Vitals[406]がGoogle検索ランキング[407]に寄与するのは今年がはじめてです。そのため、ウェブサイトのパフォーマンス向上に対する関心が高まっており、ユーザーにとっては嬉しいニュースです。

*今年の報告書の主なポイントは何ですか？*まず、良いユーザー体験を提供するためには、まだ長い道のりが必要です。たとえば、ネットワークやデバイスの高速化は、JavaScriptの配信量を無視できるレベルにはまだ達していませんし、そこまで到達していないかもしれません。2つ目は、新機能を性能のために誤用し、結果的に性能が低下します。第三に、インタラクティブ性を測定するためのより良い測定基準が必要であり、それは現在進行中です。そ

406. https://web.dev/i18n/ja/vitals/
407. https://developers.google.com/search/blog/2020/11/timing-for-page-experience

して、4つ目は、CMSやフレームワークレベルのパフォーマンスへの取り組みが、上位1000万のウェブサイトのユーザー体験に大きな影響を与える可能性があることです。

*今年の新機能は？*今回はじめて、トラフィックランキング別のパフォーマンスデータを公開します。また、過去の主要なパフォーマンス指標もすべて把握しています。最後に、また、過去の主要なパフォーマンス指標もすべて把握しています。最後に、最大コンテンツの描画（LCP）要素について、より深く掘り下げた内容を追加しました。（LCP）要素について、より深く掘り下げた内容を追加しました。

方法論に関する注記

パフォーマンスの章が他の章と異なる点は、分析に Chromeユーザー体験レポート[408] (CrUX) へ大きく依存していることです。なぜか？私たちの最優先事項がユーザー体験であるならば、パフォーマンスを測定する最良の方法は、リアルユーザーデータ（リアルユーザーメトリクス、略してRUM）です。

> *Chromeユーザー体験レポートは、実際のChromeユーザーがウェブ上の人気スポットをどのように体験しているかについてのユーザー体験メトリクスを提供します。*
>
> *– Chromeユーザー体験レポート*[409]

CrUXのデータは、ハイレベルなフィールド/RUMの指標のみを提供し、Chromeブラウザーのみを対象としています。また、CrUXは、ページ単位ではなく、オリジン（ウェブサイト）単位でデータを報告します。

CrUX RUMのデータをHTTP ArchiveのWebPageTestのラボデータで補完しています。WebPageTestには、Lighthouseのフルレポートを含む、各ページに関する非常に詳細な情報が含まれています。WebPageTestは、全米のロケーションでパフォーマンスを測定していることに注意してください。CrUXのパフォーマンスデータは、実際のユーザーのページロードを表しているので、グローバルなものです。

パフォーマンスを前年比で比較する場合、以下の点に留意してください。

- 2020年から累積レイアウトシフト（CLS）[410]の計算が変更されました。
- 2020年からコンテンツの初回ペイント（FCP）の基準値（「良い」、「改善が必要」、「悪い」）が変更[411]されました。

408. *https://developers.google.com/web/tools/chrome-user-experience-report*
409. *https://developers.google.com/web/tools/chrome-user-experience-report*
410. *https://web.dev/articles/cls-web-tooling*
411. *https://web.dev/articles/cls-web-tooling#additional-updates*

- 昨年は2020年8月のデータをもとに、今年は2021年7月の実行をもとに報告しました。

詳しくはWeb Almanacの方法論をお読みください。

ハイレベルなパフォーマンス：Core Web Vitals

個々の指標へ飛び込む前に、Core Web Vitals[412] (CWV) の複合パフォーマンスについて見てみましょう。Core Web Vitals（LCP、CLS、FID）は、ユーザー体験に焦点を当てたパフォーマンスメトリクスの集合体です。読み込み、インタラクティブ性、視覚的な安定性に重点を置いています。

ウェブパフォーマンスは、アルファベットの羅列で有名ですが、このフレームワークでコミュニティがまとまりつつあります。

ここでは、CWVの3つの指標すべてで「良好」の閾値に達したWebサイトへ焦点を当て、Webがどのように高いレベルで機能しているかを理解します。指標別分析では、各指標による同じチャートを詳しく説明し、さらにCWVにはない指標も取り上げます。

412. *https://web.dev/i18n/ja/vitals/*

デバイス別

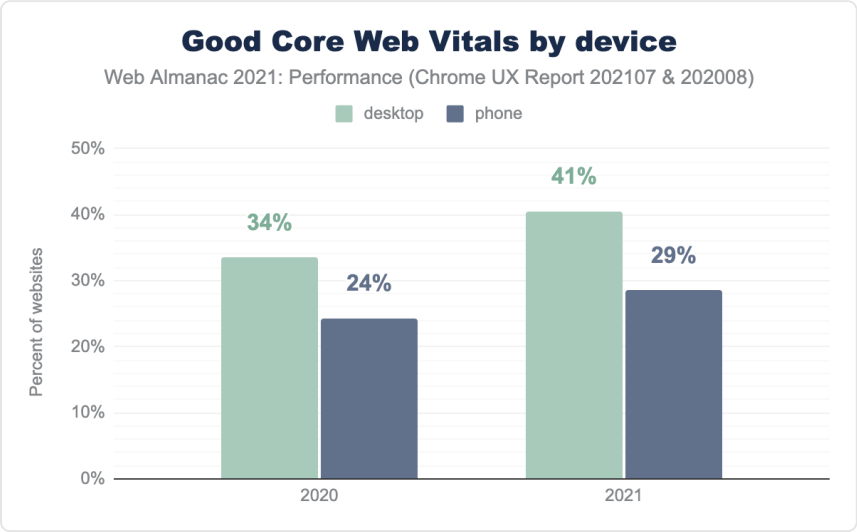


図10.1. 2020年から2021年にかけてデバイス別の優れたCore Web Vitals

備考:*昨年からCLSの計算方法が変更になったため、この数値は単純比較ではありません。*

Chromeユーザー体験レポートに掲載されたウェブサイトのコアウェブバイタルは、前年比で改善されました。しかし、この改善のかなりの部分は、必ずしもCLSの性能向上ではなく、CLSの計算方法の変更によるものです。その結果、CLSの「改善」はデスクトップで8ポイント（モバイルは2ポイント）となった。LCPはデスクトップで7ポイント改善（モバイルは2ポイント改善）。FIDは、デスクトップでは両年ともすでに100%で、モバイルでは10ポイント改善した。

例年通り、モバイル端末よりもデスクトップマシンの方が、パフォーマンスが高いという結果になりました。このため、実際のモバイル端末でサイトのパフォーマンスをテストし、実際のユーザー指標（つまり、フィールドデータ）を測定することが非常に重要です。開発者向けツールでモバイルをエミュレートすることは、研究室（＝開発）では便利ですが、実際のユーザー体験を代表するものではありません。

有効な接続タイプ別

CrUXの接続タイプ別データはわかりにくいかもしれません。トラフィックに基づくものではありません。ある接続タイプでWebサイトに何らかの経験があれば、その接続タイプの分母が増加します。もし、その接続タイプでそのウェブサイトにとって良い経験があれば、それは分子を増加させます。別の言い方をすれば、4Gの速度でページロードを経験したすべ

てのWebサイトのうち、36%はCWVが良好であったということです。

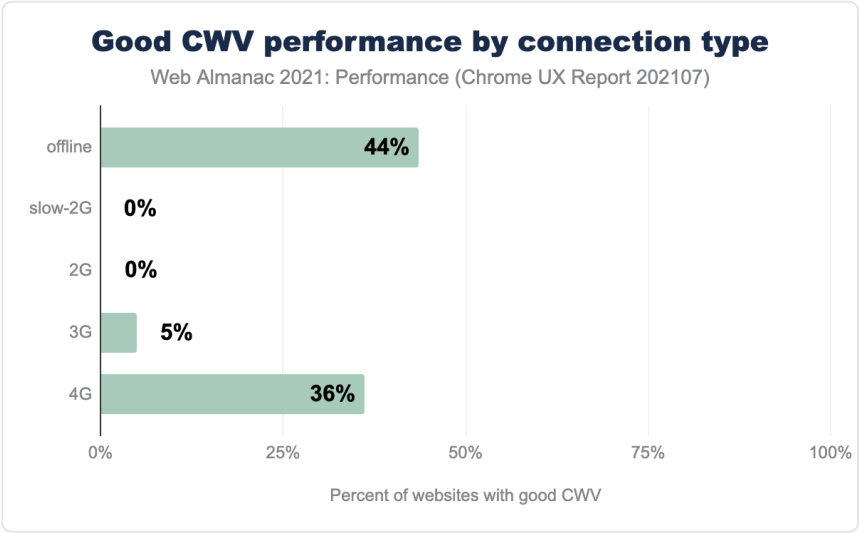


図10.2. 効果的な接続タイプによる良好なCWV性能

接続の高速化は、Core Web Vitalsのパフォーマンス向上と相関しています。オフラインのパフォーマンスが向上したのは、おそらくプログレッシブ ウェブ アプリのサービス ワーカー キャッシングが原因です。ただし、オフラインの有効な接続タイプのカテゴリのオリジン数は、合計2,634(0.02%) とごくわずかです。

3G以下の通信速度は、大幅な性能劣化と相関が、あることがトップの収穫です。低速の接続速度でのアクセス用に、簡略化したエクスペリエンスを提供することを検討してください (たとえば、データセーバーモード[413])。ユーザーの代表的なデバイスと接続を使用して、サイトのプロファイルを作成します（アナリティクスデータに基づく）。

413. *https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/save-data/*

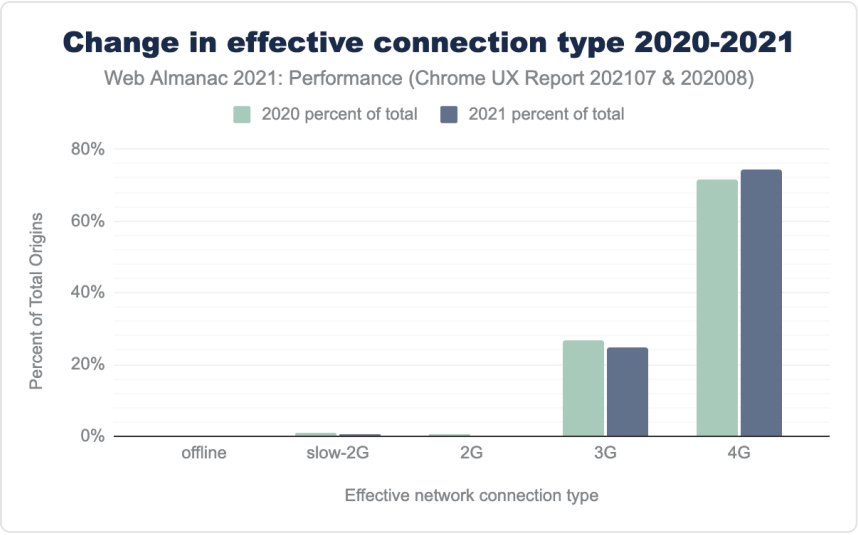


図10.3. 2020-2021年有効接続タイプの変更

先ほど、LCPとFIDの改善について、前年比で改善したと申し上げました。これらは、モバイル機器やモバイルネットワークの高速化が一因と思われます。上図では、3Gでの総アクセス元が2パーセントポイント減少し、4Gでのアクセス元が3パーセントポイント増加しています。オリジンの割合は、必ずしもトラフィックと相関があるわけではありません。しかし、人々がより高速な接続を利用できるようになれば、その接続タイプからアクセスされるオリジンもより多くなると推測されます。

接続タイプ別のパフォーマンスは、オリジンだけでなく、トラフィック別にトラッキングを開始できれば、より分かりやすくなると思います。 また、高速通信時のデータも見られるといいですね。ただし、APIは現在、4G以上のものを4Gとしてグループ化するよう制限されています[414]。

414. *https://developer.mozilla.org/docs/Glossary/Effective_connection_type*

地域別

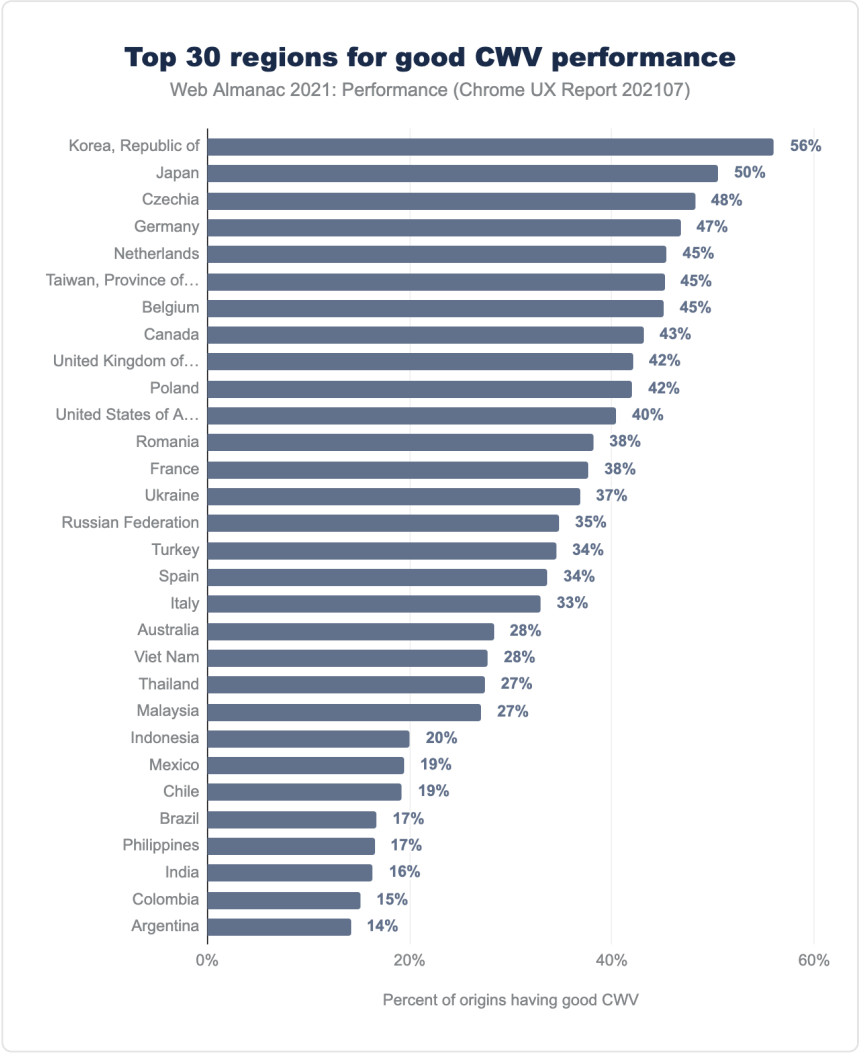


図10.4. CWVの性能が良い地域トップ30

アジアやヨーロッパの一部の地域は、引き続き高いパフォーマンスを示しています。これは、ネットワークの速度が速いこと、より高速なデバイスを持つ裕福な人口が多いこと、エッジキャッシュの位置が、近いことが原因であると考えられます。多くの結論を出す前に、このデータセットをもっと理解する必要があります。

CrUXのデータはChromeのみ収集されています。国別のオリジン比率は、相対的な人口規模とは一致しません。理由としては、ブラウザのシェア、アプリ内ブラウジング、デバイスのシェア、アクセスレベル、使用レベルの違いなどが考えられます。すべてのCrUX分析において、地域レベルの差異とその背景を評価する際には、これらの注意事項に留意してください。

順位別

今年はじめて、ランキングデータを掲載しました！CrUXでは、Chromeで計測した1サイトあたりのページビュー数でランキングを決定しています。チャートでは、カテゴリーは加算式になっています。上位10,000サイトには上位1,000サイトが含まれる、といった具合です。詳しくは方法論をご覧ください。

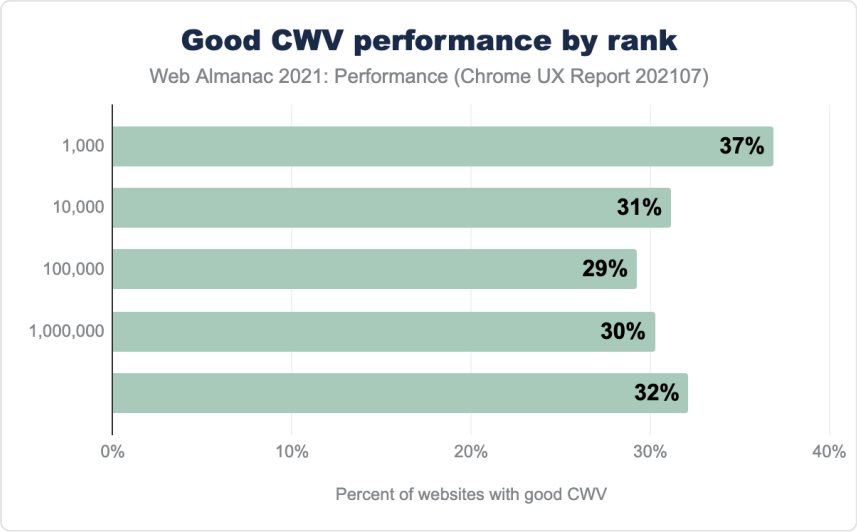


図10.5. CWVのランク別良パフォーマンス

Core Web Vitalsでは、上位1,000サイトが他を大きく引き離している。グラフの真ん中には、CLSに起因するパフォーマンスの低下という興味深い谷が発生しています。FIDはすべてのグループ化で横ばいでした。他のすべての指標は、上位のランキングほど高いパフォーマンスを示すという相関関係にあります。

相関関係は因果関係ではありません。しかし、数え切れないほどの企業がパフォーマンスの向上を示し、最終的なビジネスインパクトにつながっています（WPO stats[415]）。トラフィックの増加やエンゲージメントの増加を達成できない理由が、パフォーマンスであってはな

415. https://wpostats.com/

らないのです。

指標別分析

このセクションでは、各指標について掘り下げて説明します。あまり詳しくない方のために、各指標を詳しく説明した記事へのリンクも用意しています。

最初のバイトまでの時間（TTFB）

最初のバイトまでの時間[416] (TTFB) は、ブラウザがページを要求してから、サーバーから最初のバイトの情報を受け取るまでの時間です。ウェブサイトのロードに関する連鎖の最初の指標となるものです。TTFBが悪いと、連鎖的にFCPやLCPに影響を与えることになります。それが、私たちが最初にこの指標について話す理由です。

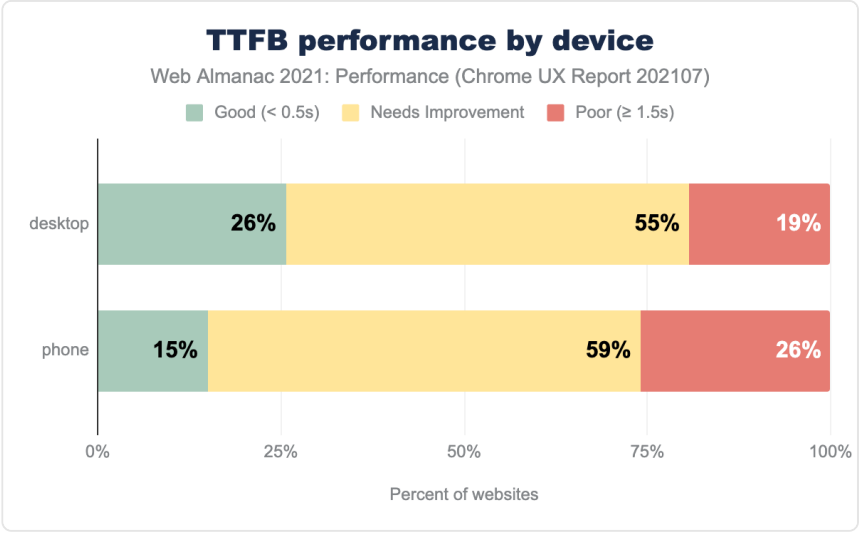


図10.6. デバイス別TTFB性能

TTFBはモバイルよりもデスクトップの方が高速であり、これはネットワーク速度が速いためと思われる。昨年[417]と比較すると、TTFBはデスクトップでわずかに改善し、モバイルで遅くなった。

416. https://web.dev/articles/ttfb
417. https://almanac.httparchive.org/ja/2020/performance#fig-17_

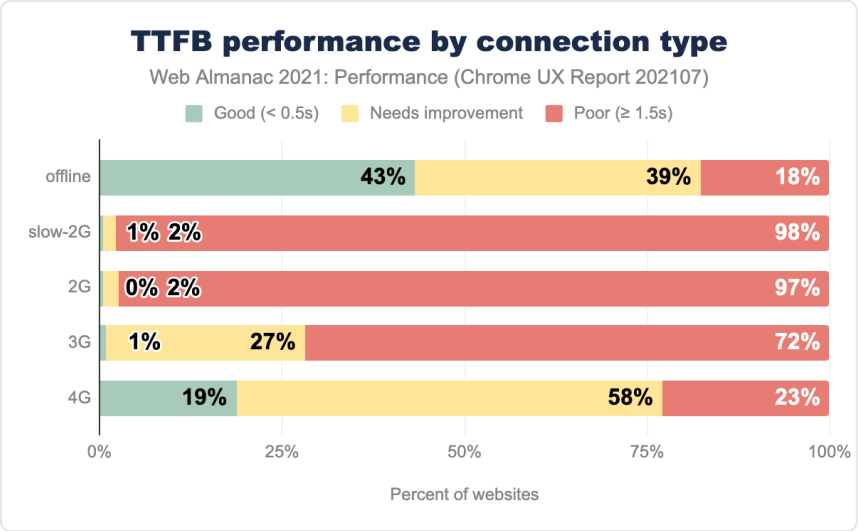


図10.7. 接続タイプ別のTTFBパフォーマンス

TTFBはまだまだこれからです。当社のウェブサイトの75%は4G回線グループ、25%は3G回線グループで、残りのものはごくわずかでした。4Gの実効速度では、19%のオリジンだけが「良好」なパフォーマンスでした。

オフライン接続でどうしてTTFBが発生するのか、疑問に思われるかもしれません。おそらく、TTFBデータを記録し送信するオフラインのサイトのほとんどは、サービスワーカーキャッシング[418]を使用していると思われます。TTFBは、そのレスポンスがCache Storage APIやHTTP Cacheからのものであっても、ページのレスポンスの最初の1バイトを受信するまでの時間を測定します。実際のサーバーが関与している必要はない。レスポンスにサービスワーカーからのアクションが必要な場合、サービスワーカー・スレッドの起動とレスポンスの処理にかかる時間も、TTFBの一因になり得ます。しかし、サービスワーカーの起動時間を考慮しても、これらのサイトは平均して、他の接続カテゴリよりも速く最初のバイトを受信します。

418. *https://developer.mozilla.org/docs/Web/Progressive_web_apps/Offline_Service_workers*

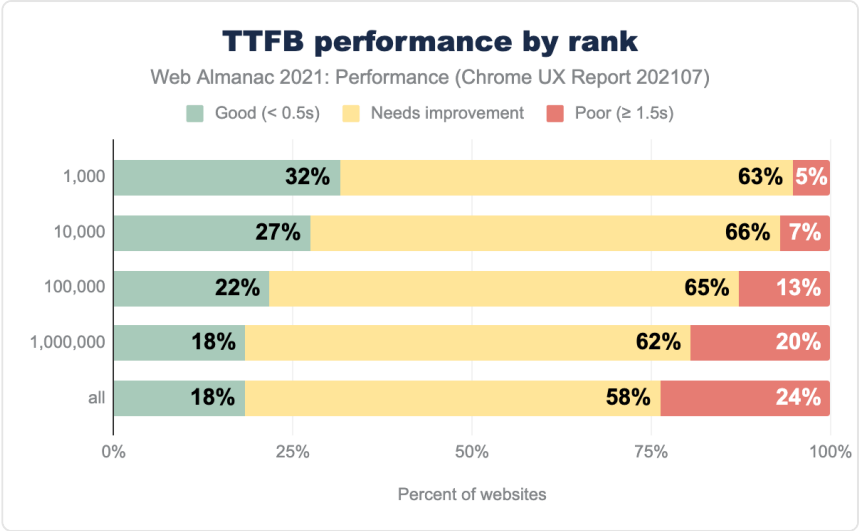


図10.8. TTFBのランク別パフォーマンス

ランクについては、高ランクのサイトほどTTFBが速かった。その理由のひとつは、これらの多くは、パフォーマンスを優先するためのより多くのリソースを持つ大企業であることでしょう。これらの企業は、サーバーサイドのパフォーマンスの向上や、エッジCDNを通じたアセット配信に注力している可能性があります。もうひとつの理由は、選択バイアスです。上位のオリジンは、サーバーが近い地域、つまり低遅延の地域でより多くアクセスされる可能性があります。

もう1つ可能性があるとすれば、CMSの導入が関係している。CMS編では、ランク別のCMS導入状況を示しています。

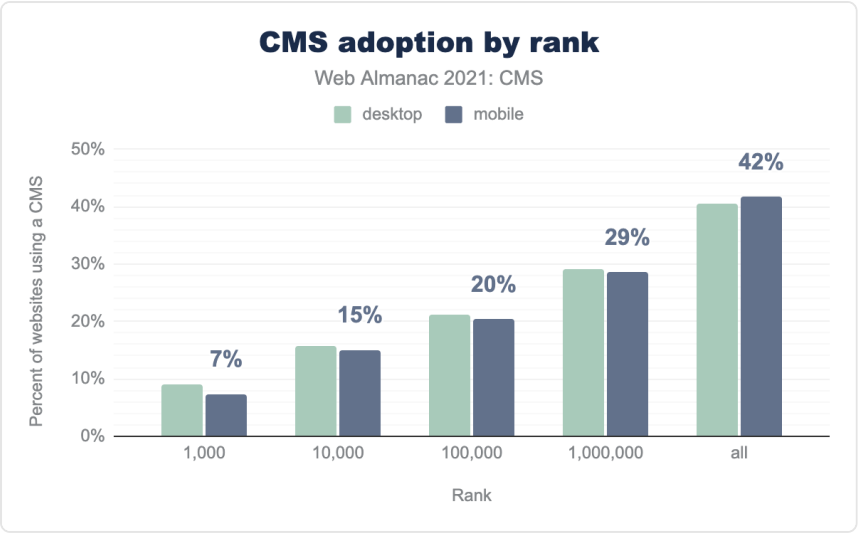


図10.9. ランク別CMS採用状況

「すべて」のグループでは42%のページ（モバイル）がCMSを使用しているのに対し、上位1,000サイトでは7%の導入にとどまっています。

次に、上位5つのCMSを順位別に見ると、WordPressが「全ページ」の33.6%を占め、もっとも導入が進んでいることがわかります。

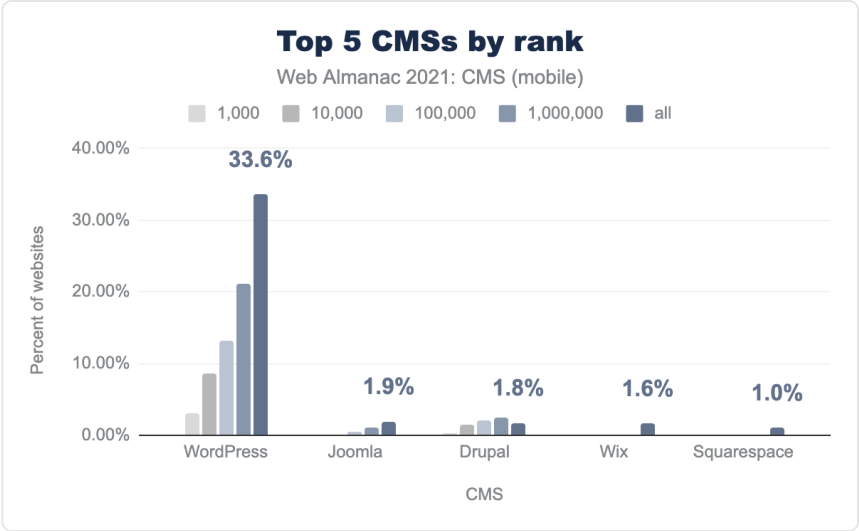


図10.10. CMSのランキングトップ5

最後に、Core Web Vitalsテクノロジーレポート[419]を見ると、各CMSが指標別にどのようなパフォーマンスを見せてくれるかがわかります。

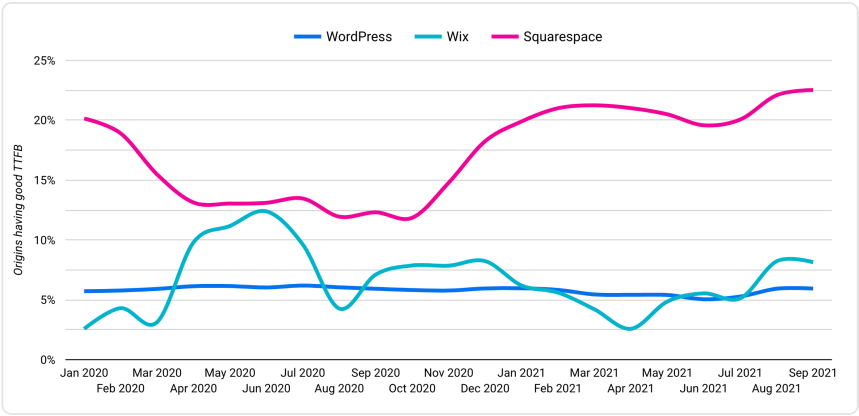


図10.11. CMSによる良いTTFBを持つオリジン (Core Web Vitalsテクノロジーレポート[420])

2021年7月に良好なTTFBを経験したWordPressのオリジンはわずか5%でした。上位1000万サイトにおけるWordPressの大きなシェアを考慮すると、そのTTFBの悪さがランクによるTTFBの劣化の一因である可能性があります。

419. *https://datastudio.google.com/s/o6zLzlTpWaI*
420. *https://datastudio.google.com/s/o6zLzlTpWaI*

コンテンツの初回ペイント (FCP)

コンテンツの初回ペイント (FCP)[421] は、読み込みが最初に開始されてから、ブラウザがページのコンテンツ部分（テキスト、画像など）を最初にレンダリングするまでの時間を測定するものです。

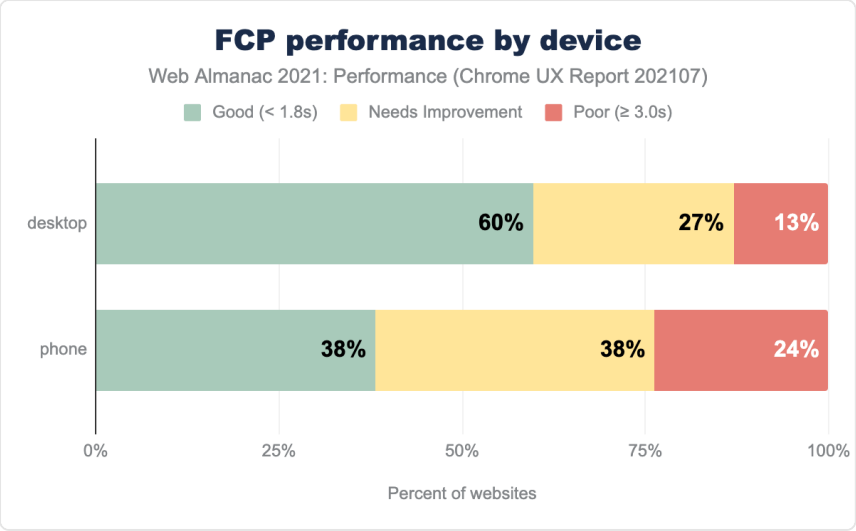


図10.12. デバイス別のFCPパフォーマンス

モバイルよりもデスクトップの方が、FCPが速いのは平均ネットワーク速度が速いことと、プロセッサが速いことの両方が原因だと思われます。モバイルでのFCPが良好だったのは、38%のオリジンだけでした。同期JavaScriptのようなレンダリングをブロックするリソースは、一般的な原因である可能性があります。TTFBはFCPの最初の部分なので、TTFBが悪いと、良好なFCPを達成するのが難しくなります。

注：昨年からFCPの閾値が変更されています。今年のデータを昨年のデータと比較しようとする場合は、注意が必要です。

421. *https://web.dev/i18n/ja/fcp/*

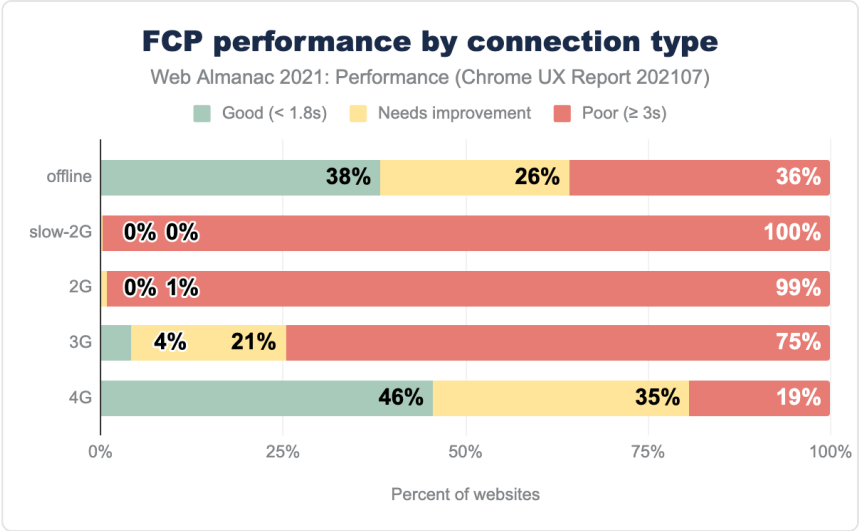


図10.13. 接続タイプ別のFCPパフォーマンス

3G以下のスピードのオリジンでは、FCPで著しい劣化が発生しました。繰り返しになりますが、分析によるユーザーデータを反映した実際のデバイスとネットワークを使用して、ウェブサイトのプロファイリングを行っていることを確認してください。光ファイバー接続のハイエンドデスクトップでプロファイリングしている場合、JavaScriptのバンドルは重要でないように見えるかもしれません。

オフライン接続は、4Gに近い性能でしたが、それほど良いものではありませんでした。サービスワーカーの起動時間や複数のキャッシュの読み込みが影響している可能性があります。FCPでは、TTFBよりも多くの要因が絡んできます。

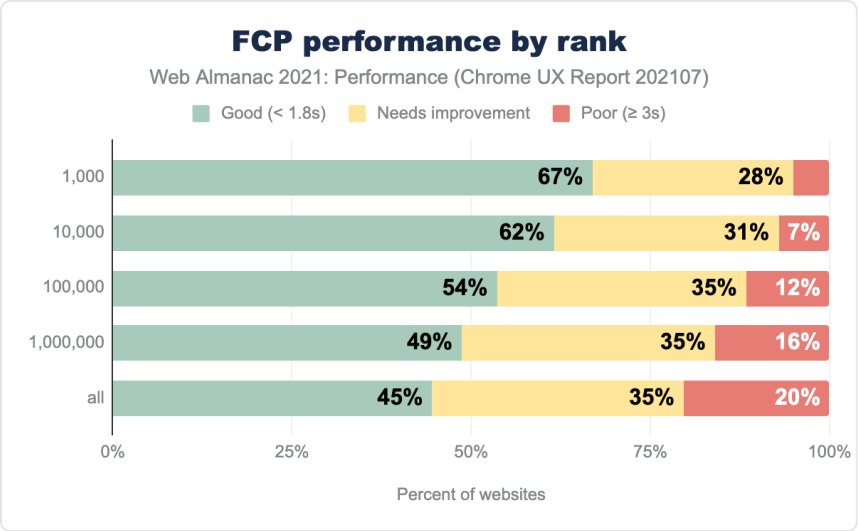


図10.14. FCPのランク別パフォーマンス

TTFBと同様、FCPも上位にランクインして改善されました。また、TTFBと同様に、WordPressのオリジンの19.5% のみが、FCPの良いパフォーマンスを経験しました[422]。TTFBの性能が低いのだから、FCPも遅いのは当然です。TTFBが遅いと、FCPやLCPで良いスコアを出すのは難しいのです。

FCPがうまくいかない原因としては、レンダリングをブロックするリソース、サーバーの応答時間（TTFBの遅さに関連するもの）、大きなネットワークペイロード、などが挙げられます。

最大コンテンツの描画（LCP）

最大コンテンツの描画（LCP）[423] は、ロード開始からブラウザがビューポートで最大の画像またはテキストを表示するまでの時間を測定します。

422. https://datastudio.google.com/s/kZ9K0d-sBQw
423. https://web.dev/i18n/ja/lcp/

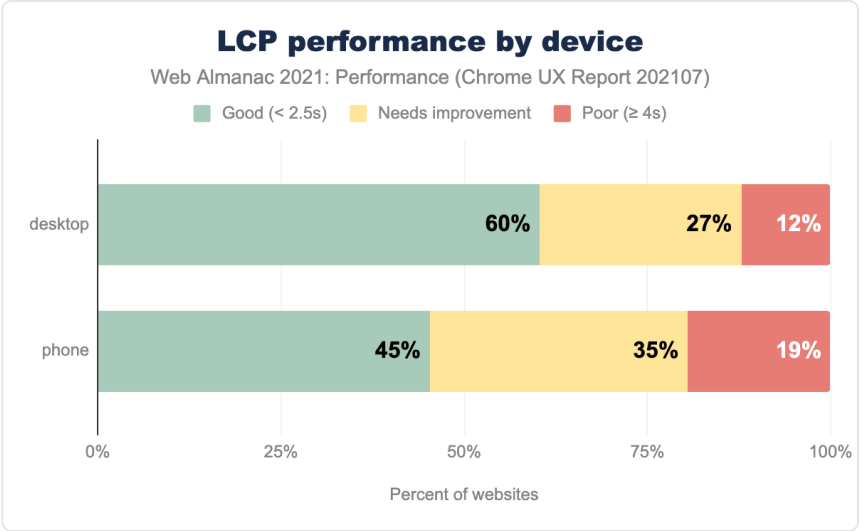


図10.15. LCPのデバイス別性能

LCPはモバイルよりデスクトップの方が速かった。TTFBはFCPのようにLCPに影響を与える。 デバイス、接続形態、ランクによる比較は、すべてFCPの傾向を反映しています。レンダーブロックリソース、総重量、ロードストラテジーはすべてLCPのパフォーマンスに影響を与えます。

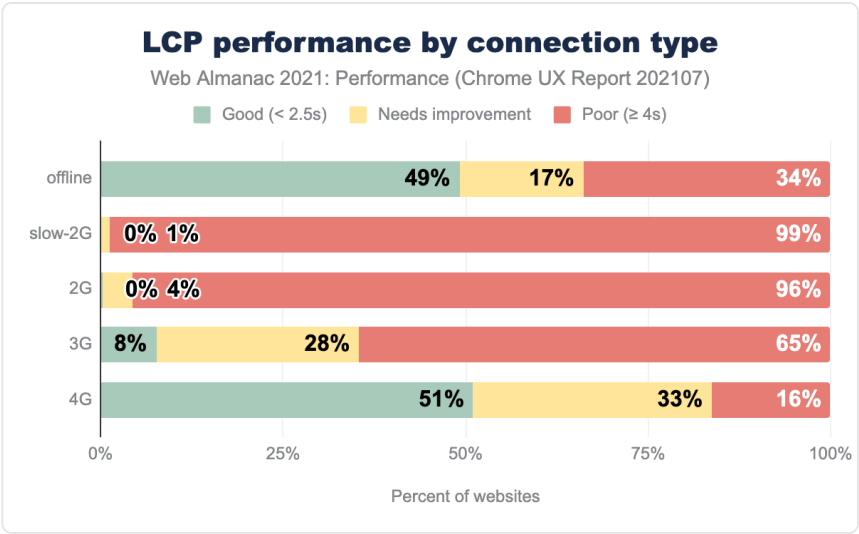


図10.16. 接続タイプ別のLCPパフォーマンス

LCPが良好なオフラインの起点は、4Gの体験とより密接に一致していますが、LCPが悪い体験はオフラインの方が高いのです。LCPはFCPの後に発生し、0.7秒の追加予算があるため、FCPよりもオフラインのウェブサイトの方が良いLCPを達成した可能性があります。

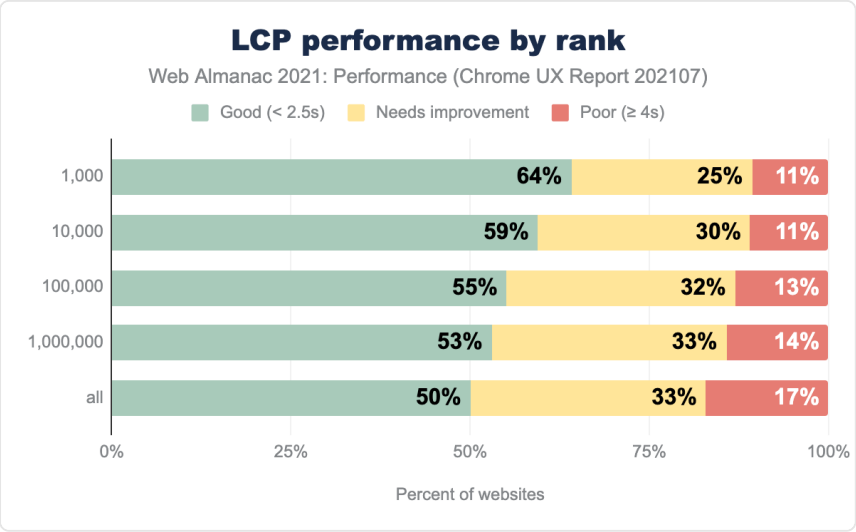


図10.17. LCPのランク別パフォーマンス

LCPについては、FCPよりも順位によるパフォーマンスの差が接近している。また、上位1,000位までのオリジンではLCPが、悪い割合が高くなっています。WordPressでは、28%のオリジンが良好なLCPを経験[424]しています。LCPの悪化は通常、一握りの問題によって引き起こされるため、これはユーザーエクスペリエンスを向上させるチャンスです。

LCPの要素

LCPの要素をより深く掘り下げてみましょう。

424. *https://datastudio.google.com/s/kvq1oJ60jaQ*

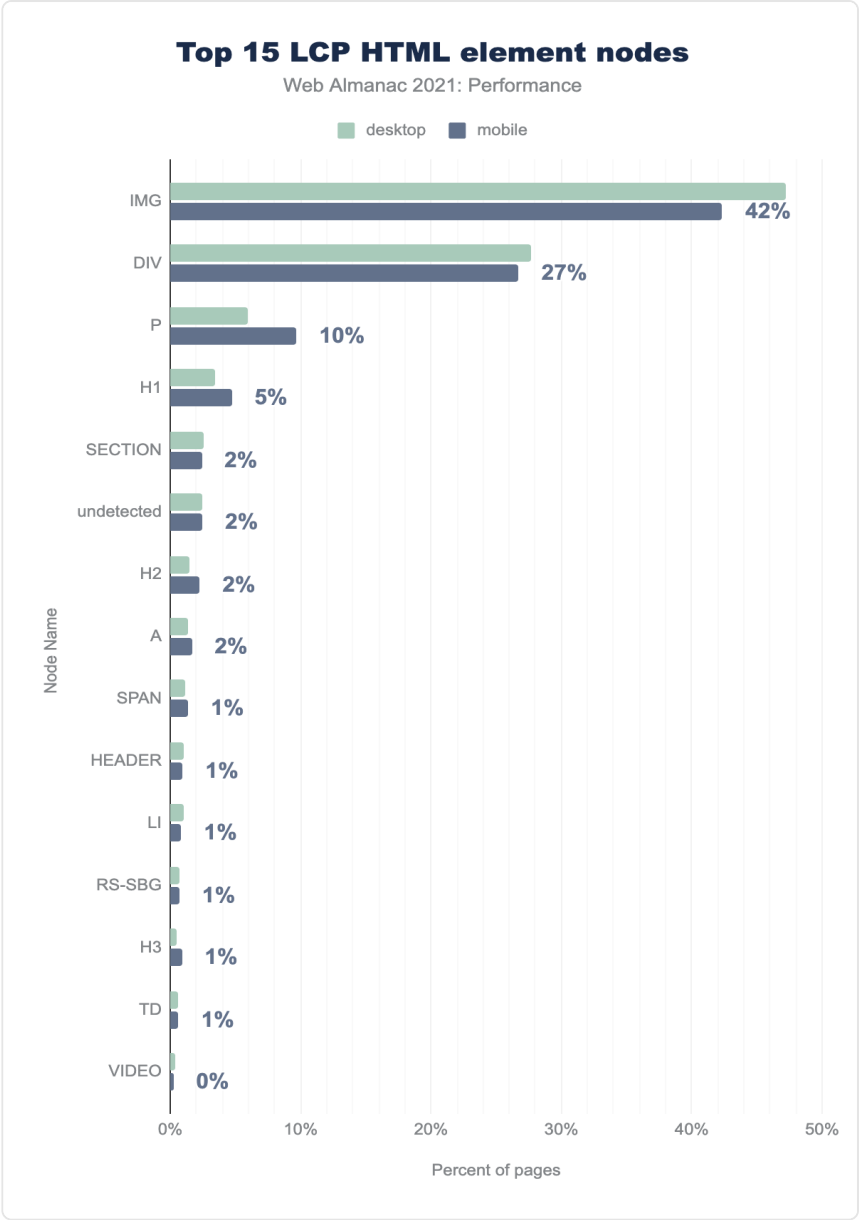


図10.18. LCP HTML要素ノード上位15個

IMG、DIV、P、H1がLCPノード全体の83%を占めた（モバイルの場合）。背景画像はCSSで適用できるため、コンテンツが画像なのかテキストなのかは、これだけではわかりませ

ん。

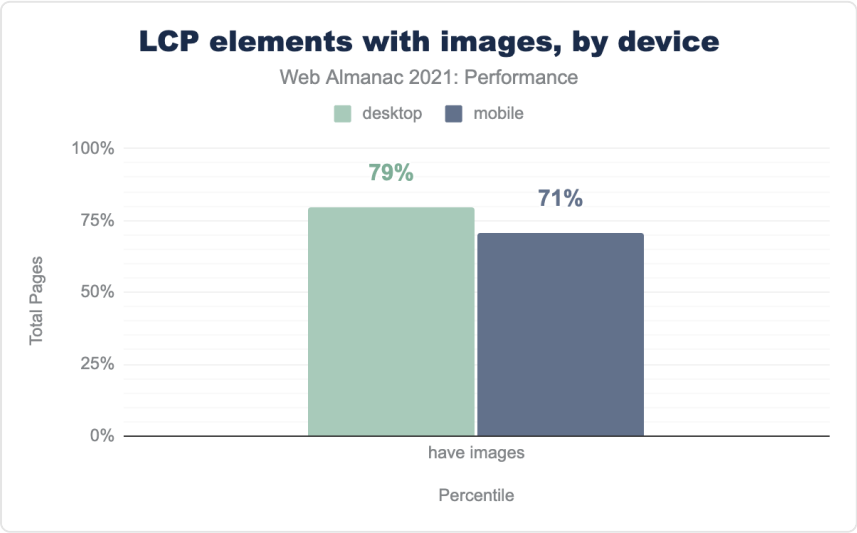


図10.19. デバイス別画像付きLCP素子

HTMLノードに関係なく、71-79%のページでLCP要素が、画像であったことがわかります。さらに、デスクトップ端末ではLCPが、画像である割合が高くなっています。これは、画面サイズが小さいため、画像がビューポートからはみ出し、見出しテキストがもっとも大きな要素になることが、原因である可能性があります。

いずれの場合も、LCP要素の大半を画像が占めています。このため、画像の読み込み方法について、より深く掘り下げる必要があります。

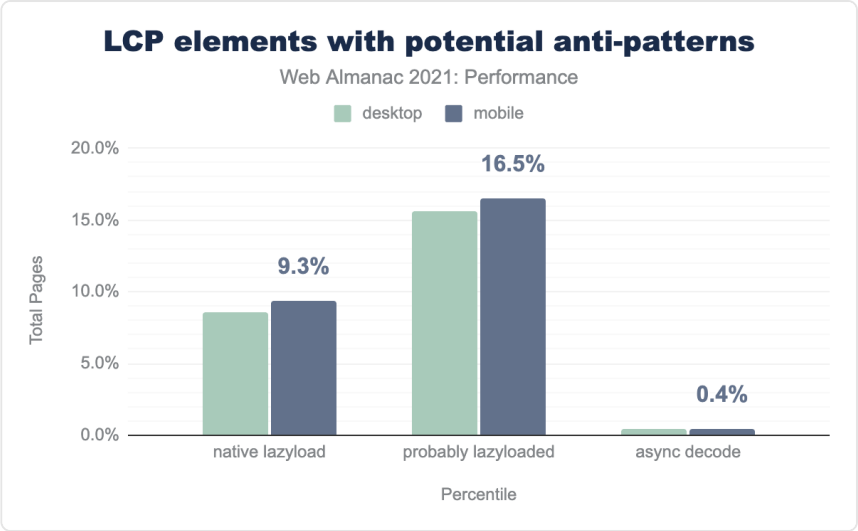


図10.20. アンチパターンの可能性があるLCP要素

ユーザー体験の観点から、LCPの要素はできるだけ速くロードされることが望まれます。LCPがCore Web Vitalの1つに選ばれた理由は、ユーザー体験です。レンダリングがさらに遅くなるため、遅延ロードさせないようにします。しかし、9.3%のページがLCPの `<img>` 要素にネイティブのloading=lazyフラグを使用していることがわかります。

すべてのブラウザがネイティブの遅延ローディングに対応しているわけではありません。一般的な遅延ロードのポリフィルは、画像要素に「lazyload」クラスがあると検出します。このように、”lazyload“クラスを持つ画像を集計に加えることで、より遅延ロードの可能性が高い画像を特定できます。LCPの `<img>` 要素を遅延ロードしていると思われるサイトの割合は、モバイルでは16.5%に跳ね上がります。

LCP要素を遅延ローディングすると、パフォーマンスが低下します。 *やめてくれ！* WordPressはネイティブの遅延ローディングを早くから採用しています。初期の方法は、すべての画像に遅延ロードを適用する素朴なソリューションで、結果は 負の性能相関[425]を示しました。このデータをもとに、より良いパフォーマンスを発揮するために、よりニュアンスの異なるアプローチを実施することができたのです。

画像の `decode` 属性は、比較的新しいものです。 `async` に設定することで、読み込みやスクロールのパフォーマンスを向上させることができます。現在、0.4%のサイトがLCP画像に非同期デコード指令を使用しています。非同期デコードがLCP画像に与える悪影響は、現在のところ不明です。したがって、LCPイメージに `decode="async"` を設定する場合は、設定前と設定後にサイトをテストしてください。

425. *https://web.dev/articles/lcp-lazy-loading*

354

図10.21. 画像やiframeでないLCP要素にネイティブ遅延ローディングを使おうとしたウェブサイト

興味深いことに、デスクトップの354のオリジンは、loading属性をサポートしないHTML要素（例：`<div>`）に対してネイティブの遅延ロードを使用しようとしました。loading属性は`<img>`と、一部のブラウザでは`<iframe>`要素にのみ対応しています（Can I use[426]を参照してください）。

累積レイアウトシフト (CLS)

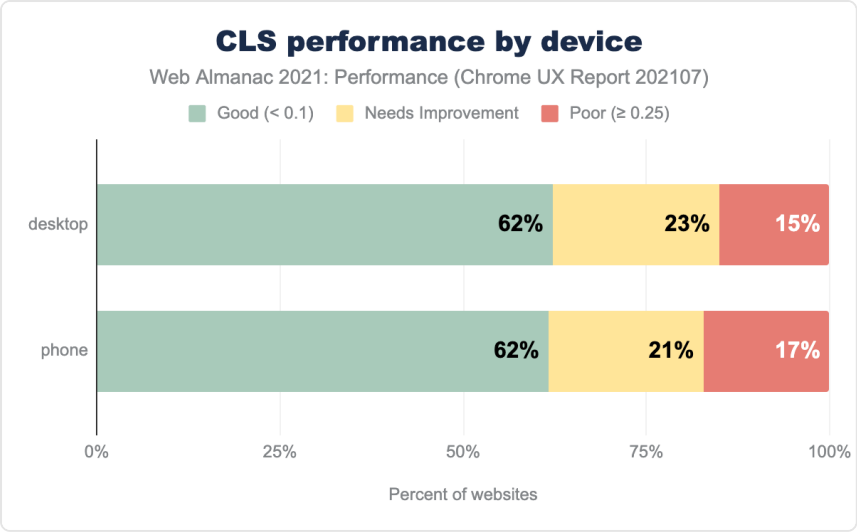


図10.22. デバイス別CLS性能

累積レイアウトシフト (CLS)[427] は、FCPやLCPなどの視覚的な表示にかかる時間ではなく、ユーザーがどの程度のレイアウト シフトを経験するかによって特徴付けられます。このように、デバイスごとの性能はほぼ同等でした。

426. https://caniuse.com/loading-lazy-attr
427. https://web.dev/i18n/ja/cls/

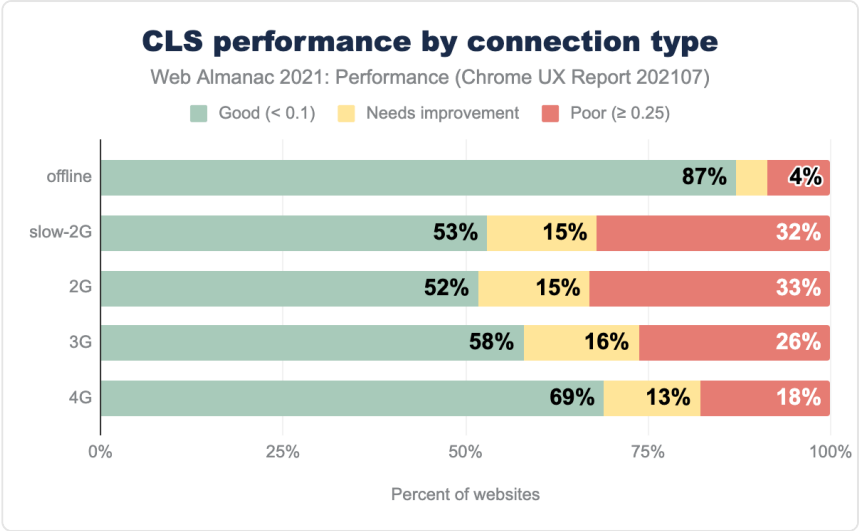


図10.23. 接続タイプ別のCLSパフォーマンス

4Gから3G以下への性能劣化は、FCPやLCPほど顕著ではありませんでした。多少の劣化はあるが、端末データには反映されず、接続タイプのみに反映されます。

オフラインのウェブサイトは、すべての接続タイプの中でもっとも高いCLSパフォーマンスを示しました。サービスワーカーキャッシングを使用しているサイトでは、レイアウトシフトの原因となる画像や広告などの一部のアセットでキャッシュされない場合があります。そのため、これらのアセットは読み込まれず、レイアウトシフトを引き起こすことはありません。このようなサイトのフォールバックHTMLは、オンラインウェブサイトのより基本的なバージョンになることがよくあります。

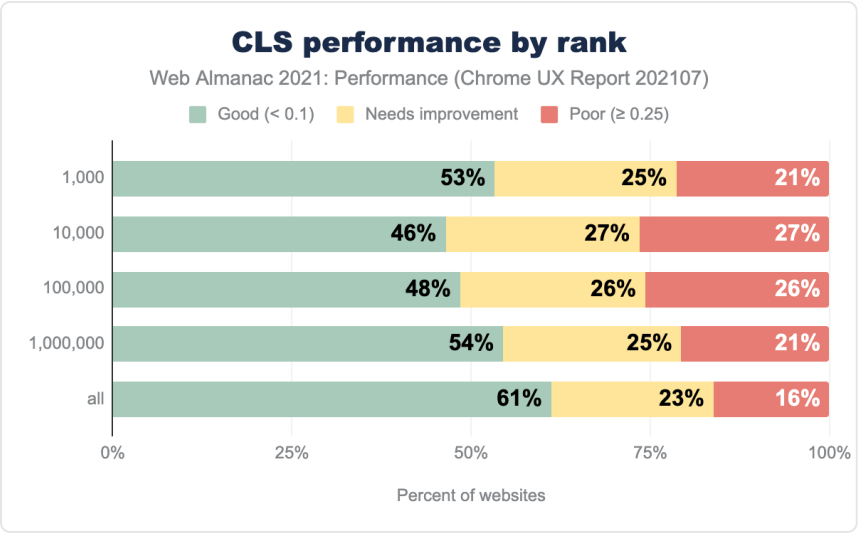


図10.24. CLSのランク別パフォーマンス

ランキングについては、CLSのパフォーマンスは、上位10,000サイトで興味深い谷を示しました。また、1M以上のランキンググループは、1M未満のランキングサイトよりもパフォーマンスが低下しています。「all」グループは、他のすべてのランク付けされたグループよりもパフォーマンスが良かったので、1M以下のグループのパフォーマンスが向上しています。WordPressを使ったオリジンの60%が良いCLSを経験した[428]ことから、WordPressが再びこれに関与していると思われます。

CLSが悪くなる原因としては画像のスペースが確保されていない、ウェブフォントを読み込むと文字がずれる、最初に描いた後トップバナーが挿入される。またアニメーションが合成されていない、iframeがある、などが挙げられます。

初回入力遅延 (FID)

初回入力遅延 (FID)[429] は、ユーザーが最初にページと対話してから、その対話に応答してブラウザがイベントハンドラーの処理を開始するまでの時間を測ります。

428. *https://datastudio.google.com/s/qG00yMxSa3o*
429. *https://web.dev/i18n/ja/fid/*

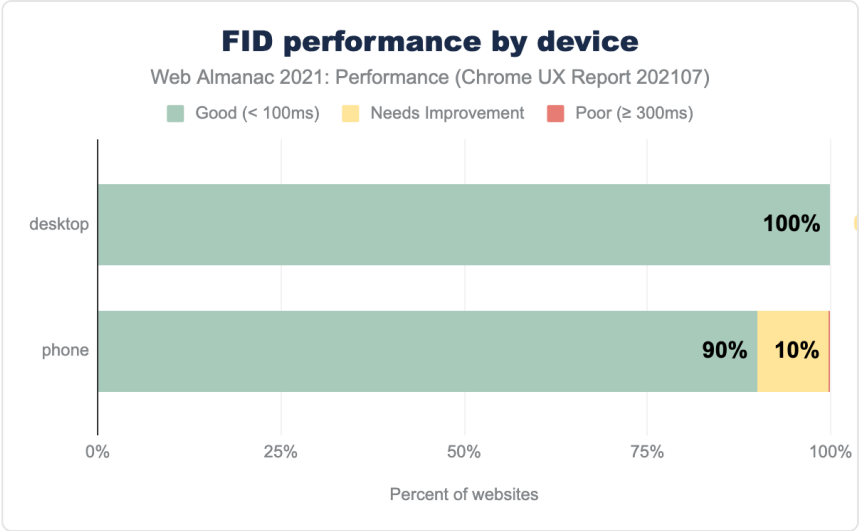


図10.25. デバイス別のFID性能

FIDのパフォーマンスは、モバイル端末よりもデスクトップ端末の方が優れており、これは、より大量のJavaScriptを処理できる端末のスピードが速いためと思われます。

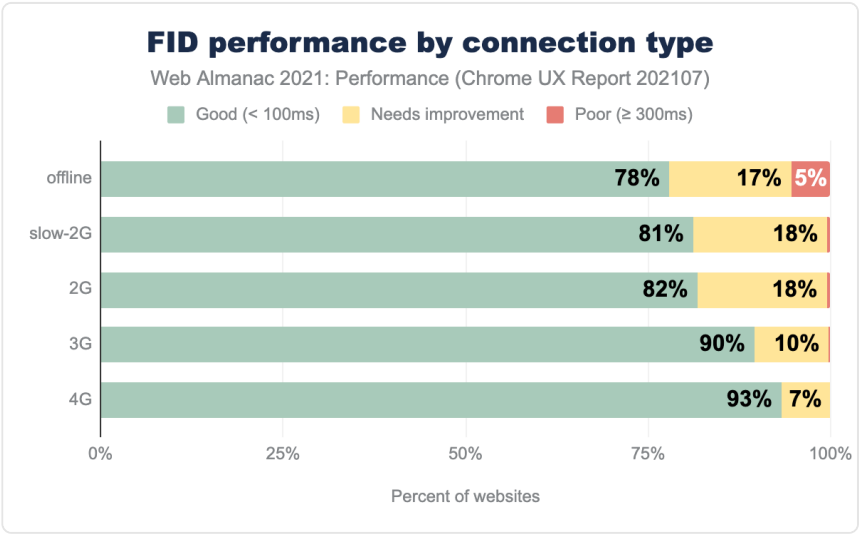


図10.26. 接続タイプ別のFID性能

FIDの性能は、接続の種類によって多少劣化しますが、他の指標に比べるとそれほどではあ

りません。スコアの分布が大きいため、結果のばらつきが少なくなっているようです。

他の指標とは異なり、オフラインのウェブサイトでは、他の接続カテゴリーよりもFIDが悪化していました。これは、サービスワーカーを持つ多くのウェブサイトがより複雑な性質を持つためと思われます。サービスワーカーがあっても、メインスレッドで実行されるクライアントサイドのJavaScriptの影響がなくなるわけではありません。

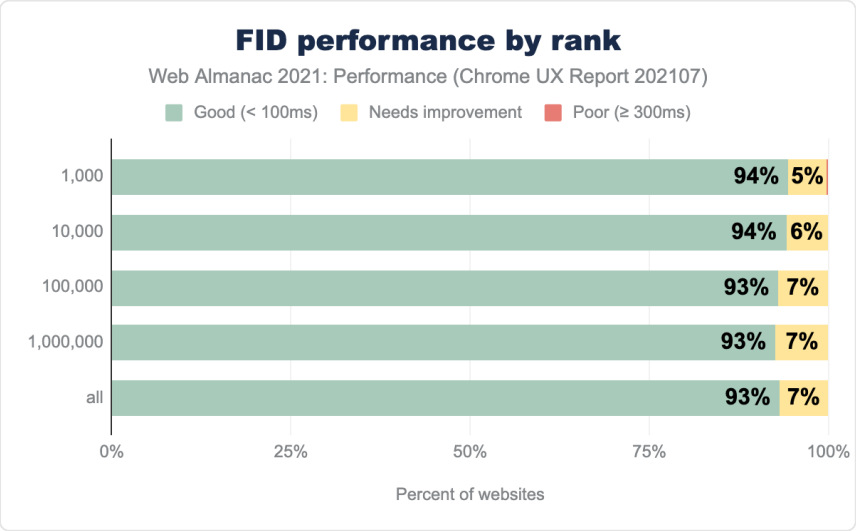


図10.27. FIDのランク別パフォーマンス

FIDのランク別実績は横ばい。

すべてのFID指標において、「良好」のカテゴリーに非常に大きなバーが表示されますが、これは本当にピーク性能に達していない限り、あまり効果がないことを示しています。良いニュースは、Chromeチームが現在これを評価しており[430]、あなたのフィードバックを求めていることです。

もし、あなたのサイトのパフォーマンスが「良い」の範疇にないのであれば、間違いなくパフォーマンスに問題があります。FID問題の一般的な原因は、長時間実行されるJavaScriptが多すぎることです。バンドルサイズを小さくし、サードパーティのスクリプトに注意を払いましょう。

430. *https://web.dev/better-responsiveness-metric/*

合計ブロッキング時間 (TBT)

> *合計ブロッキング時間 (TBT) メトリクスは、コンテンツの初回ペイント (FCP) とユーザー操作可能になるまでの時間 (TTI) の間で、メインスレッドが入力反応を妨げるほど長くブロックされた時間の総計を測定します。*
>
> *— Web.dev*[431]

合計ブロッキング時間 (TBT)[432] は、潜在的な双方向性の問題をデバッグするのに役立つラボベースのメトリックです。FIDはフィールドベースの指標であり、TBTはラボベースの類似指標である。現在、私はクライアントのWebサイトを評価する際、JavaScriptによるパフォーマンスの問題の可能性を示す別の指標として、総ブロック時間TBTに着目しています。

残念ながら、TBTはChromeユーザー体験レポートでは計測されていません。しかし、HTTP Archive Lighthouseのデータ（モバイルのみ収集）を使用すれば、状況を把握することは可能です。

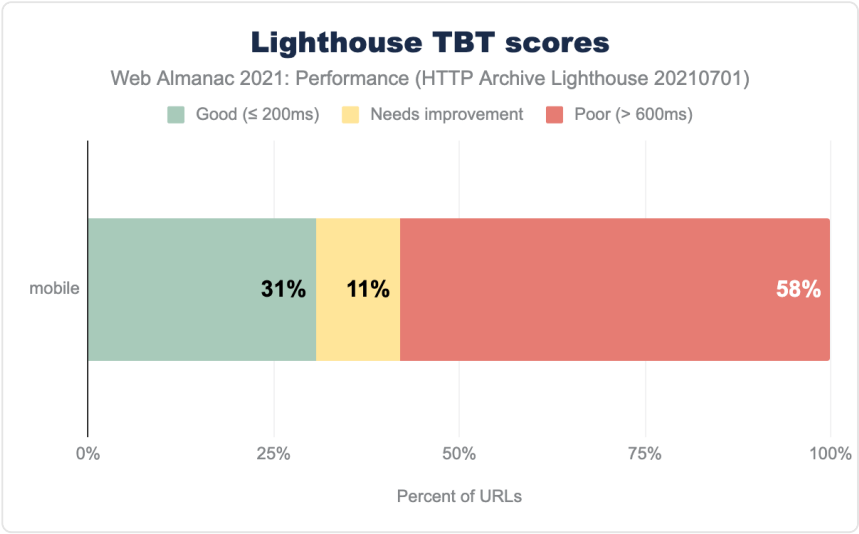


図10.28. Lighthouse TBTスコア

注：グラフのグループは、TBTのLighthouseスコアに基づいています（例：>= 0.9は「良い」結果）。スコアの四捨五入により、200msをわずかに超えるTBT値が「良好」に分類されています（600msの閾値でも同様です）。

431. https://web.dev/i18n/ja/tbt/
432. https://web.dev/i18n/ja/tbt/

このデータは、WebPageTestでスロットルCPUのLighthouseを1回実行したものであり、実際のユーザー体験を反映したものではないことに留意してください。しかし、TBTとFIDを比較すると、潜在的なインタラクティビティははるかに悪く見えます。インタラクティブ性の「本当の」評価は、おそらくこの間のどこかにあるのでしょう。ですから、もしFIDが「良い」のであれば、FIDでは捉えられないような劣悪なユーザー体験を見逃しているかもしれないので、TBTを見てみてください。FIDが悪いのと同じ問題は、TBTも悪いのです。



図10.29. 最長のTBT

結論

2020年よりパフォーマンスは向上しました。素晴らしいユーザー体験を提供するにはまだ長い道のりがありますが、改善するための手段を講じることは可能です。

まず、パフォーマンスを測定できなければ、改善することはできません。ここでは、実際のユーザーの端末を使ってサイトを測定し、リアルユーザーモニタリング（RUM）を設定することが良い第一歩となります。CrUX dashboard launcher[433] で、あなたのサイトがChromeユーザーでどのように機能しているかを知ることができます（あなたのサイトがデータセットに含まれている場合）。複数のブラウザにまたがって測定するRUMソリューションを設定する必要があります。このソリューションは自分で構築することもできますし、多くの分析ベンダーのソリューションの1つを利用することもできます。

次に、HTML、CSS、JavaScriptの新機能がリリースされたら、それを理解した上で実装することです。新しい戦略を採用した結果、パフォーマンスが向上したかどうかをA/Bテストで検証する。たとえば、フォールドより上にある画像をダラダラとロードしないなどです。RUMツールを実装していれば、自分の変更が誤ってリグレッションを引き起こしたときに、より適切に検出できます。

第三に、FID（現場・実使用データ）とTBT（実験室データ）の両方に対する最適化を継続することです。新しい応答性指標の 提案[434] を見て、フィードバックを提供して参加してください。また、新しいアニメーションの滑らかさの指標[435]も提案されています。より高速なウェブを目指す私たちにとって、変化は必然であり、より良い方向へと向かっています。私たちが最適化を続けていくためには、お客様の参加が重要です。

最後に、WordPressは上位1000万のウェブサイト、あるいはそれ以上のパフォーマンスに影響を与える可能性があることを確認しました。これは、すべてのCMSとフレームワークが留

433. *https://rviscomi.github.io/crux-dash-launcher/*
434. *https://web.dev/responsiveness/*
435. *https://web.dev/smoothness/*

意すべき教訓です。フレームワークレベルでパフォーマンスに関するスマートなデフォルトを設定することができれば、より良いウェブを作ることができ、同時に開発者の仕事を容易にできます。

あなたがもっとも興味を持ったこと、驚いたことは何ですか？あなたの感想をTwitter（@HTTPArchive）でシェアしてください。

著者



Sia Karamalegos

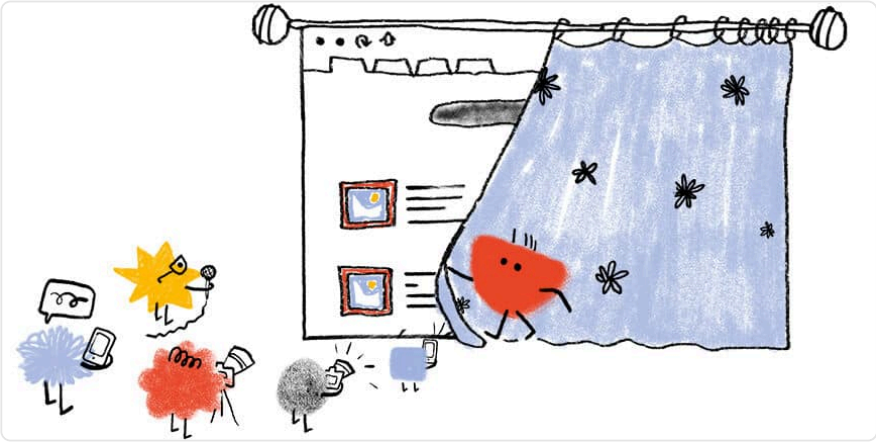
@https://front-end.social/@sia @sia.codes siakaramalegos karamalegos

https://sia.codes

Sia Karamalegosはウェブ開発者、国際会議のスピーカー、ライターです。Googleデベロッパー ウェブテクノロジーの専門家、Cloudinaryメディア開発者エキスパート、Stripeコミュニティエキスパートであり、Eleventy Meetupを共同開催しています。sia.codes[436] やTwitter[437] で、彼女の執筆、講演、ニュースレターをチェックしてください。

436. *https://sia.codes/*
437. *https://x.com/thegreengreek*

部 II 章 11
Privacy



Yana Dimova と Victor Le Pochat によって書かれた。
Maud Nalpas によってレビュ。
Victor Le Pochat と Max Ostapenko による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

*インターネットでは、あなたが犬であることを誰も知らない*確かに、そのようにインターネットを使うため匿名にすることはできるかもしれませんが、個人情報を完全に非公開にすることはかなり難しいでしょう。

業界全体[438]は、ターゲット広告、詐欺検出、価格差別化、あるいは信用スコアリングなどの目的で詳細なユーザープロファイルを構築するために、オンラインでユーザーを追跡することに専念しています。ウェブサイトと地理位置情報を共有することは、日常生活において非常に便利ですが、企業があなたのすべての行動を見る[439]ことができるようになる可能性もあります。たとえサービスがユーザーの個人情報を真摯に扱っていたとしても、個人情報を保存するという行為だけで、ハッカーはサービスに侵入し、数百万の個人記録をオンラインで

438. https://crackedlabs.org/en/corporate-surveillance/
439. https://www.nytimes.com/interactive/2019/12/19/opinion/location-tracking-cell-phone.html

流出させる[440]機会を与えてしまうのです。

欧州ではGDPR[441]など、最近の立法化の動きもあります。カリフォルニア州のCCPA[442]、ブラジルのLGPD[443]や、インドのPDP Bill[444]は、いずれも企業に個人データの保護を求め、オンラインも含めてデフォルトでプライバシーを実装しようとするものです。Google、Facebook、Amazonなどの大手テクノロジー企業は、ユーザーのプライバシー侵害の疑いで、すでに巨額の罰金[445]を受け取っています。

この新しい法律により、ユーザーは個人情報を共有することにどれだけ抵抗がないか、より大きな発言権を持つようになりました。おそらく、すでに多くのクッキー同意バナーをクリックし、この選択を可能にしていることでしょう。さらにウェブブラウザは、機密データを隠してサードパーティのクッキーをブロックすることから、個人の属性に関する正当なユースケースと個々のユーザーのプライバシーのバランスをとる革新的な方法まで、ユーザーのプライバシーを改善する技術的解決策[446] を実装しています。

この章では、ウェブにおけるプライバシーの現状を概観する。まず、ユーザーのプライバシーがどのように害されうるかを考える。ここでは、ウェブサイトがどのようにオンライントラッキングによってあなたをプロファイリングし、どのようにあなたの機密データにアクセスしているかについて議論します。次に、ウェブサイトが機密データを保護する方法と、プライバシー設定シグナルによってユーザーに選択肢を与える方法について掘り下げます。最後に、今後のブラウザによるプライバシー保護への取り組みについての展望を掲載します。

ウェブサイトがあなたをプロファイリングする方法：オンライントラッキング

HTTPプロトコルは本質的にステートレスなので、デフォルトでは2つの異なるウェブサイトへの訪問、あるいは同じウェブサイトへの2つの訪問が、同じユーザーによるものかどうかをウェブサイトが知る方法はありません。しかし、このような情報はウェブサイトがよりパーソナライズされたユーザー体験を構築するために、また第三者がウェブサイト間でユーザー行動のプロファイルを構築しターゲット広告を通じてウェブ上のコンテンツに資金を提供したり、不正行為の検出などのサービスを提供するために有用であると思われます。

残念ながら、この情報を得るには、現状ではオンライントラッキングに頼ることが多く、それを中心に多くの大小の企業がビジネスを展開している[447]。これにより、侵襲的な追跡はユーザーのプライバシーと矛盾するため、ターゲット広告の禁止[448]を求める声にもつながって

440. *https://haveibeenpwned.com/*
441. *https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu*
442. *https://www.oag.ca.gov/privacy/ccpa*
443. *https://www.gov.br/cidadania/pt-br/acesso-a-informacao/lgpd*
444. *https://www.meity.gov.in/data-protection-framework*
445. *https://en.wikipedia.org/wiki/GDPR_fines_and_notices*
446. *https://privacysandbox.com/*
447. *https://crackedlabs.org/en/corporate-surveillance/*
448. *https://www.forbrukerradet.no/wp-content/uploads/2021/06/20210622-final-report-time-to-ban-surveillance-based-advertising.pdf*

いるのです。とくにデリケートな話題のウェブサイトを閲覧する場合、ユーザーは自分の足取りを誰にも知られたくないかもしれません。ここでは、オンライントラッキングのエコシステムを構成する主な企業や技術について見ていきます。

サードパーティトラッキング

オンライントラッキングは、多くの場合、サードパーティのライブラリを通じて行われます。これらのライブラリは通常、何らかの（有用な）サービスを提供しますが、その過程で各ユーザーに固有の識別子を生成、それを使ってウェブサイト全体でユーザーを追跡し、プロファイルすることができるものもあります。WhoTracksMe[449] プロジェクトは、もっとも広く展開されているオンライントラッカーを発見することに専念しています。私たちはWhoTracksMeのトラッカーの分類を使用していますが、トラッキングが主目的の一部であるサービスをカバーする可能性がもっとも高いため、カテゴリ[450]の4つに限定しています。*広告*、*ポルノグラフィティ*、*サイト分析*および*ソーシャル・メディア*です。

449. *https://whotracks.me/*
450. *https://whotracks.me/blog/tracker_categories.html*

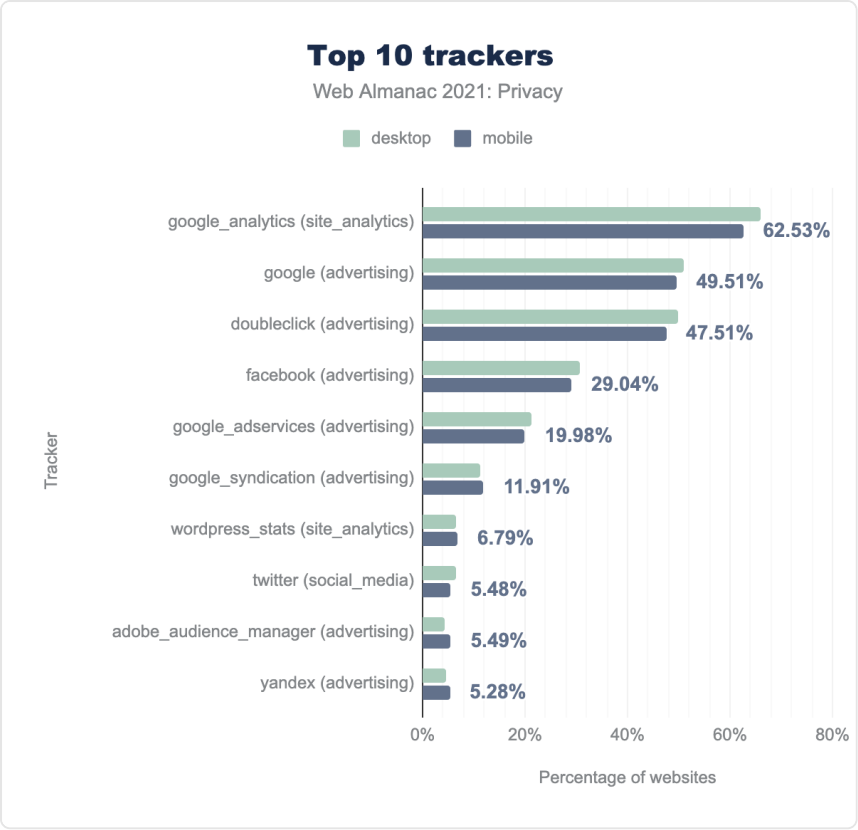


図11.1. 人気のトラッカー10選とその普及率

オンライントラッキング市場では、Googleの所有するドメインが広く普及していることがわかります。ウェブサイトのトラフィックを報告するGoogle Analyticsは、全ウェブサイトのほぼ3分の2に存在しています。約30%のサイトがFacebookライブラリを含んでおり、他のトラッカーは一桁のパーセンテージにしか達しません。

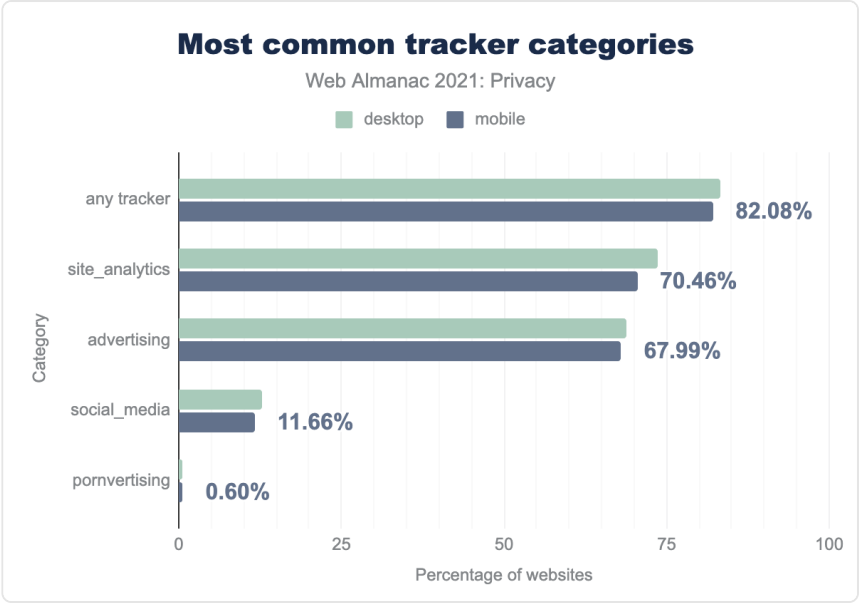


図11.2. もっとも一般的なトラッカーカテゴリ。

全体では、モバイルサイトの82.08%、デスクトップサイトの83.33%が少なくとも1つのトラッカーを含んでおり、通常はサイト分析または広告目的のために利用されています。

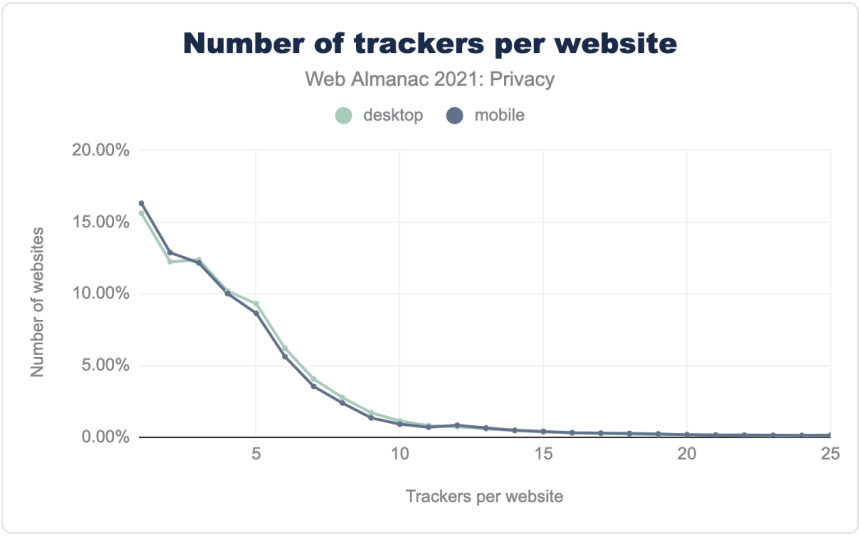


図11.3. ウェブサイトごとのトラッカー数。

4つのウェブサイトのうち3つはトラッカーが10個以下ですが、それ以上のトラッカーを持つサイトもあり、あるデスクトップ・サイトでは133個（！）の異なるトラッカーに接続しています。

サードパーティークッキー

クロスサイトのユーザー識別子を保存および取得するための主な技術的アプローチは、ブラウザに永続的に保存されるクッキーを通じて行われます。サードパーティのクッキーは、クロスサイトトラッキングによく使われますが、サイト間でサードパーティのウィジェットの状態を共有するような、トラッキング以外のユースケースにも使われることがあることに注意してください。私たちは、ウェブを閲覧しているときにもっとも頻繁に表示されるクッキーと、それを設定するドメインを検索しました。

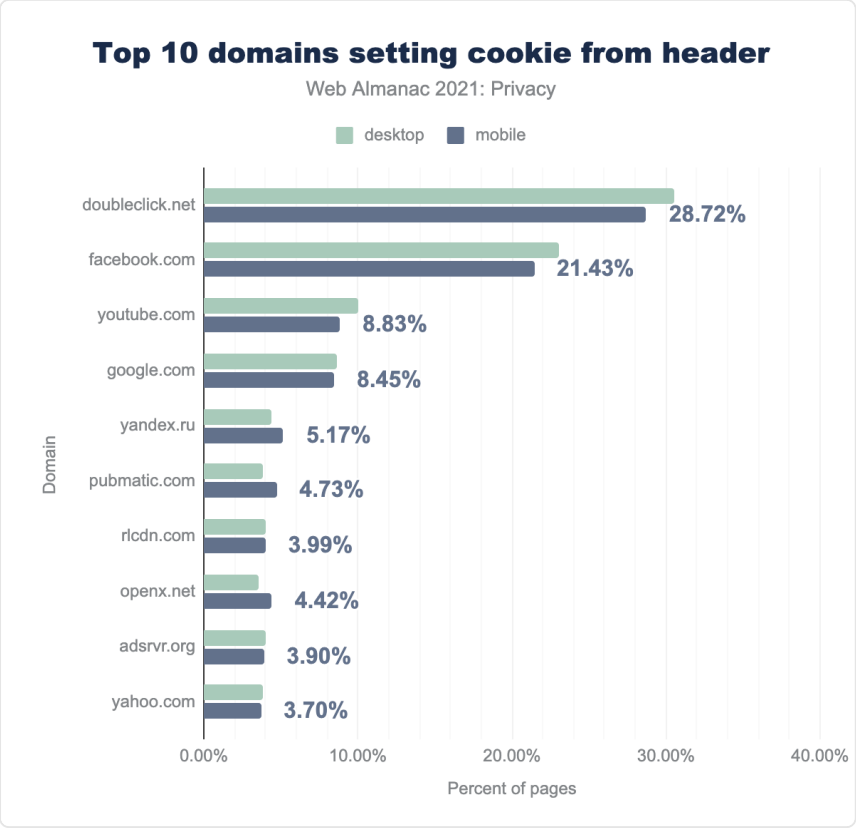


図11.4. ヘッダーからクッキーを設定するドメイントップ10。

グーグルの子会社であるDoubleClickは、デスクトップサイトの31.4%、モバイルサイトの28.7%にクッキーを設定し、首位に立っています。もう1つの主要なプレーヤーはFacebookで、モバイルウェブサイトの21.4%にクッキーを保存しています。クッキーを設定する他の上位ドメインのほとんどは、オンライン広告に関連しています。

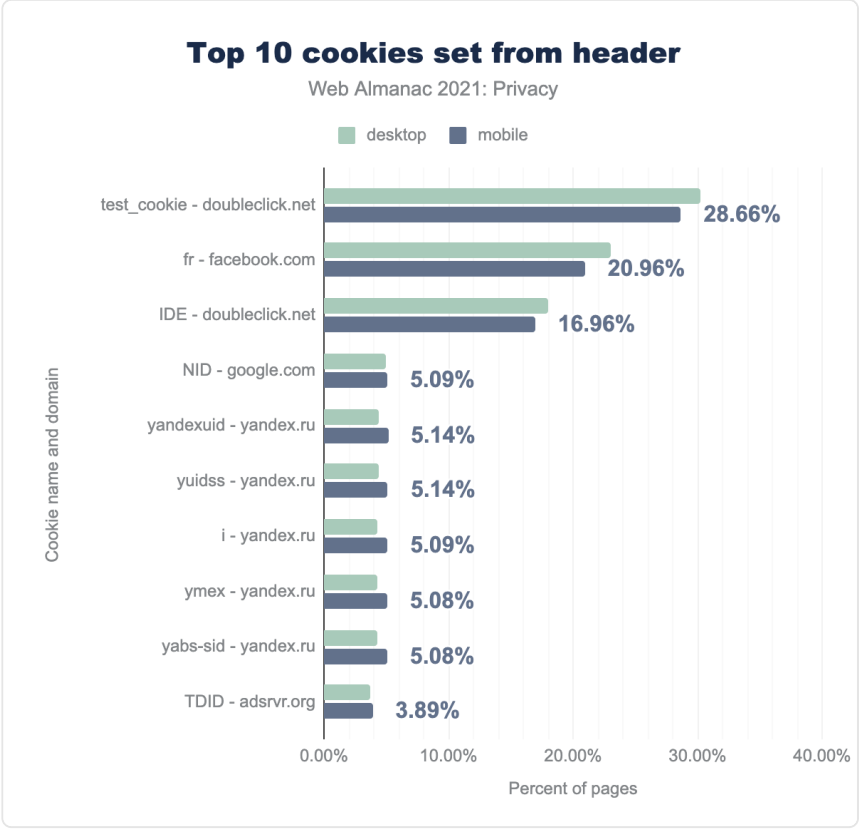


図11.5. ヘッダーから設定されたトップ10のクッキー。

これらのウェブサイトが設定する特定のクッキーを見てみると、トラッカーからのもっとも一般的なクッキーは、doubleclick.netの `test_cookie` です。次に多いのは広告関連のクッキーで、ユーザーの端末にずっと長く残ります。Facebookの `fr` クッキーは90日間[451]、DoubleClickの `IDE` クッキーはヨーロッパでは13か月、その他の地域では2年[452]持続する。

`Lax` が `SameSite` クッキー属性 のデフォルト値になったため、ウェブサイト間でサードパーティのクッキーを共有し続けたいサイトは、この属性を明示的に `None` に設定しなければ

451. *https://www.facebook.com/policy/cookies/*
452. *https://business.safety.google/adscookies/*

ならなくなりました。サードパーティーの場合、モバイルでは85%、デスクトップでは64%が、トラッキングを目的とする可能性があるとして、これまでに実施しています。`SameSite`クッキー属性については、セキュリティーの章で詳しく説明されています。

指紋認証

広告ブロックなどのプライバシー保護ツールの台頭やFirefox[453]、Safari[454]、そして2023年にはChrome[455]などの主要ブラウザからサードパーティCookieを段階的に排除する取り組みにより、トラッカーはサイト間でユーザーを追跡する、より持続的で密かな方法を探しています。

その1つが*ブラウザフィンガープリント*です。ウェブサイトは、ユーザーエージェント[456]、画面解像度、インストールされているフォントなど、ユーザーのデバイスに関する情報を収集し、それらの値のユニークな組み合わせを使用して*フィンガープリント*を作成できます。このフィンガープリントは、ユーザーがウェブサイトを訪れるたびに再作成され、照合することでユーザーを特定することができるのです。この方法は、不正行為の検出に使用できますが、繰り返し利用するユーザーを持続的に追跡したり、サイト間でユーザーを追跡するためにも使用されます。

フィンガープリントの検出は複雑です。メソッドコールとイベントリスナーの組み合わせにより効果を発揮し、追跡以外の目的にも使用されることがあります。そこで、これらの個々のメソッドに焦点を当てる代わりに、ウェブサイトがフィンガープリントを簡単に実装できるようにする5つの人気のあるライブラリに焦点を当てます。

453. *https://blog.mozilla.org/en/products/firefox/todays-firefox-blocks-third-party-tracking-cookies-and-cryptomining-by-default/*
454. *https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/*
455. *https://blog.google/products/chrome/updated-timeline-privacy-sandbox-milestones/#:~:text=Chrome%20could%20then%20phase%20out%20third-party%20cookies%20over%20a%20three%20month%20period%2C%20starting%20in%20mid-2023%20and%20ending%20in%20late%202023*
456. *https://developer.mozilla.org/docs/Glossary/User_agent*

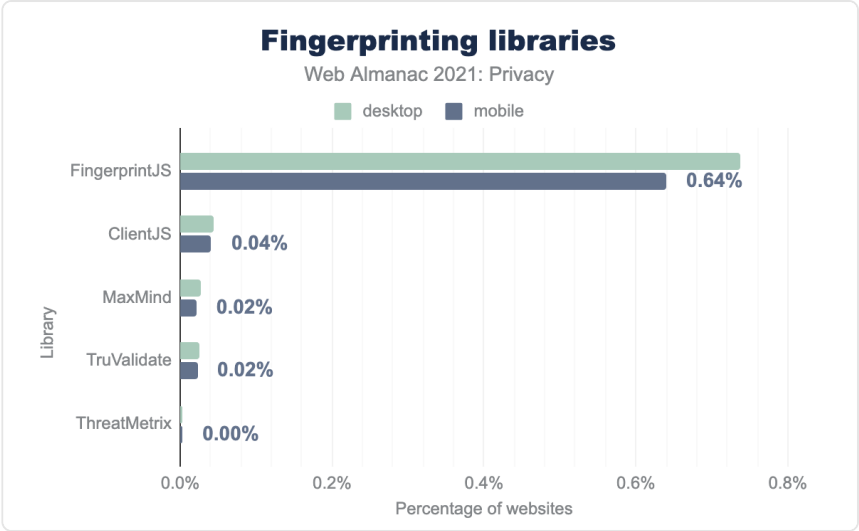


図11.6. 各フィンガープリント・ライブラリーを使用しているウェブサイト。

これらのサードパーティ サービスを使用しているWebサイトの割合から、もっとも広く使用されているライブラリである Fingerprint.js[457] は、デスクトップでは2番目に多く使用されているライブラリの19倍も使用されていることがわかります。しかし、ユーザーのフィンガープリントに外部ライブラリを使用しているWebサイトの割合は、全体としては非常に小さいものです。

CNAMEトラッキング

サードパーティトラッキングのブロックを回避するテクニックを続けると、CNAMEトラッキング[458]は、ファーストパーティのサブドメインがDNSレベル[459]でCNAMEレコードを使ってサードパーティサービスの使用をマスクするという新しい方法です。ブラウザから見ると、すべてがファーストパーティの文脈で行われるため、サードパーティの対策は一切適用されない。AdobeやOracleなどの大手トラッキング会社は、すでにCNAMEトラッキングソリューションを顧客に提供しています。 この章に含まれるCNAMEベースのトラッキングの結果については、この章の著者の一人（および他の人）によって完成した 研究[460] を参照し、DNSデータとHTTP Archiveからのリクエストデータに基づいてCNAMEベースのトラッキングを検出する方法を開発しました。

457. *https://fingerprintjs.com/*
458. *https://medium.com/nextdns/cname-cloaking-the-dangerous-disguise-of-third-party-trackers-195205dc522a*
459. *https://adguard.com/en/blog/cname-tracking.html*
460. *https://sciendo.com/article/10.2478/popets-2021-0053*

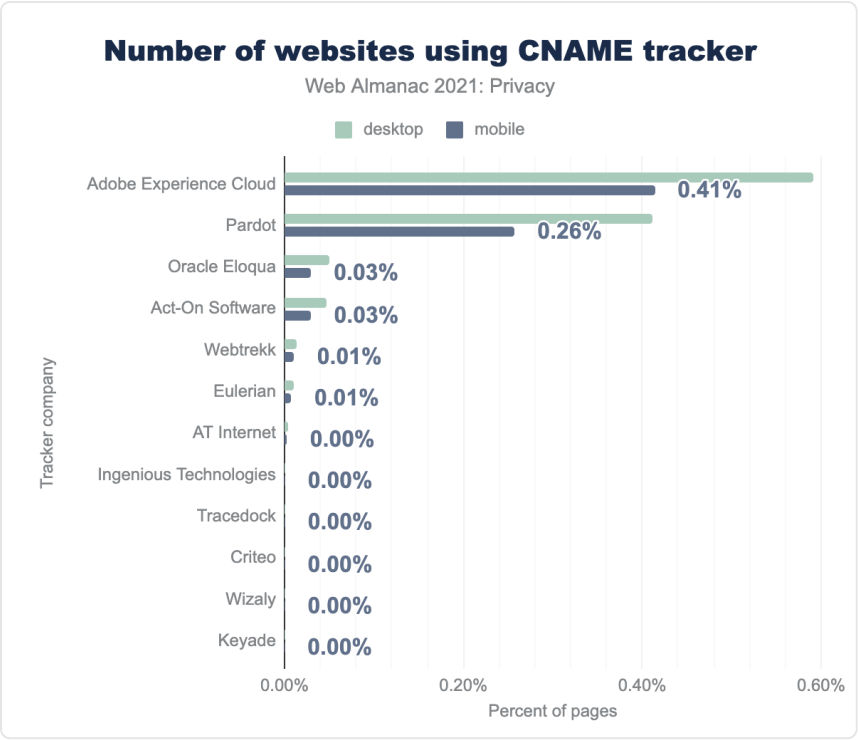


図11.7. デスクトップクライアントでCNAMEベースのトラッキングを使用しているWebサイト。

CNAMEベースのトラッキングを実行しているもっとも人気のある企業はAdobeで、デスクトップWebサイトの0.59%、モバイルWebサイトの0.41%に存在しています。また、規模ではPardot[461]が0.41%、0.26%と顕著です。

この数字は小さな割合に見えるかもしれませんが、サイトの人気度でデータを分離すると、その意見は変わります。

461. *https://www.pardot.com/*

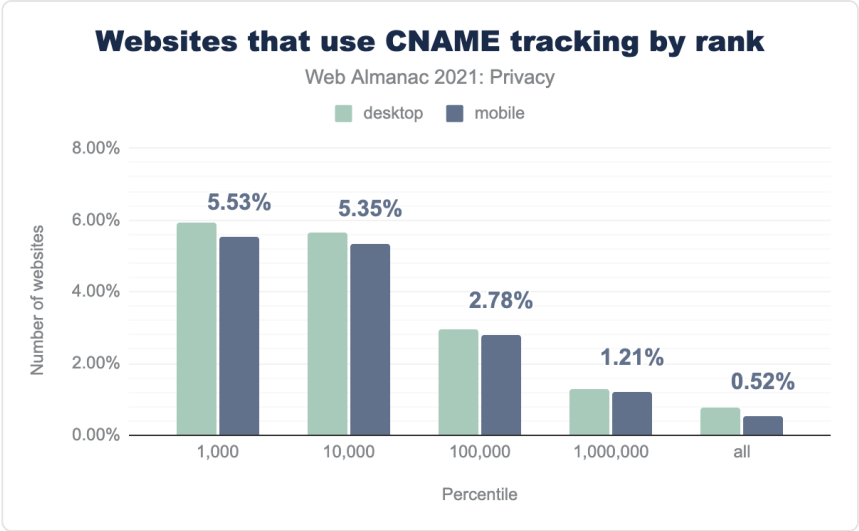


図11.8. CNAMEトラッキングを使用しているウェブサイトをランク別に紹介。

CNAMEベースのトラッキングを使用しているウェブサイトのランクを見ると、モバイルの上位1,000ウェブサイトのうち5.53%がCNAMEトラッカーを埋め込んでいることがわかります。上位100,000サイトでは、その数は2.78%に減少し、データセット全体を見ると0.52%に減少しています。

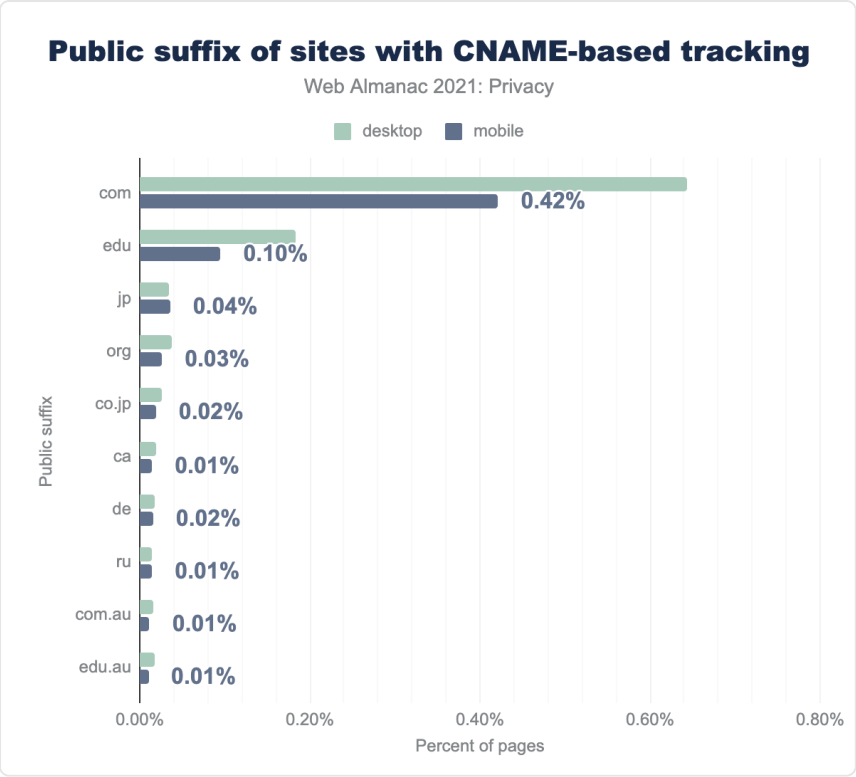


図11.9. CNAMEベースのトラッキングを行うサイトのパブリックサフィックス。

`.com` サフィックスとは別に、CNAMEベースのトラッキングを使用しているウェブサイトの多くは `.edu` ドメインを持っています。また、CNAMEトラッカーは `.jp` と `.org` ウェブサイトに多く存在します。

CNAMEベースのトラッキングは、ユーザーがサードパーティーのトラッキングに対するトラッキング保護を有効にしている可能性がある場合の対策になります。トラッカーブロックツールやブラウザ[462]は、すでにCNAMEトラッキングに対する防御機能を備えているものが少ないため、現在までに多くのウェブサイトで蔓延しているのが現状です。

（再）ターゲティング

広告再ターゲティングとは、ユーザーが見たが購入していない商品を記録し、その商品に関する広告を別のWebサイトでフォローアップすることを指します。ユーザーが訪問している

462. https://www.cookiestatus.com/

間、積極的なマーケティング戦略を選ぶのではなく、ウェブサイトは継続的にブランドと製品を思い出させることで、ユーザーが製品を購入するように誘導することを選択します。

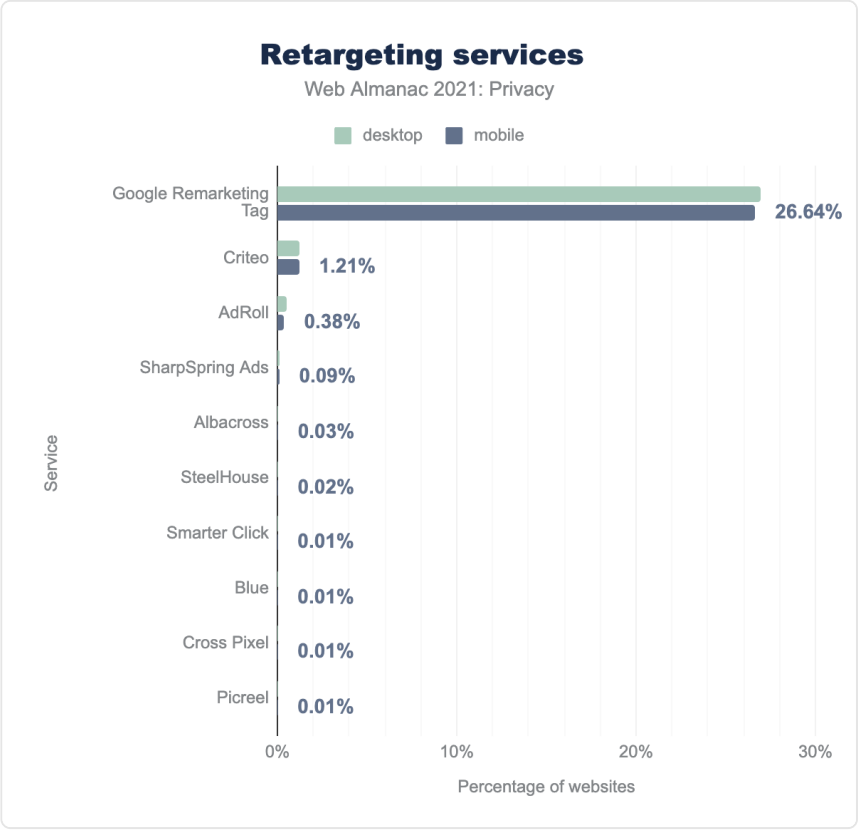


図11.10. 再ターゲティングサービスを利用しているページの割合。

多くのトラッカーが広告再ターゲティングのためのソリューションを提供しています。もっとも広く利用されているGoogle Remarketing Tagは、デスクトップのウェブサイトの26.92%、モバイルのウェブサイトの26.64%で利用されており、それぞれ1.25%未満で利用されている他のサービスよりはるかに優れています。

ウェブサイトにおけるお客様の個人情報の取り扱いについて

ウェブサイトによっては、ジオロケーションデータ、マイク、カメラなどにアクセスすることで、ユーザーのプライバシーに影響を与える可能性のある特定の機能およびブラウザAPIへのアクセスを要求するものがあります。これらの機能は通常、近くの観光スポットを発見

したり、人々が互いに通信できるようにするなど、非常に有用な目的を果たします。これらの機能はユーザーが同意した場合にのみ有効になりますが、ユーザーがこれらのリソースの使用方法を十分に理解していない場合や、サイトが誤動作した場合、機密データが、公開されるリスクがあります。

Webサイトが機密性の高いリソースへのアクセスを要求する頻度を調べました。さらに、サービスが機密データを保存する場合は常に、ハッカーがそのデータを盗み出し、漏えいさせる危険性があるのです。ここでは、この危険性が実際に存在することを証明する、最近のデータ漏洩事件について見ていきます。

デバイスセンサー

センサーは、ウェブサイトをよりインタラクティブへするのに便利ですが、指紋認証[463]のために悪用される可能性もあります。JavaScriptのイベントリスナーの使用状況から、モバイル、デスクトップクライアントで、デバイスの向きがもっともアクセスされていることがわかります。なお、Webサイト上でイベントリスナーの存在を検索したが、実際にコードが実行されたかどうかはわからない。したがって、本節のデバイスセンサーイベントへのアクセスは、上限値です。

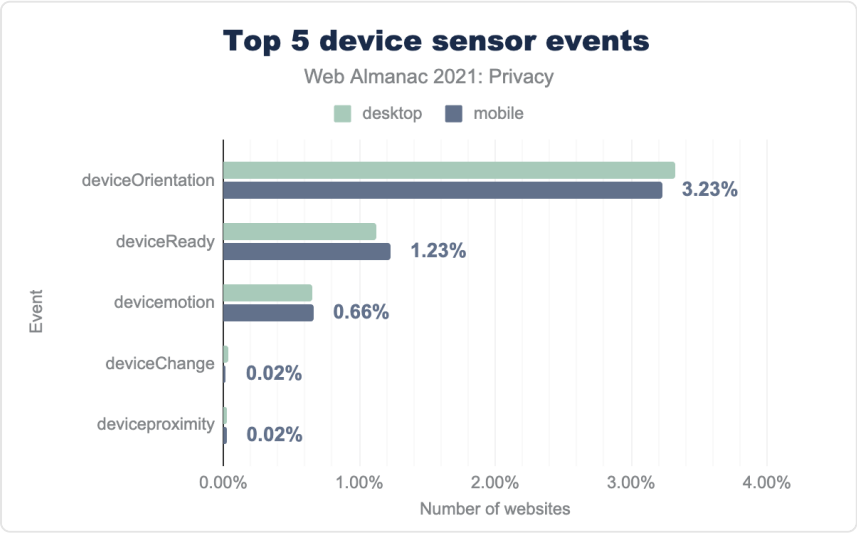


図11.11. もっとも使用されている5つのセンサーイベント。

463. https://www.esat.kuleuven.be/cosic/publications/article-3078.pdf

メディアデバイス

MediaDevices API[464]を使用すると、カメラやマイク、画面共有などの接続されたメディア入力にアクセスできます。



図11.12. `MediaDevicesEnumerateDevices` APIを使用したデスクトップページの割合。

デスクトップのウェブサイトの7.23%、モバイルのウェブサイトの5.33%で `enumerateDevices()` メソッドが呼び出され、接続されている入力デバイスのリストが提供されます。

ジオロケーションサービス

ジオロケーションサービスは、GPSやユーザーのその他の位置情報（IPアドレス[465]など）を提供し、トラッカーはとくにユーザーにより関連性の高いコンテンツを提供するために利用できます。そこで、Wappalyzerで検出したライブラリをもとに、Webサイトにおける「ジオロケーションサービス」技術の利用を分析します。

464. https://developer.mozilla.org/docs/Web/API/MediaDevices
465. https://developer.mozilla.org/docs/Glossary/IP_Address

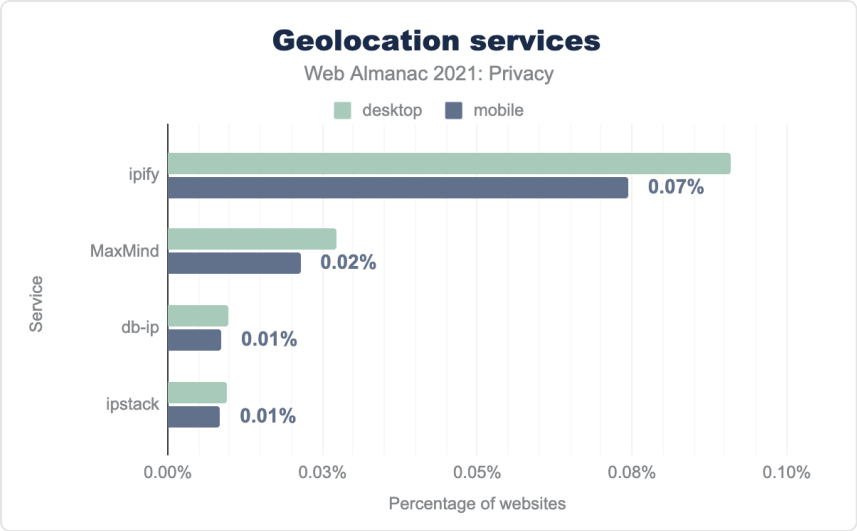


図11.13. ジオロケーションサービスを利用しているWebサイトの割合。

もっとも人気のあるサービスであるipify[466]は、デスクトップWebサイトの0.09%とモバイルWebサイトの0.07% で使用されていることがわかります。つまり、ジオロケーションサービスを利用しているウェブサイトは少ないようです。

466. https://www.ipify.org/

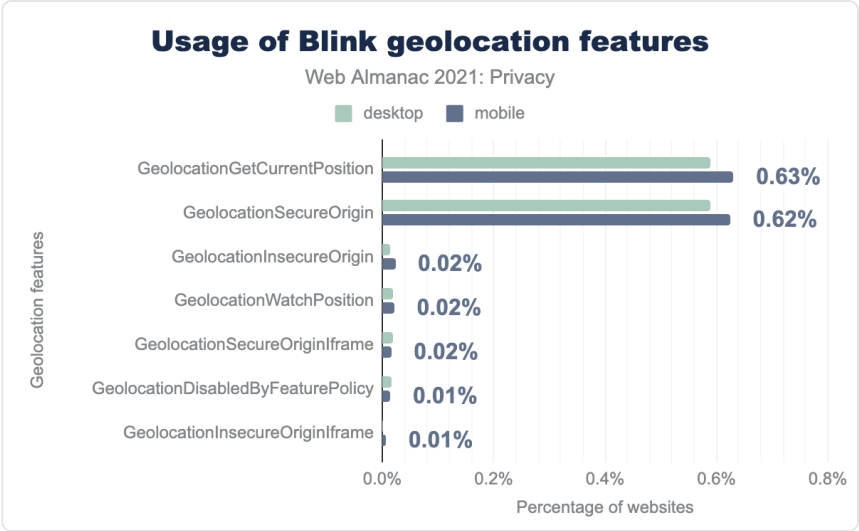


図11.14. ジオロケーション機能を利用しているWebサイトの割合。

また、WebサイトからはWebブラウザAPI[467]を介して、ジオロケーションデータにアクセスすることが可能です。デスクトップクライアントで0.59%、モバイルクライアントでは0.63%のWebサイトがユーザーの現在位置にアクセスしていることがわかります（ブリンク機能に基づいています）。

情報漏えい

企業におけるセキュリティ管理の不備は、お客様の個人情報にも大きな影響を及ぼします。Have I Been Pwned[468] は、ユーザーが自分のメールアドレスや電話番号がデータ漏洩で流出したかどうかを確認することができるサービスです。この記事を書いている時点で、Have I Been Pwnedは562件の情報漏えいを追跡し、6億4000万件の記録が漏れていることが判明しています。2020年だけでも40のサービスが侵入され、数百万人のユーザーの個人情報が流出した。このうち3件はセンシティブとされ、誰かがそのユーザーのデータを見つけた場合、ユーザーに悪影響が、及ぶ可能性があることを指しています。機密漏洩の一例として、盗まれたクレジットカードが、取引されるプラットフォーム「Carding Mafia[469]」が挙げられます。

なお、前年度の40件の違反は、多くの違反が発生してから数カ月後に発見、または公表されるため、下限値であることに注意してください。

467. https://developer.mozilla.org/docs/Web/API/Geolocation_API
468. https://haveibeenpwned.com/
469. https://www.vice.com/en/article/v7m9jx/credit-card-hacking-forum-gets-hacked-exposing-300000-hackers-accounts

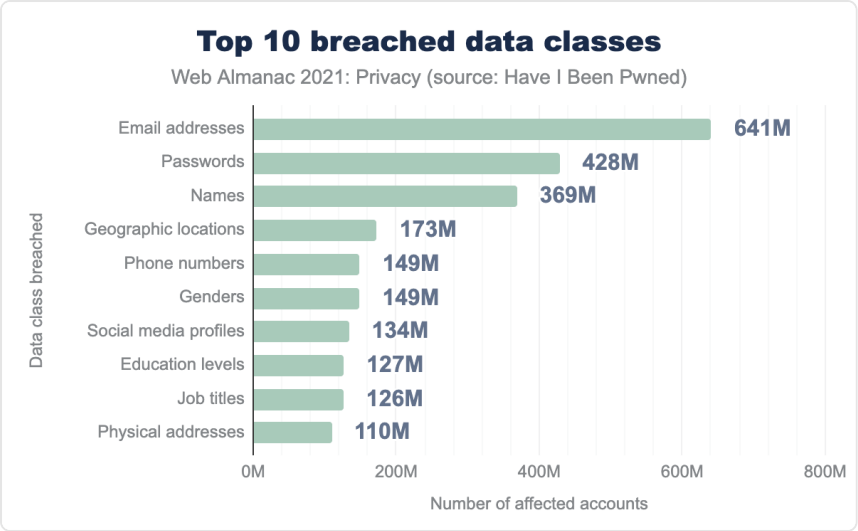


図11.15. データクラスごとの違反で影響を受けたアカウント数。(出典：Have I Been Pwned[470])

Have I Been Pwned[471] が追跡したすべてのデータ侵害では、電子メールアドレスが漏れています。これは、ユーザーが自分のデータが侵害されたかどうかを問い合わせる方法だからです。電子メールアドレスの設定には、多くのユーザーがフルネームや認証情報を採用しているため、電子メールアドレスの流出は、すでに大きなプライバシーリスクとなっています。さらに、ユーザーの性別、銀行口座番号、完全な物理的住所など、他の多くの非常に機密性の高い情報が流出するケースもあります。

ウェブサイトが機密情報を保護する方法

ウェブサイトを閲覧していると、閲覧したページ、フォームに入力した機密データ、位置情報など、非公開にしたいデータがあります。セキュリティの章では、91.1%のモバイルサイトがHTTPSを有効にし、インターネットを通過するデータを盗聴から保護していることをご紹介しています。ここでは、機密性の高いリソースのプライバシーを確保するために、ウェブサイトがブラウザにどのように指示できるかに焦点を当てます。

470. *https://haveibeenpwned.com/*
471. *https://haveibeenpwned.com/*

パーミッションポリシー／フィーチャーポリシー

パーミッションポリシー[472]（以前はフィーチャーポリシーと呼ばれていました）は、ウェブサイトがどのウェブ機能を使用するつもりで、どの機能がユーザーによって明示的に承認される必要があるか（たとえば、第三者から要求されたとき）を定義する方法を提供します。これにより、埋め込まれたサードパーティーのスクリプトがどのような機能へのアクセスを要求できるかを、ウェブサイトがコントロールできるようになります。たとえば、パーミッションポリシーを使用すると、ウェブサイトは、サードパーティーが自分のサイトでマイクアクセスを要求しないようにできます。このポリシーにより、開発者は `allow` 属性で指定することで、使用する予定のWeb APIを細かく選択できます。

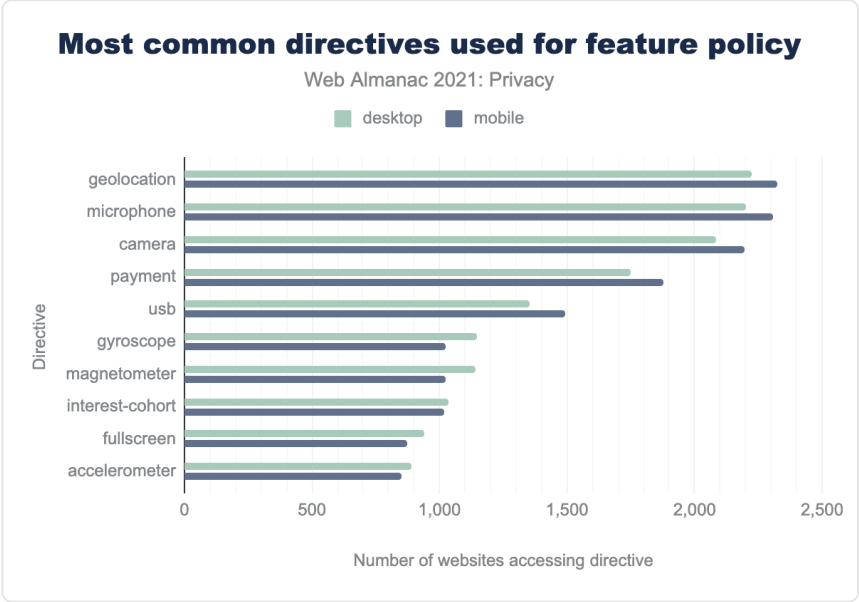


図11.16. フィーチャーポリシーディレクティブにアクセスしたウェブサイトの数。

フィーチャーポリシーに関連して、もっともよく使われるディレクティブは上記の通りです。モバイルでは3,049サイト、デスクトップでは2,901サイトで、マイク機能の利用が指定されています。これはデータセットのごく一部であり、まだニッチな技術であることを示しています。その他によく制限される機能は、ジオロケーション、カメラ、支払い機能です。

ディレクティブの使われ方をより深く理解するために、よく使われるディレクティブの上位3つと、そのディレクティブに割り当てられた値の分布について調べてみました。

472. https://www.w3.org/TR/permissions-policy-1/

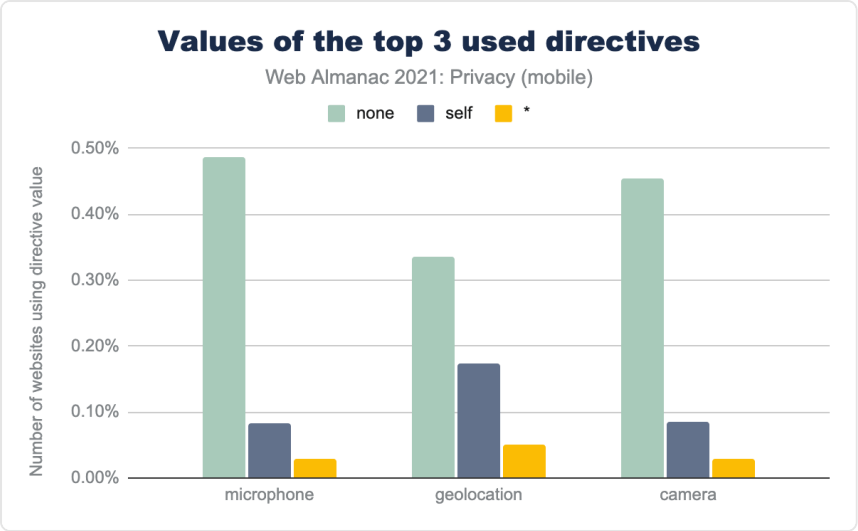


図11.17. もっとも一般的な3つの機能ポリシーディレクティブに使用される値。

`none` がもっともよく使われる値です。これは、トップレベルおよびネストされたブラウジングのコンテキストで、その機能が無効であることを指定します。2番目によく使われる値である `self` は、現在のドキュメントと同じオリジン内でその機能を許可することを指定するために使われます。一方 `*` は、オリジンをまたいだ完全なアクセスを許可します。

Refererポリシー

HTTPリクエストはオプションで `Referer` ヘッダーを含むことができます。これはリクエストがどのオリジンまたはWebページのURLから行われたかを示すものです。`Referer` ヘッダーはさまざまなタイプのリクエストに含まれる可能性があります。

- ユーザーがリンクをクリックしたときのナビゲーション要求。
- サブリソースリクエスト。ブラウザが画像、iframe、スクリプトなど、ページが必要とするリソースを要求すること。

ナビゲーションやiframeの場合、このデータはJavaScriptで `document.referrer` を使用してアクセスすることもできます。

`Referer` の値は、洞察力を高めることができます。しかし、パスとクエリ文字列を含む完全なURLが `Referer` としてオリジン間で送信される場合、プライバシーを侵害する可能性があります。URLには、個人情報、時には個人を特定するような重要な情報が含まれていることがあります。このような情報は、送信元を越えて静かに流出することで、ユーザーのプ

ライバシーを侵害し、セキュリティ上のリスクをもたらします。`Referrer-Policy` HTTPヘッダーは、開発者が自分のサイトからのリクエストで利用できるリファラーデータを制限して、このリスクを軽減することを可能にします。

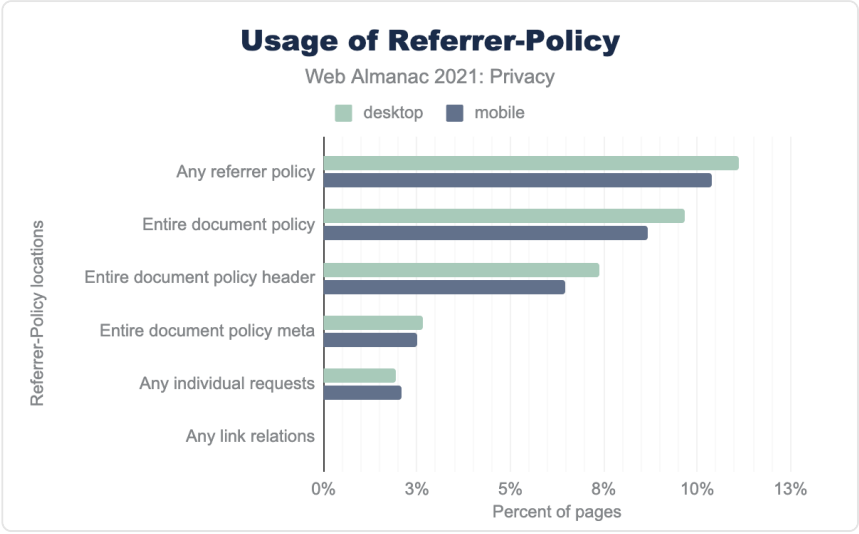


図11.18. リファラーポリシーを指定しているWebサイトの割合。

まず注目すべき点は、ほとんどのサイトがリファラーポリシーを明示的に設定していないことです。デスクトップ用ウェブサイトの11.12%、モバイル用ウェブサイトの10.38%だけが、明示的にリファラーポリシーを定義しています。残りの部分（デスクトップでは88.88%、モバイルでは89.62%）は、ブラウザのデフォルトポリシーにフォールバックされます。最近ほとんどの主要なブラウザ[473]は、2020年8月にChrome[474]、2021年3月にFirefox[475]など、`strict-origin-when-cross-origin`というデフォルトポリシーを導入しています。`strict-origin-when-cross-origin`はクロスオリジンリクエストの際にURLのパスとクエリフラグメントを削除し、セキュリティとプライバシーのリスクを軽減します。

473. *https://web.dev/i18n/ja/referrer-best-practices/#default-referrer-policies-in-browsers*
474. *https://developers.google.com/web/updates/2020/07/referrer-policy-new-chrome-default*
475. *https://blog.mozilla.org/security/2021/03/22/firefox-87-trims-http-referrers-by-default-to-protect-user-privacy/*

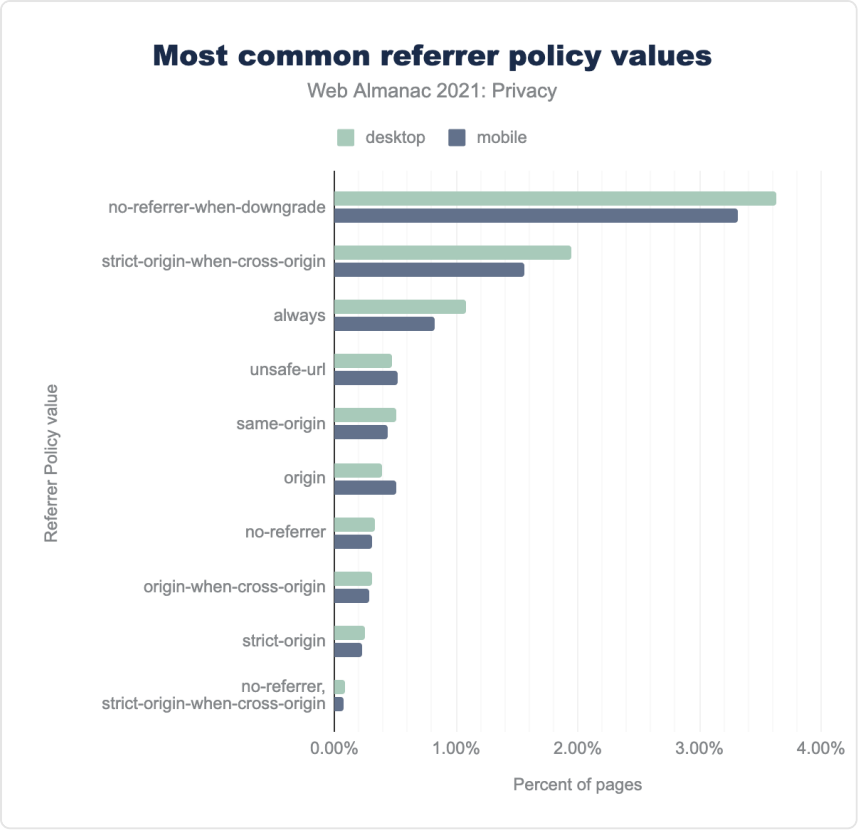


図11.19. Percentage of pages using Referrer Policy values.

明示的に設定されるもっとも一般的なReferrer Policyは、`no-referrer-when-downgrade`です。モバイルクライアントのウェブサイトの3.38%、デスクトップクライアントのウェブサイトの3.81%で設定されています。`no-referrer-when-downgrade`はプライバシーを強化するものではありません。このポリシーでは、ユーザーがあるサイトで訪れたページの完全なURLが、クロスオリジンのHTTPSリクエスト（リクエストの大部分）で共有され、この情報は他の当事者（オリジン）がアクセスできるようになります。

さらに、約0.5%のウェブサイトがリファラーポリシーの値を`unsafe-url`に設定しており、受信者のセキュリティレベルに関係なく、*あらゆるリクエストでオリジン、ホスト、クエリー文字列が送信されるようになっています。この場合、リファラーが平文で送信され、個人情報を漏えいさせる可能性があります。心配なことに、この動作を可能にするようにサイトが積極的に設定されているのです。*

注：ウェブサイトは、リファラー情報をURLのパラメータとして送信先サイトに送ることも

あります。このレポートでは、そのような仕組みの使用状況は測定していません。

User-Agent クライアントヒント

ウェブブラウザがHTTPリクエストを行う際、クライアントのブラウザ、デバイス、ネットワーク機能に関する情報を提供する `User-Agent` ヘッダーが含まれます。しかし、これを悪用してユーザーをプロファイリングしたり、フィンガープリンティングによって一意に特定することが可能です。

User-Agentクライアントヒント[476] は `User-Agent` 文字列と同じ情報へのアクセスを可能にしますが、よりプライバシーを保護する方法でアクセスします。これにより、ChromeがUser Agent削減[477] の段階的な計画で提案しているように、ブラウザが `User-Agent` 文字列をデフォルトで提供する情報量を最終的に削減することができるようになります。

サーバーは `Accept-CH` ヘッダーを指定することで、これらのClient Hintsをサポートすることを示すことができます。このヘッダーは、デバイス固有またはネットワーク固有のリソースを提供するために、サーバーがクライアントに要求する属性をリストアップしています。一般に、Client Hintsはサーバがコンテンツを効率的に提供するために必要な最小限の情報のみを取得する方法を提供します。

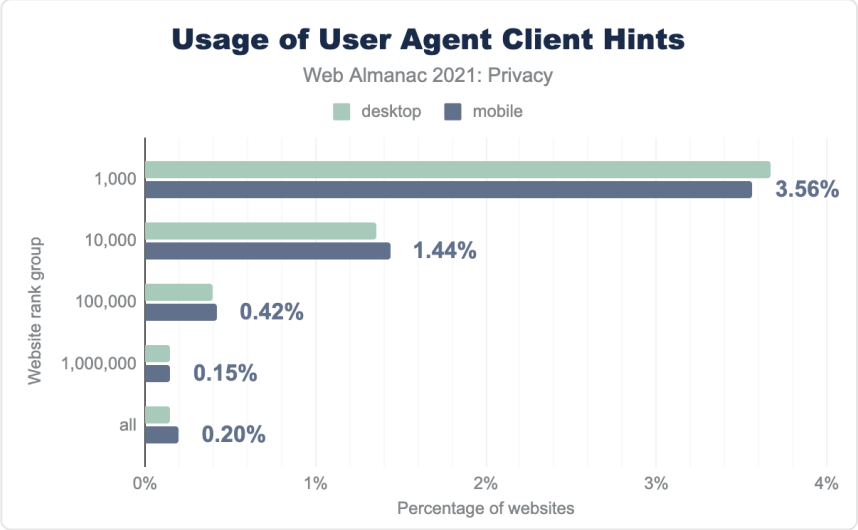


図11.20. User-Agentクライアントヒントを使用しているページの割合。

しかし、現時点では、クライアントヒントを導入しているWebサイトはほとんどありませ

476. *https://wicg.github.io/ua-client-hints/*
477. *https://www.chromium.org/updates/ua-reduction*

ん。また、人気のあるWebサイトとそうでないWebサイトでは、クライアントヒントの利用状況に大きな差があります。モバイルで人気のある上位1,000のWebサイトのうち、3.67%がClient Hintsをリクエストしています。上位10,000のウェブサイトでは、実装率は1.44%に低下しています。

ウェブサイトがプライバシーを選択できるようにする方法 プライバシー・プリファレンス・シグナル

冒頭で述べたような近年のプライバシー規制の導入に伴い、ウェブサイトは、マーケティングや分析といった本質的でない機能のための個人データの収集について、ユーザーの明確な同意を得ることが求められています。

そのため、ウェブサイトは、クッキーの同意バナーやプライバシーポリシー、その他の仕組み（経時変化[478]）を利用して、これらのサイトが処理するデータについてユーザーに知らせ、選択を与える方向に向かいました。このセクションでは、このようなツールの普及状況について見ていきます。

同意書管理プラットフォーム

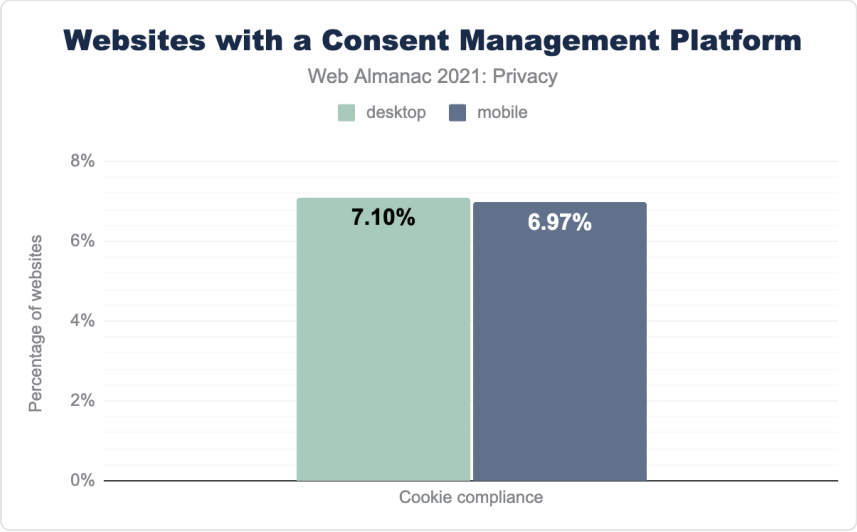


図11.21. 同意管理プラットフォームを使用しているウェブサイトの割合。

478. https://sciendo.com/article/10.2478/popets-2021-0069

同意書管理プラットフォーム（CMP）は、ウェブサイトがユーザーのためにクッキーの同意バナーを提供するために組み込むことができるサードパーティライブラリです。約7%のWebサイトがCMPを利用していることが確認されています。

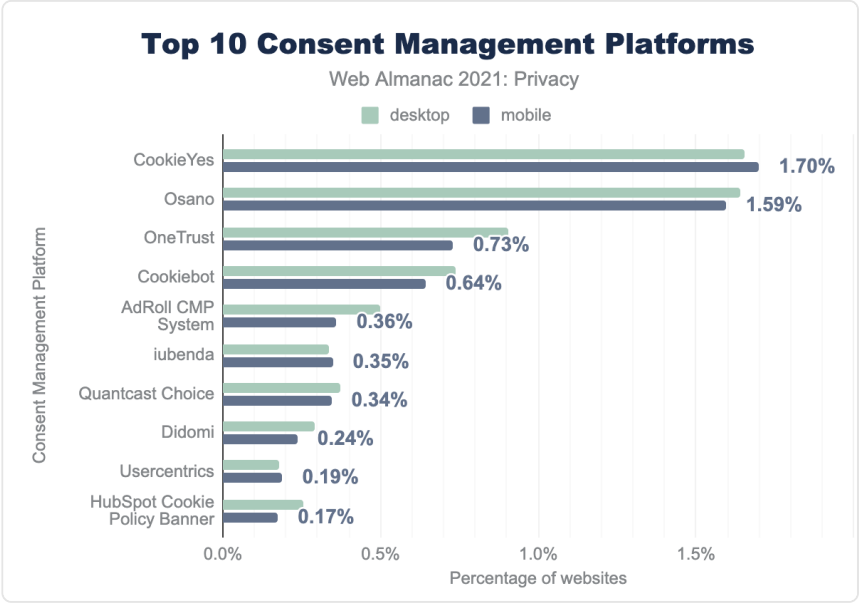


図11.22. もっとも人気のある10の同意管理プラットフォーム。

もっとも人気のあるライブラリは、CookieYes[479] と Osano[480] ですが、ウェブサイトがクッキー同意バナーを掲載できるライブラリは20種以上見つかりました。各ライブラリは、それぞれ2%未満と、わずかな割合でしか存在しない。

IABの同意フレームワーク

透明性と同意のフレームワーク[481] (TCF) は、インタラクティブ広告協会ヨーロッパ（IAB）が主導する、広告主にユーザーの同意を伝えるための業界標準を提供するためのものです。このフレームワークは、ベンダーが処理するデータの正当な目的を指定できるグローバルベンダーリスト[482]と、ベンダーとパブリッシャーの仲介をするCMPのリストで構成されています。各CMPは、法的根拠を伝達し、ユーザーから提供された同意の選択肢をブラウザに保存する責任を負っています。私たちは、保存されたクッキーをコンテントストリングと呼んでいます。

479. https://www.cookieyes.com/
480. https://www.osano.com/
481. https://iabeurope.eu/transparency-consent-framework/
482. https://iabeurope.eu/vendor-list/

TCFは、欧州ではGDPRに準拠した仕組みとして意味づけられていますが、ベルギーデータ保護局による最近の決定[483]では、この仕組みはまだ侵害されていると判断されています。カリフォルニア州でCCPAが施行されたとき、IAB Tech Lab USは同じコンセプトでU.S. プライバシー[484] (USP) 技術仕様を策定しました。

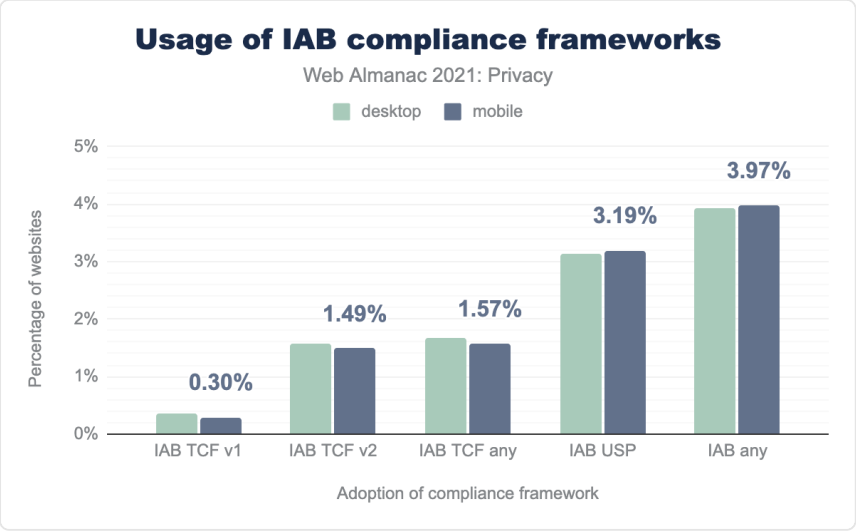


図11.23. IAB準拠のフレームワークを使用しているウェブサイトの割合。

上図は、TCFとUSPの両バージョンの使用率分布を示したものです。なお、今回のクロールは米国をベースにしているため、TCFを導入しているウェブサイトは多くないと思われる。TCFを使用しているサイトは全体の2%以下ですが、USプライバシーフレームワークを使用しているサイトはその2倍です。

483. https://iabeurope.eu/all-news/update-on-the-belgian-data-protection-authoritys-investigation-of-iab-europe/
484. https://iabtechlab.com/standards/ccpa/

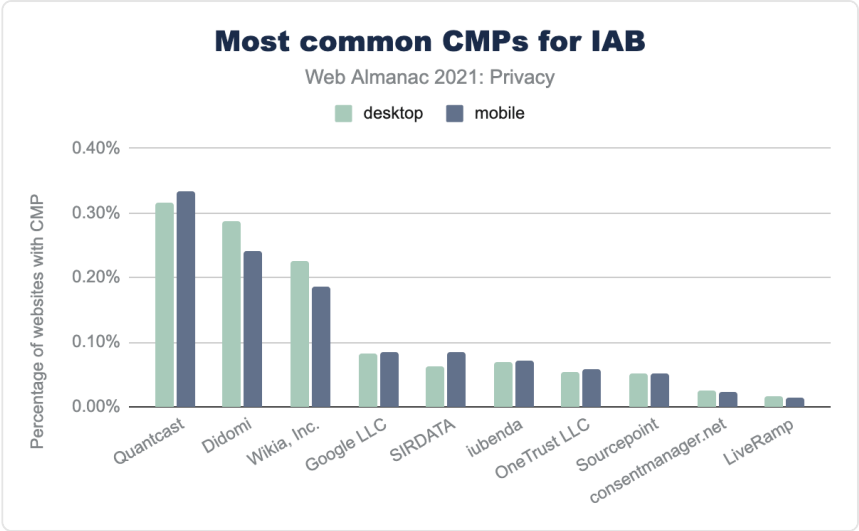


図11.24. IABでもっとも人気のある10の同意管理プラットフォーム。

フレームワークに含まれる同意管理プラットフォームでもっとも人気のある10個の中で、トップはQuantcast[485]でモバイルでは0.34%となっています。その他、Didomi[486]が0.24%、Wikiaが0.30%と、人気の高いソリューションとなっています。

USPフレームワークでは、ウェブサイトとユーザーのプライバシー設定は、*プライバシー文字列*でエンコードされています。

485. *https://www.quantcast.com/products/choice-consent-management-platform/*
486. *https://www.didomi.io/*

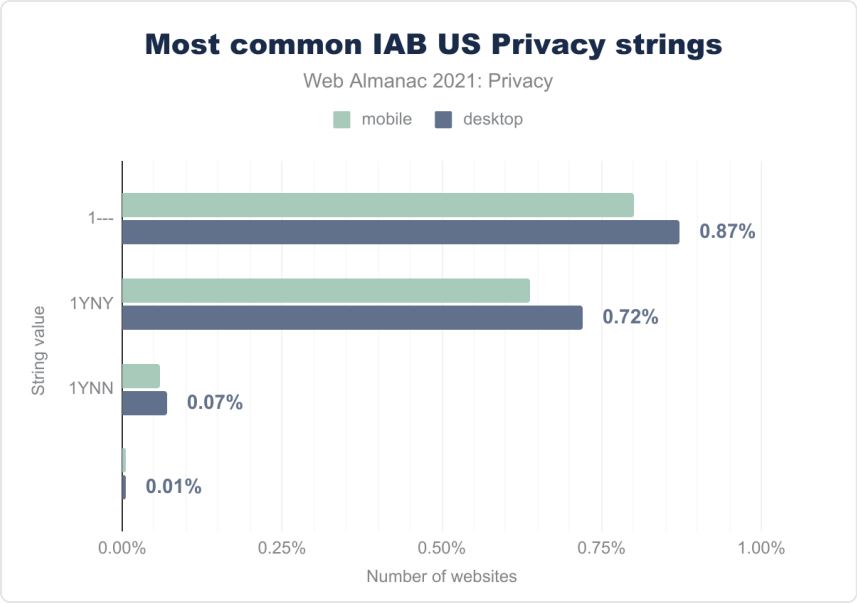


図11.25. IAB USプライバシー文字列を使用しているウェブサイトの割合。

もっとも一般的なプライバシーに関する文字列は、`1----`です。これは、CCPAがウェブサイトに適用されないことを示し、したがってウェブサイトはユーザーに対してオプトアウトを提供する義務がないことを示します。CCPAは、個人情報の販売を主業務とする企業、またはデータ処理を行う企業で年間売上高が2500万ドル以上の企業にのみ適用されます。2番目に多い文字列は、`1YNY`です。これは、ウェブサイトが「データの販売をオプトアウトするための通知と機会」を提供したが、ユーザーが個人データの販売をオプトアウトしていないことを示すものです。

プライバシーポリシー

現在、ほとんどのウェブサイトにはプライバシーポリシーがあり、ユーザーは自分について保存、処理される情報の種類を知ることができます。

39.70%

図11.26. プライバシーポリシーのリンクがあるモバイルウェブサイトの割合。

「プライバシーポリシー」「クッキーポリシー」などのキーワードを多くの言語[487]で検索してみると、モバイルサイトの39.70%、デスクトップサイトの43.02%が何らかのプライバシーポリシーに言及していることが分かります。一部のウェブサイトでは、このようなポリシーの策定が義務付けられていませんが、多くのウェブサイトでは個人情報を取り扱っているため、ユーザーに対して十分な透明性を確保するために、プライバシーポリシーを策定することが必要です。

追跡禁止 – グローバルなプライバシー管理

追跡禁止[488] (DNT) HTTPヘッダーは、ユーザーが追跡を希望しないことをウェブサイトへ伝えるために使用できます。以下に、DNTの現在値にアクセスしていると思われるサイトの数を、`Navigator.doNotTrack` JavaScriptコールの存在に基づいて確認できます。

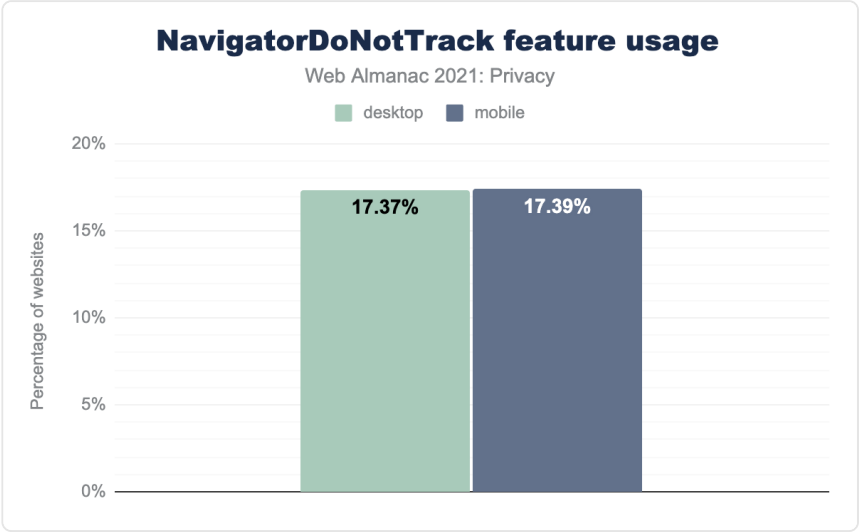


図11.27. 追跡禁止 (DNT)を使用しているWebサイトの割合。

モバイルクライアントとデスクトップクライアントでは、ほぼ同じ割合のページでDNTが使用されています。しかし、実際には、DNTのオプトアウトを尊重するウェブサイトはほとんどありません。DNTを規定するTracking Protection Working Groupは、2018年に閉鎖[489]された。「サポート不足」[490]のためです。Safariはその後、DNTのサポートを止め[491]、フィンガープリント への悪用の可能性を防止するようにしました。

487. *https://github.com/RUB-SysSec/we-value-your-privacy/blob/master/privacy_wording.json*
488. *https://www.eff.org/issues/do-not-track*
489. *https://www.w3.org/2016/11/tracking-protection-wg.html*
490. *https://lists.w3.org/Archives/Public/public-tracking/2018Oct/0000.html*
491. *https://developer.apple.com/documentation/safari-release-notes/safari-12_1-release-notes#:~:text=Removed%20support%20for%20the%20expired%20Do%20Not%20Track*

DNTの後継となるグローバル・プライバシー・コントロール[492]（GPC）は2020年10月にリリースされ、より強制力のある代替手段を提供するものであり、より良い普及を期待するものです。このプライバシー設定シグナルは、すべてのHTTPリクエストに1ビットで実装されています。まだ取り込みは確認できていませんが、主要なブラウザがGPCを実装し始めている[493]ため、今後改善されることが期待できます。

ブラウザはどのようにプライバシーへのアプローチを進化させているのか

ウェブ閲覧中のユーザーのプライバシー保護を強化するため、主要ブラウザはユーザーの機密情報をより安全に保護する新機能を実装しています。`Referrer-Policy` ヘッダーや `SameSite` クッキー に対して、ブラウザがよりプライバシーを保護するデフォルト設定を強制し始めたことは、すでに説明しました。

さらに、Firefoxはトラッキング防止機能の強化[494]、Safariはインテリジェントなトラッキング防止[495]によりトラッキングをブロックしようとしています。

トラッカーのブロックにとどまらず、Chromeはプライバシー・サンドボックス[496] を立ち上げ、広告や詐欺防止などさまざまなユースケースでよりプライバシーに配慮した機能を提供する新しいウェブ標準を開発中です。サイトがユーザーを追跡する機会を減らすために設計された、これらの新進気鋭の技術について詳しく見ていきます。

プライバシー・サンドボックス

エコシステムのフィードバックを求めるため、プライバシー サンドボックスAPIの初期および実験バージョンは、最初は個々の開発者によるテストのために 機能フラグ[497] の背後に用意され、その後 *オリジントライアル* を通じてChromeで使用されるようになります。このオリジントライアルに参加することで、Webプラットフォームの実験的な機能をテストし、その機能の使い勝手、実用性、有効性について、すべてのWebサイトにデフォルトで提供される前にWeb標準化コミュニティへフィードバックできます。

***免責事項：** Originのトライアルは限られた時間のみ利用可能です。以下の数字は、この記事を書いている2021年10月時点のプライバシーサンドボックスのオリジントライアルの状態を表しています。*

492. *https://globalprivacycontrol.org/*
493. *https://www.washingtonpost.com/technology/2021/10/26/global-privacy-control-firefox/*
494. *https://developer.mozilla.org/docs/Web/Privacy/Tracking_Protection*
495. *https://webkit.org/tracking-prevention/*
496. *https://privacysandbox.com/*
497. *https://www.chromium.org/developers/how-tos/run-chromium-with-flags*

FLoC

プライバシー・サンドボックスの実験でもっとも話題になったものの1つが、*コホートのフェデレート学習*、略して*FLoC*です。FLoCのオリジントライアルは、2021年7月に終了しました。

ウェブ上では、興味関心に基づく広告選択が一般的に行われています。FLoCは、その特定のユースケースを満たすために、個々のユーザーを識別し追跡する必要のないAPIを提供しました。FLoCは、いくつかのflak[498]を取りました。Firefox[499] や Chromiumベースの他のブラウザ[500]は実装を拒否し、電子フロンティア財団は 新しいプライバシーリスクをもたらすかもしれないという懸念[501]を唱えている。しかし、FLoCは最初の実験でした。今後のAPIの改良で、これらの懸念が解消され、より広く採用される可能性があります。

FLoCでは、ユーザーに固有の識別子を割り当てる代わりに、ブラウザがユーザーの*コホート*（同じようなページを訪れた、したがって同じ広告主が関心を持つ可能性のある何千もの人々のグループ）を決定したのです。

FLoCは実験的なものであったため、広く展開されることはなかった。その代わり、オリジン・トライアルに登録することで、ウェブサイトが、テストすることができました。デスクトップとモバイルでそれぞれ62と64のウェブサイトがFLoCをテストしていることがわかりました。

最初のFLoC実験の仕組みはこうです。ユーザーがウェブ上を移動すると、ブラウザはFLoCアルゴリズムを使って、最近の閲覧履歴が同じような何千ものブラウザに対して同じ*興味のあるコホート*を算出します。ブラウザは、個々の閲覧データをブラウザベンダーや他の関係者と共有することなく、ユーザーのデバイス上で定期的にコホートを再計算しました。コホートをワークアウトする際、ブラウザは センシティブなカテゴリーを明らかにしなかった[502]というコホートの間で選択を行っていました。

個々のユーザーやウェブサイトは、コホート計算の対象から外れることも可能です。

498. *https://www.economist.com/the-economist-explains/2021/05/17/why-is-floc-googles-new-ad-technology-taking-flak*
499. *https://blog.mozilla.org/en/privacy-security/privacy-analysis-of-floc/*
500. *https://www.theverge.com/2021/4/16/22387492/google-floc-ad-tech-privacy-browsers-brave-vivaldi-edge-mozilla-chrome-safari*
501. *https://www.eff.org/deeplinks/2021/03/googles-floc-terrible-idea*
502. *https://www.chromium.org/Home/chromium-privacy/privacy-sandbox/floc#:~:text=web%20pages%20on%20sensitive%20topics*

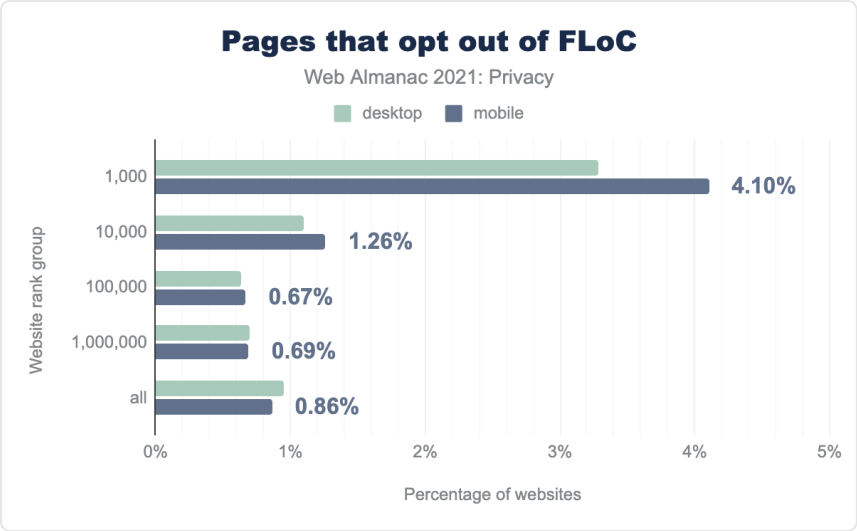


図11.28. FLoCコホートでオプトアウトするウェブサイトの割合。

上位1,000サイトのうち、4.10%がFLoCをオプトアウトしていることがわかりました。全ウェブサイトのオプトアウト率は1%未満です。

その他のプライバシー・サンドボックスの実験

GoogleのPrivacy Sandbox構想の中では、さまざまな実験が行われています。

アトリビューションレポートAPI（旧称：*換算測定*）は、広告とユーザーのインタラクションがいつコンバージョンに結びついたか、たとえば広告クリックが最終的に購買につながったかなどを測定できるようにするものです。最初のオリジントライアル（2021年10月に終了）が10オリジンで有効になっているのを確認しました。

FLEDGE（第1回「グループ上の局所実行型判定」実験）は、広告ターゲティングに対応することを目指したものです。このAPIは、現在のバージョンのChromeローカルで個々の開発者で[503]試すことができますが、2021年10月現在、オリジントライアルは行われていません。

*トラストトークン*は、ウェブサイトがある閲覧状況から別の閲覧状況へと限られた量の情報を伝達し、パッシブ追跡なしで詐欺へ対抗できるようにするものです。私たちは、サードパーティのプロバイダーとして多くのサイトへ組み込まれていると思われる7つのオリジンで、最初の オリジン トライアル[504]（2022年5月に終了予定）が有効になっていることを確認しました。

503. *https://developer.chrome.com/ja/docs/privacy-sandbox/fledge/*
504. *https://developer.chrome.com/blog/third-party-origin-trials*

CHIPS (Cookies Having Independent Partitioned State) では、ウェブサイトがクロスサイトのクッキーを「パーティショニド」としてマークし、トップレベルのサイトごとに別のクッキー入れへ入れられるようにします。（Firefoxでは、Cookieのパーティショニングについて、同様の *トータル・クッキー・プロテクト* 機能がすでに導入されています）。2021年10月現在、CHIPSのオリジントライアルはありません。

フェンスフレーム は、埋め込みページからのデータへのフレームアクセスを保護します。2021年10月現在、オリジントライアルはありません。

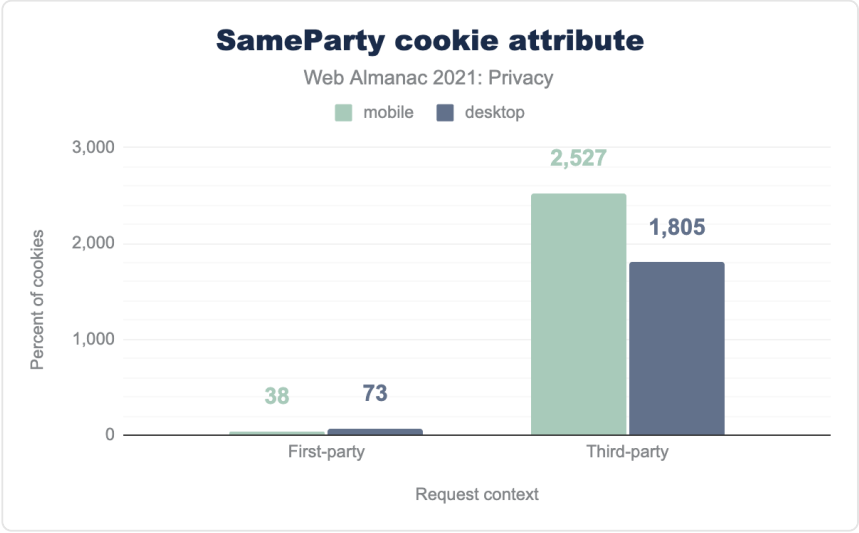


図11.29. SamePartyクッキー属性を持つクッキーの割合。

最後に、*First-Party Sets* により、ウェブサイトの所有者は、実際には同じエンティティに属している個別のドメインのセットを定義できます。所有者は、サイトが同じファーストパーティセットである限り、クロスサイトコンテキストで送信されるべきクッキーに `SameParty` 属性を設定できます。最初のオリジン・トライアルは2021年9月に終了しました。私たちは数千のクッキーに `SameParty` 属性があることを確認しました。

結論

現在もWeb上ではユーザーのプライバシーが危険にさらされています。全Webサイトの80%以上が何らかのトラッキングを有効にしており、CNAMEトラッキングのような新しいトラッキングメカニズムも開発されています。また一部のサイトでは、ジオロケーションなどの機密データを扱っており、注意を怠ると、潜在的な侵害によってユーザーの個人情報を流出させる可能性があります。

幸いなことに、ウェブ上でのプライバシーの必要性についての認識が高まり、具体的な行動につながっています。現在、ウェブサイトは、機密性の高いリソースへのアクセスを保護するための機能を利用できるようになっています。世界中の法律が、個人データの共有について、ユーザーの明示的な同意を義務付けています。ウェブサイトは、プライバシーポリシーやクッキーのバナーを実装し、これに準拠しています。最後にブラウザは、広告や不正行為の検出などのユースケースをよりプライバシーに配慮した方法でサポートし続けるために、革新的な技術を提案し、開発しています。

最終的には、ユーザーは自分の個人データがどのように扱われるかについて発言する権限を与えられるべきです。一方、ブラウザやウェブサイトの所有者は、ユーザーのプライバシーが保護されていることを保証する技術的な手段を開発し、配備する必要があります。ウェブとのインタラクション全体にプライバシーを組み込むことで、ユーザーは自分の個人データが十分に保護されていることをより確信できます。

著者



Yana Dimova

ydimova

Yana Dimovaはimec-DistriNetの博士課程に在籍し、ウェブプライバシーについて研究しています。オンライントラッキング、プライバシーの脆弱性、プライバシーの法律と政策に関心を持ち、研究している。



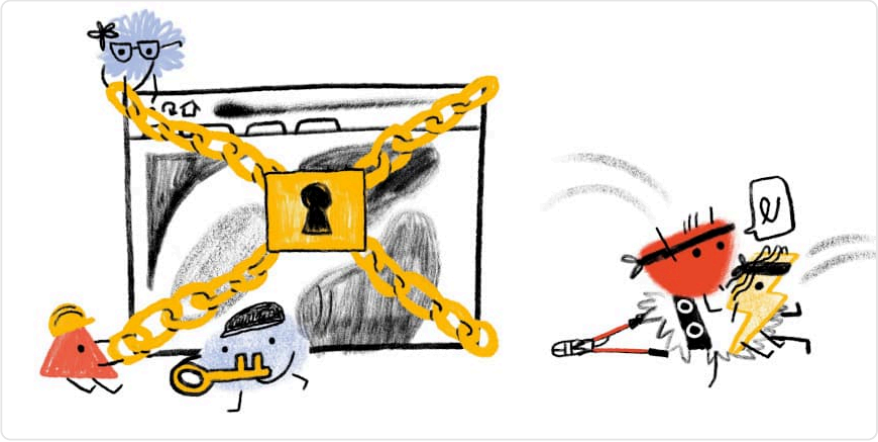
Victor Le Pochat

@VictorLePochat VictorLeP victor-le-pochat https://lepoch.at

Victor Le Pochatは、ベルギーのKU LeuvenのimecDistriNet[505]研究グループの博士課程研究者である。彼の興味は、ウェブのエコシステムの探求と、ウェブのセキュリティとプライバシーの研究方法論にあり、現在の方法の分析と改良の両方に取り組んでいます。

505. *https://distrinet.cs.kuleuven.be/*

部 II 章 12
セキュリティ



Saptak Sengupta、 Tom Van Goethem と Nurullah Demir によって書かれた。
Caleb Queern、 Edmond W. W. Chan と Matteo Große-Kampmann によってレビュ。
Gertjan Franken による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

現代はどんどんデジタル化が進んでいます。ビジネスだけでなく、私生活もデジタル化されています。私たちはオンラインで人と連絡を取り、メッセージを送り、友人と瞬間を共有し、ビジネスを行い日常生活を整理しています。同時に、この変化は、より多くの重要なデータがデジタル化され、私的、商業的にも処理されるようになったことを意味します。このような状況において、ユーザーデータの可用性、完全性、機密性を提供することでユーザーを保護することを目的とするサイバーセキュリティの重要性も増してきています。今日の技術に目を向けると、デジタルで提供されるソリューションを提供するために、ウェブリソースがますます利用されていることがわかります。それはまた、その普及により、私たちの現代生活とWebアプリケーションのセキュリティとの間に強い結びつきがあることを意味します。

この章では、Web上のセキュリティの現状を分析し、Webコミュニティが自分たちの環境を守るために使っている（そして見逃している）手法の概要を説明します。具体的には、この

レポートでは、一般的な実装、プロトコルのバージョン、暗号スイートなどの トランスポートレイヤーセキュリティ (HTTPS) に関するさまざまな指標を分析しています。また、クッキーを保護するための技術についても概観しています。そして、コンテンツ・インクルージョンの話題と攻撃を阻止するための方法（例：特定のセキュリティヘッダーの使用）についての包括的な分析が掲載されています。また、セキュリティメカニズムがどのように採用されているか（国別や特定の技術別など）についても見ています。また、Cryptojackingのようなウェブ上の不正行為について議論し、最後に `security.txt` URL の使用について見ています。

分析対象ページをデスクトップとモバイルの両方でクロールしていますが、多くのデータで同様の結果が得られているため、本章で紹介する統計情報は、とくに断りのない限り、モバイルページのセットを参照しています。データの収集方法の詳細については、方法論 ページを参照してください。

トランスポートセキュリティ

最近の傾向として、今年もHTTPSを採用するWebサイトの数が継続的に増加していると思われます。トランスポート・レイヤー・セキュリティは、Webサイトの安全な閲覧を可能にするため重要であり、利用者に提供されるリソースやWebサイトに送信されるデータが転送中に改ざんされないことを保証するものです。現在、主要なブラウザのほとんどにHTTPS専用の設定があり、ウェブサイトがHTTPSではなくHTTPを使用している場合は、ユーザーに警告が表示されるようになっているため、より広範な導入が進んでいます。



図12.1. モバイルでHTTPSを使用するリクエストの割合です。

現在、デスクトップではウェブサイトへの総リクエストの91.9%、モバイルでは91.1%がHTTPSで提供されていることがわかります。Let's Encryptのような非営利の認証局のおかげで、増え続ける証明書[506]が日々発行されているのを目にします。

506. *https://letsencrypt.org/stats/#daily-issuance*

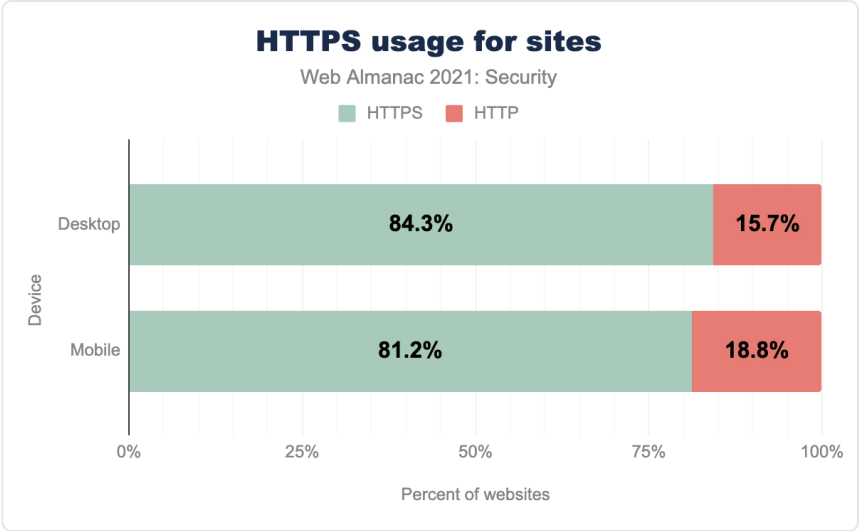


図12.2. サイトのHTTPS利用状況。

現在、デスクトップで84.3%、モバイルでは81.2%のWebサイトのホームページがHTTPSで提供されているため、HTTPSを利用しているWebサイトとHTTPSを利用しているリクエストの間にはまだギャップがあると考えられます。これは、HTTPSリクエストの印象的な割合の多くは、フォント、分析、CDNなどのサードパーティサービスによって占められており、最初のウェブページ自体ではないことが多いからです。

HTTPSを使用しているサイトは継続的に改善しています（昨年[507]から約7-8% の増加）。しかし、ブラウザがデフォルトでHTTPS専用モード[508]を採用し始めると、まもなく多くのメンテナンスされていないウェブサイトが警告を表示し始めるかもしれません。

プロトコルのバージョン

トランスポートレイヤーセキュリティ（TLS）とは、HTTPリクエストを安全かつプライベートにするためのプロトコルです。時代とともに、TLSには新たな脆弱性が発見され、修正されています。したがって、ウェブサイトをHTTPSで提供するだけでなく、そのような脆弱性を回避するために最新のTLS設定が使用されていることを確認することが重要です。

その一環として、最新バージョンの採用によるセキュリティと信頼性の向上を目指し、2021年3月25日をもってTLS 1.0および1.1がインターネット技術タスクフォース（IETF）[509]により非推奨とされました。また、すべてのアップストリームブラウザは、TLS 1.0および

507. https://almanac.httparchive.org/ja/2020/security#fig-3
508. https://blog.mozilla.org/security/2021/08/10/firefox-91-introduces-https-by-default-in-private-browsing/
509. https://datatracker.ietf.org/doc/rfc8996/

1.1のサポートを完全に削除するか、非推奨としています。たとえば、FirefoxはTLS1.0と1.1を非推奨としていますが、完全に削除していません[510]。これは、パンデミックの間、ユーザーが、しばしばまだTLS1.0で動いている政府のウェブサイトにアクセスする可能性があるからです。ユーザーは、ブラウザの設定で `security.tls.version.min` を変更し、ブラウザが許可するもっとも低いTLSバージョンを決定することもできます。

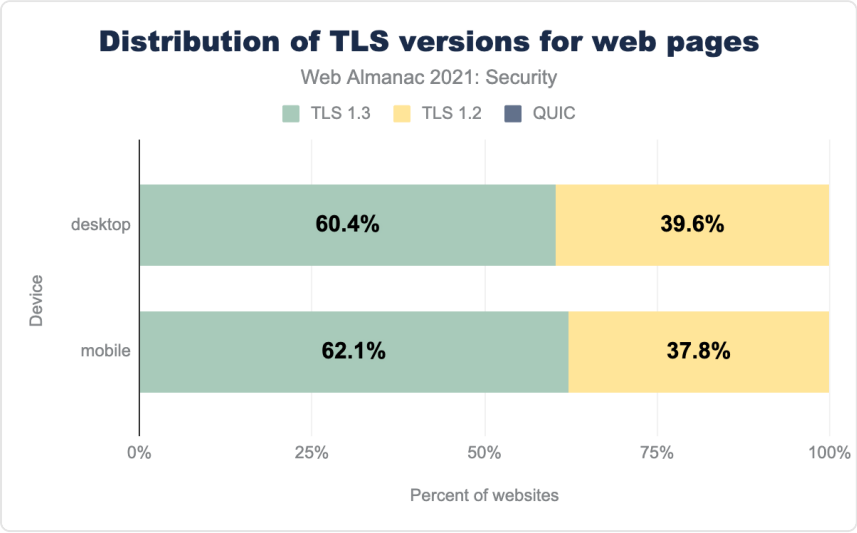


図12.3. サイトでのTLSバージョン使用状況。

デスクトップで60.4%、モバイルでは62.1%のページがTLSv1.3を使用しており、TLSv1.2を超えるプロトコルバージョンが主流となっています。TLSv1.3を使用しているページ数は、それぞれ43.2%、45.4%であった昨年[511]から約20%増加しました。

暗号スイート

暗号スイートは、TLSで使用されるアルゴリズムのセットで、安全な接続を行うために使用されます。現代の ガロアカウンターモード[512] (GCM) 暗号モードは、古い Cipher Block Chaining Mode[513] (CBC) と比較して、はるかに安全であると考えられています。暗号は、パディング攻撃[514]に対して脆弱であることが示されています。TLSv1.2は新しい暗号スイートと古い暗号スイートの両方をサポートしていましたが、TLSv1.3は古い暗号スイートを一切サポートしていません[515]。これが、TLSv1.3がより安全な接続オプションである理由の1つです。

510. *https://www.ghacks.net/2020/03/21/mozilla-re-enables-tls-1-0-and-1-1-because-of-coronavirus-and-google/*
511. *https://almanac.httparchive.org/ja/2020/security#protocol-versions*
512. *https://ja.wikipedia.org/wiki/Galois/Counter_Mode*
513. *https://ja.wikipedia.org/wiki/%E6%9A%97%E5%8F%B7%E5%88%A9%E7%94%A8%E3%83%A2%E3%83%BC%E3%83%89#Cipher_Block_Chaining_(CBC)*
514. *https://blog.qualys.com/product-tech/2019/04/22/zombie-poodle-and-goldendoodle-vulnerabilities*
515. *https://datatracker.ietf.org/doc/html/rfc8446#page-133*

96.8%

図12.4. フォワードセキュリティーを利用したモバイルサイト。

最近の暗号スイートはほとんどすべてフォワードセキュリティー鍵交換をサポートしています。つまり、サーバーの鍵が漏洩した場合、その鍵を使った古いトラフィックは復号化できないのです。デスクトップで96.6%、モバイルでは96.8%が前方秘匿を使用しています。TLSv1.2ではオプションだったフォワードセキュリティーが、TLSv1.3では必須となったことも、より安全である理由の1つです。

暗号モードとは別に考慮すべきは、認証された暗号化および認証された復号化[516] アルゴリズムの鍵のサイズです。鍵のサイズが大きいと、暗号化するのに時間がかかり、接続の暗号化と復号化のための集中的な計算が、サイトのパフォーマンスにほとんど影響を与えません。

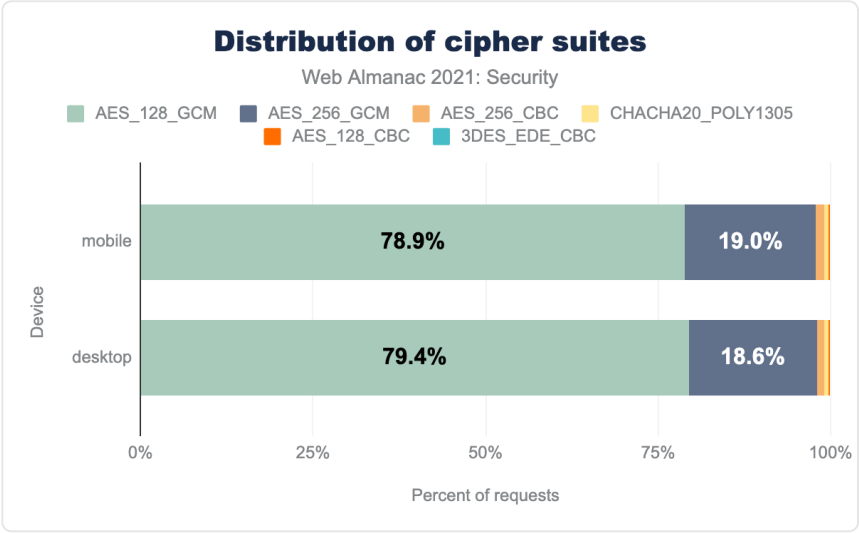


図12.5. 暗号スイートの配布。

`AES_128_GCM`は、デスクトップで79.4%、モバイルで78.9%と、依然としてもっとも広く利用されている暗号スイートです。`AES_128_GCM`は、暗号化と復号化にキーサイズ128ビットの*Advanced Encryption Standard* (AES) とGCM暗号モードを使用することを表しています。128ビットの鍵はまだ安全だと考えられていますが、ブルートフォース攻撃により長く耐えられるように、256ビットの鍵が徐々に業界標準になりつつあります。

516. https://datatracker.ietf.org/doc/html/rfc5116#section-2

認証局

*認証局*とは、デジタル証明書を発行する企業や組織のことで、WebサイトなどWeb上のエンティティの所有権や身元を検証するのに役立っています。認証局は、ブラウザが認識するTLS証明書を発行し、ウェブサイトがHTTPSで提供できるようにするため必要です。昨年と同様に、サードパーティのサービスやリソースではなく、Webサイト自身が使用する認証局について再び調べます。

発行者	*アルゴリズム*	*デスクトップ*	*モバイル*
R3[517]	*RSA*	*46.9%*	*49.2%*
Cloudflare Inc ECC CA-3	*ECDSA*	*11.7%*	*11.5%*
Sectigo RSA Domain Validation Secure Server CA [518]	*RSA*	*8.3%*	*8.2%*
cPanel, Inc. Certification Authority	*RSA*	*5.0%*	*5.5%*
Go Daddy Secure Certificate Authority - G2[519]	*RSA*	*3.6%*	*3.0%*
Amazon[520]	*RSA*	*3.4%*	*3.0%*
Encryption Everywhere DV TLS CA - G1[521]	*RSA*	*1.3%*	*1.6%*
AlphaSSL CA - SHA256 - G2[522]	*RSA*	*1.2%*	*1.2%*
RapidSSL TLS DV RSA Mixed SHA256 2020 CA-1 [523]	*RSA*	*1.2%*	*1.1%*
DigiCert SHA2 Secure Server CA[524]	*RSA*	*1.1%*	*0.9%*

図12.6. Webサイト向け証明書発行会社トップ10

Let's Encryptは、新しい証明書のバイト数を節約するために、その主題のコモンネーム[525]を"Let's Encrypt Authority X3" から単に "R3" に変更しました。つまり、R3が署名したSSL証明書は、Let's Encrypt[526]が発行していることになるのです。このように、Let's Encryptが発行する証明書を使用しているデスクトップWebサイトは46.9%、モバイルサイトでは49.2%と、例年通りLet's Encryptがチャートをリードしていることがわかります。これは昨年より2〜3%増加している。Let's Encryptの無料証明書自動生成機能は、誰もが簡単にHTTPSでウェブサイトを提供できるようにする上で、画期的な役割を果たしています。

517. https://letsencrypt.org/certificates/
518. https://sectigo.com/knowledge-base/detail/Sectigo-Intermediate-Certificates/kA01N000000rfBO
519. https://certs.godaddy.com/repository
520. https://www.amazontrust.com/repository/
521. https://www.digicert.com/kb/digicert-root-certificates.htm
522. https://support.globalsign.com/ca-certificates/intermediate-certificates/alphassl-intermediate-certificates
523. https://www.digicert.com/kb/digicert-root-certificates.htm
524. https://www.digicert.com/kb/digicert-root-certificates.htm
525. https://letsencrypt.org/2020/09/17/new-root-and-intermediates.html#why-we-issued-an-ecdsa-root-and-intermediates
526. https://letsencrypt.org/certificates/

Cloudflareは、同様に顧客向けに無料の証明書を提供しており、引き続き2位を維持しています。また、Cloudflare CDNは、*Elliptic Curve Cryptography*（ECC）証明書の利用を増やしています。RSA証明書よりも小型で効率的ですが、古いクライアントに対して非ECC証明書も提供し続ける必要があり、しばしば導入が困難となります。CloudflareのようなCDNを利用することで、その複雑さを解消できます。最新のブラウザ[527]はすべてECC証明書に対応していますが、Chromeなど一部のブラウザはOSに依存しています。そのため、Windows XPなどの古いOSでChromeを使う人は、ECC以外の証明書にフォールバックする必要があるのです。

HTTPストリクトトランスポートセキュリティ

HTTPストリクトトランスポートセキュリティ (HSTS)は、ウェブサイトとの通信に安全なHTTPS接続を*常に*使用するようブラウザに指示する応答ヘッダーです。



図12.7. モバイルでHSTSヘッダーを持つリクエストの割合です。

`Strict-Transport-Security` ヘッダーは、そのサイトへのリクエストが行われる前に、`http://` URLを `https://` URLへ変換するのに役立つものです。モバイル用レスポンスの22.2%、デスクトップ用レスポンスの23.9%にHSTSヘッダーがあります。

***HSTS*ディレクティブ**	***デスクトップ***	***モバイル***
Valid `max-age`	*92.7%*	*93.4%*
`includeSubdomains`	*34.5%*	*33.3%*
`preload`	*17.6%*	*18.0%*

図12.8. HSTSディレクティブの使用方法。

HSTSヘッダーを持つサイトのうち、デスクトップで92.7%、モバイルでは93.4%が有効な `max-age`（つまり、値がゼロでなく、空でないこと）を持ち、ブラウザは何秒間だけHTTPSでウェブサイトにアクセスすべきか判断しています。

モバイルで33.3%、デスクトップでは34.5%のリクエストレスポンスがHSTS設定に `includeSubdomain` を含んでいます。 `preload` ディレクティブは HSTS仕様の一部ではな

527. *https://developers.cloudflare.com/ssl/ssl-tls/browser-compatibility*

い[528]ため、レスポンス数が少なくなっています。また、最低でも `max-age` が31,536,000秒（または1年）で、さらに `includeSubdomain` ディレクティブも必要なため、このディレクティブは存在します。

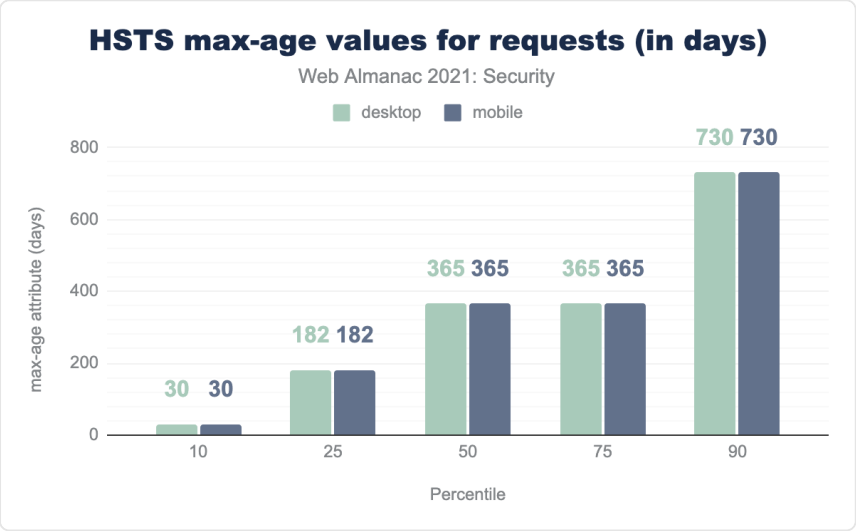


図12.9. すべてのリクエストに対するHSTS `max-age` の値（日数単位）。

HSTSヘッダーの `max-age` 属性の中央値は、モバイルとデスクトップの両方で365日であることがわかりました。https://hstspreload.org/ では、HSTSヘッダーが適切に設定され問題が、発生しないことが確認された場合、 `max-age` を2年間とすることを推奨しています。

クッキー

HTTPクッキーとは、サーバーがウェブブラウザに送信する、ウェブサイトにアクセスするユーザーに関する小さな情報のことです。ブラウザはこの情報を保存し、その後のサーバーへのリクエストで送り返します。クッキーは、セッション管理に役立ち、ユーザーが現在ログインしているかどうかなど、ユーザーの状態情報を維持します。

Cookieを適切に保護しないと、攻撃者はセッションを乗っ取り、ユーザーになりすましてサーバーに不要な変更を送ることができます。また、*クロスサイトリクエストフォージェリ*という攻撃にもつながり、ユーザーのブラウザがユーザーに気づかれないように、クッキーを含むリクエストを不用意に送信してしまうことがあるのです。

528. *https://developer.mozilla.org/docs/Web/HTTP/Headers/Strict-Transport-Security#preloading_strict_transport_security*

他にも、*クロスサイトスクリプトインクルージョン* (XSSI) や *XS-リークス* の脆弱性クラスのさまざまなテクニックなど、クロスサイトのリクエストにクッキーを含めることに依存しているタイプの攻撃もいくつか存在します。

特定の属性や接頭辞を追加することで、クッキーが安全に送信され、意図しない相手やスクリプトによってアクセスされないようにできます。

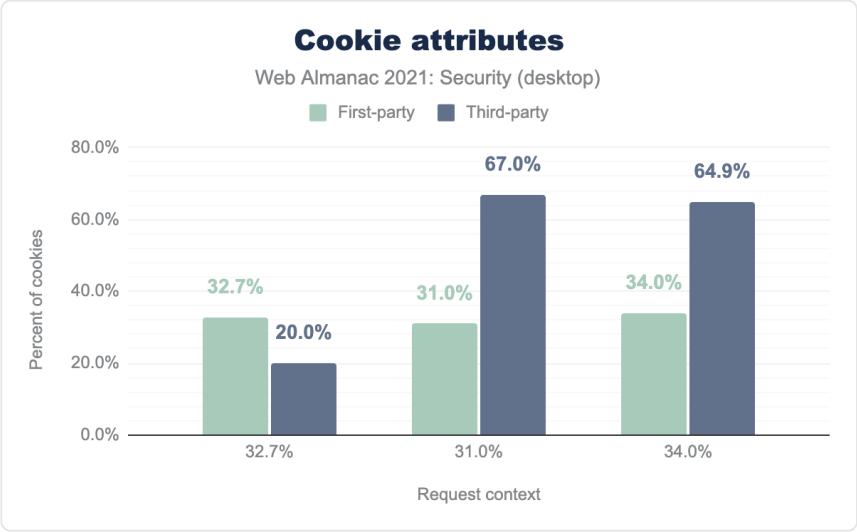


図12.10. Cookieの属性（デスクトップ）。

Secure

`Secure` 属性が設定されたクッキーは、安全なHTTPS接続でのみ送信され、*Manipulator-in-the-middle* 攻撃で盗まれないようにします。HSTSと同様に、TLSプロトコルが提供するセキュリティの強化にも貢献します。ファーストパーティのクッキーについては、デスクトップとモバイルの両方で、30%強のクッキーに `Secure` 属性が設定されています。しかし、デスクトップにおけるサードパーティ製クッキーのうち、`Secure` 属性を持つものの割合が、昨年[529]の35.2%から今年は67.0%に大きく増加していることがわかります。この増加は、後述する `SameSite=none` のクッキーに対して `Secure` 属性が必須である ことに起因すると思われます。

529. *https://almanac.httparchive.org/ja/2020/security#クッキー*

`HttpOnly`

`HttpOnly` 属性が設定されたクッキーは、JavaScriptの `document.cookie` APIを使ってアクセスすることができません。このようなクッキーはサーバにのみ送ることができ、クッキーを悪用したクライアントサイドの *クロスサイトスクリプティング*(XSS) 攻撃を緩和するのに役立ちます。サーバーサイドのセッションにのみ必要なクッキーに使用されます。`HttpOnly` 属性を持つクッキーの割合は、他のクッキー属性がそれぞれ32.7%と20.0%で使用されているのに比べ、ファーストパーティークッキーとサードパーティーの差はより小さくなっています。

`SameSite`

クッキーの `SameSite` 属性により、ウェブサイトはクロスサイトリクエストでクッキーを送信するタイミングとその有無をブラウザに通知できます。これはクロスサイトリクエストフォージェリ攻撃を防ぐために使用されます。`SameSite=Strict` はクッキーをそれが発生したサイトのみに送信することを可能にします。`SameSite=Lax` では、ユーザーがリンクをたどって元のサイトに移動していない限り、クッキーはクロスサイトリクエストに送られません。`SameSite=None` では、クッキーはオリジンサイトとクロスサイトリクエストの両方で送信されます。

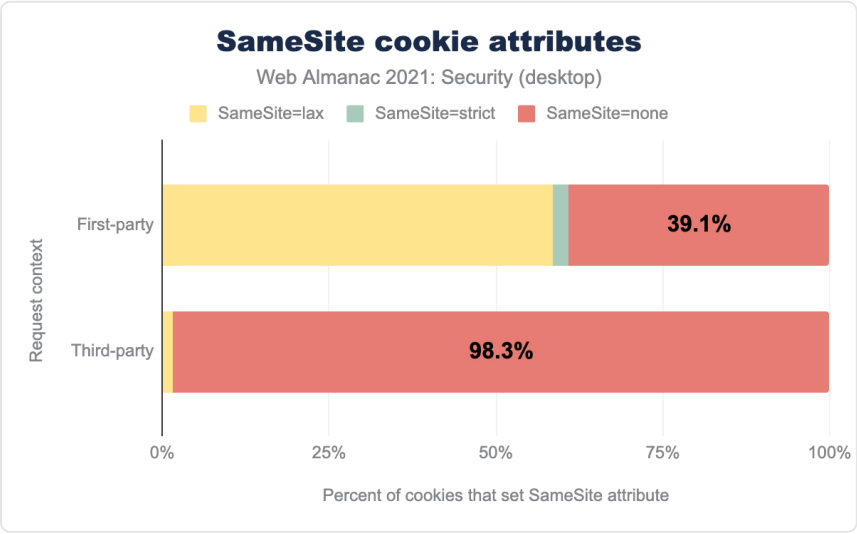


図12.11. 同一サイトのCookie属性。

`SameSite` 属性を持つファーストパーティークッキーの58.5%が `Lax` に設定されていることがわかります。一方、`SameSite` 属性が `none` に設定されているクッキーがまだ39.1%

あり、かなり大変ですが、数は着実に減少しています。現在のほぼすべてのブラウザは、`SameSite`属性が設定されていない場合、`SameSite=Lax`をデフォルトとしています。ファーストパーティークッキー全体の約65%は`SameSite`属性を持っていません。

プレフィックス

クッキープレフィックス`__Host-`と`__Secure-`は、セッションフィクスチャ攻撃[530]のためにセッションクッキーの情報を上書きする攻撃を軽減するのに役立ちます。`__Host-`はクッキーをドメインロックするのに役立ちます。クッキーは`Secure`属性と`Path`属性を`/`に設定し、`Domain`属性を持たず、安全な場所から送信される必要があります。一方、`__Secure-`はクッキーが`Secure`属性のみを持ち、安全な場所から送信されることを要求します。

クッキーの種類	*__Secure*	*__Host*
ファーストパーティ	*0.02%*	*0.01%*
サードパーティ	*< 0.01%*	*0.03%*

図12.11. モバイルでの`__Secure`と`__Host`のCookieプレフィックスの使用について。

どちらの接頭辞もクッキーのかなり低い割合で使われていますが、`__Secure-`は前提条件が低いため、ファーストパーティークッキーでより一般的に見られます。

クッキー寿命

永続的なクッキーは、`Expires`属性で指定された日付、または`Max-Age`属性で指定された期間の経過後に削除されます。`Expires`と`Max-Age`の両方が設定されている場合、`Max-Age`が優先されます。

530. https://owasp.org/www-community/attacks/Session_fixation

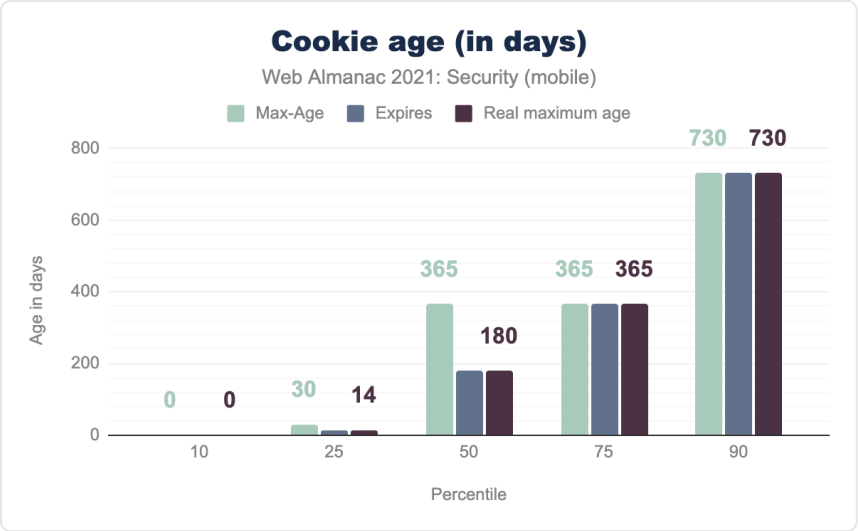


図12.12. クッキー使用日数（モバイル）。

`Max-Age` を持つクッキーの約20.5%が31,536,000という値を持っていることから、中央値の `Max-Age` は365日であることがわかります。しかし、ファーストパーティークッキーの64.2%は `Expires` を持ち、23.3%は `Max-Age` を持ちます。クッキーの間では `Expires` がずっと優勢なので、実際の最大寿命の中央値は、期待されるように `Max-Age` ではなく `Expires`（180日）と同じになっています。

コンテンツ搭載

ほとんどのWebサイトでは、多くのメディアやCSS、JavaScriptライブラリが、さまざまな外部ソース、CDN、クラウドストレージサービスから読み込まれています。どのソースのコンテンツが信頼できるかを確認することは、Webサイトのセキュリティだけでなく、Webサイトのユーザーのセキュリティにとっても重要です。そうでなければ、信頼できないコンテンツが読み込まれた場合、ウェブサイトはクロスサイトスクリプティング攻撃にさらされる可能性があります。

コンテンツセキュリティポリシー

コンテンツセキュリティポリシー(CSP) は、さまざまなコンテンツの読み込みを許可するオリジンを制限することにより、クロスサイトスクリプティングやデータインジェクション攻撃を緩和するために使用される主要な方法です。ウェブサイトには、さまざまな種類のコンテンツのソースを指定するために使用できる数多くのディレクティブがあります。たとえ

ば、`script-src` はスクリプトを読み込むことができるオリジンやドメインを指定するために使用されます。また、インラインスクリプトと `eval()` 関数が許可されているかどうかを定義するための値も持っています。

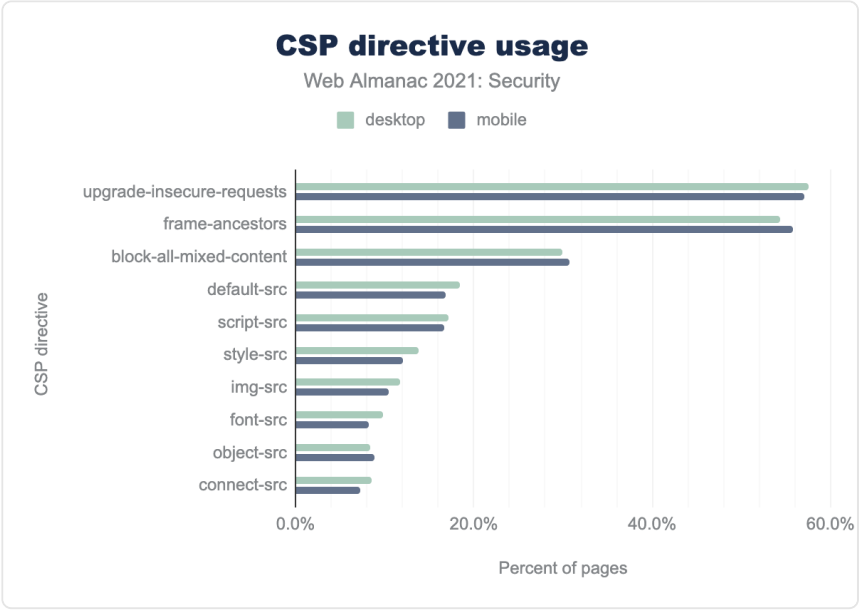


図12.13. CSPで使用されるもっとも一般的なディレクティブ。

モバイル向けホームページでは、昨年の7.2%から9.3%がCSPを使用しており、ますます多くのウェブサイトがCSPを使用し始めていることがわかります。`upgrade-insecure-requests` は、引き続きもっとも頻繁に使用されているCSPです。このポリシーの採用率が高いのは、昨年[531]と同じ理由によるものと思われます。これは簡単でリスクの少ないポリシーで、すべてのHTTPリクエストをHTTPSへアップグレードするのに役立ち、またページで使用されている混合コンテンツをブロックするのにも役に立ちます。`frame-ancestors` は、ページを埋め込むことができる有効な親を定義するのを助ける、僅差で2番目のものです。

コンテンツの読み込み元を定義するポリシーの採用は、依然として低い水準にとどまっています。これらのポリシーのほとんどは、破損を引き起こす可能性があるため、実装がより困難になっています。外部コンテンツを許可するための `nonce`、ハッシュ、ドメインなどを定義するために、実装に労力を要するのです。

厳格なCSPは攻撃に対する強力な防御となりますが、ポリシーの定義が正しくない場合、望

531. *https://almanac.httparchive.org/ja/2020/security#content-security-policy*

ましくない効果をもたらし有効なコンテンツが、ロードされなくなることがあります。異なるライブラリやAPIがさらにコンテンツを読み込むと、これはさらに難しくなります。

Lighthouse[532] は最近、CSPにそのようなディレクティブがない場合に重大度警告のフラグを立て、XSS攻撃を防ぐためにより厳しいCSPを採用するよう奨励するようになりました。CSPがXSS攻撃の阻止にどのように役立つかは、この章の攻撃の阻止 のセクションで詳しく説明します。

ウェブ開発者がCSPポリシーの正しさを評価できるように、非強化の代替案もあります。これは、`Content-Security-Policy-Report-Only` 応答ヘッダーでポリシーを定義することによって有効になります。このヘッダーの普及率はまだかなり低く、モバイルでは0.9%です。しかし、ほとんどの場合、このヘッダーはテスト段階で追加され、後に強制CSPへ置き換えられるので使用率の低さは予想外ではありません。

サイトは `report-uri` ディレクティブを使用して、CSPエラーを解析できる特定のリンクにCSP違反を報告することもできます。これらは、CSPディレクティブが追加された後に、有効なコンテンツが新しいディレクティブによって偶然にブロックされていないかどうかをチェックするのに役に立ちます。この強力なフィードバック・メカニズムの欠点は、ブラウザの拡張機能や、ウェブサイト所有者がコントロールできないその他の技術によって、CSP報告がノイズになる可能性があることです。

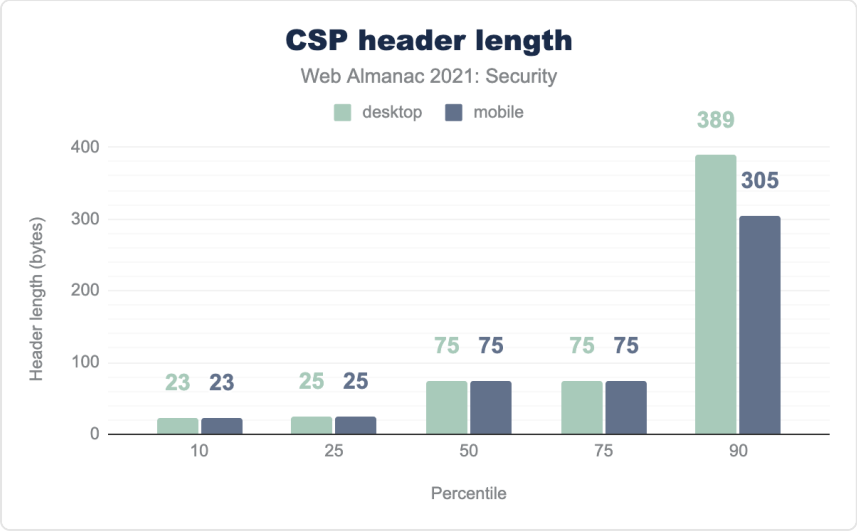


図12.14. CSPヘッダーの長さ。

CSPヘッダーの長さの中央値は75バイトと、かなり短い状態が続いています。ほとんどの

532. https://web.dev/csp-xss/

Webサイトでは、長い厳密なCSPの代わりに、特定の目的のために単一のディレクティブがまだ使用されています。たとえば、24.2%のウェブサイトは `upgrade-insecure-requests` ディレクティブのみを持っています。



図12.15. 観測された最長のCSPのバイト数。

一方、最長のCSPヘッダーは、昨年の2倍近い長さになっています。43,488バイトです。

オリジン	デスクトップ	モバイル
https://www.google-analytics.com	0.29%	0.22%
https://www.googletagmanager.com	0.26%	0.22%
https://fonts.googleapis.com	0.22%	0.16%
https://fonts.gstatic.com	0.20%	0.15%
https://www.google.com	0.19%	0.14%
https://www.youtube.com	0.19%	0.13%
https://connect.facebook.net	0.16%	0.11%
https://stats.g.doubleclick.net	0.15%	0.11%
https://www.gstatic.com	0.14%	0.11%
https://cdnjs.cloudflare.com	0.12%	0.10%

図12.16. CSP ポリシーでもっとも頻繁に許可されるホスト。

`*-src` ディレクティブでもっともよく使われるオリジンは、引き続きGoogleが大きく占めています（フォント、広告、分析）。また、今年はCloudflareの人気ライブラリCDNが10位に表示されています。

サブリソースの整合性

多くのWebサイトでは、JavaScriptライブラリやCSSライブラリを外部のCDNから読み込んでいます。これは、CDNが侵害されたり、攻撃者が頻繁に使用されるライブラリを置き換える他の方法を見つけたりした場合、特定のセキュリティ上の意味を持つことがあります。サ

ブリソースの整合性（SRI）は、そのような結果を回避するのに役立ちますが、悪意のない変更でそのリソースがないとウェブサイトが、機能しない可能性がある場合は他のリスクが発生します。可能であれば、サードパーティからロードするのではなく、セルフホスティングすることが、より安全な選択肢となります。



図12.17. 携帯電話のSRIにSHA384ハッシュ関数を使用。

ウェブ開発者は、ウェブサイトにJavaScriptやCSSのコードを含めるために使われる `<script>` と `<link>` タグに `integrity` 属性を追加できます。`integrity` 属性は、リソースの期待される内容のハッシュ値からなる。ブラウザは取得したコンテンツのハッシュと `integrity` 属性に記述されたハッシュを比較してその妥当性を確認し、一致した場合にのみリソースをレンダリングできます。

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"
  integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
  crossorigin="anonymous"></script>
```

ハッシュは `SHA256`、`SHA384`、`SHA512` の3種類のアルゴリズムで計算される。現在、もっとも利用されているのは `SHA384` (モバイルでは66.2%) であり、次いで `SHA256` (モバイルでは31.1%) となっています。現在、この3つのハッシュアルゴリズムはすべて安全に使用できると考えられています。



図12.18. モバイル向け `<script>` 要素に含まれるSRIの割合。

ここ数年、SRIの利用がやや増加しており、デスクトップで17.5%、モバイルでは16.1%の要素にintegrity属性が含まれています。そのうち82.6%は、モバイルの `<script>` 要素に含まれています。

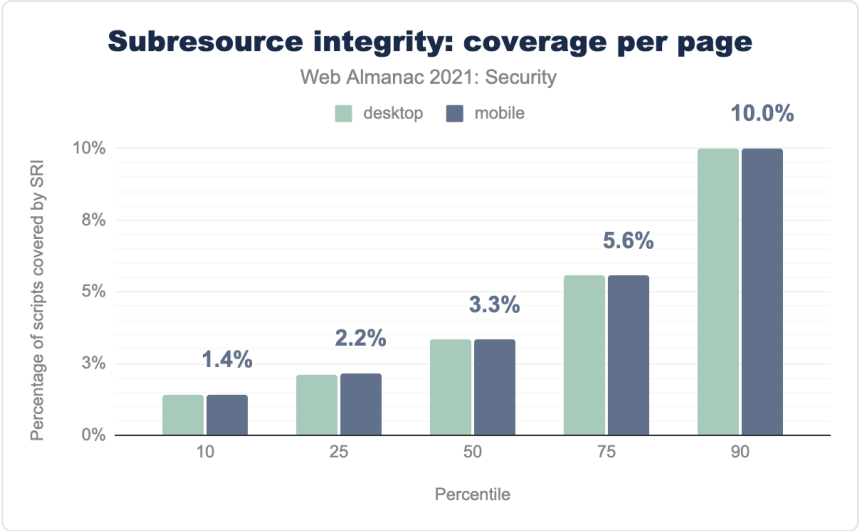


図12.19. サブリソースの整合性：1ページあたりのカバー率。

しかし、`<script>`要素については、まだ少数派の選択肢です。ウェブサイト上の`<script>`要素で`integrity`属性を持つものの割合の中央値は、3.3%です。

ホスト	デスクトップ	モバイル
www.gstatic.com	*44.3%*	*44.1%*
cdn.shopify.com	*23.4%*	*23.9%*
code.jquery.com	*7.5%*	*7.5%*
cdnjs.cloudflare.com	*7.2%*	*6.9%*
stackpath.bootstrapcdn.com	*2.7%*	*2.7%*
maxcdn.bootstrapcdn.com	*2.2%*	*2.3%*
cdn.jsdelivr.net	*2.1%*	*2.1%*

図12.20. SRIで保護されたスクリプトが含まれるもっとも一般的なホスト。

SRIで保護されたスクリプトが含まれる一般的なホストのうち、そのほとんどがCDNで構成されていることがわかります。異なるライブラリを使用する場合、複数のウェブサイトで使

用される非常に一般的なCDNが3つあります。jQuery[533]、cdnjs[534]とBootstrap[535]。これら3つのCDNが、サンプルのHTMLコードにintegrity属性を備えているのは、おそらく偶然ではないでしょう。開発者がサンプルを使ってこれらのライブラリを埋め込むとき、SRIで保護されたスクリプトが読み込まれていることを確認することになります。

パーミッションポリシー

最近のブラウザは、トラッキングや悪意のある目的に使用できる無数のAPIや機能を提供しており、ユーザーのプライバシーに悪影響を与えることが判明しています。*パーミッションポリシー*は、ウェブサイトが自身のフレームや埋め込んだiframe内のブラウザ機能の使用を許可またはブロックする機能を提供するウェブプラットフォームAPIです。

`Permissions-Policy`レスポンスヘッダーにより、ウェブサイトはどの機能を使用したいか、また悪用を制限するためにウェブサイト上でどの強力な機能を禁止したいかを決定できます。パーミッションポリシーは、ジオロケーション、ユーザーメディア、ビデオ自動再生、暗号化メディアデコードなどのAPIを制御するために使用できます。これらのAPIのいくつかは、ユーザーからのブラウザの許可を必要としますが、悪意のあるスクリプトはユーザーが許可のポップアップを取得せずにマイクをオンにすることはできません、ウェブサイトが必要としない場合、特定の機能の使用を完全に制限するために許可ポリシーを使用することは依然として良い習慣です。

このAPI仕様は、以前は*フィーチャーポリシー*として知られていましたが、名称が変更されただけでなく、他にも多くの更新が行われています。この`Feature-Policy`レスポンスヘッダーはまだ使用されていますが、モバイル向けウェブサイトの0.6%しか使用していません。`Permissions-Policy`レスポンスヘッダーには、異なるAPIに対する許可リストが含まれています。たとえば、`Permissions-Policy: geolocation=(self "https://example.com")`は、自身のオリジンと"`https://example.com`"オリジン以外のGeolocation APIの利用を許可しないことを意味します。たとえば、`Permissions-Policy: geolocation=()`のように、空のリストを指定することで、ウェブサイトでのAPIの使用を完全に無効にすることができる。

モバイルサイトでは、すでに1.3%のサイトが`Permissions-Policy`を使用していることが確認されています。この新しいヘッダーの使用率が予想以上に高い理由として、一部のウェブサイトの管理者が、ユーザーのプライバシーを保護するためにコホートのフェデレート学習またはFLoC[536]（Chromeで実験的に実装）のオプトアウトを選択している可能性が挙げられます。プライバシーの章に詳しい分析が載っています。

533. *https://code.jquery.com/*
534. *https://cdnjs.com/*
535. *https://www.bootstrapcdn.com/*
536. *https://privacysandbox.com/proposals/floc*

ディレクティブ	デスクトップ	モバイル
`encrypted-media`	46.8%	45.0%
`conversion-measurement`	39.5%	36.1%
`autoplay`	30.5%	30.1%
`picture-in-picture`	17.8%	17.2%
`accelerometer`	16.4%	16.0%
`gyroscope`	16.4%	16.0%
`clipboard-write`	11.2%	10.9%
`microphone`	4.3%	4.5%
`camera`	4.2%	4.4%
`geolocation`	4.0%	4.3%

図12.21. フレームにおける `allow` 指令の普及率。

また、`<iframe>` 要素の `allow` 属性を使用すると、埋め込みフレームで使用することを許可されている機能を有効または無効にできます。モバイルの1680万フレームの18.3%が、許可や機能のポリシーを有効にするため `allow` 属性を含んでいます。

この章の以前のバージョンでは、フレームの合計数と `allow` 属性を持つフレームの割合の値が間違っていました。これらのエラーは修正されました。詳細については、このGitHub PR[537] を参照してください。

例年通り、iframeの `allow` 属性でもっとも使用されているディレクティブは、埋め込みビデオやメディアのコントロールに関連するものです。もっともよく使われるディレクティブは、引き続き `encrypted-media` で、これは暗号化メディア拡張APIへのアクセスを制御するために使用されます。

Iframeサンドボックス

iframe内に信頼できない第三者が存在すると、そのページに対してさまざまな攻撃を仕掛けることができます。たとえば、トップページをフィッシングページに誘導したり、偽のアンチウイルス広告を表示するポップアップを起動したり、その他のクロスフレームスクリプテ

537. https://github.com/HTTPArchive/almanac.httparchive.org/pull/3912

ィング攻撃を行うことができます。

iframeの `sandbox` 属性は、コンテンツに制限をかけるため、埋め込まれたWebページから攻撃を仕掛ける機会を減らすことができます。属性の値は、すべての制限を適用する場合は空、特定の制限を解除する場合はスペースで区切られたトークンのいずれかになります（いくつかの制限を挙げると埋め込みページはJavaScriptコードを実行できず、フォームは送信できず、ポップアップを作成できません）。広告やビデオなどのサードパーティーコンテンツをiframeで埋め込むことは、ウェブ上では一般的な行為であり、その多くが `sandbox` 属性によって制限されていることは驚くことでありません。デスクトップ用ページのiframeの19.7%が `sandbox` 属性を持っており、モバイル用ページでは21.0%となっています。

この章の以前のバージョンでは、`sandbox` 属性を持つフレームの割合の値が間違っていました。詳細については、このGitHub PR[538] を参照してください。

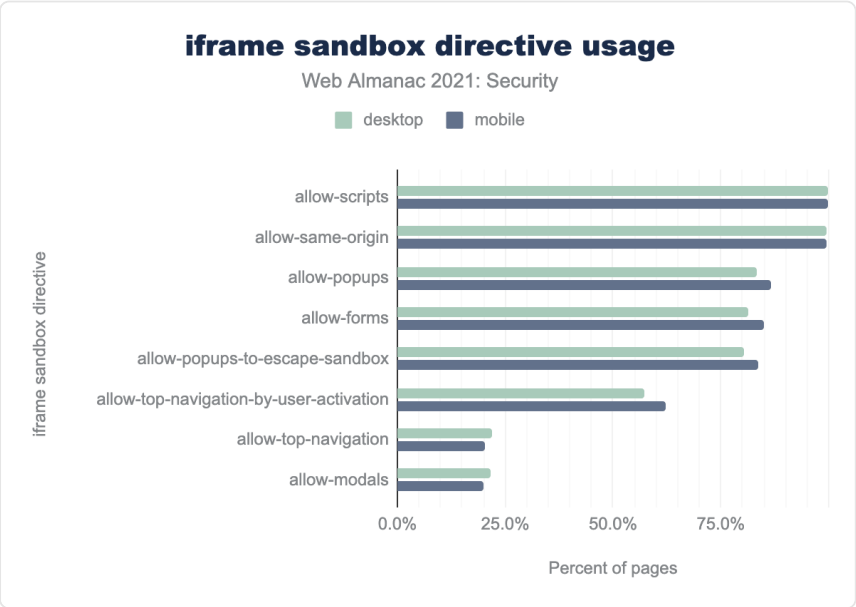


図12.22. フレームに対するサンドボックスディレクティブの普及率。

もっともよく使われるディレクティブは `allow-scripts` で、デスクトップページのすべてのサンドボックスポリシーの99.98%に存在し、埋め込みページがJavaScriptコードを実行することを許可しています。もう1つのディレクティブは事実上すべてのサンドボックスポリシーに存在する `allow-same-origin` で、埋め込みページがそのオリジンを保持し、たとえば、そのオリジンに設定されたクッキーへアクセスすることを許可します。

538. *https://github.com/HTTPArchive/almanac.httparchive.org/pull/3912*

攻撃を阻止する

ウェブアプリケーションは、複数の攻撃に対して脆弱である可能性があります。幸いある種の脆弱性を防ぐメカニズムがいくつか存在します（たとえば、クリックジャッキング攻撃に対抗する[539]には`X-Frame-Options`や、CSPの`frame-ancestors`ディレクティブによるフレーム保護が必要です）、あるいは攻撃の結果を制限することが可能です。これらの保護機能のほとんどはオプトイン方式であるため、ウェブ開発者が適切なレスポンスヘッダーを設定することで有効化する必要があります。大規模な場合、ヘッダーの存在は、ウェブサイトのセキュリティ衛生状態や、開発者がユーザーを保護するインセンティブについて、何かを教えてくれるかもしれません。

セキュリティ機能の採用

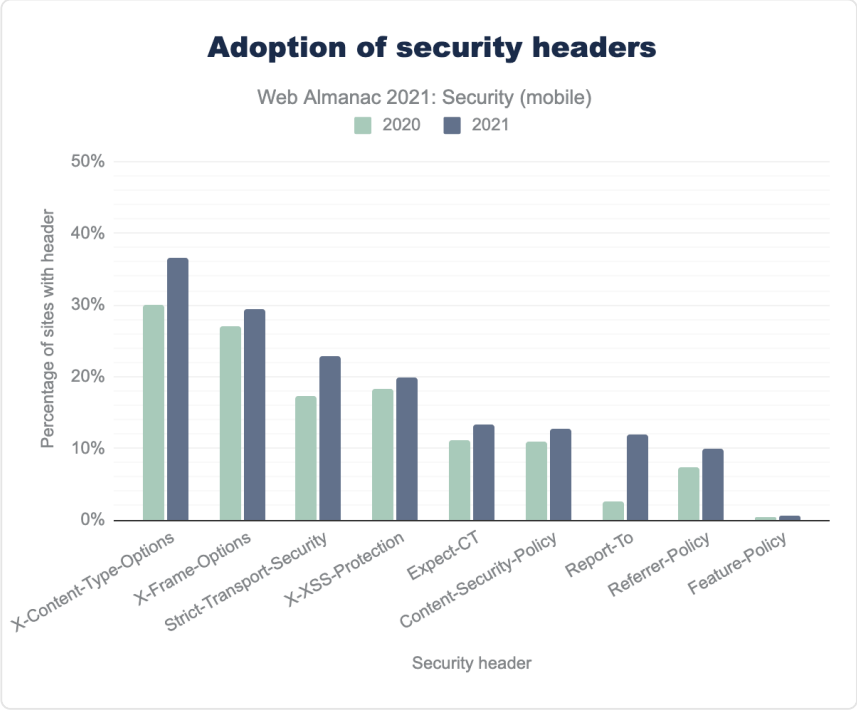


図12.23. モバイルページにおけるサイトリクエストのセキュリティヘッダーの採用。

本章でもっとも有望かつ明るい発見は、おそらく、セキュリティ機構の一般的な採用が増え続けていることです。これは、攻撃者が特定のウェブサイトを悪用するのがより困難になる

539. *https://pragmaticwebsecurity.com/articles/securitypolicies/preventing-framing-with-policies.html*

ことを意味するだけでなく、より多くの開発者が、自分たちが構築するウェブ製品のセキュリティを重視していることを示すものです。全体として、昨年と比較して、セキュリティ機能の採用が10〜30%相対的に増加していることがわかります。セキュリティ関連の仕組みでもっとも導入が進んでいるのは、the Reporting API[540] の `Report-To` ヘッダーで、導入率は2.6%から12.2%と、ほぼ4倍に増加しています。

このように、セキュリティ機構の採用率が上昇し続けていることは注目に値しますが、まだまだ改善の余地は残されています。もっとも広く利用されているセキュリティメカニズムは、依然として `X-Content-Type-Options` ヘッダーであり、モバイルでクロールしたウェブサイトの36.6% でMIMEスニッフィング攻撃から保護するために利用されています。このヘッダーに続くのは、`X-Frame-Options` ヘッダーで、全サイトの29.4%で有効になっています。興味深いことに、CSPのより柔軟な `frame-ancestors` ディレクティブを使用しているウェブサイトは、わずか5.6%に過ぎません。

もう1つの興味深いのは、`X-XSS-Protection` ヘッダーの進化です。この機能は、レガシーブラウザのXSSフィルターを制御するために使用されます。Edge[541] とChrome[542] は意図しない新たな脆弱性をもたらす恐れがあるとして、XSSフィルターをそれぞれ2018年7月、2019年8月に引退させたそうです。しかし、`X-XSS-Protection` ヘッダーは昨年より8.5%も多いことがわかりました。

`<meta>` 要素で有効な機能

レスポンスヘッダーを送信することに加えて、いくつかのセキュリティ機能は `<meta>` 要素に `name` 属性を `http-equiv` に設定するとHTMLレスポンスボディで有効にできます。セキュリティ上の理由から、この方法で有効にできるポリシーは限られています。より正確には、コンテンツセキュリティポリシーとリファラーポリシーのみが `<meta>` タグで設定できます。それぞれ、0.4%と2.6%のモバイルサイトがこの方法でメカニズムを有効にしていることがわかりました。

3,410

図12.24. `X-Frame-Options` を `<meta>` タグに記述しているが、実際にはブラウザに無視されているサイトの数。

他のセキュリティ機構が `<meta>` タグで設定されている場合、ブラウザはこれを実際に無視します。興味深いことに、3,410のサイトが `<meta>` タグを使って `X-Frame-Options` を有効にしようとしており、その結果、クリックジャック攻撃から守られているという誤った

540. https://developers.google.com/web/updates/2018/09/reportingapi
541. https://blogs.windows.com/windows-insider/2018/07/25/announcing-windows-10-insider-preview-build-17723-and-build-18204/
542. https://www.chromium.org/developers/design-documents/xss-auditor

認識を抱いていたことがわかりました。同様に、数百のウェブサイトが、セキュリティ機能をレスポンスヘッダーの代わりに `<meta>` タグへ記述することで導入に失敗しました（`X-Content-Type-Options` : 357, `X-XSS-Protection` : 331, `Strict-Transport-Security` : 183）。

CSPによるXSS攻撃の阻止

CSPは、クリックジャック攻撃、混合コンテンツの取り込みの防止、コンテンツを取り込む信頼できるソースの決定（上で述べたとおり）など、多数のものから守るために使用できます。

さらに、XSS攻撃から身を守るために必要不可欠な仕組みです。たとえば、制限的な `script-src` ディレクティブを設定することで、ウェブ開発者はアプリケーションのJavaScriptコードだけが実行されるようにできます（攻撃者のコードは実行されません）。さらに、DOMベースのクロスサイトスクリプティングを防御するために、*信頼できるタイプ*を使うことができます。これは、CSPの `require-trusted-types-for` ディレクティブを使って有効にすることが可能です。

キーワード	デスクトップ	モバイル
`strict-dynamic`	5.2%	4.5%
`nonce-`	12.1%	17.6%
`unsafe-inline`	96.2%	96.5%
`unsafe-eval`	82.9%	77.2%

図12.25. `default-src` または `script-src` ディレクティブを定義するポリシーに基づくCSPキーワードの普及率です。

CSPの採用は全体的に緩やかな増加（17%）ですが、それ以上に興味深いのは、`strict-dynamic` とnoncesの利用が同じ傾向を維持しているか、わずかに増加していることです。たとえば、デスクトップサイトでは、`strict-dynamic` の使用率が2.4%昨年[543]から、今年は5.2%に増加しました。同様に、noncesの使用率も8.7%から12.1%に増加しています。

一方、問題の多いディレクティブである `unsafe-inline` と `unsafe-eval` の使用率は、まだかなり高いことがわかります。しかし、これらが `strict-dynamic` と共に使用されている場合、モダンブラウザはこれらの値を無視し、`strict-dynamic` をサポートしていない古いブラウザは引き続きウェブサイトを使用することができることに留意すべきです。

543. *https://almanac.httparchive.org/ja/2020/security#CSPによるXSS攻撃の防止*

XS-Leaksに対する防御

Web開発者がSpectre[544]のような攻撃などのマイクロアーキテクチャ攻撃や、一般にXS-Leaks[545]と呼ばれる攻撃からWebサイトを防御できるように、さまざまな新しいセキュリティ機能を導入しています。これらの攻撃の多くがここ数年で発見されたものであることを考えると、それらに対処するためのメカニズムも明らかにごく最近のものであり、これが比較的低い普及率の理由かもしれません。とはいえ、昨年[546]と比較すると、クロスオリジンポリシーの採用が大幅に増えています。

リソースをどのように含めるか（クロスオリジン、同一サイト、同一オリジン）をブラウザへ示すために使用される `Cross-Origin-Resource-Policy` は、昨年[547]1,712サイトから106,443サイト(1.5%)で存在することがわかりました。この理由としてもっとも考えられるのは、クロスオリジンの分離[548]が `SharedArrayBuffer` やハイレゾタイマーなどの機能を使うために必要で、それにはサイトの `Cross-Origin-Embedder-Policy` を `require-corp` に設定する必要があるからだと思われます。要するに、これらの機能を使いたいサイトのために、ロードされたすべてのサブリソースに `Cross-Origin-Resource-Policy` レスポンスヘッダーを設定することを要求しているのです。

その結果、several[549] CDN[550] は現在、ヘッダーの値を `cross-origin` に設定します（CDNのリソースは通常クロスサイトのコンテキストに含まれるべきものだからです）。CORPヘッダーの値を `cross-origin` に設定しているサイトは96.8%であるのに対し、 `same-site` に設定しているサイトは2.9%、より限定的な `same-origin` を使用しているサイトは0.3%であり、これは実際にそうであるということがわかります。

この変化に伴い、 `Cross-Origin-Embedder-Policy` の採用が着実に増えているのは当然のことです。2021年には、911サイトがこのヘッダーを有効にし、昨年の6サイトを大幅に上回りました。来年、これがさらにどのように発展していくのか、興味深いところです。

最後に、もう1つのXS-Leak対策ヘッダーである `Cross-Origin-Opener-Policy` も昨年と比較して大幅に増加しています。現在、このセキュリティ機構を有効にしているサイトは15,727件で、特定のXS-Leak攻撃から保護されているサイトが31件しかなかった昨年と比較すると、大幅に増加していることがわかりました。

ウェブ暗号化API

Web開発において、セキュリティは中心的な課題の1つとなっています。ウェブ暗号化API[551] W3C勧告は2017年に導入され、クライアントサイドでサードパーティのライブラリなしで

544. *https://ja.wikipedia.org/wiki/Spectre*
545. *https://xsleaks.dev*
546. *https://almanac.httparchive.org/ja/2020/security#defending-against-xs-leaks-with-cross-origin-policies*
547. *https://almanac.httparchive.org/ja/2020/security#defending-against-xs-leaks-with-cross-origin-policies*
548. *https://web.dev/i18n/ja/cross-origin-isolation-guide/*
549. *https://github.com/cdnjs/cdnjs/issues/13782*
550. *https://github.com/jsdelivr/bootstrapcdn/issues/1495*
551. *https://www.w3.org/TR/WebCryptoAPI/*

基本的な暗号操作（ハッシュ、署名生成・検証、暗号化・復号化など）を実行できるようになりました。このJavaScript APIの利用状況を分析しました。

暗号化API	デスクトップ	モバイル
CryptoGetRandomValues	70.4%	67.4%
SubtleCryptoDigest	0.4%	0.5%
SubtleCryptoEncrypt	0.4%	0.3%
CryptoAlgorithmSha256	0.3%	0.3%
SubtleCryptoGenerateKey	0.3%	0.2%
CryptoAlgorithmAesGcm	0.2%	0.2%
SubtleCryptoImportKey	0.2%	0.2%
CryptoAlgorithmAesCtr	0.1%	< 0.1%
CryptoAlgorithmSha1	0.1%	0.1%
CryptoAlgorithmSha384	0.1%	0.2%

図12.26. よく使われる暗号API

機能の人気は前年とほぼ同じで71.8%から72.5%へと0.7%の微増にとどまっています。今年も `Cypto.getRandomValues` がもっとも人気のある暗号化APIです。開発者は強力な擬似乱数を生成することができる。Google Analyticsのスクリプトがこの機能を利用しているため、やはりGoogle Analyticsの影響が大きいと思われます。

なお、受動的なクローリングを行っているため、機能実行前に何らかのインタラクションが必要なケースを特定できず、本節の結果は限定的なものとなっています。

ボット対策サービスの活用

多くのサイバー攻撃は自動化されたボット攻撃に基づいており、それに対する関心も高まっているようです。Imperva社の2021年の悪意のあるBotレポート[552]によると、今年に入ってから悪質ボットの数が25.6%増加したとのことです。なお、2019年から2020年の増加率は、前回のレポート[553]によると24.1%となっています。以下の表では、悪意あるボットからの防御のために、ウェブサイトがどのような対策をとっているかについての結果を示していま

552. https://www.imperva.com/blog/bad-bot-report-2021-the-pandemic-of-the-internet/
553. https://www.imperva.com/blog/bad-bot-report-2020-bad-bots-strike-back/

す。

サービスプロバイダー	デスクトップ	モバイル
reCAPTCHA	10.2%	9.4%
Imperva	0.3%	0.3%
Sift	0.1%	0.1%
Signifyd	0.03%	0.03%
hCaptcha	0.03%	0.02%
Forter	0.03%	0.03%
TruValidate	0.03%	0.02%
Akamai Web Application Protector	0.02%	0.02%
Kount	0.02%	0.02%
Konduto	0.02%	0.02%
PerimeterX	0.02%	0.01%
Tencent Waterproof Wall	0.01%	0.01%
Others	0.03%	0.04%

図12.27. ボット対策サービスのプロバイダー別利用状況。

当社の分析によると、悪意のあるボットと戦うためのメカニズムを使用しているのは、デスクトップWebサイトの10.7%未満、モバイルWebサイトの9.9%未満であることがわかりました。昨年は8.3%、7.3%でしたので、前年比で約30%増加しています。今年も、モバイル版よりもデスクトップ版の方がボット対策の仕組みが多く確認されています（10.8%対9.9%）。

また、我々のデータセットでは、ボット保護プロバイダーとして新たに人気のあるプレーヤーが見られます（例：hCaptcha）。

セキュリティメカニズム採用のドライバー

Webサイトがセキュリティ対策に投資する背景には、さまざまな影響が考えられます。そのような要因の例としては社会的なもの（例：特定の国でよりセキュリティ志向の教育が行われている、またはデータ侵害の場合により懲罰的な措置を取る法律）、技術的なもの（例：

特定の技術スタックでセキュリティ機能を採用することが容易または特定のベンダーがデフォルトでセキュリティ機能を有効にする）、または脅威ベースのもの（例：広く普及しているウェブサイトは、知名度が低いウェブサイトより標的型攻撃に直面するかもしれない）などがあります。このセクションでは、これらの要因がセキュリティ機能の採用にどの程度影響するかを評価しようとします。

ウェブサイトの訪問者はどこから来るか

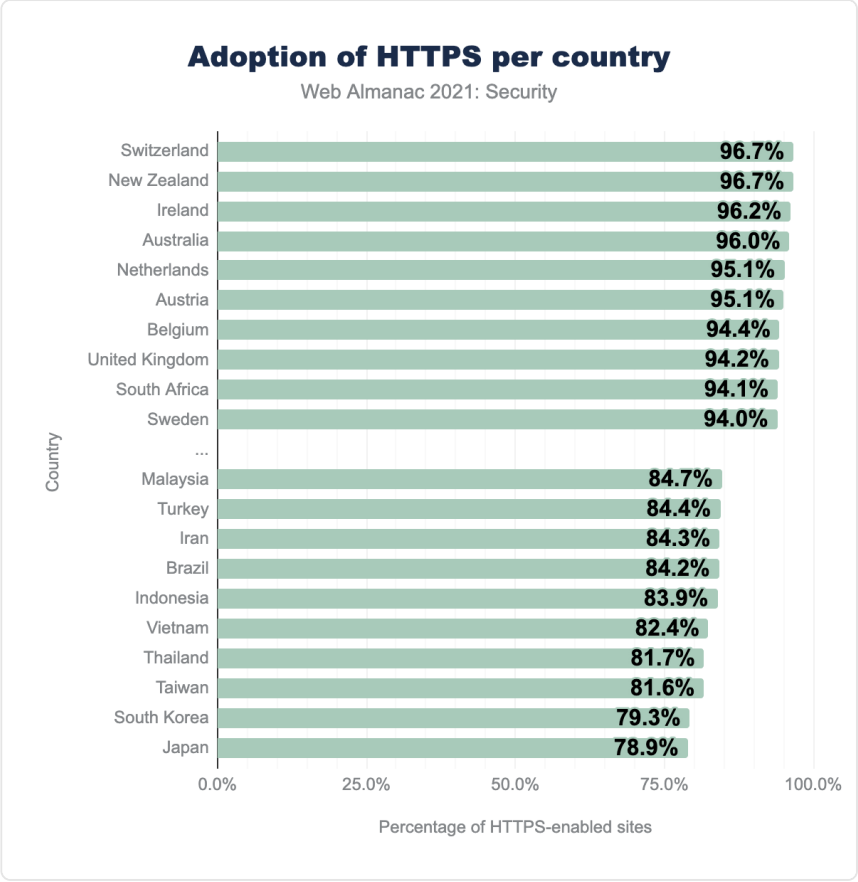


図12.28. 国ごとのHTTPSの採用状況。

デフォルトでHTTPSを採用するのが全般的に進んでいることがわかりますが、訪問者の出身国によって、サイト間の採用率にばらつきがあります。

昨年と比較して[554]、オランダがトップ5に入ったことがわかり、オランダ人はトランスポート層攻撃に対して比較的保護されていることがわかります。オランダの人々が頻繁に訪れるサイトの95.1%がHTTPSを有効にしています（昨年は93.0%）。実際、オランダだけでなく、ほぼすべての国でHTTPSの導入が進んでいることがわかります。

また、昨年は最悪の結果だったいくつかの国が大きく躍進したことは、非常に心強いことです。たとえば、イラン（HTTPSの普及率がもっとも高かった国）の人々が訪問したサイトは、昨年と比較して13.4%も多くHTTPSに対応しています（74.3%から84.3%）。もっとも導入が進んでいる国ともっとも進んでいない国の差は小さくなってきていますが、まだ大きな努力が必要です。

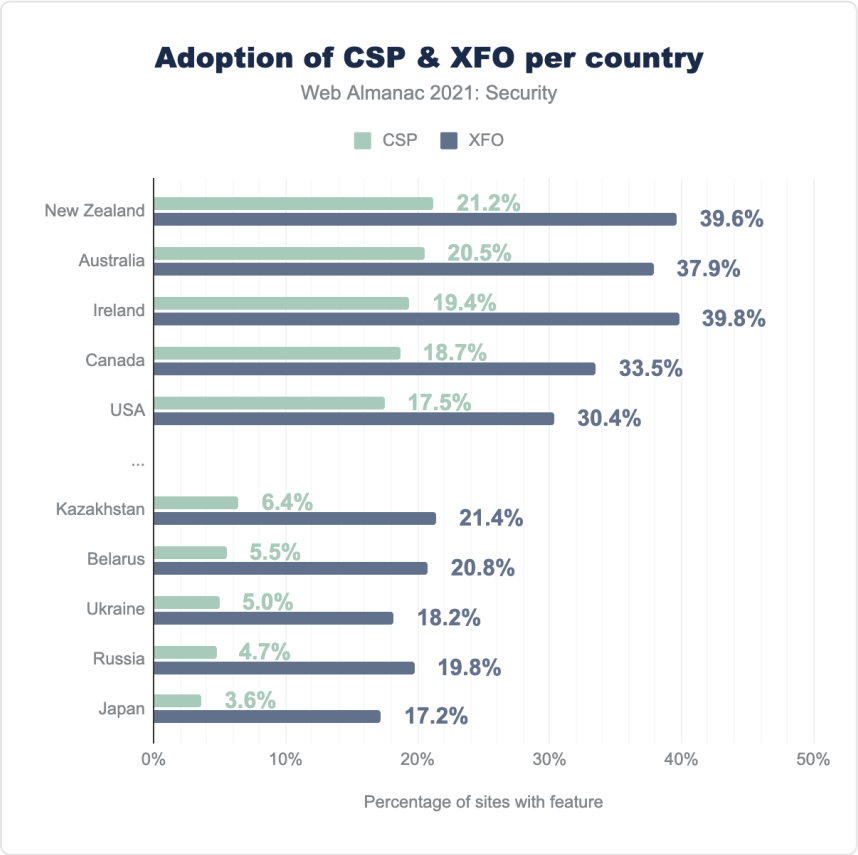


図12.29. 国ごとのCSPとXFOの採用状況。

CSPや `X-Frame-Options` などの特定のセキュリティ機能の採用状況を見ると、国によって

554. *https://almanac.httparchive.org/ja/2020/security#サイトの訪問者の国*

さらに顕著な違いがあり、成績上位の国のサイトは成績下位の国に比べて、これらのセキュリティ機能を採用する傾向が2〜4倍も高いことが分かっています。また、HTTPSの採用率が高い国は、他のセキュリティメカニズムの採用率も高い傾向にあることがわかります。これは、セキュリティはしばしば全体的に考えられ、さまざまな角度からカバーする必要があることを示しています。攻撃者は、悪用可能な脆弱性を1つ見つけるだけでよいのに対して、開発者は、あらゆる側面が確実に保護されていることを確認する必要があります。

Technology stack

技術紹介	*デフォルトで有効なセキュリティ機能*
オートマティック (PaaS)	*Strict-Transport-Security (97.8%)*
ブロガー（ブログ）	*X-Content-Type-Options (99.6%), X-XSS-Protection (99.6%)*
Cloudflare (CDN)	*Expect-CT (93.1%), Report-To (84.1%)*
Drupal (CMS)	*X-Content-Type-Options (77.9%), X-Frame-Options (83.1%)*
Magento (E-commerce)	*X-Frame-Options (85.4%)*
Shopify (E-commerce)	*Content-Security-Policy (96.4%), Expect-CT (95.5%), Report-To (95.5%), Strict-Transport-Security (98.2%), X-Content-Type-Options (98.3%), X-Frame-Options (95.2%), X-XSS-Protection (98.2%)*
Squarespace (CMS)	*Strict-Transport-Security (87.9%), X-Content-Type-Options (98.7%)*
Sucuri (CDN)	*Content-Security-Policy (84.0%), X-Content-Type-Options (88.8%), X-Frame-Options (88.8%), X-XSS-Protection (88.7%)*
Wix (Blogs)	*Strict-Transport-Security (98.8%), X-Content-Type-Options (99.4%)*

図12.30. さまざまな技術によるセキュリティ機能の採用。

特定のセキュリティメカニズムの採用に強く影響するもう1つの要因は、ウェブサイトを構築するために使用されている技術スタックです。場合によってはセキュリティ機能がデフォルトで有効になっていたり、ブログシステムによっては、レスポンスヘッダーに対する制御

がウェブサイト所有者の手を離れ、プラットフォーム全体のセキュリティ設定が行われていたりすることがあります。

また、CDNは、とくにトランスポートのセキュリティに関わる場合、追加のセキュリティ機能を追加することがあります。上の表では、少なくとも25,000のサイトで使用され、特定のセキュリティメカニズムの採用率が著しく高い9つのテクノロジーをリストアップしました。たとえば、Shopifyのeコマースシステムで構築されたサイトでは、7つのセキュリティ関連ヘッダー（`Content-Security-Policy`, `Expect-CT`, `Report-To`, `Strict-Transport-Security`, `X-Content-Type-Options`, `X-Frame-Options` と `X-XSS-Protection`）の適用率が非常に高く（95%以上）なっていることが見て取れるでしょう。



図12.31. Shopifyサイトでの採用率が95％を超えるセキュリティ機能の数々。

これらの技術を使用するコンテンツにばらつきがあるにもかかわらず、これらのセキュリティ機構を統一的に採用することが可能であることは、素晴らしいことだと思います。



図12.32. DrupalサイトがデフォルトのXFOヘッダーを保持している割合です。

このリストのもう1つの興味深い項目はDrupalで、そのウェブサイトでは `X-Frame-Options` ヘッダーの採用率が83.1%でした（昨年の81.8%と比較して若干改善されています）。このヘッダーはデフォルトで有効[555]なので、Drupalサイトの大多数がこれを使用し、クリックジャック攻撃から守っていることは明らかです。近い将来、古いブラウザとの互換性のために `X-Frame-Options` ヘッダーを残すことは理にかなっていますが、サイトオーナーは同じ機能を持つ推奨の `Content-Security-Policy` ヘッダーディレクティブ `frame ancestors` への移行を検討すべきことに留意してください。

セキュリティ機能の採用の文脈で探るべき重要な点は、多様性です。たとえば、Cloudflareは最大のCDNプロバイダーであり、何百万ものウェブサイトを支えています（これに関するさらなる分析については、CDNの章を参照してください）。Cloudflareがデフォルトで有効にする機能は、全体として大きな採用率になります。実際、`Expect-CT` 機能を採用しているサイトの98.2%はCloudflareによって提供されており、このメカニズムの採用がかなり限定的な分布であることを示しています。

555. *https://www.drupal.org/node/2735873*

しかし、全体的に見ると、DrupalやCloudflareのような単一のアクターがセキュリティ機能の採用を技術的に牽引するという現象は異常値であり、時間の経過とともに少なくなっているように見受けられます。これは、ますます多様化するウェブサイトがセキュリティ機構を採用し、より多くのウェブ開発者がその利点を認識するようになっていることを意味します。たとえば、昨年はコンテンツセキュリティポリシーを設定したサイトの44.3%がShopifyによるものでしたが、今年はCSPを有効にしたサイト全体の32.9%がShopifyによるものにとどまっています。一般的に普及率が高まっていることと合わせると、これは素晴らしいニュースです。

ウェブサイトの人気度

多くの人が訪れるWebサイトでは、潜在的な機密情報を持つユーザーが多く、攻撃者が集まりやすいため、標的型攻撃に遭いやすいと考えられます。したがって、多くの人が訪れるウェブサイトは、ユーザーを保護するために、より多くのセキュリティ投資を行うことが予想されます。この仮説が成り立つかどうかを評価するために、実際のユーザーデータを使って、どのウェブサイトがもっとも訪問されているかを調べるChromeユーザー体験レポートが提供するランキング（上位1000、1万、10万、100万とデータセット内の全サイトでランキング）を使用しました。

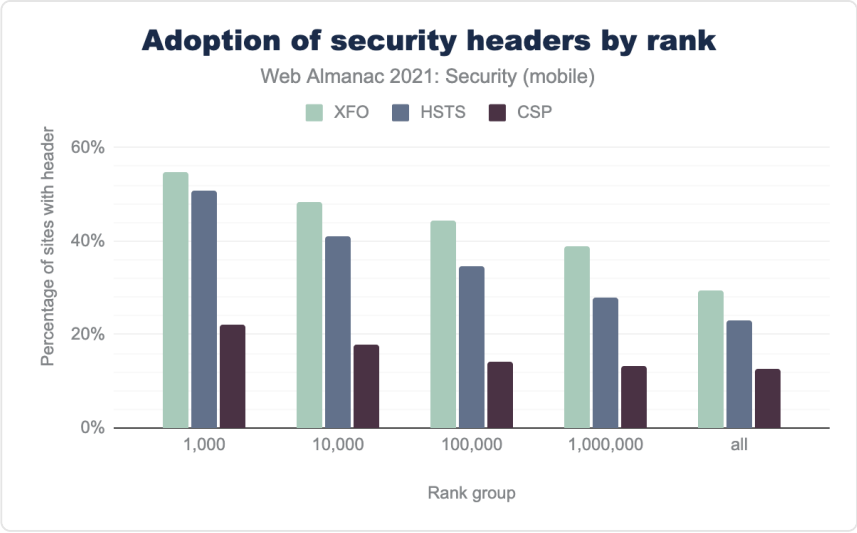


図12.33. ファーストパーティコンテキストに設定されたセキュリティヘッダーのランク別普及率。

X-Frame-Options (XFO)、Content Security Policy (CSP)、Strict Transport Security (HSTS) という特定のセキュリティ機能の採用が、サイトのランキングに大きく関係していることがわ

かります。たとえば、アクセス数の多い1,000のサイトでは、全体の採用率と比較して、あるセキュリティヘッダーを採用する割合が約2倍になっています。また、各機能の採用率は、ランキング上位のサイトほど高いことがわかります。

一方では、より多くの訪問者を集めるサイトでより優れた「セキュリティ衛生」を実現すれば、より多くのユーザー（よく知られた信頼できるサイトと個人データを共有する傾向がある）の利益につながるということです。一方、訪問者の少ないサイトでのセキュリティ機能の採用率が低いのは、これらの機能を（正しく）実装するためには、まだかなりの投資が必要であることを示しているのかもしれません。小規模なWebサイトでは、この投資が必ずしも有効であるとは限りません。願わくは、特定のテクノロジースタックで、デフォルトで有効になっているセキュリティ機能がさらに増え、ウェブ開発者がそれほど努力しなくても多くのサイトのセキュリティがさらに強化されることを期待します。

ウェブ上での不正行為

暗号通貨は、現代の社会でますます身近な存在になっています。世界的な暗号通貨の普及は、パンデミック発生当初から急増しています[556]。その経済的な効率性から、サイバー犯罪者も暗号通貨に関心を持つようになりました。そのため、新たな攻撃ベクトルが生まれたのです。クリプトジャッキング[557]です。攻撃者は、WebAssemblyの力を発見し、ウェブサイト訪問者がウェブサイト上でサーフィンしている間に暗号通貨をマイニングするために悪用されています。

そこで、Web上でのクリプトマイナーの利用状況について、次の図に示すような調査結果を得た。

556. *https://blog.chainalysis.com/reports/2021-global-crypto-adoption-index*
557. *https://ja.wikipedia.org/wiki/%E3%82%AF%E3%83%AA%E3%83%97%E3%83%88%E3%82%B8%E3%83%A3%E3%83%83%E3%82%AD%E3%83%B3%E3%82%B0*

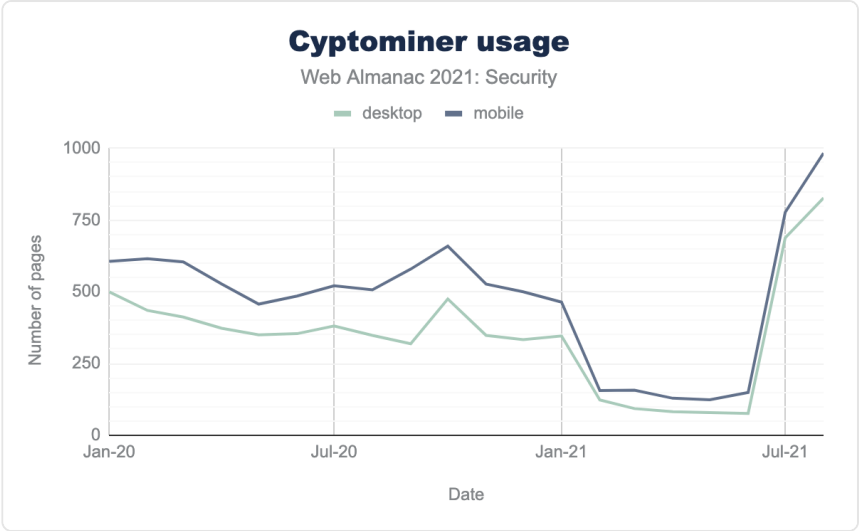


図12.34. クリプトマイナーの使用方法。

私たちのデータセットによると、最近まで、クリプトマイナーを使用したウェブサイトの数は非常に安定して減少していました。しかし、現在では、そのようなウェブサイトの数が過去2カ月間で10倍以上に増加していることが確認されています。このようなピックは、たとえば、広範なクリプトジャック攻撃が行われたときや、人気のあるJSライブラリが感染したときなど非常に典型的なものです。

次に、クリプトマイナーの市場シェアについて、次の図に示す。

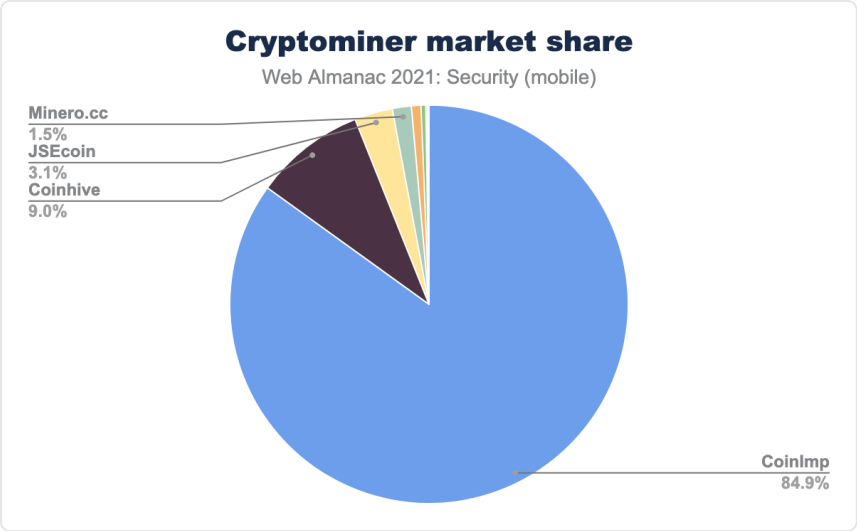


図12.35. クリプトマイナーの市場シェア（モバイル）。

Coinhive[558]は、CoinImpに抜かれ、クリプトマイニングサービスの主流となったことがわかります。その大きな理由のひとつが、2019年3月にCoinhiveが閉鎖されたこと[559]です。興味深いことに、このドメインは現在Troy Hunt[560]によって所有されており、Coinhiveスクリプトをまだホストしているサイト（デスクトップ: 5.7%, モバイル: 9.0%）に、しばしば彼らの知らないところでホストしていることを意識させようとウェブサイトに積極的にバナーを表示させているのだそうです。これは、Coinhiveのスクリプトが運営停止から2年以上経過しても普及していることと、サードパーティーのリソースが運営停止になった場合、引き継がれる可能性があることを反映しています。Coinhiveの消滅により、CoinImpは明らかに市場リーダー（シェア84.9%）となった。

この結果は、クリプトジャッキングが依然として深刻な攻撃ベクトルであることを示唆しており、必要な対策を講じる必要があることを示しています。

なお、これらのWebサイトのすべてが感染しているわけではありません。ウェブサイト運営者は、ウェブサイトの資金調達のために、（広告を表示する代わりに）この技術を展開することもあります。しかし、この手法の使用については、技術的、法的、倫理的にも大きく議論されています。

また、今回の結果は、クリプトジャッキングに感染したウェブサイトの実態を表していない可能性があることをご了承ください。クローラーの実行は月に1回であるため、クリプトマイナーを実行しているすべてのWebサイトを発見できるわけではありません。たとえば、あ

558. https://en.wikipedia.org/wiki/Monero#Mining_malware
559. https://www.zdnet.com/article/coinhive-cryptojacking-service-to-shut-down-in-march-2019/
560. https://www.troyhunt.com/i-now-own-the-coinhive-domain-heres-how-im-fighting-cryptojacking-and-doing-good-things-with-content-security-policies/

るWebサイトがX日間だけ感染したままで、私たちのクローラーが走った日には感染していない場合などがこれにあたります。

`security.txt`

`security.txt` は、ウェブサイトが脆弱性報告の基準を提供するためのファイル形式です。ウェブサイト提供者は、このファイルに連絡先、PGPキー、ポリシーなどの情報を記載できます。ホワイトハットハッカーは、この情報を利用して、これらのウェブサイトのセキュリティ分析を行ったり、脆弱性を報告したりできます。

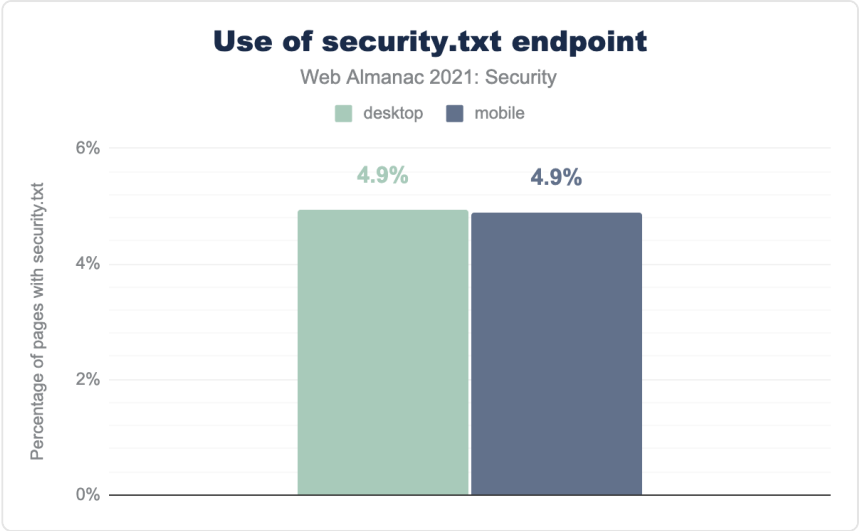


図12.36. Use of `security.txt`.

その結果、`/.well-known/security.txt` のURLを要求したところ、5%弱のウェブサイトが応答を返していることがわかりました。しかし、これらの多くは基本的に404ページであり、ステータスコード200を誤って返しているため、使用率はもっと低いと思われます。

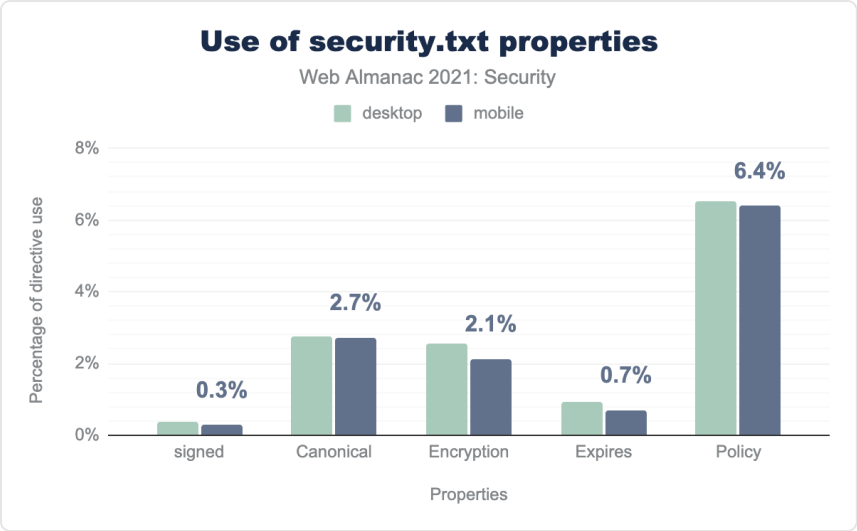


図12.37. security.txtのプロパティを使用する。

`Policy` は `security.txt` ファイルでもっとも使用されているプロパティですが、それでも `security.txt` のURLを持つサイトの6.4%でしか使用されていないことがわかります。このプロパティには、ウェブサイトの脆弱性開示ポリシーへのリンクが含まれており、研究者が従うべき報告慣習を理解するのに役立ちます。ほとんどのファイルが `Policy` 値を持つことが予想されるため、このプロパティは `security.txt` の実際の使用状況のより良い指標となるでしょう。つまり、すべてのサイトのうち、上記の5%ではなく、0.3%に近い数の「本物の」 `security.txt` ファイルが、存在する可能性があるということです。

もう1つの興味深い点は、この「本物の」security.txt URLのサブセットだけを見ると、Tumblrが63%〜65%の使用率を占めていることです。これらのドメインでは、デフォルトでTumblrの連絡先情報に設定されているようです。これは、単一のプラットフォームがこれらの新しいセキュリティ機能の採用を促進できることを示す一方で、実際のサイト利用がさらに減少していることを示す素晴らしいことです。

その他によく使われるプロパティは `Canonical` と `Encryption` です。 `Canonical` は `security.txt` ファイルがどこにあるのかを示すために使用されます。もし `security.txt` ファイルを取得するために使用したURIが `Canonical` フィールドのリストURIと一致しない場合は、ファイルの内容を信頼すべきではありません。 `Encryption` はセキュリティ研究者へ暗号化された通信に使用するための暗号鍵を提供します。

結論

当社の分析によると、プロバイダー側に関するWebセキュリティの状況は、以前と比較して改善されていることがわかります。たとえば、HTTPSの利用は、過去12か月で10%近く増加していることがわかります。また、クッキーの保護やセキュリティヘッダーの使用も増えていることがわかります。

これらの増加は、私たちがより安全なウェブ環境に移行していることを示していますが、今日の私たちのウェブが十分に安全であることを意味するものではありません。まだまだ改善しなければならないことがあります。たとえば、私たちは、ウェブコミュニティがセキュリティヘッダーをもっと重視すべきであると考えています。これらは、ウェブ環境やユーザーを攻撃から守るために非常に有効な拡張機能です。

また、悪意のあるボットからプラットフォームを保護するために、ボット保護機構をより多く採用できます。さらに昨年[561]の私たちの分析や、HTTP Archiveデータセットを用いたupdate behavior of websites[562]に関する別の研究では、ウェブサイトのコンポーネントが熱心にメンテナンスされておらず、ウェブ環境の攻撃表面を増大させていることが示されました。

また、攻撃者は、私たちが採用しているセキュリティの仕組みを回避するために、新しい技術の開発に熱心に取り組んでいることも忘れてはならない。

この分析によって、私たちのウェブのセキュリティの概要を結晶化することを試みました。私たちの調査は広範囲におよびますが、私たちの方法論では、現代のウェブセキュリティのすべての側面のサブセットを見ることができるだけです。たとえば、*クロスサイトリクエストフォージェリ*（CSRF）や*クロスサイトスクリプティング*（XSS）のような攻撃を軽減・防止するためにサイトが採用している追加措置についてはわかりません。このように、この章で描かれた絵は不完全なものですが、今日のWebセキュリティの状況を示す確かな方向性を示しています。

この分析から得られることは、私たちウェブコミュニティはより良くより安全な明日のために、ウェブ環境をより安全にするため、より多くの関心と資源を投資し続けなければならないということです。

561. *https://almanac.httparchive.org/ja/2020/security#ソフトウェアアップデートの実践*
562. *https://www.researchgate.net/publication/349027860_Our_inSecure_Web_Understanding_Update_Behavior_of_Websites_and_Its_Impact_on_Security*

著者



Saptak Sengupta

@Saptak013 saptaks https://saptaks.website/

Saptak Sは人権を中心としたウェブ開発者であり、ウェブ開発におけるユーザビリティ、セキュリティ、プライバシー、アクセシビリティのトピックに焦点を当てています。A11Y Project[563]、OnionShare[564]、Wagtail[565] など様々なオープンソースプロジェクトの貢献者、メンテナである。ブログはsaptaks.blog[566]でご覧いただけます。



Tom Van Goethem

@tomvangoethem tomvangoethem https://tom.vg/

Tom Van Goethemは、ベルギーのルーヴェン大学DistriNet group[567]の研究者です。彼の研究は、セキュリティやプライバシーの問題につながるウェブ上の新しいサイドチャネル攻撃を発見し、その原因となるリークを修正する方法を見つけ出すことに重点を置いています。



Nurullah Demir

@nrllah nrllh https://ndemir.com

Nurullah Demirは、インターネットセキュリティ研究所[568]のセキュリティ研究者であり、博士課程に在籍しています。研究テーマは、堅牢なWebセキュリティ機構と敵対的機械学習です。

563. *https://www.a11yproject.com*
564. *https://onionshare.org/*
565. *https://wagtail.io/*
566. *https://saptaks.blog*
567. *https://distrinet.cs.kuleuven.be/*
568. *https://www.internet-sicherheit.de/en/*

部 II 章 13
モバイルウェブ



Jamie Indigo、 Dave Smart と Ashley Berman Hale によって書かれた。
David Fox と Fili Wiese によってレビュ。
Ruth Everett と David Fox による分析。
Shaina Hantsis 編集。
Sakae Kotaro によって翻訳された。

序章

2021年1月、世界人口の59.5%がインターネットを利用していた。全世界のアクティブなインターネットユーザー46億6000万人のうち、92.6% がモバイルデバイスでインターネットにアクセスしている[569]ということです。

ポケットに収められたモバイルウェブのユビキタス化により、Statista[570]によると、世界人口の80.8%がスマートフォンを所有しているとのことです。これは、前年比0.0%の比較的小さな伸びです。これに対し、2016年のスマートフォン所有率は49.4%でした。

この章では、世界的な接続性、技術の普及、モバイルフレンドリーな機能の利用など、モバイルウェブの最近の動向について考察した。

569. https://www.statista.com/statistics/617136/digital-population-worldwide/
570. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

方法論に関するメモ

モバイルウェブとの関連でタブレット端末の体験をどのように分類するかという課題を考えたとき、このデータセットは分析から除外することにしました。多くの場合、タブレットのデータは、デスクトップまたはモバイルに分類されます。どちらをデフォルトとすべきか、統一された基準はありません。

データソースについて

この章では、いくつかの異なるデータソースを使用しました。

- CrUX
- HTTP Archive
- Lighthouse
- Wappalyzer
- Akamai[571]

なお、HTTP ArchiveとLighthouseのデータは、ウェブサイトのホームページのみから特定されるデータに限定されており、サイト全体のデータではありません。詳しくは、Methodologyのページをご覧ください。

ワールドワイドな接続性

2021年も世界的なCOVID-19の大流行の影響を受け、地域によってその影響も異なり、対策も地域によって異なるという状況です。これにより、ノートパソコンやパソコンとモバイル端末の使い分けが変わってきたのでしょうか。

モバイル**Web**アクセスのコスト

モバイルウェブアクセスの金銭的コストは、2021年には大きく変化していた。ある分析[572]によると、イスラエルで1GBの平均価格はわずか0.05米ドルであることが判明しました。赤道ギニアで同じデータ通信料を使用した場合、ユーザーは49.67米ドルを支払うことになります。

パフォーマンス編」のデータによると、サイトの中央値は現在2,205KBとなっています。市

571. *https://x.com/paulcalvano/status/1454866401781587969*
572. *https://www.cable.co.uk/mobiles/worldwide-data-pricing/*

場データを使用して、自分のサイトのコストについて[573]は中央のサイトをロードするための最良のシナリオの価格を計算しました。

もっとも高額な有料ロードはカナダのユーザーで0.26米ドル、次いでブラジルで0.18米ドルでした。ポーランドやロシアの一般的なデータプランで同じページを読み込んでも、ユーザーの請求書にはほとんど記載されず、0.01米ドル未満にとどまります。

モバイルからのサイトへのトラフィックとデスクトップからのトラフィック (CrUX)

モバイル端末からのトラフィックとデスクトップからのトラフィックの割合はどうなっているのでしょうか？また、サイトの種類や業界によって、これらのユーザーの構成は大きく変わる可能性があります。

トラフィック利用の人気順

77.4%

図13.1. 2021年7月データの817,4923オリジンのうち、モバイルトラフィックがデスクトップトラフィックを上回った割合。

今年の新しいCrUXデータセットでは、もっとも人気のあるサイト ranked by magnitude[574]を、これらのオリジンへのトラフィック記録によって照会することが可能です。

573. *https://whatdoesmysitecost.com/#usdCost*
574. *https://developers.google.com/web/updates/2021/03/crux-rank-magnitude*

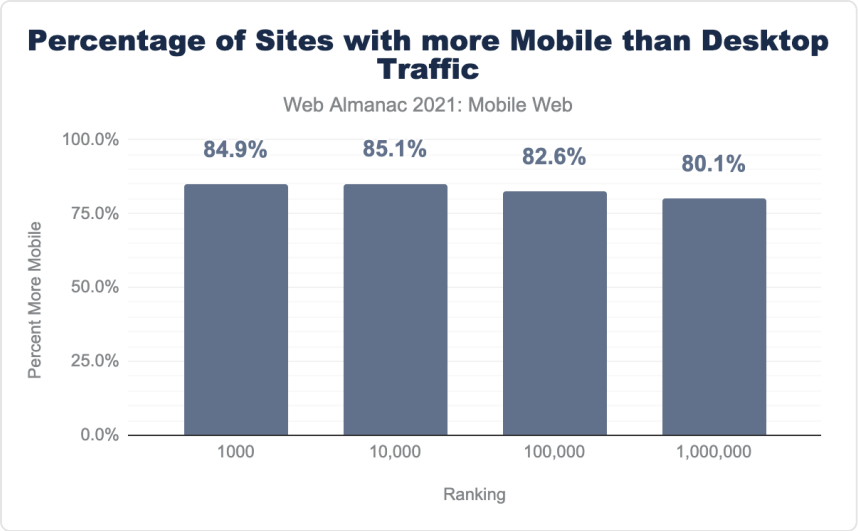


図13.2. デスクトップよりモバイルのトラフィックが多いサイトの割合。

CrUXランキング（データセット内のトラフィック上位1,000、10,000など）でグループ化すると、トラフィックが多いサイトほど、モバイルからのトラフィックの割合がわずかに増加します。上位1,000を除くすべてのサイトは、デスクトップに対してモバイルがわずかに少なく（84.9%対85.1%）なっています。

トラフィック分布

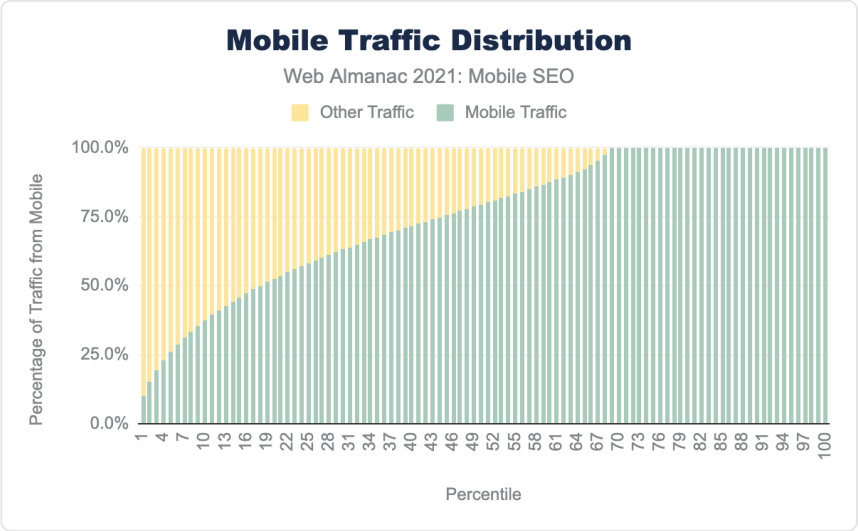


図13.3. モバイルとその他のトラフィックの分布。

分布も同様に、モバイルヘビーな傾向を示しています。50パーセンタイルでは、トラフィックの79.4%がモバイルデバイスからのもので、2020年の77.6%を上回り、2019年の79.9%に追いついています。

CrUXデータを超えて

CrUXのデータセットは、ログインしており、同期が有効で、*検索やブラウジングをより快適にする / あなたが閲覧したURLをGoogleに送信* の設定を無効にしていないChromeユーザーのデータしか収集できないという制限があります。こういう意味です。

- FirefoxやSafariなど、他の主要なブラウザは欠落している
- iOSユーザーからのデータはまったくありません（ChromeはiOSデバイスの他のブラウザと同様にiOSでWebKitを使用します）

幸いなことに、他にもいくつかの資料があります。Paul Calvanoは、2021年7月の Akamai mPulse[575] リアルユーザー モニタリング データについて、いくつかの分析を実施しました。また、モバイル端末からのトラフィックは59.4%で、モバイル端末とデスクトップ端末の比率はやや拮抗していることがわかりました。mPulseのデータは1時間ごとに集計されている

575. *https://www.akamai.com/products/mpulse-real-user-monitoring*

ため、いくつかの興味深い傾向を見ることができます。

すべての日々は平等ではない

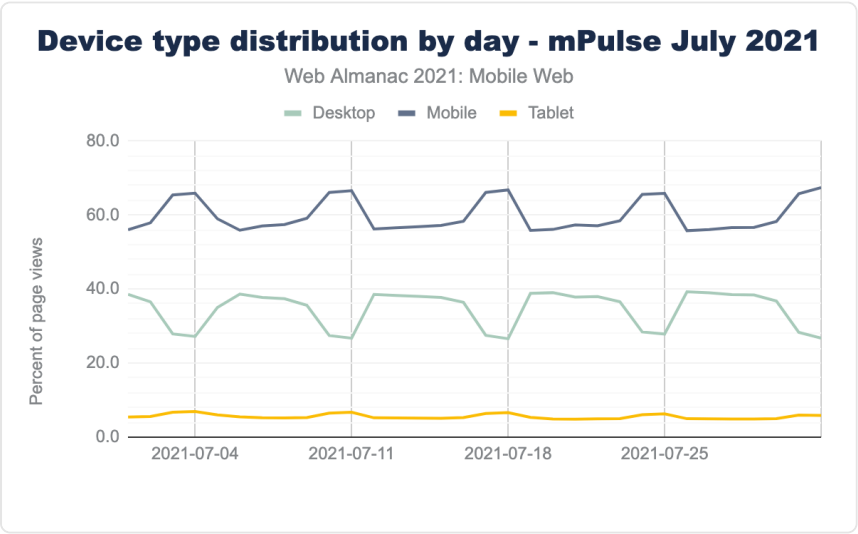


図13.4. 日別デバイスタイプ分布 - mPulse 2021年7月。

週末のモバイルトラフィックの割合は、55〜56%から65〜67%へと10%程度増加しています。世界的に見ると、すべての国が月曜日から金曜日までの労働週ではありません。日曜日から木曜日もよくあるパターン[576]で、これは金曜日にわずかに上昇し、土曜日と日曜日にモバイル利用の大きなジャンプにつながることが見て取れるものです。

すべての時代が平等ではない

平日はモバイルの利用が減少し、デスクトップの利用がトラフィック全体に占める割合が増加しています。これは、インターネットユーザーがモバイル端末とデスクトップ端末を切り替えて利用していることを示しています。午前5時（UTC）頃から上昇し、午後7時（UTC）頃に再び上昇し始めます（午前10時/11時頃にも若干の上昇があります）。これは、就業時間と一致しています。

576. *https://en.wikipedia.org/wiki/Workweek_and_weekend*

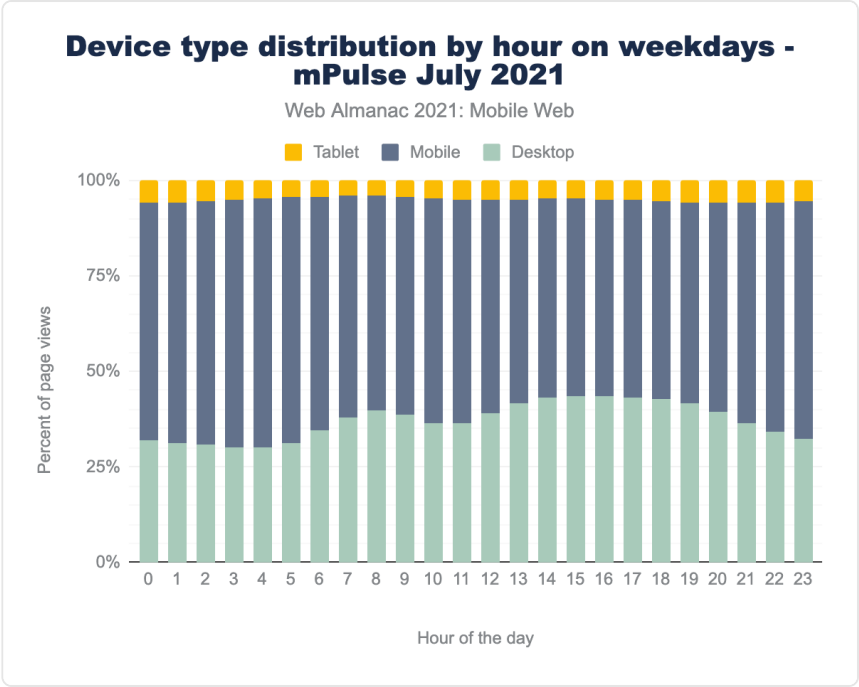


図13.5. 週末の時間帯別デバイスタイプ分布 - mPulse 2021年7月号。

週末は、モバイルとデスクトップのトラフィックの割合がより安定的に推移しています。

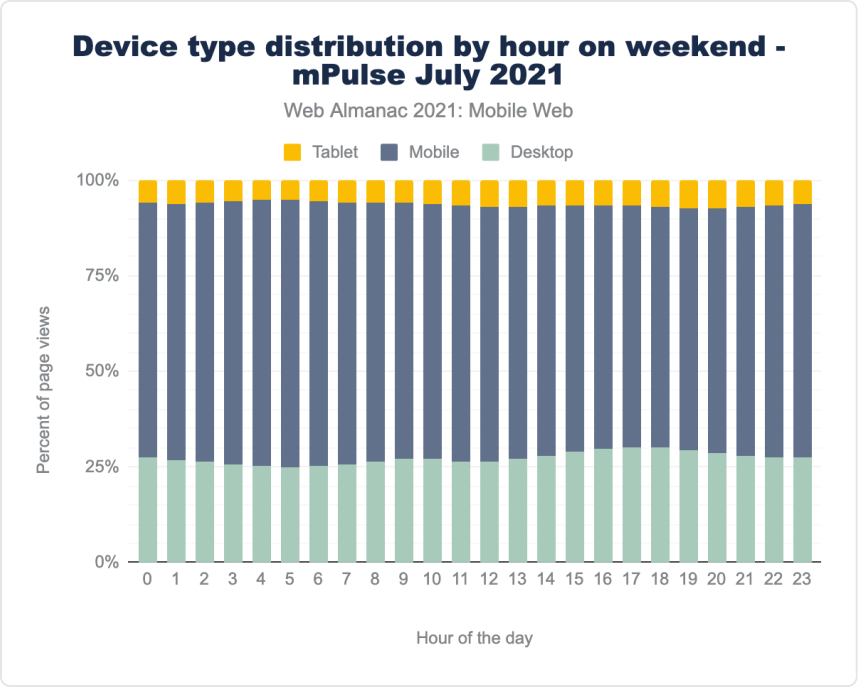


図13.6. 週末の時間帯別デバイスタイプ分布 - mPulse 2021年7月号。

このことから、異なるデバイスを選択できる人は、プライベートな時間ではモバイルのものを使う可能性が、高いことがわかります。

Cloudflareも素晴らしい調査結果を発表しています。アカマイのデータと同様に、この調査でも、CrUXのデータセットよりもモバイルデバイスとデスクトップデバイスがより近い割合で分かれていることが示されています。10月4日までの30日間で、トラフィックの52%はモバイルでした。

> *この1ヶ月間で、モバイルインターネットトラフィックの割合がもっとも高い国を探しました。その答えは... スーダン、インターネットトラフィックの83%がモバイルデバイスによるもので、実はイエメンと同率でした。*
>
> *– João Tomé、モバイルトラフィックがもっとも多い場所ともっとも少ない場所は？*[577]

577. https://blog.cloudflare.com/where-mobile-traffic-more-and-less-popular/

クラウドフレアのRadar[578]トレンドレポートでは、地域ごとにトラフィックをセグメントできます。モバイルとデスクトップの割合が地域ごとに異なるのは興味深いことで、スーダンやイエメンは83%の使用率で並ぶのに対し、セイシェルはわずか29%の使用率です。

結論を出すこと

モバイル端末の利用は引き続き堅調で、世界的に在宅勤務が増える傾向にあるにもかかわらず（保健当局や政府による規制や勧告のため）、モバイル端末によるウェブサイトへのアクセスは依然としてもっとも一般的な手段であることが明らかになりました。デスクトップに対するモバイルの人気は、昨年失ったものの大部分を取り戻したように見えますが、それ自体はかなり小さな後退です。

当然ながら、この数字からその理由を知ることはできませんが、多くのウェブユーザーにとってモバイル端末が唯一のデバイスであり、モバイルを使うかデスクトップを使うかの選択肢がないことは覚えておいて損はないでしょう。

モバイルトラフィックの割合が予想通りかどうかを予測するのは難しいですが、地域や分野に対して低いと思われる場合は、この部分のユーザーへのサービスが不十分であることを示している可能性があります。

モバイルの方法論と技術スタック

モバイルウェブは非常によく利用されていますが、これらの体験は一般的に処理能力が低く、インターネットの相互接続も遅いものです。 このような制限を緩和するために、多くの技術が登場しています。 たとえば、接続の種類を識別して、その接続に最適なアセットを提供するクライアントヒントやAPIなどがあります。

このセクションでは、モバイルウェブの全体的なアプリの使用状況や、プログラミング言語、コンテンツ管理システム、ウェブサーバーがデスクトップのエクスペリエンスと比較してどうであるかについても見ていきます。

クライアントヒント

*クライアントヒント*は、サーバーがアクセスするクライアントに要求して、デバイスやその機能、ネットワーク状況、その他のエージェント設定やプリファレンスに関する情報を取得できるHTTPリクエストヘッダーフィールドの集合体です。

これにより、そのデバイスに合わせた判断やコード、コンテンツ、エクスペリエンスの提供

578. *https://radar.cloudflare.com/*

が可能になります。

モバイルウェブでは、劣悪なネットワーク環境と低性能なデバイスがより一般的であり、この情報を積極的に要求するサイトは、単にデスクトップページをモバイル画面に収まるように縮小する以上のことを考えている可能性が高いです。

HTTPクライアントヒントは比較的新しい機能で、RFCは今年の2月に発行されたばかり[579]で、やや実験的なものです。したがって、1.4%のサイトがモバイルユーザーに対して少なくとも1つのクライアントヒントを要求していることがわかったのは、非常に心強いことです。

このような情報をもとにサイトが何をするのか、どのようにモバイルユーザー向けにカスタマイズするのかは分かりませんが、最初の兆候としては良いことだと思います。

これらのヒントは、大きく分けて3つのグループに分けられます。

- **デバイスクライアントのヒント**: サイトにアクセスするデバイスの機能および特徴の詳細。
- **ネットワーククライアントのヒント**: 機器とサーバー間のネットワーク接続の詳細。
- **ユーザーエージェントのヒント**: アクセスしているエージェントに関する詳細。

579. *https://www.rfc-editor.org/rfc/rfc8942#section-3.1*

デバイスクライアントのヒント

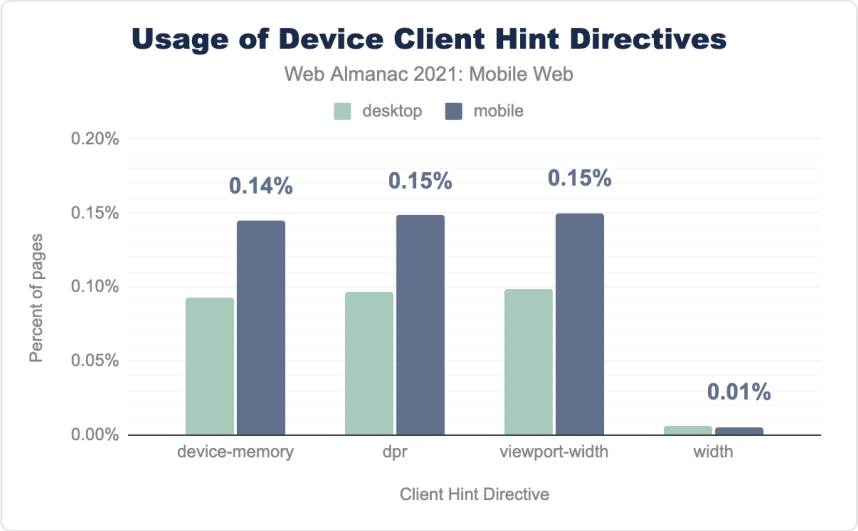


図13.7. Device Client Hintディレクティブの使用方法。

`DPR` と `Viewport-Width` はモバイルサイトの0.15%でトップ、 `Device-Memory` は0.14%で少し遅れ `Width` はわずか0%ですが、これは現在非推奨で代替案はSec-CH-Widthで、これを要求するサイトは見つかりませんでした。

現在、これらのヘッダーをサポートしているのは、Chrome（およびMicrosoftのEdgeなどのChromiumベースのブラウザ）とOperaのみで、SafariとFirefoxはまだ搭載されていません[580]。

580. *https://caniuse.com/client-hints-dpr-width-viewport*

ネットワーククライアントのヒント

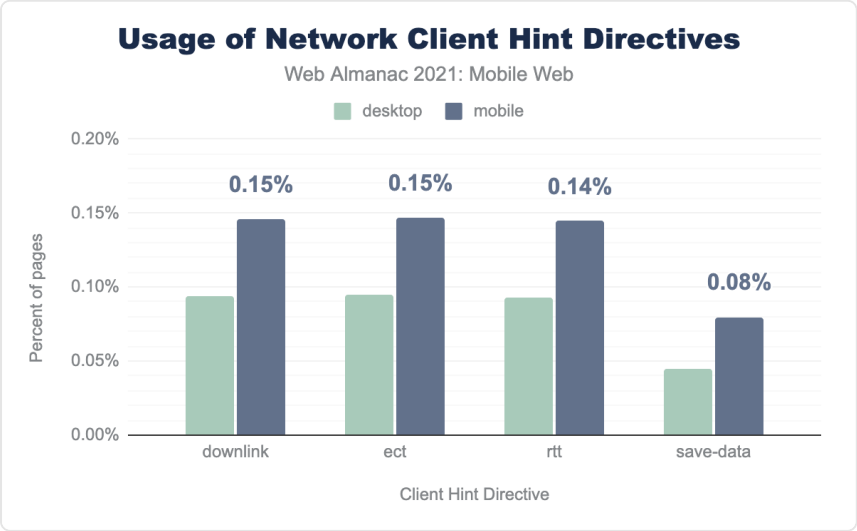


図13.8. Network Client Hintディレクティブの使用方法。

ネットワーククライアントヒントは、デバイスクライアントヒントと同様の利用率を示しており、Downlink[581] と ECT[582]（エフェクティブコネクションタイプ）はモバイルの負荷の0.2%が要求し、 RTT[583]（ラウンドトリップタイム）はモバイルの負荷の0.1%が要求しています。

Google Web Fundamentalsの記事、セーブデータで高速・軽快なアプリケーションを実現する[584]で詳しく述べられているように、ユーザーの利点を考えると、この機会を逃したように思われます。

User-Agent クライアントヒント

Chrome[585]、Safari[586]、Firefox[587] などの主要なブラウザは、パッシブフィンガープリント[588] を減らすため `User-Agent` 文字列に削減と上限設定を行いました。

従来は、サイトがこの情報を使って、それらのデバイスに合わせたエクスペリエンスを提供していたかもしれません。この方法は、刻々と変化するデバイスの状況に対応しようとすると、ユーザーエージェントの文字列が簡単に変更でき、なりすましが可能であるという欠点

581. https://developer.mozilla.org/docs/Web/HTTP/Headers/Downlink
582. https://developer.mozilla.org/docs/Web/HTTP/Headers/ECT
583. https://developer.mozilla.org/docs/Web/HTTP/Headers/RTT
584. https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/save-data/
585. https://blog.chromium.org/2021/05/update-on-user-agent-string-reduction.html
586. https://bugs.webkit.org/show_bug.cgi?id=216593
587. https://bugzilla.mozilla.org/show_bug.cgi?id=1679929
588. https://www.w3.org/2001/tag/doc/unsanctioned-tracking/#unsanctioned-tracking-tracking-without-user-control

がありました。

User-Agent クライアントヒント はこの情報を取得する方法を提供しますが、デバイスヒントやネットワークヒントとは異なり、`Accept-CH` ヘッダーを介してサーバーがこの情報を要求する必要はありません。これが、おそらくこれを要求しているサイトがほんの一握りしか検出されなかった理由です。

ネットワーク情報APIおよびデバイスメモリAPIの使用状況

ネットワーク情報API と `Navigator.deviceMemory` は、クライアントヒントで公開されるものと同様のスコープで、デバイスと接続情報を収集するためのJavaScriptへのインタフェースを提供します。

ネットワーク情報API

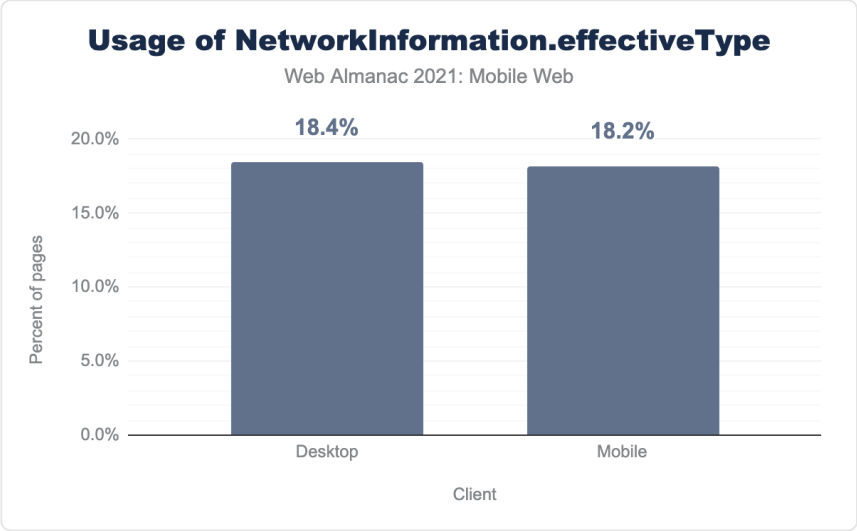


図13.9. `NetworkInformation.effectiveType` の使用法。

モバイルとデスクトップのページロードの比較には、`NetworkInformation.effectiveType` を使用しました。これは、有効な接続タイプ、`slow-2g`、`2g`、`3g`、`4g` に基づく文字列を返します。最上位は `4g` なので、5gやブロードバンド、固定接続を含む「4g以上の高速接続」と考えてもよいでしょう。

モバイルリクエストの18.2%に `NetworkInformation.effectiveType` を利用したページロードがありましたが、驚くべきことに、デスクトップリクエストの18.4%がこのAPIの利

用を検知しています。

デバイスメモリAPI

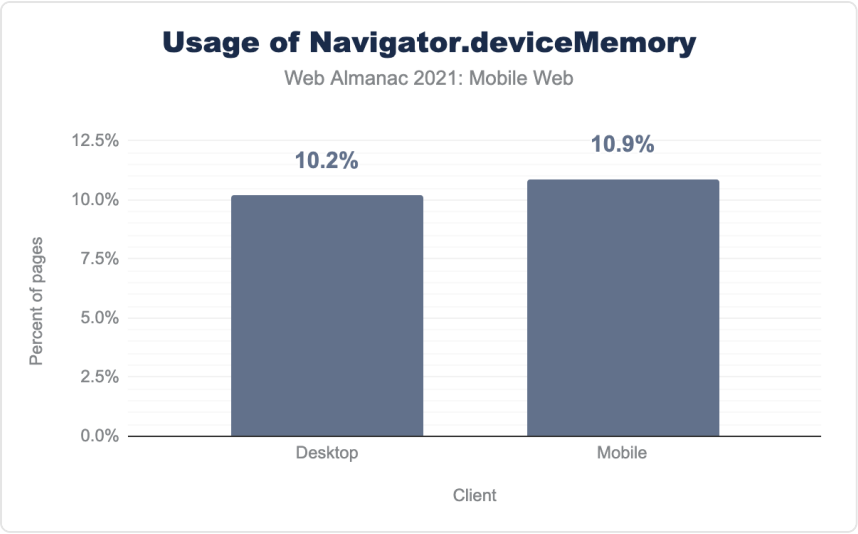


図13.10. `Navigator.deviceMemory` の使用法。

このAPIは、クライアントが処理できるかもしれないものを判断し、それに応じて適応するのに役立つ、デバイスメモリの概算量を返します。

モバイルページの読み込みの10.9%がこのAPIを利用しており、デスクトップの読み込みの10.2%をわずかに上回っています。

Client Hintsと同様、これらのAPIはまだ実験的なものです。また、各ブラウザでユニバーサルサポートされていません。（引用元: ネットワーク情報API[589]と `Navigator.deviceMemory`）しかし、より広く採用されています。

普及の理由のひとつは、サードパーティのスクリプトがページロード時にこれらを要求することだろう。もうひとつの理由は、実装のしやすさかもしれません。HTTPヘッダーの設定と読み込みは、より複雑で、インフラの変更を伴う可能性が高いと考えられるからです。

クライアントヒント、ネットワーク情報API、デバイスメモリAPIの結論

実験的なAPIや機能については、すでにいくつかの有望な導入事例があります。ブラウザの

589. https://caniuse.com/netinfo

サポートが進み、APIが実験的な状態から移行するにつれて、さらに利用が進むことを期待しています。

もしあなたがネットワークやデバイスの能力に制限のあるウェブアプリケーションを使用していて、低性能のデバイスや貧弱なネットワーク接続からアクセスするユーザーがかなりの割合を占めているなら、これらのAPIによって、より良いユーザー体験を提供できるかどうかを調査する良い機会かもしれません。

モバイルウェブでのアプリ利用状況

モバイルウェブでもっともよく使われるライブラリやテクノロジーは、パフォーマンスに影響を与え、テクノロジーの採用について情報を与えてくれます。

Wappalyzer[590]のデータによると、JavaScriptライブラリJQueryは、モバイルウェブの主要ライブラリで、テストサイトの84.4%に搭載されています。Googleは、上位5位中3位を占め、圧倒的なプロバイダーです。

アプリ	モバイル	デスクトップ	デスクトップとモバイルの使い分け
jQuery	84.4%	84.4%	1.0%
Google Analytics	65.4%	68.6%	3.2%
PHP	50.5%	50.5%	-0.4%
Google Font API	47.6%	47.6%	-0.1%
Google Tag Manager	43.4%	43.4%	2.6%

図13.11. 人気のある技術利用。

上位5つのモバイルWeb技術のうち、3つの技術の採用率はデスクトップサイトより高かった。 これらのアプリは、Googleがパフォーマンスの問題を診断するために推奨しているオープンソースの監査ツールであるLighthouseによって頻繁にフラグが立てられるため、モバイルパフォーマンスへの取り組みによってモバイルでの採用率が低下したと考えるのは妥当でしょう。

2021年、Googleはページ体験ランキングシグナル[591]をアルゴリズムに追加しました。このランキングシグナルは、モバイルデバイスで提供される検索エンジンの結果ページに特化したもので、実際のユーザーのページロードから収集したデータを使用してパフォーマンスを測定しています。

590. https://www.wappalyzer.com/
591. https://developers.google.com/search/docs/advanced/experience/page-experience

JavaScriptライブラリJQueryは、モバイルウェブで圧倒的なシェアを誇り、モバイルページ読み込みの84.4%で利用されています。Googleは、上位5位中3位を占める圧倒的なプロバイダーです。

コンテンツ管理システム

コンテンツ管理システムは、サイトオーナーが認証されたバックエンドを通じて、コンテンツの公開、更新、制御を行うことを可能にします。2021年のモバイルウェブにおけるコンテンツ管理システムの上位5つは以下の通りです。

CMS	モバイル	デスクトップ
WordPress	33.6%	32.9%
Joomla	2.0%	1.7%
Drupal	1.8%	2.1%
Wix	1.6%	1.2%
Squarespace	1.0%	1.2%

図13.12. モバイルとデスクトップのCMSを比較すると突出している。

2021年のCMSは、PHPで書かれたオープンソースのCMSであるWordPressが主流となった。この技術は33.6%のサイトに登場しました。

デスクトップ技術の採用率を比較する

モバイルウェブの技術採用率は、デスクトップと歩調を合わせて推移しました。もっとも顕著な違いは、サードパーティピクセルの使用という形で現れました。デスクトップサイトの68.6%がGoogle Analyticsを使用しているのに対し、モバイルサイトでは65.4%でした。

カテゴリー	テクノロジー	デスクトップ	モバイル	デスクトップ普及率向上
分析	Google Analytics	68.6%	65.4%	3.2%
タグマネージャー	Google Tag Manager	46.0%	43.4%	2.6%
分析	Facebook Pixel	20.6%	18.9%	1.7%
ウィジェット	Facebook	28.0%	26.3%	1.6%
JavaScript ライブラリ	jQuery UI	23.8%	22.2%	1.5%

図13.13. デスクトップ普及率の高い技術。

パフォーマンス測定と優先順位付けの変更を考慮すると、これらのJavaScriptを多用するサードパーティ製アセットがないのは、モバイルページのエクスペリエンスを向上させるための意図的な努力の一環と考えるのが妥当でしょう。Facebookピクセル分析スクリプトは、デスクトップに比べてモバイルサイトでは、-1.7%減少しました。

モバイルサイトでは、特定の技術を採用する傾向が強いが、その差は小さかった。Bloggerはモバイルサイトの3.1%、デスクトップサイトの1.7%で採用されています。

カテゴリー	テクノロジー	デスクトップ	モバイル	モバイルの普及率向上
ブログ	ブロガー	1.7%	3.1%	1.5%
ウェブサーバー	OpenGSE	1.7%	3.2%	1.5%
プログラミング言語	Python	2.2%	3.6%	1.4%
プログラミング言語	Java	2.8%	4.0%	1.2%

図13.14. モバイルの普及率が高い技術。

モバイルウェブアプリの利用状況に関する結論の導き出し

2021年、JQuery経由のJavaScriptがモバイルWebに浸透。サードパーティーの分析ツールは、モバイルでの採用率が低かった。

このデータで明らかになったのは、CMSやウェブサーバのレベルでは、モバイルとデスクトップはサイト開発方法において密接な相関関係があるということです。おそらく、レスポンシブデザインはオーバーヘッドが少なく、ひとつのコードベースですべての体験を提供でき

ることが大きな要因となっています。

WordPressがモバイルサイトでの人気を維持・拡大し、他のCMSもデスクトップと同様のシェアを獲得していることから、CMSのコアの改善と最適化がモバイルウェブ全体に大きな利益をもたらす絶好の機会となっているのです。

そのため、提案されたWordPressパフォーマンス チーム[592]のような活動は重要かつ貴重なものとなっています。

モバイルウェブとの対話

ユーザージャーニーにおける摩擦を減らすためには、モバイルデザインと使いやすさに気を配ることが重要です。ユーザーは、マウスやトラックパッドのような洗練された操作ではなく、指のタップでモバイルウェブをナビゲートします。

代替プロトコルリンク

ウェブはリンクで成り立っている。モバイルウェブでは、http/sを超えるUnique Resource Identifier[593] スキームにより、ユーザーは `tel:` を使って電話番号をダイヤルしたり 、メールを開始したりといった作業を最小限の摩擦で完了できます。

もっとも一般的なURIスキームは、93.2%のサイトで見られた `https:` と、その非セキュア版である `http:` で、56.7%に見られた。2020年には、コンテンツが安全でない場合に警告を発してユーザーの安全を守ることを、ブラウザが大きく発表したため、安全でないリンクプロトコルの高い使用率は注目される。

ウェブページのリンクの次に、モバイルウェブでアンカーhrefの値でよく使われるプロトコルは次の5つです。

592. *https://make.wordpress.org/core/2021/10/12/proposal-for-a-performance-team/*
593. *https://ja.wikipedia.org/wiki/Uniform_Resource_Identifier*

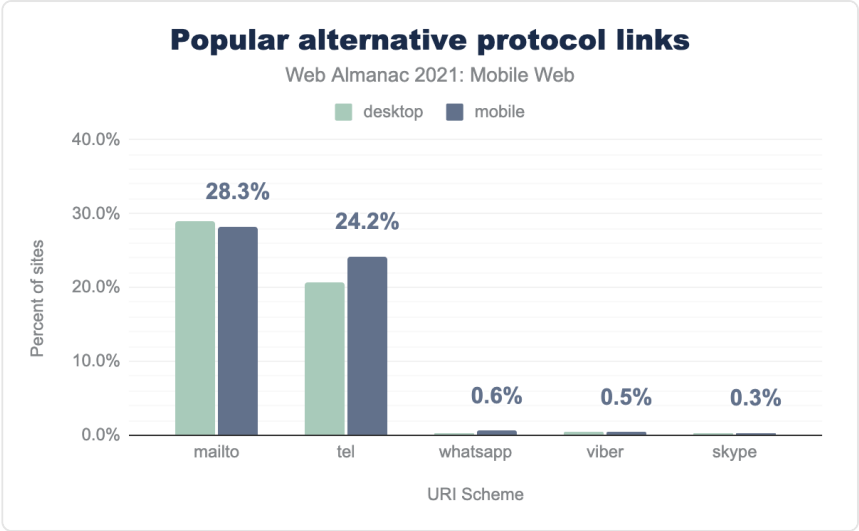


図13.15. 人気のある代替プロトコルのリンク集です。

モバイルデバイスは、電話であり、SMSやその他のメッセージングサービスを備えていますが、デスクトップクライアントは備えていない場合があります。標準的な `http:` / `https:` 以外のリンクプロトコルを使用することで、これらの機能のいくつかを解放できます。コピー&ペーストすることなく、通話やメッセージ送信のためのタップ可能なリンクを提供することで、よりスムーズで統合されたユーザーインタラクションを実現できます。

`mailto`

`mailto:` をクリックすると、ユーザーが選択したメールクライアントを起動します。

```
<a href="mailto:enquiries@example.com?subject=Enquiring about Red
Widgets">
  enquiries@example.com
</a>
```

指定されたメールアドレスと件名でメールをプリフィルする。モバイルで便利ですが、デスクトップにも関連します。

`tel`

`tel:` は電話をかけます。

```
<a href="tel:+44123467890">
  電話 +44 (0)123 4567890
</a>
```

電話アプリを開き、その番号にダイヤルする準備ができます。これにより、コピー＆ペーストの手間が省け、電話による問い合わせを重視するビジネスでは摩擦が少なくなります。

`sms`

`sms:` は、クライアントのデフォルトのSMSメッセージングアプリを呼び出します。

```
<a href="sms:+441234567890">
  テキストで問い合わせる
</a>
```

クリックすると正しい番号のメッセージがプリフィルされる場合、メッセージ本文をプリフィルすることもできます。これはトップ5から外れ、モバイルサイトの読み込みのわずか0.3%がこれを利用しています。

その他のメッセージングアプリ

他のメッセージングアプリは、`<a href="">` を開かせるプロトコルを登録できます。上の表にあるように、ここではWhatsAppとViberが主要で、ネイティブアプリ `sms:` の使用率を上回ります。

代替プロトコルリンクの結論

`mailto:` はインターネット上で長い歴史を持ち、1994年までさかのぼります[594]。しかし、モバイルデバイスでのさらなる有用性を考えると、`tel:` が24%の使用率に達し、遠く及ばないのは頼もしいことです。

594. https://datatracker.ietf.org/doc/html/rfc1738#section-3

smsの取り込みがこれほど少ないのは驚きであり、WhatsAppやViberなどの独自アプリを下回っているのは残念です。

SMSはデフォルトで利用できる可能性が高く、追加インストールも必要ないため、一見すると利用しやすいように見えます。しかし、WhatsAppとViberのメッセージは無料ですが、SMSのメッセージはユーザーの携帯電話会社から料金が、発生する場合があります。このことが、相対的な人気の高さを説明しているのかもしれない。

`https:` 以降のプロトコルがユーザーに提供できる通信のための拡張機能を使用しておらず、モバイルウェブサイトに適している場合、これらはシンプルでユーザーフレンドリー、かつ低開発の利点を提供できる可能性があります。

入力欄

URIスキームが、ユーザーがウェブサイトからアクションを起こすことを可能にするのに対し、入力欄はユーザーがウェブサイトに情報を提供することを可能にする。

入力要素は、HTMLの中でもっとも強力で複雑な機能の1つです。入力要素は、ウェブベースのフォームのためのインタラクティブなコントロールを作成するために使用されます。ウェブユーザーは、ボタン、チェックボックス、カレンダー、検索など、ユーザーの入力に基づきページの内容を制御できるこれらの要素を経験します。

71.5%

図13.16. モバイルページで入力を使用している割合。

テストしたモバイルページの71.5%が入力を含んでいました。これは、デスクトップの71.1%よりわずかに高い値です。

タイプ宣言

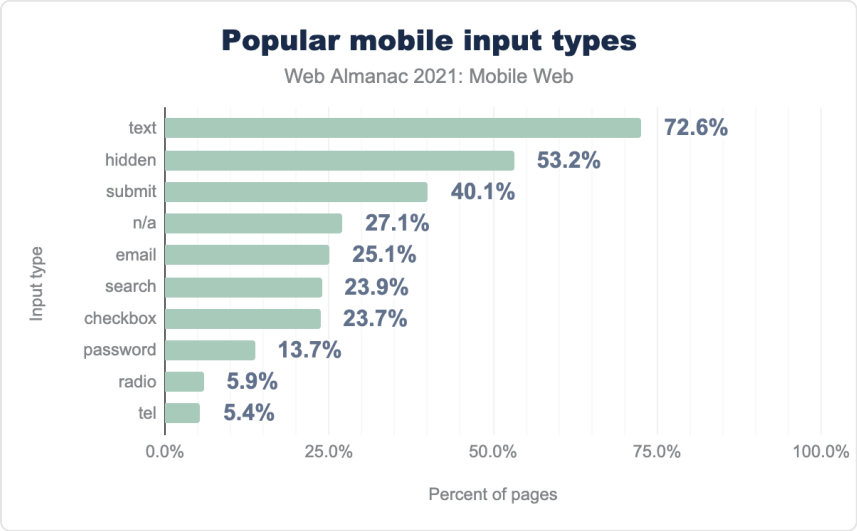


図13.17. モバイルで人気の入力タイプ。

入力によって作成されたインタラクティブ・コントロールの出現を追跡するには、`type`属性を探せばよいのです。`type`属性はinput要素がどのように動作するかを制御するため、もっとも重要です。テストしたサイトの70.9%で`type`属性の値が宣言されていました。

`type`属性がない場合、入力のデフォルトは`text`で、1行のテキストフィールドになります。input要素を使用しているページの分析では、27.1%のページが入力タイプを宣言しておらず、デフォルトの`text`文字列値を使用していました。

全ページのうち、72.6%が少なくとも1つの`text`型の入力を含んでいた。これはもっともよく使われている。

宣言された`text`値とフォールバック値を合わせると、input要素を使用しているサイトの99.7%がtext値を取得していることがわかります。

高度な入力タイプ

44.8%

図13.18. モバイルページで入力を使用している割合。

入力項目が1つ以上あるページのうち、44.8%が1つ以上の「高度な入力タイプ」を使用しています。高度な入力タイプには、`color`、`date`、`datetime-local`、`email`、`month`、`number`、`range`、`reset`、`search`、`tel`、`time`、`url`、`week`、`datalist`があります。

電話番号

電話番号の入力を求めるページは5.4%でした。モバイルユーザーにとって、アルファベットキーボードから数字キーボードへの移行は摩擦の大きいポイントです。電話番号を入力するページの62.6%は、入力フィールドに`type=tel`の値がないものを使用していました。

Eメール

入力タイプ`email`は、ユーザーに有効な電子メールアドレスを送信することを要求します。フォームに電子メール以外の値を入力すると、フォームの送信時にエラーが表示されます。

25.1%のページが、ユーザーに電子メールを求めるフィールドを少なくとも1つ含んでいました。

電子メールの収集は、ユーザージャーニーにおける重要なマイクロコンバージョンであることが多いため、最小限の摩擦で電子メールを収集することは、サイトにとってより高いコンバージョン率という利益をもたらします。このように明確なビジネス価値があるにもかかわらず、ユーザーの電子メールを尋ねるページの42%は、少なくとも1つのインスタンスでtype=emailの入力タイプが使用されていません。

検索条件入力

サイト内検索は、ユーザーを目的のコンテンツに誘導するための強力なツールです。検索入力は、機能的にはテキストと同じテキストフィールドです。検索入力フィールドとテキスト入力フィールドの主な違いは、ブラウザによる処理のされ方です。

検索入力タイプを使用すると、十字のアイコンが表示され、ユーザーは既存の検索テキストを素早く消去できます。また、最近のブラウザの多くは、検索クエリをドメイン間で保存します。検索タイプが指定されている場合、保存されているクエリを使用してフィールドをオートコンプリートできます。

テストしたページの23.9%に検索入力欄があった。このようなフィールドは、テキストまたは宣言されていない入力タイプを使用しているにもかかわらず存在する可能性があることに留意する必要があります。これは、17%のサイトが検索入力を使用していた2020年よりも若干増加しています。

ビジネス上の価値が入力タイプの採用に影響を与えているようだ。Eコマースサイトは、取引というビジネス目標を達成するために、ユーザーを目的の商品まで迅速に移動させるという既得権益を有している。

テストしたECサイトの43.3%がモバイル体験で検索入力を使用しています。興味深いことに、これはデスクトップクライアントで入力タイプを使用しているサイトの42.6%より高い数字です。

オートコンプリート

`オートコンプリート`属性は、フォームや入力がブラウザの自動入力機能でどのように動作するかをある程度制御できるようにするものです。完全に無効にする方法から、名前や住所などの自動入力のヒントを提供する方法まで、さまざまなオプションがあります。

モバイル端末でのテキストやデータの入力は、フルキーボードを備えた端末よりも一般的に面倒な作業となるため、自動入力はデスクトップユーザーよりもさらに便利で時間の節約になる機能です。Googleは、自動入力を使用するとフォーム送信が25% 増加することを発見しました[595]。

モバイルページの読み込みでは、24.8%のページが`オートコンプリート`属性を利用しており、デスクトップページの読み込みの27%よりも低い数値となっています。

HTTP Archiveのデータはホームページのみを対象としているため、チェックアウトや問い合わせなど、入力が必要と思われる場所での利用率はもっと高い可能性があります。しかし、間違いなくもっとも有用であるはずのモバイル体験での利用率が低いのは、残念なことかもしれません。

入力欄の結論

入力タイプの宣言は、摩擦を減らすために非常に重要です。入力要素が適切なタイプでマークアップされていれば、入力要素は異なるキーボードを促し、体験を向上させることができます。ユーザー体験への恩恵により、入力タイプを低リフトで採用することは有意義な投資となります。

電話や電子メールといった入力タイプの採用率が低いことは、モバイルウェブの入力欄がユビキタスであることを考えると、驚くべきことです。ビジネスゴールとユーザー体験との間のこのギャップは、モバイルウェブでのユーザー体験が重要であることを示している。Webサイトがもたらす最大のチャンスは、自社で機能を開発することではなく、モダンブラウザにネイティブ搭載されつつある機能性を活用することかもしれません。

595. *https://www.youtube.com/watch?v=m2a9hlUFRhg&t=1433s*

モバイルウェブにおけるアクセシビリティ

パンデミックによって、世界中の人間が友人、家族、地域社会から孤立せざるを得なくなった。また、COVID後の状態[596]により、障害に直面する人の数が増加しました。このシフトにより、対面でのサービスや商取引、コミュニケーションが阻害され、デジタル空間が新たなデフォルトとなることを余儀なくされたのです。

アクセシビリティの目標は、すべてのユーザーに対して機能と情報の平等性を提供するWeb体験を作り出すことです。アクセシビリティの実践により、低速のインターネット接続を使用している人や、データプランが限られている人、あるいは高価なデータプランを持っている人も情報を利用できるようになるため、モバイルのユーザーはアクセシビリティの恩恵を受けることができます。

ARIAロール

アクセスしやすいリッチなインターネットアプリケーション (ARIA) は、一般的に使用されるインタラクションやウィジェットを支援技術に渡すことができるように、HTMLを補足する属性のセットです。これらの属性は、検索エンジンがページの内容を理解するのに役立つ[597]ものでもあります。

サイトが支援技術を使用してアクセスされる場合、要素のARIAロールは、ユーザーがどのように対話できるかの情報を伝えます。

596. *https://www.hhs.gov/civil-rights/for-providers/civil-rights-covid19/guidance-long-covid-disability/index.html#footnote10_0ac8mdc*
597. *https://webaim.org/blog/web-accessibility-and-seo/*

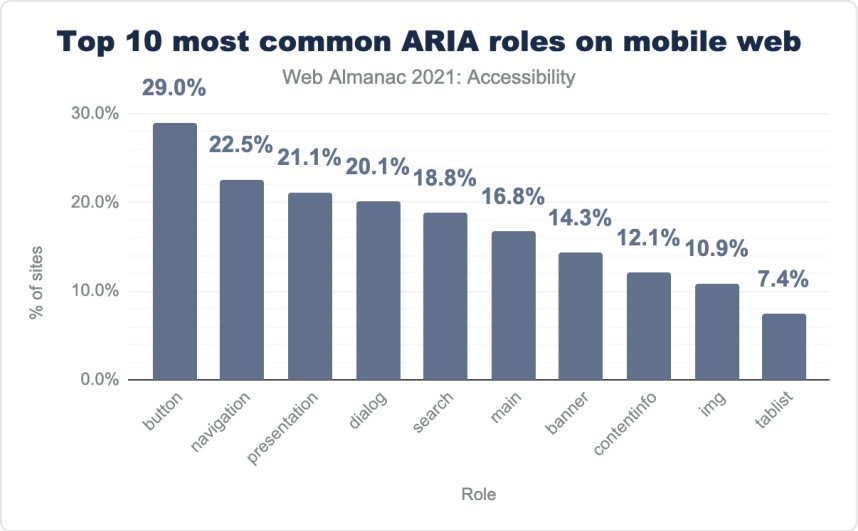


図13.19. ARIAの代表的な役割トップ10。

2021年にもっとも普及したARIAロールは`ボタン`で、29%のサイトに掲載されています。`ボタン`の役割は、ユーザーによってアクティブにされたとき反応を引き起こす、クリック可能な要素を示しています。

71%以上のモバイルサイトがウェブベースのフォームにインタラクティブ・コントロールを搭載している一方で、もっとも一般的に採用されているARIA属性であるaria-labelは、テストサイトの11.2%にしか搭載されていないことがわかりました。このアクセシビリティに特化した属性は、テキスト文字列で入力にラベルを付けるために使用されます。

カラーコントラスト

色覚異常や高齢者に多い低色感のユーザーには、色のコントラスト不足が影響します。十分なカラーコントラストは、コンテンツへの平等なアクセスを可能にし、ビジネス目標にポジティブな影響を与えることができます。Googleのケーススタディでは、eコマースサイトのEastpakは、コールトゥアクションボタンにテキスト色とその背景の十分なコントラストを使用すると、クリック率が20%増加した[598]と報告しています。

598. *https://www.thinkwithgoogle.com/intl/en-154/marketing-strategies/app-and-mobile/5-lessons-eastpak-learned-its-mobile-audience/*

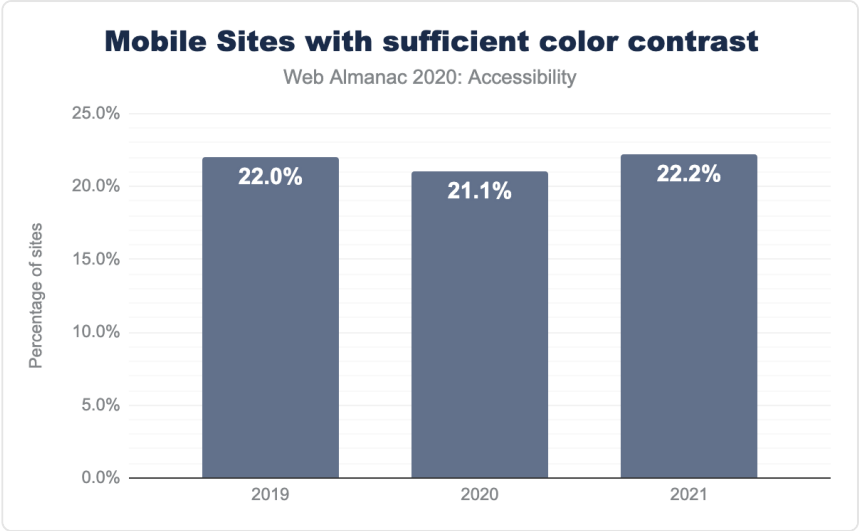


図13.20. 十分なカラーコントラストが確保されたモバイルサイト

コンバージョン率を高める可能性があるにもかかわらず、77.8%のサイトで十分なカラーコントラストが使用されているかどうかのLighthouse監査に不合格となりました。これは、前年比でわずかに改善されています。

タップ対象

タップ対象は、ユーザーの入力に反応する要素です。リンク、ボタン、フォームフィールドなど、さまざまなものが含まれます。

効果的なユーザー交流を実現するためには、タップターゲットのサイズとページ上の他のタップ対象との間隔は適切であることが必要です。インタラクティブ要素は、少なくとも48x48ピクセルで、他のインタラクティブ要素から少なくとも8ピクセルのパディングを取る必要があります。

39.3%

図13.21. モバイルサイトにおいて、十分な大きさのタップ対象を使用している割合。

全体では、テストしたサイトの39.3%が十分な大きさのモバイルタップ対象を使用していました。タップ対象の採用率は、ドメインランクのグループ間で一貫していました。これは、適切なサイズのタップ対象が36.3%であった2020年からわずかに増加したものです。

ズームとスケーリング

ビューポートメタ要素は、ユーザーのデバイスでページをどのようにレイアウトするかをブラウザへ通知するために重要です。また、`user-scalable="no"`や小さな`maximum-scale:`パラメーターを追加して、ユーザーがコンテンツを拡大することを完全に防止するか、制限するように設定することも可能です。モバイルデバイスでは、これは一般的にピンチズームです。

ズームインできないようにすることは、弱視のユーザーにとって問題であり、WCAG2.0ガイダンスに引っかかるもの[599]です。

残念なことに、モバイルページの29.4%がこの要件を満たせず、ズームを妨げるビューポートを含んでいました。（引用元：2020 Web Almanacアクセシビリティ[600]の章）

ドメインランキングで使用率を見ると、さらに状況は悪くなる。

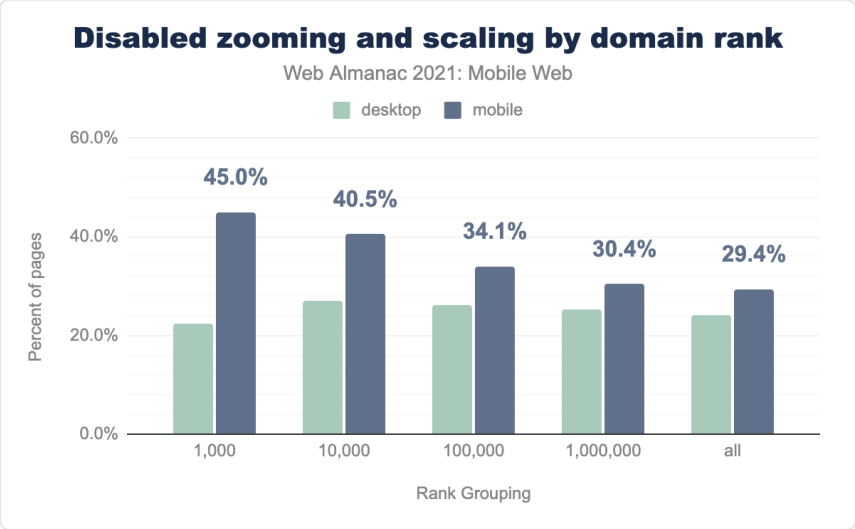


図13.22. ドメインランクによる拡大・縮小を無効化しました。

人気のあるサイトほどこれに該当する可能性が高く、全体として、より多くのユーザーが準拠していないモバイルサイトにアクセスしていることを意味します。

599. https://dequeuniversity.com/rules/axe/3.3/meta-viewport
600. https://almanac.httparchive.org/ja/2020/accessibility#ズームと拡大縮小

アクセシビリティの結論

ウェブがアクセスしやすくなれば、より多くの人々がウェブを知覚して理解してナビゲートして交流、貢献できます。ウェブアクセスの成長と必要性に対応するため、平等で包括的なアクセスを優先させなければならない。

ここで取り上げたのは、アクセシビリティのほんの一部です。ARIA、ズーム、カラーコントラストは最低限必要なものです。W3Cのウェブアクセシビリティ・イニシアチブ[601]の調査によると、世界人口の15%（10億人以上）が認知障害を抱えていることが分かっています。さらに多くの人が未登録のまま、あるいは人生のある時点で障害を持つようになり、サイトへのアクセスに影響を与えるかもしれません。アクセシビリティは、ごく一部の人のためのものではありません。

優れたアクセシビリティの実践が不十分なため、これらのユーザーには、人間として邪魔になる技術的な障害が発生するのです。というのも、このような潜在的なユーザーに対して適切に対応することは、明らかに商業的な機会だからです。

多くの法域において、アクセシビリティは単なるグッドプラクティスではありません。

> *昨年は米国障害者法に関連する訴訟が20%増加[602]しています。*
>
> *— Web Almanac 2021年 アクセシビリティの章*

モバイルウェブでのアクセシビリティについて詳しく知りたい方は、アクセシビリティの章をご覧ください。

モバイル検索エンジン最適化(SEO)

どんなウェブサイトでも、獲得は重要なステップです。最適化されたモバイルウェブサイトも、誰にも発見されず、訪問されなければ、悪いことに変わりはありません。

検索エンジン、ソーシャルメディア、他のウェブサイトからのリンクが主な発見経路となる可能性が高い。

検索エンジンは、多くのサイトにとって主要な獲得源であり、さらに多くのサイトにとって今もなお規模が大きいため、SEOはほとんどすべてのサイトにとって重要な検討事項となっています。

SEOではモバイルに特化した分野や懸念事項があります。

601. *https://www.w3.org/WAI/business-case/#increase-market-reach*
602. *https://info.usablenet.com/2020-report-on-digital-accessibility-lawsuits*

モバイルファーストインデックス

Googleは、ウェブへのアクセス方法の主流がモバイルであることを認識し、現在ではモバイルユーザエージェント[603]を持つウェブサイトを主にインデックスしています。2019年7月以降、すべての新規サイトがこの方法でインデックスされ、現在、ほとんどの既存サイトもモバイルファーストインデックスに移行しています。

つまり、デスクトップ端末にしか提供されないコンテンツやマークアップがある場合、googleはその部分をインデックスしなくなるのです。

モバイルフレンドリー

Google[604] と Bing[605] などの検索エンジンは、モバイルフレンドリーという何らかの概念を直接ランキングシグナルとして使っています。これは主に、コンテンツがビューポートに収まるか、テキストが読みやすいか、タップ対象が適度な大きさであるかを確認するためのテストです。

Googleはモバイルフレンドリーテスト[606]を提供しており、Bing[607] もあなたのページが合格かどうかを診断するのに役立つとされています。

これを実現する推奨される方法は、レスポンシブWebデザインを使用することです。web.devでは、素晴らしい学習リソース[608]を提供しています。

コアウェブバイタル＆ページエクスペリエンス

2021年7月15日、Googleはページ体験ランキングアップデート[609]を展開することを発表しました。これは、モバイルフレンドリーを含むいくつかの異なるシグナルで構成されており、主な新機能としてCore Web Vitals metrics[610]が追加されました。

モバイルウェブにとってとくに興味深いのは、コアウェブバイタルの部分がモバイル専用[611]で、これらの指標は今のところモバイル結果の一部のみを担っていますが、2022年2月[612]にデスクトップへの展開が予定されていることです。

モバイルフレンドリーやコアウェブバイタルのSEOにおける役割については、SEOの章で詳しく説明されています。

603. *https://developers.google.com/search/mobile-sites/mobile-first-indexing*
604. *https://developers.google.com/search/blog/2015/04/rolling-out-mobile-friendly-update*
605. *https://blogs.bing.com/webmaster/2015/11/12/mobile-friendly-test*
606. *https://search.google.com/test/mobile-friendly*
607. *https://www.bing.com/webmaster/tools/mobile-friendliness*
608. *https://web.dev/learn/design/*
609. *https://developers.google.com/search/blog/2021/04/more-details-page-experience*
610. *https://web.dev/i18n/ja/vitals/*
611. *https://support.google.com/webmasters/thread/104436075/core-web-vitals-page-experience-faqs-updated-march-2021*
612. *https://developers.google.com/search/blog/2021/11/bringing-page-experience-to-desktop*

モバイルパフォーマンス

モバイル端末は、デスクトップ端末に比べて消費電力が低く、ネットワーク接続も遅くて信頼性が、低い可能性があります。このような状況を考えると、パフォーマンスはより大きな課題であり、より優先度の高いものとなりえます。

読み込み性能

新しく獲得したユーザーの注意を引きつける、あるいはリピーターの注意を引きつけるには、サイトの重要なコンテンツを素早く見てもらうことから始まります。

最大のコンテンツフルペイント

最大のコンテンツフルペイント[613] (LCP) は、この経験を捉えるために設計された指標です（Core Web Vitalsの1つです）。ビューポートで最大の要素がレンダリングされるときの指標で、`<img>`、`<svg>`内の`<image>`、`<video>`（posterが設定されている場合）、背景画像付きのブロック要素、テキストブロックに限定されます。

LCPは2.5秒以下が好スコアとされています。

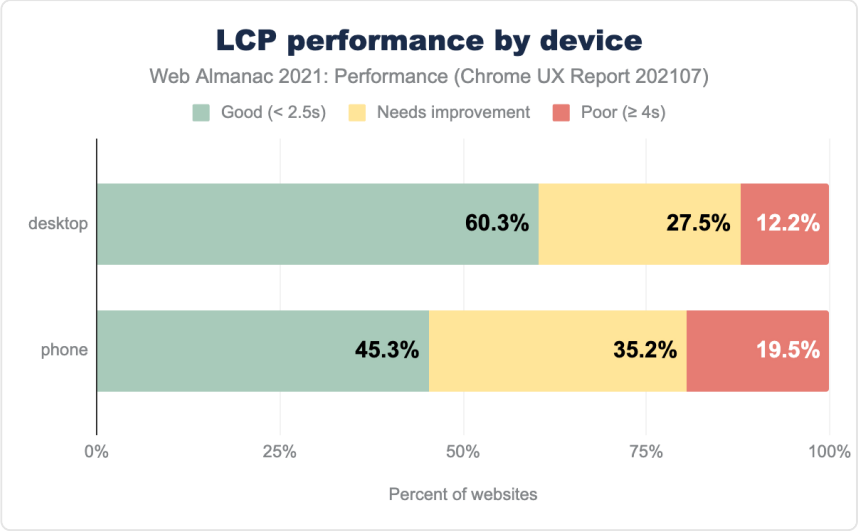


図13.23. LCPのデバイス別パフォーマンス。パフォーマンスの章のデータです。

613. https://web.dev/articles/lcp

このデータによると、CrUXのデータセットに記録されたモバイルページの読み込みのうち、2.5秒以下の目標を達成しているのはわずか45%で、デスクトップの60%よりはるかに低いことがわかります。

しかし、2.5秒以下の基準を満たしたのはモバイルページの読み込みの43%[614]であった2020年と比べると、わずかながら改善されていることがわかります。

モバイル層に対して良好なLCPスコアを達成するためには、より大きな課題があることは明らかですが、追い求める価値のある課題です。最近のボーダフォン[615]の調査では、LCP時間をわずか8%短縮しただけで31%ものコンバージョン増加につながったことが示されています。パフォーマンスは、収益に直接影響を与える可能性があります。

画像

CSSやJavaScriptなど、さまざまなアセットがモバイルのロードタイムに影響を与えます。しかし、大きな要因は画像です。

レスポンシブWebデザインのアプローチとして、デスクトップユーザーに適したネイティブサイズの画像を用意し、CSSで画面に合わせて拡大縮小することがよくあります。

適切な大きさの画像

56.6%

図13.24. モバイルページの読み込みで、適切なサイズの画像が表示された割合

2020年の58.8%から一歩後退しているのが悲しいところです。つまり、43.4%のモバイルユーザーが間違ったサイズの画像を取得していることになります。

レスポンシブ画像

画像もレスポンシブに提供[616]できます。`srcset`属性と`<picture>`要素で適切なサイズと適切なフォーマットの画像を指定し、画面とデバイスにもっとも適した画像をブラウザにダウンロードさせることができます。

614. *https://almanac.httparchive.org/ja/2020/performance#デバイス別LCP*
615. *https://web.dev/vodafone/*
616. *https://developer.mozilla.org/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images*

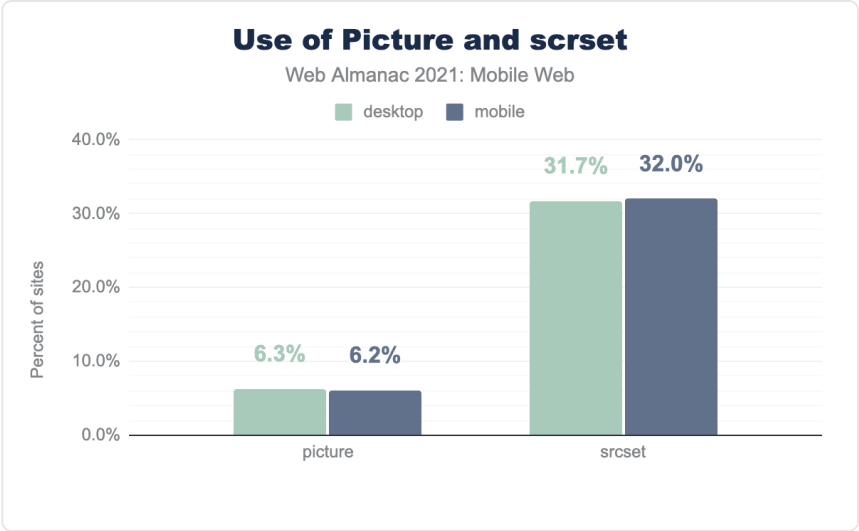


図13.25. レスポンシブな画像を提供するために、`<picture>` と `srcset` を使用します。

画像を含むモバイルページの読み込みで `<picture>` 要素が使われたのはわずか6.2%で、デスクトップよりわずかに低い数値でした。

画像を含むモバイルページのロードのうち、32%が `srcset` 属性を使用しています。この属性は `<picture>` 要素と `<img>` 要素の両方で使用できるため、ここでいくつかのクロスオーバーが、発生する可能性があることは言及しておく価値があるでしょう。

遅延ローディング

初期ビューポートにない画像を遅延ロードさせることは、リソースを表示可能なもののロードに集中させるための良い戦略です。Chrome、Opera、そして2021年9月からAndroid版Firefox（引用元：caniuse.com[617]）でサポートされているネイティブのlazy-load属性は、JavaScriptの回避策なしにこれを実現できます。

18.4%

図13.26. 画像を含むモバイルページのロードに `loading="lazy"` が使用されている。

これは、2020年のわずか4.1%から大きく飛躍しています。

617. https://caniuse.com/loading-lazy-attr

HTTP Archiveのネイティブ画像のレイジーローディングレポート[618]を見ると、とくに `<img>` タグに属性を使用することが、同じように目覚ましい成長を示していることがわかります。

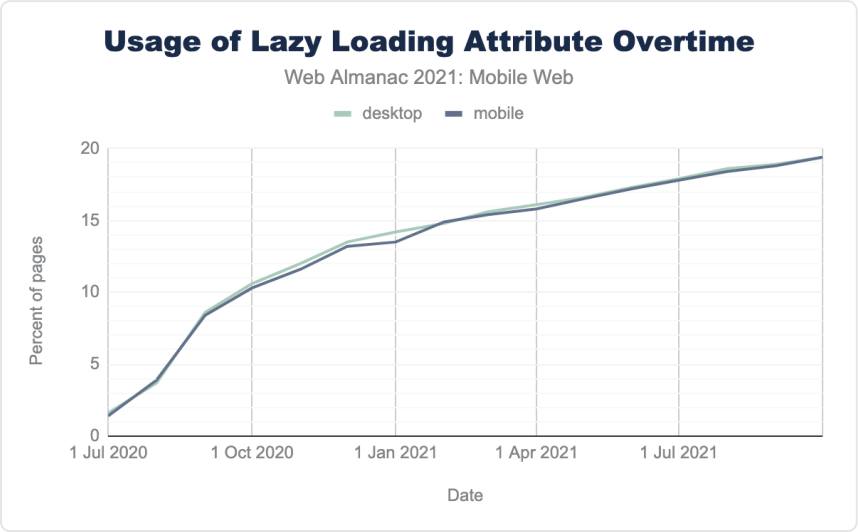


図13.27. 遅延ローディング属性の経時的な使用状況。

この成長の原動力は、WordPressの普及にあると考えられます（引用元：Rick Viscomiのツイッター[619]）。WordPressは、2020年8月11日に一般公開されたバージョン5.5で遅延ローディングに対応[620]しました。

また、遅延ローディングLCP候補[621]を誤って使用すると、パフォーマンスに悪影響を及ぼす可能性があることも述べておく必要があります。折り返し位置より下の画像にのみ `loading="lazy"` を適用するようにすることが、ベストプラクティスです。

画像の結論

今年、より多くのモバイルページの読み込みでサイズが、正しくない画像があったことは残念です。おそらく `<img>` 要素と比較した複雑さに基づいて、 `<picture>` の取り込みも低いままです。

しかし、 `loading="lazy"` 属性の採用は、わずか1年で大きな飛躍を遂げました。

618. *https://httparchive.org/reports/state-of-images#imgLazy*
619. *https://x.com/rick_viscomi/status/1344380340153016321?s=20*
620. *https://make.wordpress.org/core/2020/07/14/lazy-loading-images-in-5-5/*
621. *https://web.dev/articles/lcp-lazy-loading*

画像は依然としてWebの重要な要素であり、それはモバイルユーザーに対しても変わりません。もしあなたのサイトが、モバイルに適した画像を提供するために利用可能ないくつかのアプローチを活用していないのであれば、これを調査する時期が来ています。

レイアウトの安定性

一般的にフォームファクターが小さく、画面占有面積が限られているモバイルデバイスでは、予期せぬコンテンツの移り変わりがとくに気になるものです。

記事を読んでいて上に広告が表示され、今いる段落が下に飛んだり、フォントが読み込まれて目の前で変わったりするのは、不快でネガティブな体験です。

レイアウトの累積移動量

Core Web Vitalsの1つである レイアウトの累積移動量[622] (CLS) は、この種の要素のシフトの影響を把握するために設計されたメトリックです。

この指標は、インパクト・フラクションとディスタンス・フラクションを掛け合わせた計算です。インパクト・フラクションは画面の面積がどれだけ移動したかを、ディスタンス・フラクションは画面の面積がどれだけ移動したかを表しています。

CLSのスコアが0.1以下は良好、0.25以下は確かに改善、それ以上は不良と判断される

画面サイズが小さいほどずれが大きく、360×640pxの場合、この例のブロックではCLSスコアが0.22となる。

622. *https://web.dev/articles/cls*

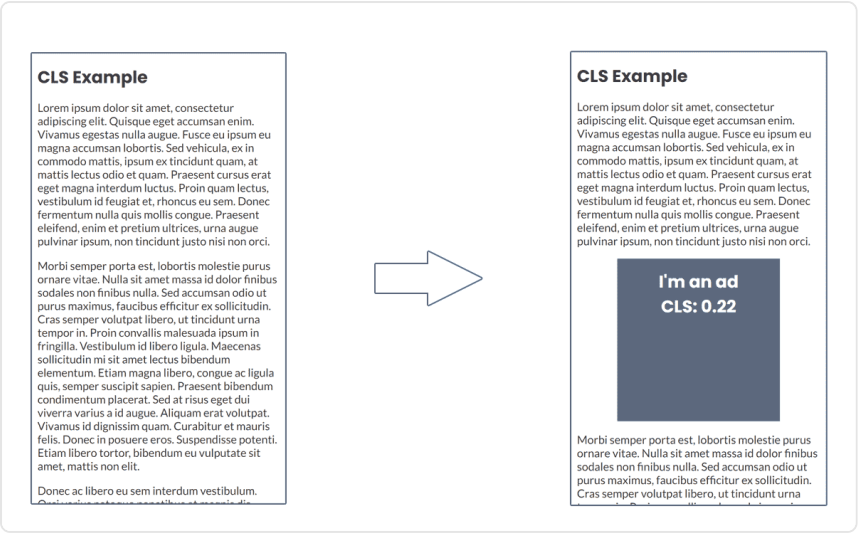


図13.28. モバイルサイズの画面上にCLSを引き起こす広告を表示した画面キャプチャモックアップ。

デスクトップ画面サイズでは、同じ要素が表示されてもCLSスコアは0.07にとどまります。

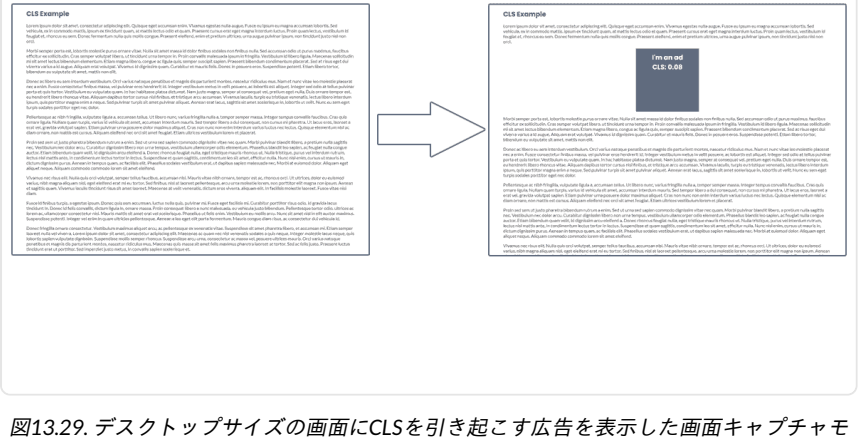


図13.29. デスクトップサイズの画面にCLSを引き起こす広告を表示した画面キャプチャモックアップ。

CrUXデータセットによると、モバイルページの読み込みの62%でCLSが、0.1以下であった

ことが示されています。

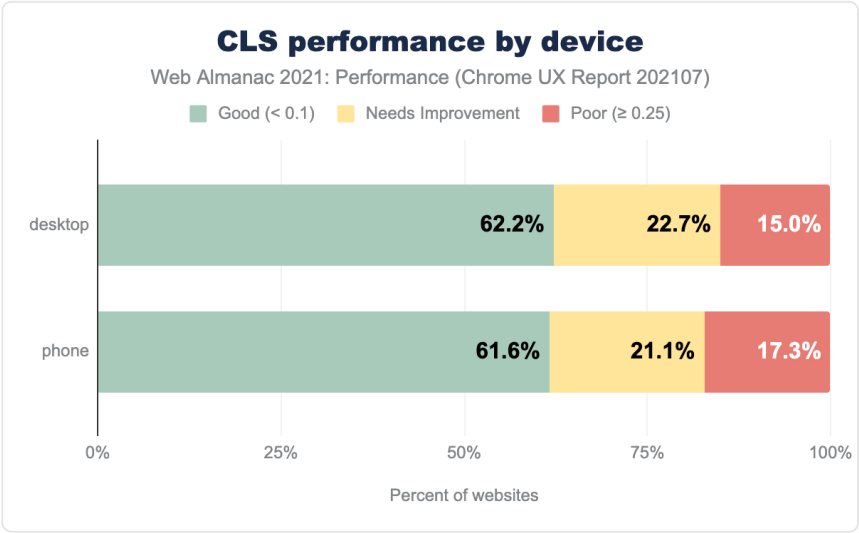


図13.30. デバイス別のCLS性能。

これは、昨年達成した43%を大きく上回るものですが直接の比較は難しく、2021年6月1日[623]に長寿命ページの体験をよりよく捉えるために指標を変更したので、このジャンプの一部はこれに起因する可能性があります。

ユーザーとの対話への対応

ユーザーがサイトを利用する際、何かをクリックしてから実際に何かが起こるまでの時間が長いと、ウェブサイトやアプリが重く感じられ、遅いと感じることがあります。このような入力と動作の間の遅延は多くの場合、重いJavaScriptの処理がメインスレッドをブロックし、その処理が完了するまで、ユーザーが発行したコマンドをブラウザが処理できない状態になること起因します。

一般にモバイル機器はデスクトップやノートパソコンに比べ、はるかに低電力であるため、その影響は増幅される可能性があります。

最初の入力遅延

最初の入力遅延[624] (FID) は、これを捉えるために設計された3つ目のCore Web Vitalメトリッ

623. https://web.dev/evolving-cls/
624. https://web.dev/articles/fid

クです。最初のインタラクション（タップまたは要素のクリック）が発生してからブラウザが、それが起こったという処理を開始するまでの時間を測定します。タップが引き金となった可能性のある処理にかかる時間は測定しません。

FIDスコアは100ms以下が良好、300ms以上が不良となります。

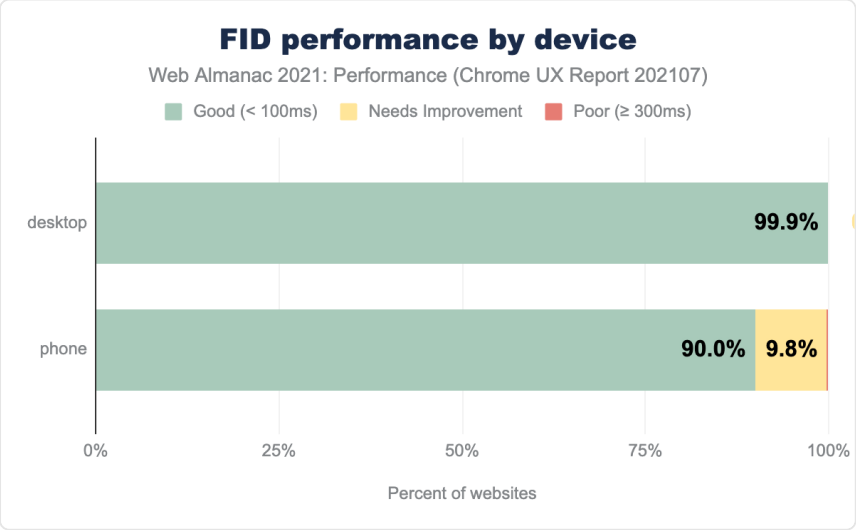


図13.31. デバイス別のFID性能。

心強いことに、CrUXデータセットのモバイルページロードの90%が良好なFIDスコアを獲得し、2020年の80%から上昇しました。

Chrome Speed Metricsチームは、新しい応答性メトリックについて、いくつかの計画を共有し、フィードバックを求めています[625]。

Core Web Vitals全般について知りたい場合は、パフォーマンスの章にCore Web Vitalsの詳細がたくさん載っています。

サービスワーカー

サービスワーカー[626]は、モバイルデバイスに限らずオフライン機能を追加したり、キャッシュからWebアプリケーションへの読み込みをよりよく制御したりすることで、ユニークな機能を発揮します。この2つの機能は、接続性の低下や全喪失に遭遇する可能性が高いモバイルユーザーにとって、しばしばより適切なものとなります。

625. https://web.dev/responsiveness/
626. https://developer.mozilla.org/docs/Web/API/Service_Worker_API

14.8%の事業所がサービス従事者を登録、2020年の0.9%から大きく上昇

サービスワーカーとPWA（プログレッシブWebアプリ）について詳しく知りたい方は、PWAの章をご覧ください。

モバイル性能の結論

全体として2020年に向けて一歩進んだ性能になっており、とくにレイアウトの安定性が大きく向上しています。

また、`loading="lazy"`の使用率が目覚ましく伸びていることや、サービスワーカーの普及など、良い兆候も見られます。開発者がこれらを受け入れているという事実は、パフォーマンスが真剣に受け止められていることを示すポジティブなサインです。

しかし、最大のコンテンツフルペイントの改善や画像のハンドリングは、他の分野よりも開発者が苦労しているようです。うまくいけば、next/image[627]のようなツールやライブラリが提供されます。 Next.jsのフレームワークや、WordPressのような人気のあるCMSでの採用は、開発者がこれらのペインポイントを克服するのに役立ちます。

結論

2021年には、明確な「モバイルウェブ」という認識は時代遅れになっています。

複数のデータソースによると、モバイルはユーザーがデジタルコンテンツとインタラクションするための多くの方法の1つであり、実際、デジタルインタラクションの大半を占めているようです。

多くのユーザーにとって、モバイルデバイスはウェブに接するための主要な、あるいは唯一の手段です。にもかかわらず、方法論、パフォーマンス戦略、アクセシビリティの原則の採用や、ブラウザでサポートされる機能の採用は低いままです。

いくつかの分野では大きな進展があり、ほとんどの業績評価指標は2020年のデータよりも改善されています。しかし、まだまだ成長の余地がある分野も残っています。

アクセシビリティは、より多くの努力と時間を費やすことが望まれる分野であり、画像のベストプラクティスはまだいくらか残っているのです。

モバイルユーザー分野の成長と規模が拡大し続ける中、多くの業界にとって、もはやモバイルウェブをサポートするためのビジネスケースを作る必要はなく、完全に受け入れ、2021年に開発者が利用できる多くのツールやテクニックを活用することが重要となっています。

627. *https://nextjs.org/docs/api-reference/next/image*

著者



Jamie Indigo

@Jammer_Volts @not-a-robot.com fellowhuman1101 jamie-indigo

https://not-a-robot.com

Jamie Indigoはロボットではなく、ボットをしゃべるんです。Deepcrawl[628]のテクニカルSEOコンサルタントとして、検索エンジンがどのようにウェブをクロールし、レンダリングし、インデックスを作成するかを研究しています。野生のJavaScriptフレームワークを手なずけ、レンダリング戦略を最適化することが大好きです。仕事以外では、ホラー映画、グラフィック小説、Dungeons & Dragonsが好きです。



Dave Smart

@https://seocommunity.social/@dwsmart @tamethebots.com dwsmart davewsmart

https://tamethebots.com

Dave Smart は、Tame the Bots[629] の開発者であり、検索エンジンの技術コンサルタントです。ツールの構築やモダンウェブの実験が好きで、ライブの最前線にいることもしばしば。



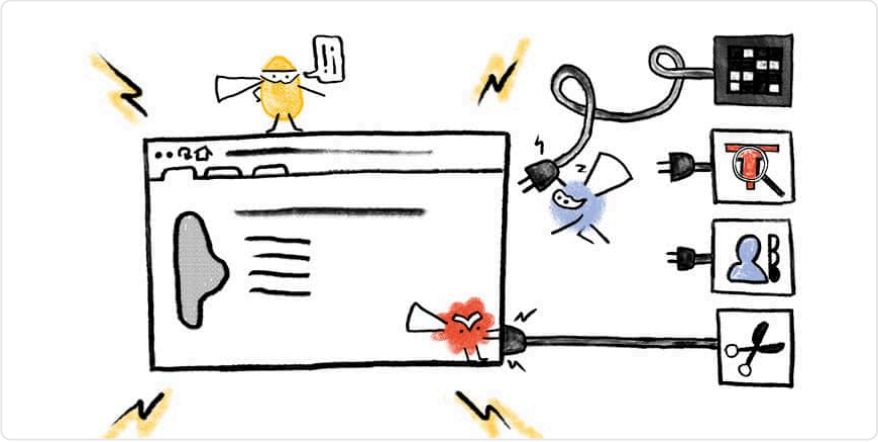
Ashley Berman Hale

ashleyish

Ashley Berman Hale は、Deepcrawl[630] のテクニカル SEO およびプロフェッショナルサービス担当副社長です。植物、動物、そして小さな人間のママでもあります。アシュリーは、地元のローラーダービーリーグでプレーし、新しいSEOを指導しています。

628. *https://www.deepcrawl.com*
629. *https://tamethebots.com*
630. *https://www.deepcrawl.com*

部 II 章 14
ケイパビリティ



***Christian Liebel* によって書かれた。**
Thomas Steiner と Hemanth HM によってレビュ。
Thomas Steiner による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

ケイパビリティは、Webアプリケーションのまったく新しいユースケースを解放する新しいWebプラットフォームAPIです。それらの新しいAPIは、Webベースのアプリケーションモデルであるプログレッシブ・ウェブ・アプリケーション(PWA)に不可欠なものです。PWAとは、ユーザーが自分のシステムにインストールできるWebアプリのことです。PWAはオフラインでも動作し、素早く起動する。基本的なオペレーティングシステムと統合するために、PWAはウェブプラットフォームAPIのみを使用できます。ブラウザはすでにいくつかの低レベルの機能をウェブに公開していますが（たとえば、geolocation[631], gamepad[632], またはwebcam[633]アクセス）、多くのAPIはまだ欠けていたり使い勝手が悪かったり（たとえば、ファイルシステムやクリップボードへのアクセスを）していました。

631. *https://developer.mozilla.org/docs/Web/API/Geolocation_API*
632. *https://developer.mozilla.org/docs/Web/API/Gamepad_API*
633. *https://developer.mozilla.org/docs/Web/API/MediaDevices/getUserMedia*

プロジェクト・フグ

ケイパビリティ・プロジェクト[634]（コードネームフグ）は、Microsoft、Intel、Google、およびその他のChromium貢献者による共同作業です。これは、新しい強力なWebプラットフォームAPIを安全かつプライバシーを保護する方法で設計・実装することにより、プラットフォーム固有のアプリケーションとWebアプリケーションの間のギャップを埋めようとしています（プライバシーの章もご覧ください）。機能がより多くのユースケースを解放するにつれて、最終的にWebへの移行を行うための新しいアプリケーションカテゴリ全体への道を開きます（例：IDE、画像エディター、オフィスアプリケーションなど）。

> *プロジェクト・フグは、Webの機能のギャップを埋め、新しいクラスのアプリケーションをWeb上で実行できるようにする取り組みです。プロジェクト・フグが提供するAPIは、セキュリティ、低摩擦、クロスプラットフォーム配信というWebの中核的な利点を維持しながら、Web上での新しい体験を可能にします。Project Fugu API の提案はすべてオープンで標準化されたトラックで行われます。*
>
> *– Webケイパビリティチーム*[635]

この2年間、フグチームは、デスクトップ生産性アプリケーションのための機能とハードウェア関連のAPIに焦点を合わせてきました。この章では、いくつかの新しい機能を簡単に紹介し、デスクトップとモバイルのさまざまなウェブサイトがそれらをどの程度使用しているかを分析します。ケイパビリティは、アプリのようなウェブサイトにとってとくに興味深いものであるため、その相対的な使用率は比較的低くなっています。このため、この章ではウェブサイトの絶対数が使用されています。各ケイパビリティには、それを利用するデモ用のWebサイトやアプリが用意されています。

方法論

この章では、HTTP Archiveのデータセットを使用します。セキュリティ上の理由から、一部のAPIは機能するためにユーザーのジェスチャー（クリックやキープレスなど）を必要とします。HTTP Archiveのクローラーは、実行時にそれらのAPIを検出することをサポートしていないため、代わりにウェブサイトのソースコードを静的に解析しています。たとえば、正規表現 `/navigator\.share\s*\(/g` をウェブサイトのソースコードにマッチさせ、*Web Share API* を使用しているかどうかを判断します。

この方法は、APIの実際の使用状況を測定していないため、完全に正確というわけではありません。開発者は、異なる構文を使用してAPIを呼び出したり、簡略化したコードで作業し

634. https://www.chromium.org/teams/web-capabilities-fugu
635. https://www.chromium.org/teams/web-capabilities-fugu

たりする可能性があるからです。しかし、この方法で十分な概観を得ることができるはずです。このソースファイル[636]には、サポートされる30の機能の正確な正規表現が記載されています。

本章の使用データはすべて2021年7月のクロールを基準としています。生データは能力2021結果シート[637]に掲載されています。

本章でよりよく使われる2つのAPIについては、Chrome Platform Statusの追加データを紹介します。このデータは、本章の出版前の過去12か月間でAPIの使用状況がどのように変化したかを示しています。

提供するAPIの状況

ここで紹介するAPIのほとんどは、いわゆるインキュベーションであることに注意してください。とくに断りのない限り、それらは（まだ）W3C勧告、すなわち公式のWeb標準ではありません。その代わり、これらのAPIは、ブラウザベンダーと開発者が新機能について議論できるウェブプラットフォームインキュベータ コミュニティ グループ（WICG）で作業されています。

すでにいくつかのブラウザで出荷されているAPIもあれば、Chromiumベースのものでしか利用できないAPIもあります。これらのブラウザには、Google Chrome、Microsoft Edge、Opera、Brave、またはSamsung Internetが含まれます。Chromiumベースのブラウザのベンダーは、特定の機能を無効にすることができるため、ChromiumベースのすべてのブラウザですべてのAPIが使用できるとは限らないことに注意してください。また、一部の機能は、ブラウザの設定でフラグを有効にしたあとでのみ利用できる場合があります。

非同期クリップボードAPI

*非同期クリップボードAPI*を使用すると、クリップボードからのデータの読み取りやクリップボードへのデータの書き込みを行うことができます。非同期のため、UIをブロックすることなく、画像を縮小しながら貼り付けるような使い方が可能です。これまでクリップボードの操作に使われていた `document.execCommand()` のような性能の低いAPIを置き換えることができます。

書き込みアクセス

非同期クリップボードAPIは、データをクリップボードにコピーするための2つのメソッドを

636. https://github.com/HTTPArchive/legacy.httparchive.org/blob/master/custom_metrics/fugu-apis.js
637. https://docs.google.com/spreadsheets/d/1b4moteB9EiLYkH1Ln9qfi1tnU-E4N2UQ87uayWytDKw/

提供します。省略記法の `writeText()` はプレーンテキストを引数にとり、ブラウザはそれをクリップボードにコピーします。`write()` メソッドは、任意のデータを含むことができるクリップボード項目の配列を受け取ります。ブラウザは、特定のデータ形式のみを実装することを決めることができます。Clipboard API仕様は、プレーンテキスト、HTML、URIリスト、PNG画像など、ブラウザが最低限サポートしなければならない必須データ型の リストを指定しています[638]。

```
await navigator.clipboard.writeText('hello world');

const blob = new Blob(['hello world'], { type: 'text/plain' });
await navigator.clipboard.write([
  new ClipboardItem({
    [blob.type]: blob,
  }),
]);
```

読み取りアクセス

クリップボードにデータをコピーするのと同様に、クリップボードからデータを貼り付けて戻すには2つの方法があります。まず、クリップボードからプレーンテキストを返す `readText()` という別のショートハンドメソッドを紹介します。`read()` メソッドを用いると、ブラウザがサポートするデータ形式でクリップボード内のすべてのアイテムにアクセスできます。

```
const item = await navigator.clipboard.readText();
const items = await navigator.clipboard.read();
```

ブラウザは、ウェブサイトがクリップボードの内容にアクセスすることを許可する前に、プライバシー上の理由から許可プロンプトを表示したり、別のUIを表示したりすることがあります。非同期クリップボードAPIは、Chrome、Edge、Safariで利用できます（current browser support for the Async Clipboard API[639]）。Firefoxは `writeText()` メソッドのみをサポートしています。

638. https://www.w3.org/TR/clipboard-apis/#mandatory-data-types-x
639. https://caniuse.com/async-clipboard

560,359

図14.1. Async Clipboard APIを使用したデスクトップWebサイト。

デスクトップ560,359（8.91%）、モバイル618,062（8.25%）のサイトにおいて、Async Clipboard API (`writeText()` method) はもっとも使われているフグAPIの1つです。`write()` メソッドは、デスクトップ1,180サイト、モバイル1,227サイトで使用されています。例として、商用サイトClipping Magic[640]では、AIアルゴリズムの助けを借りて画像の背景を除去できます。クリップボードから画像を貼り付けるだけで、ウェブサイトがその背景を削除してくれます。

このAPIの使用率が高いのは、おそらくYouTubeの埋め込み動画に含まれるスクリプトが関係していると思われます。ビデオプレーヤーでユーザーが “copy link” ボタンをクリックすると、`writeText()` メソッドが呼び出されます。

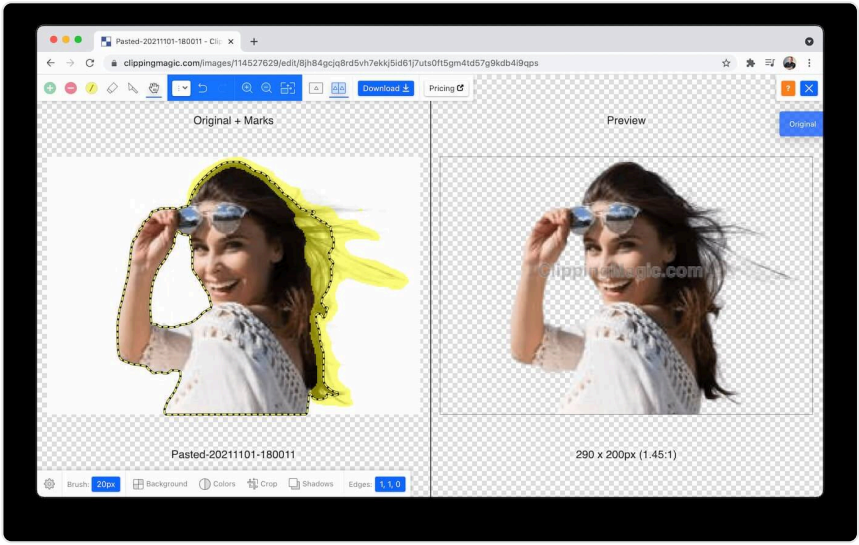


図14.2. クリッピングマジックは、非同期クリップボードAPIで貼り付けられた画像の背景を人工知能で除去するものです。

ここ数ヶ月、APIの利用が低水準で急激に増加しています。2020年11月には、`read()` メソッドは全ページロードの0.00032%しかアクティブになっていなかったが、2021年10月には0.002921%まで使用率が指数関数的に増加した。`write()` メソッドは、同期間に0.000674

640. https://clippingmagic.com/

から0.001601パーセントに増加しました。

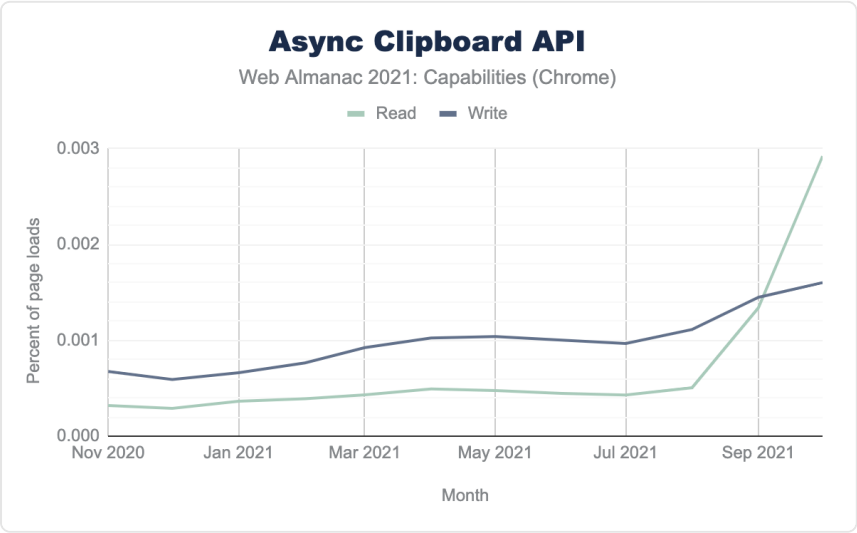


図14.3. ChromeでAsync Clipboard APIを使用してページを読み込む割合。（提供元：非同期クリップボード読み込み[641]、非同期クリップボード書き込み[642]）

ファイルシステムアクセスAPI

次に生産性関連のAPIとして、*ファイルシステムアクセスAPI*を紹介します。Webアプリは、すでにファイルを扱うことができました[643]。`<input type="file">` では、ユーザーはファイルピッカーを使って1つまたは複数のファイルを開くことができます。また、`<a download>` により、ダウンロードフォルダーにファイルを保存できます。File System Access APIは、さらなるユースケースをサポートする。ディレクトリを開いたり、変更したり、ユーザーが指定した場所にファイルを保存したり、ユーザーが開いたファイルを上書きしたりします。また、`IndexedDB` にファイルハンドルを持続させて、ページを再読み込みしたあとでも（パーミッション制限された）アクセスを継続できるようにすることも可能です。とくに、APIはファイルシステムへのランダムアクセスを許可しておらず、特定のシステムフォルダーはデフォルトでブロックされている。

641. *https://chromestatus.com/metrics/feature/timeline/popularity/2369*
642. *https://chromestatus.com/metrics/feature/timeline/popularity/2370*
643. *https://web.dev/browser-fs-access/#the-traditional-way-of-dealing-with-files*

書き込みアクセス

グローバルな `window` オブジェクトに対して `showSaveFilePicker()` メソッドを呼び出すと、ブラウザはオペレーティングシステムのファイルピッカーを表示するようになります。このメソッドはオプションのオプションオブジェクトを受け取り、保存を許可するファイルタイプ (`types` , default: all types) と、ユーザーが “accept all” オプションでこのフィルターを無効にできるかどうか (`excludeAcceptAllOption` , default: `false`) を指定できます。

ユーザーがローカルファイルシステムからファイルを正常に選択すると、そのハンドルを受け取ることができます。そのハンドルに対して `createWritable()` メソッドを実行すると、ストリームライターにアクセスできます。以下の例では、このライターはテキスト `hello world` をファイルに書き込み、その後ファイルを閉じます。

```
const handle = await window.showSaveFilePicker({
  types: [{
    description: 'PNG files',
    accept: { 'image/png': ['.png'] }
  }],
  excludeAcceptAllOption: true
});
const writable = await handle.createWritable();
await writable.write('hello world');
await writable.close();
```

アクセスを読み取る

オープンファイルピッカーを表示するには、グローバルな `window` オブジェクトの `showOpenFilePicker()` メソッドを呼び出してください。このメソッドには、オプションのオプションオブジェクトも渡されます。このオブジェクトには、上記と同じプロパティ (`types` , `excludeAcceptAllOption`) が設定されます。さらに、ユーザーが1つのファイルを選択するか、複数のファイルを選択するか (`multiple` , default: `false`) を指定できます。

ユーザーは複数のファイルを選択する可能性があるため、ファイルハンドルの配列を受け取ります。配列の再構築式 `[handle]` を用いると、最初に選択されたファイルのハンドルが配列の最初の要素として受け取られます。ファイルハンドルに対して `getFile()` メソッドを呼び出すと、 `File` オブジェクトが得られ、ファイルのバイナリデータへアクセスできる

ようになります。`text()` メソッドを呼び出すと、オープンされたファイルからプレーンテキストを受け取ることができます。

```
const [handle] = await window.showOpenFilePicker({
  multiple: false
});
const blob = await handle.getFile();
const text = await blob.text();
console.log(text);
```

ディレクトリーを開く

最後に、このAPIでは、ウェブアプリ（統合開発環境など）がディレクトリ全体のハンドルを取得できます。このハンドルを使って、開いたディレクトリ内で既存のファイルやフォルダーを作成、更新、削除できます。今回のメソッドは `showDirectoryPicker()` という名前になっています。

```
const handle = await window.showDirectoryPicker();
```

ファイルシステムアクセスAPIは、Chromiumベースのブラウザとデスクトップシステムでのみ利用可能です（ファイルシステムアクセスAPIの現在のブラウザサポート[644]）。幸いなことに、ウェブプラットフォームは、モバイルデバイスや他のブラウザーでも同様の機能を提供するために、前述のフォールバックアプローチを提供しています。開発者はGoogleが開発したライブラリ browser-fs-access[645] を使うことができます。このライブラリはファイルシステムアクセスAPIがあればそれを使い、なければ別の実装にフォールバックします。

29

図14.4. ファイルシステムアクセスAPIを使用したデスクトップWebサイト。

HTTP Archiveに登録されているデスクトップ向け6,286,373サイト、モバイル向け7,491,840サイトのうち、ファイルシステムアクセスAPIが使用されているのはデスクトップ向け29サイト、モバイル向け23サイトです。それらのサイトの例としては、手描き風の図をスケッチ

644. https://caniuse.com/native-filesystem-api
645. https://github.com/GoogleChromeLabs/browser-fs-access

してディスクに保存できる画像エディターエクスカリッドロー[646]があります。また、画像編集ソフトCorelDRAWのWeb版であるCorelDRAW.app[647]もその一例です。

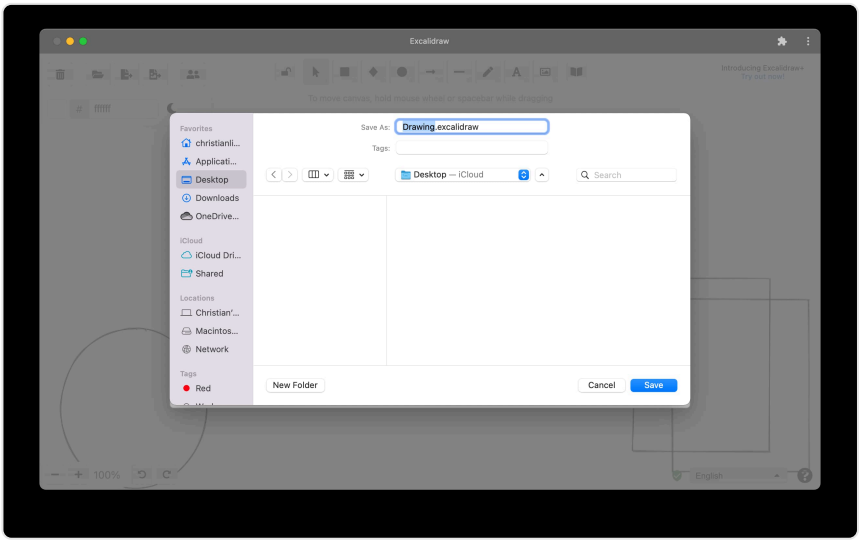


図14.5. Excalidraw PWAは、ファイルシステムアクセスAPIを使用して、内蔵の保存ダイアログからローカルファイルシステムに画像を保存します。

WebシェアAPI

*Web*シェア*API*を使用すると、WebサイトやWebアプリケーションのテキスト、URL、ファイルを、メールクライアントやメッセンジャーなど、他のアプリケーションと共有できます。そのためには、`navigator.share()`メソッドを呼び出してください。他のアプリケーションと共有するためのデータをオブジェクトとして取得します。ブラウザは内蔵の共有シートを開き、ユーザーはそこからターゲットアプリケーションを選択できます。このメソッドは、コンテンツが正常に共有された場合に解決するプロミスを返し、そうでない場合は拒否されます。

```
await navigator.share({
  files: picturesArray,
  title: 'Holiday pictures',
```

646. https://excalidraw.com/
647. https://coreldraw.app/

```
    text: 'Our holiday in the French Alps'
  })
```

WebシェアAPIは、iOSおよびmacOSのSafari、WindowsおよびChrome OSのChromeおよびEdgeでサポートされています（現在のWeb Share API対応ブラウザ[648]）。現在、Webアプリケーションワーキンググループで 草案[649] が作成されています。これは、W3C勧告になるための軌道の最初の段階の1つです。



図14.6. Web Share APIを利用したデスクトップWebサイト。

デスクトップ566,049サイト（9.00%）、モバイル642,507サイト（8.58%）で、WebシェアAPIはもっとも利用されているフグのAPIです。たとえば、PaintZアプリのベータ版[650]では、保存ダイアログを通じて、ローカルにインストールされている別のアプリケーションと図面を共有できます。

このAPIの使用率が高いのは、YouTubeの埋め込み動画に含まれるスクリプトが関係していると思われる。デバイス上でWebシェアAPIが利用可能な場合、ユーザーが動画プレーヤーの「Share」ボタンをクリックすると、このAPIが実行されます。

648. *https://caniuse.com/web-share*
649. *https://www.w3.org/TR/web-share/*
650. *https://beta.paintz.app/*

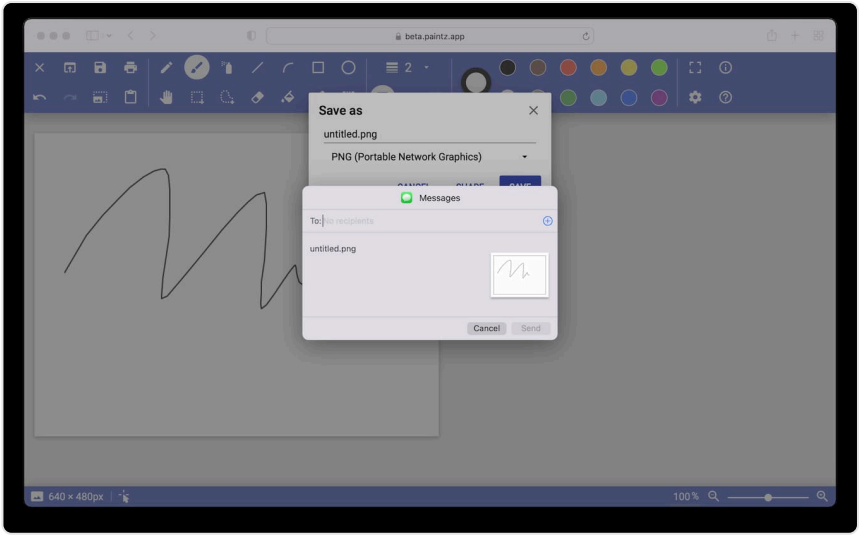


図14.7. PaintZのベータ版では、WebシェアAPIを使用してローカルアプリケーションとドローイングを共有しています。

ここ数カ月、WebシェアAPIの利用が全体的に増えています。Chrome Platform Statusのデータでは、2020年11月に全ページロードの0.0097%でAPIが呼び出されてから、2021年10月には0.0136%まで、かなり直線的に伸びていることがわかります。

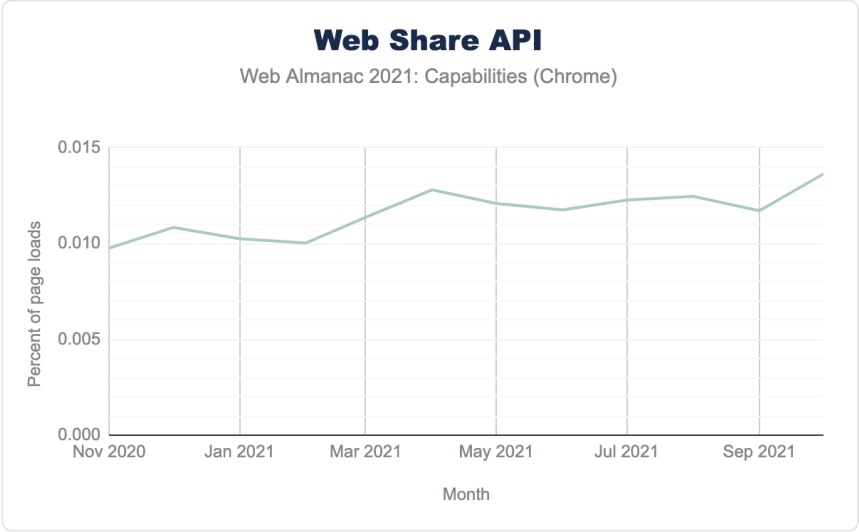


図14.8. WebシェアAPIを使用したChromeのページロードの割合。（提供元[451]）

URLハンドラーと宣言型リンクキャプチャ

この章で説明する生産性に関連する最後の2つの機能は、*URLハンドラー*と*宣言型リンクキャプチャ*で、オペレーティングシステムとさらに深く統合するための追加メソッドです。

URLの取り扱い

URLハンドラー[652]の助けを借りて、PWAはインストール時に特定のURLスキームに対するハンドラーとして自身を登録できます。たとえば `https://*.example.com` のハンドラーとして登録できます。ユーザーがこのスキームに一致するURLを開くと、新しいブラウザタブではなく、インストールされたPWAが開きます。URLハンドリングは、*Webアプリケーションマニフェスト*というWebアプリケーションのメタデータ[653]を含むファイルを拡張したものです。URLスキームに登録するには、マニフェストに `url_handlers` プロパティを追加する必要があります。このプロパティは `origin` プロパティを持つオブジェクトを含む配列を取ります。

```
{
  "url_handlers": [{
    "origin": "https://*.example.com"
  }]
}
```

ウェブアプリのオリジン以外のオリジンを登録したい場合は、所有権を確認する[654]必要があります。この機能は比較的初期の段階にあり、デスクトップのChromeとEdgeにのみ対応しています。URLハンドリングは現在、オリジン・トライアル[655]として提供されています。つまり、この機能はまだ一般的に利用できるものではありません。その代わり、開発者はこの実験的なAPIを使うために、まずOrigin Trialトークンに登録し、このトークンをウェブサイトと一緒に配信して、この機能を使うことを選択する必要があります。詳細は、Web開発者向けOriginトライアルガイド[656]に掲載されています。



図14.9. デスクトップのWebサイトでは、URLハンドリングを使用しています。

651. *https://chromestatus.com/metrics/feature/timeline/popularity/1501*
652. *https://web.dev/pwa-url-handler/*
653. *https://developer.mozilla.org/docs/Web/Manifest*
654. *https://web.dev/pwa-url-handler/#the-web-app-origin-association-file*
655. *https://developer.chrome.com/blog/origin-trials*
656. *https://github.com/GoogleChrome/OriginTrials/blob/gh-pages/developer-guide.md*

デスクトップ44サイト、モバイル41サイトがURLハンドリングを利用しています。たとえば、Pinterest PWAはインストール時にPinterestの異なるオリジン（例：`*.pinterest.com`と`*.pinterest.de`）のURLハンドラーとして自身を登録する。

宣言型リンクキャプチャ

宣言型リンクキャプチャ[657]の助けを借りて、ユーザーがPWAを開いたときの動作をさらに制御できます。たとえば、オフィス系アプリケーションでは、新しい文書を作成するために別のウィンドウを開きたいが、音楽プレーヤーでは1つのウィンドウを開いたままにしておきたい。そこで、宣言型リンクキャプチャでは、3種類のモードを定義しています。

1. `none`はリンクをまったくキャプチャしません（デフォルト）
2. `new-client`はPWAの新しいウィンドウを開きます。
3. `existing-client-navigate`は既存のクライアントを新しいURLに移動させるか、クライアントが存在しない場合は新しいウィンドウを開きます。

宣言型リンクキャプチャもWebアプリケーションマニフェストの拡張機能です。これを使用するには、マニフェストに`capture_links`プロパティを追加する必要があります。このプロパティは、上記の3つのモードに一致する文字列または文字列の配列を取ります。配列を使用する場合、ブラウザが特定のモードをサポートしない場合は、次のエントリにフォールバックします。

```
{
  "capture_links": [
    "existing-client-navigate",
    "new-client",
    "none"
  ]
}
```



図14.10. デスクトップ型Webサイトでは、宣言型リンクキャプチャを使用しています。

このケイパビリティも初期段階です。Chrome OSのみの対応となります。現在、デスクトップ36サイト、モバイル11サイトがこの機能を利用しており、たとえば、元素の周期表を表

657. https://web.dev/declarative-link-capturing/

示するPWAであるPeriodex[658]は、その例です。このアプリでは、上のリストにあるように `capture_links` という設定を使っています。つまり、サポートされていればブラウザは既存のウィンドウを再利用し、そうでなければ新しいウィンドウを開き、サポートされていなければ通常通りに動作します。

ハードウェアAPI

次のケイパビリティは、ハードウェア関連のAPIに焦点を当てたものです。Chromiumベースのブラウザでは、USB、ブルートゥース、シリアルデバイスなど、ハードウェアインターフェイスにアクセスするためのAPIが多数用意されています。さらに、汎用センサーAPIを使えば、デバイスのセンサーから読み取ることができます。このセクションで説明するすべての機能は、Chromiumベースのブラウザと、それぞれのハードウェアインタフェースまたはセンサーが存在するシステムでのみ利用可能です。

Web USB API

Web USB API により、開発者はドライバーやサードパーティのアプリケーションを使用せずにUSBデバイスへアクセスできます。たとえば、ファームウェアのアップデートを行う場合、開発者はそれぞれのプラットフォームに特化したアプリとして実装しなければならないが、この機能は興味深い。USBデバイスにアクセスするには、`navigator.usb.requestDevice()` メソッドを呼び出す必要があります。接続されているすべてのUSBデバイスのリストに対するフィルターを定義するオブジェクトを受け取ります。少なくとも `vendorId` を指定する必要があります。ブラウザにはデバイスピッカーが表示され、ユーザがマッチするデバイスを選択できます。そこから、デバイスセッションを開始できます。

```
try {
  const device = await navigator.usb.requestDevice({
    filters: [{ vendorId: 0x8086 }]
  });
  console.log(device.productName);
  console.log(device.manufacturerName);
} catch (err) {
  console.log(err);
}
```

658. https://periodex.co/

182

図14.11. デスクトップ用Webサイトでは、Web USBを使用します。

このAPIは、Chromiumベースのブラウザーではバージョン61から一般で利用できるようになっています (Web USB APIをサポートする現在のブラウザ[659])。182のデスクトップと155のモバイルサイトがこのAPIを使用しており、たとえば、AndroidやiOSデバイスの画面をミラーリングできるPWA Vysor[660] は、すべてコンピューターに追加のソフトウェアをインストールせずに使用できます。

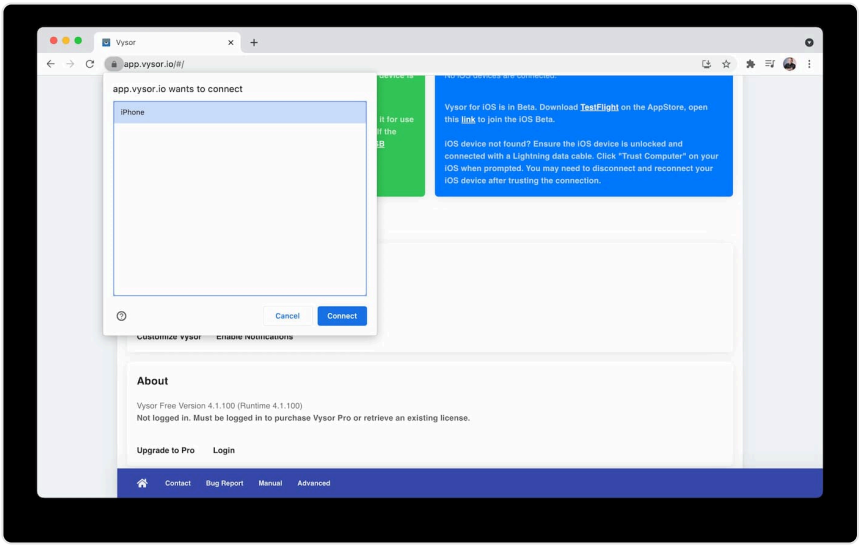


図14.12. Vysor PWAは、Web USBを利用してUSB機器と接続し、その画面の内容をデスクトップに投影できます。

Webブルートゥース API

*Webブルートゥース API*を使用すると、一般属性プロファイル (GATT)[661]を使用して近くのブルートゥース・ローエナジーデバイスと通信することが可能です。一致するデバイスを見つけるには、`navigator.bluetooth.requestDevice()` メソッドを呼び出します。次の例では、ブルートゥースデバイスのリストは、バッテリーサービスを提供しているかどうかでフィルタリングされています。ブラウザにデバイスピッカーが表示され、ユーザーがブルート

659. https://caniuse.com/webusb
660. https://app.vysor.io/#/
661. https://www.bluetooth.com/bluetooth-resources/intro-to-bluetooth-gap-gatt/

ゥースデバイスを選択できます。その後、リモートデバイスに接続し、データを収集できます。

```
try {
  const device = await navigator.bluetooth.requestDevice({
    filters: [{ services: ['battery_service'] }]
  });
  console.log(device.name);
} catch (err) {
  console.log(err);
}
```



図14.13. WebブルートゥースAPIを使用したデスクトップWebサイト。

このAPIは、Chrome OS、Android、macOS、WindowsのChromiumベースのブラウザで、バージョン56から一般的に利用できます（現在のWebブルートゥースAPIの対応ブラウザ[662]）。Linuxでは、フラグの後ろでAPIが提供されています。71のデスクトップと45のモバイルサイトがこの機能を利用しています。たとえば、家庭のビール醸造家を対象としたBrewfather[663] PWAでは、ビールのレシピをブルートゥース対応の醸造システムに無線で送信できます。ここでもまた、サードパーティのソフトウェアをインストールすることなく、すべてが行われます。

662. https://caniuse.com/web-bluetooth
663. https://web.brewfather.app/

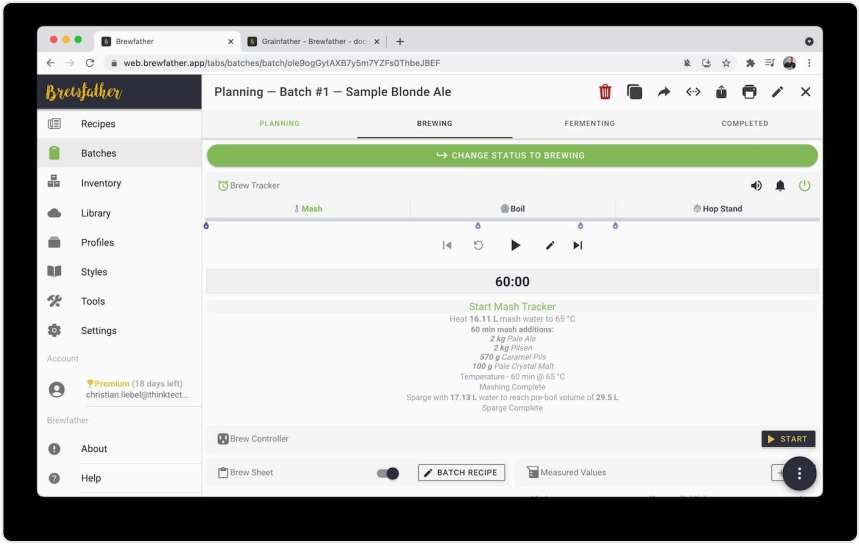


図14.14. Brewfatherアプリは、Webブルートゥースを利用してレシピをBrewコントローラーに送信します。

WebシリアルAPI

*Webシリアル*APIは、マイコンなどのシリアルデバイスと接続するためのものです。そのためには、`navigator.serial.requestPort()` メソッドを呼び出します。オプションでデバイスリストをフィルタリングするためのメソッドを渡すことができます。ブラウザにはデバイスピッカーが表示され、ユーザーがデバイスを選択できます。次に、ポートの `open()` メソッドを呼び出して、接続を開くことができます。

```
try {
  const port = await navigator.serial.requestPort();
  await port.open({ baudRate: 9600 });
} catch (err) {
  console.log(err);
}
```

15

図14.15. WebシリアルAPIを利用したデスクトップWebサイト。

この機能は、2021年3月にChromium 89で提供された比較的新しいものです（現在のWebシリアルAPIのブラウザサポート[664]）。現在、デスクトップ15サイト、モバイル14サイトでWebシリアルAPIが利用されており、ブラウザ上でArduinoやESPマイコン用のプログラムを開発できるDuino App[665]もその1つです。これらのプログラムは、リモートサーバーでコンパイルされ、WebシリアルAPIを介して接続されたボードにアップロードされます。

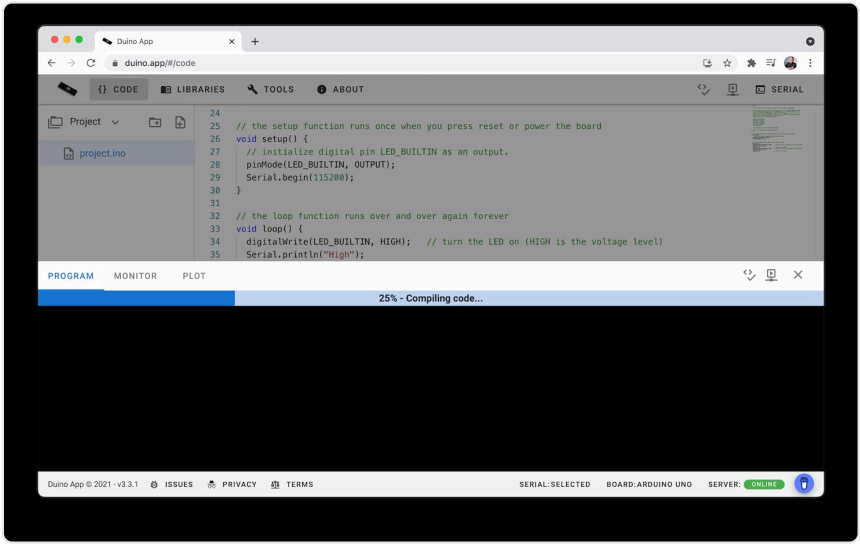


図14.16. Duinoアプリは、Webシリアルを使用してArduinoマイコンにプログラムをアップロードするWebベースのIDEです。

汎用センサーAPI

最後に*汎用センサーAPI*により、加速度計、ジャイロスコープ、または方向センサーなど、デバイスのセンサーからセンサーデータを読み込むことができます。センサーにアクセスするには、センサークラス、たとえば `Accelerometer` の新しいインスタンスを作成する。コンストラクターは、要求された周波数を持つ設定オブジェクトを受け取ります。`onreading` と `onerror` イベントにアタッチすることで、センサーの値が更新されたとき

664. *https://caniuse.com/web-serial*
665. *https://duino.app/*

やエラーが発生したときに、それぞれ通知を受けることができます。最後に、`start()` メソッドを呼び出して、読み取りを開始する必要があります。

```
try {
  const accelerometer = new Accelerometer({ frequency: 10 });
  accelerometer.onerror = (event) => {
    console.log(event);
  };
  accelerometer.onreading = (e) => {
    console.log(e);
  };
  accelerometer.start();
} catch (err) {
  console.log(err);
}
```

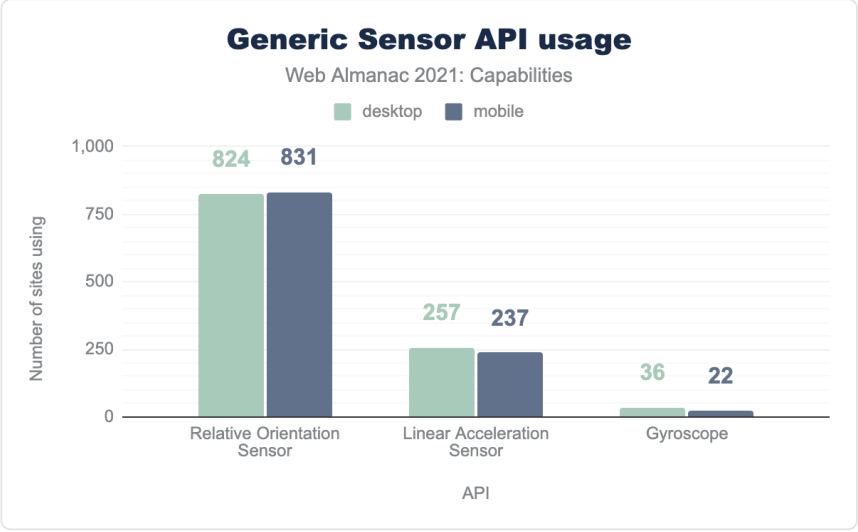


図14.17. デスクトップおよびモバイルウェブサイトでの汎用センサーAPIの使用状況。

この機能は、バージョン67以降のChromiumブラウザでサポートされています（汎用センサーAPIをサポートする現在のブラウザ[666]）。相対方位センサーはデスクトップ824サイト、モバイル831サイト、直線加速度センサーはデスクトップ257サイト、モバイル237サイト、ジ

666. https://caniuse.com/mdn-api_sensor

ャイロセンサーはデスクトップ36サイト、モバイル22サイトで利用されています。この3つを使ったアプリケーションの例としてVDO.Ninja[667]、旧OBS Ninjaがあります。OBSなどの映像放送ソフトとリモートで接続するためのソフトウェアです。このアプリを使うことで、接続した放送ソフトがデバイスからセンサーデータを読み取ることができます。たとえば、バーチャルリアリティコンテンツを配信する際に、スマートフォンの動きを取り込むことができるようになります。Fuguのコントリビューターであるintelは、汎用センサーAPIの追加のdemos[668] を提供しています。

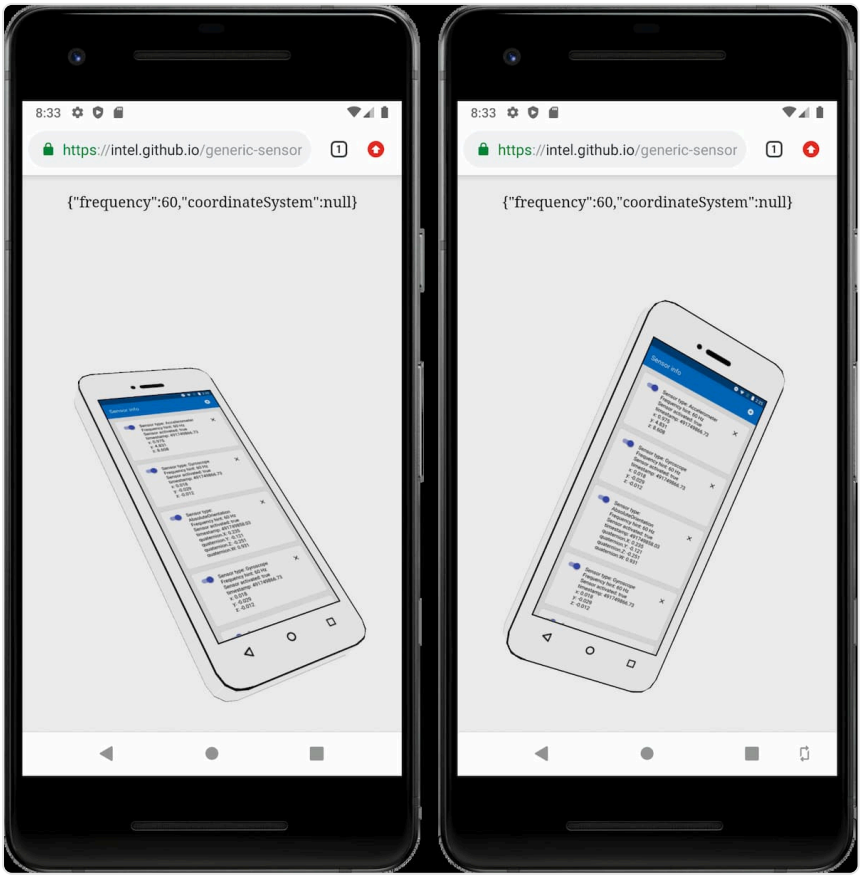


図14.18. 汎用センサーAPIを使用することで、デバイスの向きに応じて3Dモデルを回転させることができます。

667. https://obs.ninja/
668. https://intel.github.io/generic-sensor-demos/

もっとも多くの機能を使用しているサイト

また、HTTP Archiveのデータセットから、もっとも多くの機能を使用しているウェブサイトを特定する解析も行いました。この検出スクリプトは、合計30個のフグAPIを識別できます。そこで、もっとも多くのフグAPIを使用しているWebサイトに賞を贈りましょう。盛り上がってきましたね〜。

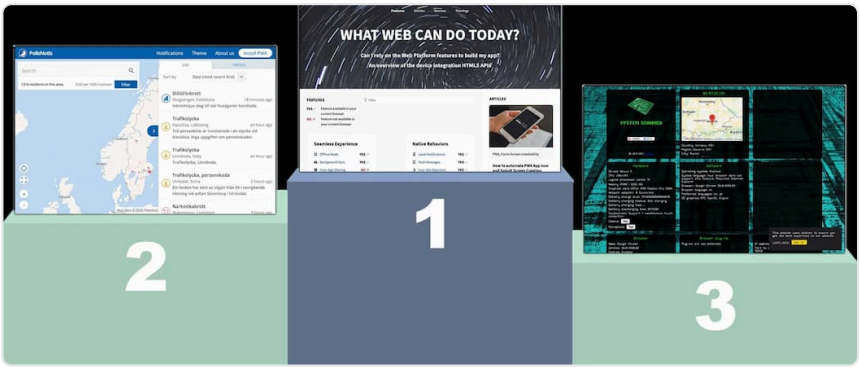


図14.19. フグのAPIをもっとも多く利用している3つのWebサイト。

1. 1位はwhatwebcando.today[669]で、28の機能を使用しています。HTML5のデバイス統合APIを、各機能のライブデモを提供することで、紹介しています。当然ながら、使用されているAPIの数は非常に多い。結果セットでは、whatpwacando.today[670]という同様のサイトがPWA機能を紹介し、8つのAPIを使用していることがわかります。
2. 次点は、スウェーデンの警察告知を表示するPolisNotis[671] PWAです。PWA関連のリンクをクリックすると必ず新しいウィンドウが開くように定義する宣言型リンクキャプチャAPIなど、10個のAPIを使用しています。WebシェアAPIはソースコードで使用されていますが、共有機能はUIに公開されていません。また、Badging APIを利用して、新しいお知らせがあった場合にアプリアイコンを通じてユーザーに注意を促しています。
3. 僅差の3位は、9つのAPIを使用するウェブサイトシステムスキャナー[672]です。汎用センサーAPIで提供されるセンサー情報を含む、ブラウザが公開するシステム情報の概要を表示します。
4. 8つのサイトが8つのフグAPIを使用しています。その1つが、前述のExcalidraw[673]で、手書き風の図面を作成するためのオンラインドローイングツールです。従来の生産性アプリとして、新機能の恩恵を受けています。

669. *https://whatwebcando.today/*
670. *https://whatpwacando.today/*
671. *https://polisnotis.se/*
672. *https://system-scanner.net/*
673. *https://excalidraw.com/*

結果セットの中には、Discourse[674]に基づいたインターネット・フォーラムであるウェブサイトもあります。合計8つのフグAPIに対応したフォーラムソフトです。Discourseベースのフォーラムがインストール可能で、とくに未読通知数を表示するBadging APIをサポートしています。

この結果には、APIを積極的に利用していないサイトも含まれています。たとえば、理論的には機能にアクセスできるライブラリコードを配布しているサイトがあります。ユーザーのブラウザを特定するために、フグAPIの存在を確認するサイトもあります。

結論

開発者がより多くのユースケースを利用できるようにすることで、ウェブを前進させることができるのです。この章で示すように、開発者は新しいWebプラットフォームAPIを使用して、強力なアプリケーションを構築します。プラットフォーム固有のものとは対照的に、これらのアプリケーションは、必ずしもシステムにインストールする必要はなく、動作するためにサードパーティのランタイムやプラグインを追加で必要としません。強力なブラウザを実行できるプラットフォームであれば、どのような環境でも動作するのです。

このコンセプトが機能している一例として、Visual Studio Codeがあります。このアプリケーションは常にWebベースでしたが、それでもElectronのようなプラットフォーム固有のアプリケーションラッパーに依存していました。ファイルシステムアクセスAPIなどの機能により、Microsoftは2021年10月にブラウザアプリケーション（vscode.dev[675]）として公開することができました。デバッグやターミナルアクセスを除いて、ほとんどすべての機能がここで動作します（まだ、この機能はありません！）。

また、Adobe Photoshop[676]も2021年10月にウェブアプリケーションとしてリリースされました[677]。Photoshopは、ここで紹介した機能のいくつかと、WebAssemblyを使用して、既存のコードをWebに移行できます。ベクターベースの対応するIllustratorは、現在クローズドベータ版として提供されており、後日リリースされる予定です。最初のエディションはまだ限られた機能しかありませんが、アドビはすでにこれにとどまらず、webへのさらなる拡大を計画している[678]ことを発表しています。

このように、ケイパビリティプロジェクトは、アプリケーションの全カテゴリーを最終的にWebに移行させる道を開くものです。

674. *https://www.discourse.org/*
675. *https://vscode.dev*
676. *https://photoshop.adobe.com*
677. *https://web.dev/ps-on-the-web/*
678. *https://web.dev/ps-on-the-web/#what's-next-for-adobe-on-the-web*

著者



Christian Liebel

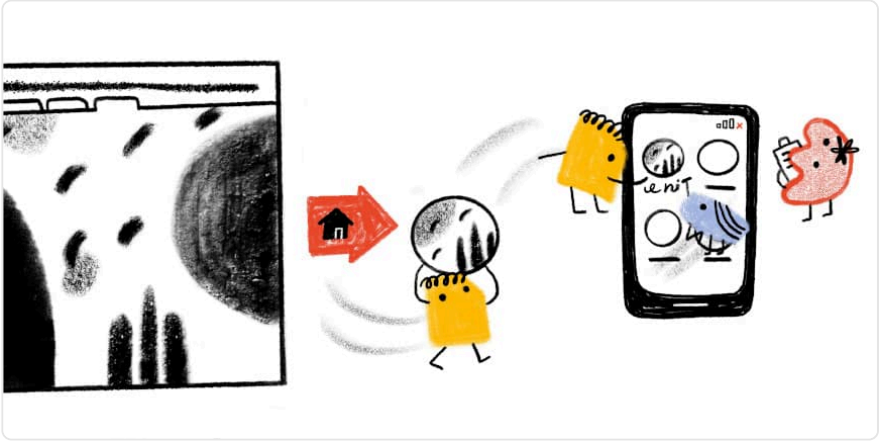
@christianliebel @https://mastodon.cloud/@christianliebel christianliebel

christianliebel https://christianliebel.com

Christian Liebel は、Thinktecture[679] のコンサルタントで、さまざまなビジネス分野のクライアントが優れた Web アプリケーションを実装できるように支援しています。Microsoft MVPのデベロッパー技術部門、Google GDEのWeb/ケイパビリティとAngularが取得しており、W3C Webアプリケーションワーキンググループに参加しています。

679. *https://thinktecture.com*

部 II 章 15
PWA



Demian Renzulli によって書かれた。
Barry Pollard、 Maxim Salnikov、 Jeff Posnick、 André Cipriani Bandarra、 Kai Hollberg、 Hemanth HM、 Pascal Schilp と Adriana Jara によってレビュ。
Barry Pollard と Demian Renzulli による分析。
Rick Viscomi 編集。
Sakae Kotaro によって翻訳された。

序章

Frances Berriman[680] と Alex Russell[681] が、ネイティブアプリと同じように没入できるウェブアプリのビジョンを示す“プログレッシブ・ウェブ・アプリ”(PWA)[682] という言葉を作ってから6年が経ちます。 このような体験が従来のWebサイトと異なる点として、次のような属性が挙げられた。

- レスポンシブ
- サービスワーカーで順次強化
- アプリのようなインタラクションを持つ

680. *https://x.com/phae*
681. *https://x.com/slightlylate*
682. *https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/*

- 新鮮
- 安全
- ディスカバブル
- リエンジニアリング可能
- リンク可能

ここ数年、ウェブプラットフォームは進化を続け、ウェブアプリとOS固有の体験との間のギャップを減らし、開発者はより豊かな機能と新しいエンゲージメントの方法をユーザーに提供することができるようになりました。

とはいえ、何がPWAなのかを明確に線引きすることはまだ困難です。専門家の中には、シェルとコンテンツアプリケーションモデル[683]の特徴である「アプリらしい」体験を生み出すことを重視する人もいれば、サービスワーカーとウェブアプリのマニフェストを持ち、オフライン体験やその他の高機能を提供するなど特定のコンポーネントや動作に重きを置く人もいます。

今年のPWA編では、サービスワーカーとその関連APIの使用、Webアプリのマニフェスト、PWAを構築するためのもっとも人気のあるライブラリやツールなど、PWAの測定可能なすべての側面に焦点を当てます。PWAは、これらの機能のすべてまたは一部を使用できます。ここでは、ウェブエコシステムにおけるこれらの技術の浸透度合いを知るために、各コンポーネントとAPIの採用レベルについて見ていきます。

注: この章では、一般的に使われているサービスワーカー関連のAPIに主に焦点を当てます。さらに最先端のAPIについては、ケイパビリティの章を必ずご覧ください。

サービスワーカー

サービスワーカー[684]（2014年12月導入）は、PWAの中核をなすコンポーネントの1つです。ネットワークプロキシとして機能し、オフライン、プッシュ通知、バックグラウンド処理など、「アプリらしい」体験に特徴的な機能を可能にします。

サービスワーカーが広く採用されるには時間がかかりましたが、現在では ほとんどの主要なブラウザ[685]でサポートされています。しかし、これはすべてのサービスワーカー機能がブラウザ間で動作することを意味するものではありません。たとえば、ネットワーク プロキシなどの中核的な機能のほとんどは利用可能ですが、 `Push` などのAPIは WebKitではまだ利用で

683. *https://developers.google.com/web/fundamentals/architecture/app-shell*
684. *https://developer.mozilla.org/docs/Web/API/Service_Worker_API*
685. *https://caniuse.com/serviceworkers*

きません[686]。

サービスワーカーの利用状況

2021年には、測定方法にもよりますが、1.22%から3.22%のサイトがサービスワーカーを使用していると推測されます。今年は、次に説明する理由から、3.22%にもっとも近い値を採用することにしました。



図15.1. モバイルサイトにおいてサービスワーカーを利用している割合。

サービスワーカーが使われているかどうかを測定するのは、案外簡単ではありません。たとえば、Lighthouseは1.5% を検出しますが、この定義には サービス ワーカーの使用だけでなく、いくつかの追加チェック[687]が加えられているので下限と見なすことができます。Chrome自体は、サービス ワーカーを使用しているサイトの割合は1.22%です[688]。これは、私たちが把握できていない理由でLighthouseよりも奇妙なほど少なくなっています。

今年のPWA編では、新しい測定基準[689]を作成し、測定技術を更新しました。たとえば、サービスワーカー登録[690]の呼び出しがあるか、サービスワーカー固有のメソッド、ライブラリ、イベントを使用しているかなど、いくつかのサービスワーカーの特徴をチェックするヒューリスティックを現在使用しています。

収集したデータから、デスクトップサイトの約3.05%、モバイルサイトの約3.22%がサービスワーカー機能を利用していることがわかり、昨年の章[691]で計測した値（デスクトップ0.88%、モバイル0.87%）よりサービスワーカーの利用率が高いかもしれないことがわかります。

モバイルとデスクトップでサービスワーカーを登録しているサイトが3%強というのは少ない数字だと思われるかもしれませんが、これがWebトラフィックにどう反映されるのでしょうか。

Chromeプラットフォームの状況[692] は、Chromeブラウザから取得した使用統計情報を提供します。この統計によると、2021年7月にページロードの19.26%[693] を支配するのはサービスワーカーだそうです。昨年の測定値16.6%[694] と比較すると、サービスワーカーが制御するペー

686. *https://caniuse.com/push-api*
687. *https://web.dev/service-worker*
688. *https://httparchive.org/reports/progressive-web-apps#swControlledPages*
689. *https://github.com/HTTPArchive/legacy.httparchive.org/blob/master/custom_metrics/pwa.js*
690. *https://developer.mozilla.org/docs/Web/API/ServiceWorkerRegistration*
691. *https://almanac.httparchive.org/ja/2020/pwa#サービスワーカーの利用状況*
692. *https://www.chromestatus.com/features*
693. *https://www.chromestatus.com/metrics/feature/timeline/popularity/990*
694. *https://almanac.httparchive.org/ja/2020/pwa#service-worker-usage*

ジ負荷は年間12%増加していることになります。

19.26%

図15.2. サービスワーカーを登録したページのページビューの割合。（提供元：Chromeプラットフォームの状況[695]）

また、約3%のサイトがWebトラフィックの約19%を占めていることをどう説明すればよいのでしょうか。直感的には、トラフィックが多いサイトほど、サービスワーカーを導入する理由があると考えるかもしれません。ユーザー数が多いということは、ユーザーがさまざまなデバイスやコネクティビティからサイトにアクセスする可能性があるため、パフォーマンス上のメリットや信頼性を提供するAPIを採用するインセンティブが高くなるのです。また、これらの企業はネイティブアプリを持っていることが多いので、サービスワーカーを介して高度な機能を実装し、プラットフォーム間のUXギャップを埋める理由がより多くあります。次のデータは、その仮定を証明するのに役立ちます。

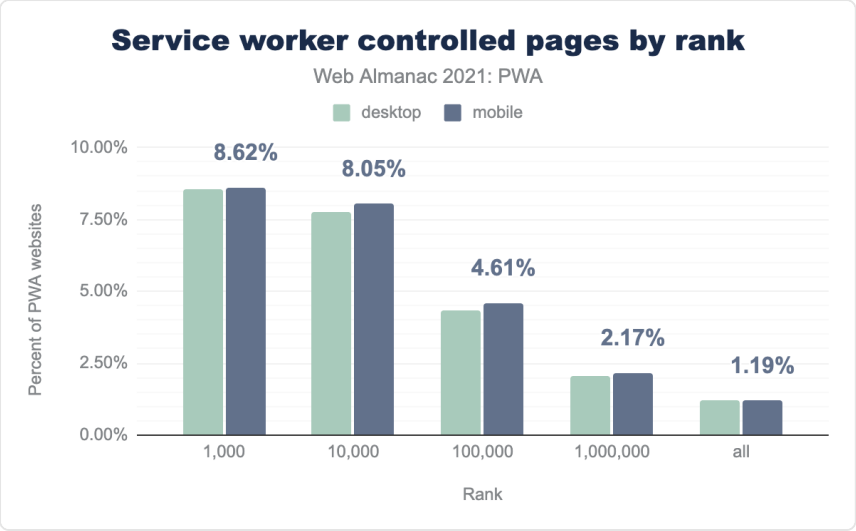


図15.3. サービスワーカーがランク別に管理したページ。

上位1,000サイトを計測すると、8.62%のサイトがサービスワーカーを利用しています。分析対象のサイト数を広げると、全体の割合は減少に転じています。これは、人気のあるサイトほど、サービスワーカーや高度な機能といった機能を利用する傾向があることを示しています。

695. https://www.chromestatus.com/metrics/feature/timeline/popularity/990

サービスワーカーの機能

このセクションでは、もっとも一般的なPWAタスク（オフライン、プッシュ通知、バックグラウンド処理など）に対するさまざまなサービスワーカー機能（イベント[696]、プロパティ[697]、メソッド[698]）の採用について分析します。

サービスワーカーのイベント

ServiceWorkerGlobalScope[699] インターフェイスは、サービスワーカーのグローバルな実行コンテキストを表し、異なる events[700] によって管理されます。イベントリスナーやサービスワーカーのプロパティを介して、2つの方法でそれらをリッスンできます。

たとえば、サービスワーカーで `install` イベントをリスニングする方法を2つ紹介します。

```
// Via event listener:
this.addEventListener('install', function(event) {
  // …
});

// Via properties:
this.oninstall = function(event) {
  // …
};
```

イベントリスナーの実装方法について、両方の方法を計測して組み合わせたところ、以下のような統計が得られました。

696. https://developer.mozilla.org/docs/Web/API/ServiceWorkerGlobalScope#event
697. https://developer.mozilla.org/docs/Web/API/ServiceWorkerGlobalScope#properties
698. https://developer.mozilla.org/docs/Web/API/ServiceWorkerGlobalScope#methods
699. https://developer.mozilla.org/docs/Web/API/ServiceWorkerGlobalScope
700. https://developer.mozilla.org/docs/Web/API/ServiceWorkerGlobalScope#event

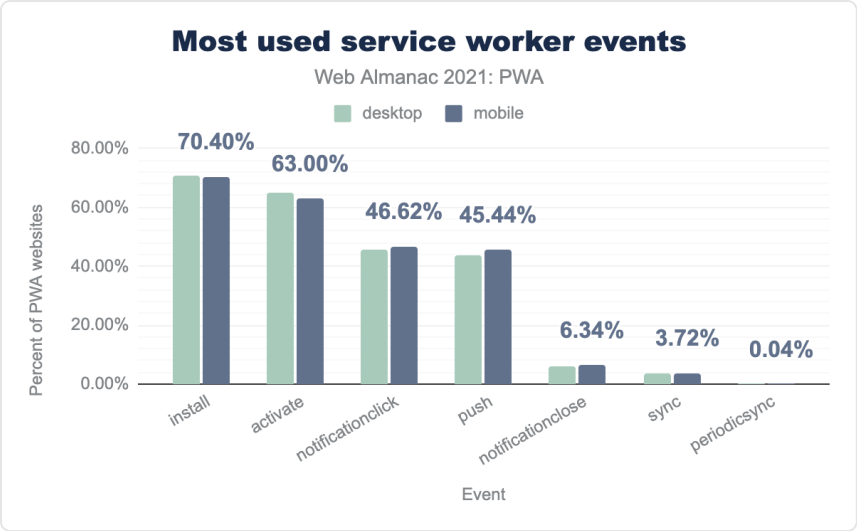


図15.4. もっとも使用されるサービスワーカーのイベント。

これらのイベント結果は、3つのサブカテゴリーに分類できます。

- ライフサイクルイベント
- 通知関連イベント
- バックグラウンド処理イベント

ライフサイクルイベント

図中の最初の2つのイベントリスナーは、ライフサイクルイベント[701]に属しています。これらのイベントリスナーを実装すると、イベントが実行されたときに、オプションで追加のタスクを実行できます。`install` はワーカーが実行されると同時に起動され、サービスワーカーごとに一度だけ呼び出されます。これにより、サービスワーカーが制御を開始する前に必要なものをすべてキャッシュしておくことができます。新しいサービスワーカーがクライアントをコントロールできるようになり、古いサービスワーカーがいなくなると、`activate` が実行されます。これは、前のサービスワーカーが使っていたがもう必要ない古いキャッシュをクリアするなどのことをする良いタイミングです。

どちらのイベントリスナーも高い採用率を誇っています。モバイルPWAの70.40%とデスクトップPWAの70.73%が `install` イベントリスナーを実装し、モバイルの63.00%とデスク

701. *https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle*

トップの64.85%が `activate` をリッスンしています。これらのイベントの内部で実行できるタスクは、パフォーマンスと信頼性にとって重要であるため、これは予想されることです（たとえば、プリキャッシング[702]など）。ライフサイクル イベントをリッスンしない理由としては、サービス ワーカーを通知のみに使用する（キャッシュ戦略なし）、キャッシュ技術をサイトの実行中のリクエストのみに適用する、ランタイムキャッシング[703] という技術がありプリキャッシュ技術と組み合わせてよく使われる（ただしこれだけではない）ことなどがあります。

通知関連イベント

図16.4に示すように、次に人気のあるイベントリスナー群は、Webプッシュ通知[704]に関連する `push`, `notificationclick`, `notificationclose` です。もっとも広く採用されているのは `push` で、サーバーから送信されるプッシュイベントを待ち受けることができ、サービスワーカーを持つデスクトップサイトの43.88%とモバイルサイトの45.44%で使用されています。これは、まだすべてのブラウザで利用できない[705]場合でも、PWAでWebプッシュ通知がいかに人気であるかを示しています。

バックグラウンド処理イベント

図16.4の最後のイベントグループは、サービスワーカーの特定のタスクをバックグラウンドで実行できます。たとえば、データの同期や接続に失敗したときのタスクの再試行などです。バックグラウンド同期[706]（`sync` イベントリスナーを介して）は、Webアプリがタスクをサービスワーカーに委任し、失敗したり接続がない場合に自動的に再試行できるようにします（その場合サービスワーカーは、接続が回復するのを待ち自動的に再試行する）。周期的なバックグラウンド同期[707]（`periodicSync` 経由）は、サービスワーカーのタスクを定期的に実行させます（たとえば、毎朝トップニュースを取得してキャッシュします）。バックグランドフェッチ[708] のような他のAPIは、その使用率がまだかなり低いため、グラフには表示されていません。

このように、バックグラウンド同期技術は、他の技術に比べてまだ広く採用されていません。これは、バックグラウンド同期のユースケースが少ないことと、APIがまだすべてのブラウザで利用可能でないことが一因です。また、Periodic Background Sync[709]を利用するにはPWAをインストールする必要があるため、「ホーム画面に追加」[710]機能を提供しないサイトでは利用することができません。

702. *https://developers.google.com/web/tools/workbox/modules/workbox-precaching*
703. *https://web.dev/runtime-caching-with-workbox/*
704. *https://developers.google.com/web/fundamentals/push-notifications*
705. *https://caniuse.com/push-api*
706. *https://developers.google.com/web/updates/2015/12/background-sync*
707. *https://web.dev/periodic-background-sync/*
708. *https://developers.google.com/web/updates/2018/12/background-fetch*
709. *https://developer.mozilla.org/docs/Web/API/Web_Periodic_Background_Synchronization_API*
710. *https://developer.mozilla.org/docs/Web/Progressive_web_apps/Add_to_home_screen*

その1つがオフライン分析（Workbox Analyticsはバックグラウンド同期を使用しています[711]）、または接続性の欠如による失敗したクエリの再試行（某検索エンジン[712]）などです。

備考: 前回とは異なり、`fetch` と `message` イベントはサービスワーカーの外にも現れる可能性があり、誤検出が多くなる可能性があるため、この分析には含めないことにしました。つまり、上記の解析は、サービスワーカー固有のイベントに対するものです。2020年のデータでは、`fetch` は `install` とほぼ同じ頻度で使われています。

その他、人気のあるサービスワーカーの機能

イベントリスナー以外にも、サービスワーカーには重要な機能があり、その有用性と人気を考えると、呼び出すのは興味深いことです。

次の2つのイベントは、かなり人気があり、よく併用されています。

- `ServiceWorkerGlobalScope.skipWaiting()`
- `Clients.claim()`

`ServiceWorkerGlobalScope.skipWaiting()` は通常 `install` イベントの最初に呼ばれ、新しくインストールされたサービスワーカーが、他にアクティブなサービスワーカーがあったとしても、すぐに `active` 状態へ移行できるようにするものです。我々の分析では、デスクトップPWAの60.47%とモバイルPWAの59.60%で使用されていることがわかりました。



図15.5. サービスワーカーが `skipWaiting()` を呼び出すモバイルサイトの割合

`Clients.claim()` は `skipWaiting()` と組み合わせてよく使われ、アクティブなサービスワーカーがその範囲内のすべてのクライアントの「コントロールを主張」できるようにします。デスクトップでは48.98%、モバイルでは47.14%のページで表示されます。



図15.6. サービスワーカーが `clients.claim()` を呼び出すモバイルサイトの割合

711. https://developers.google.com/web/tools/workbox/modules/workbox-google-analytics
712. https://web.dev/google-search-sw/

前の2つのイベントを組み合わせることで、デフォルトの動作であるアクティブなクライアント（たとえばタブ）が閉じられ、後の時点（たとえば新しいユーザーセッション）で再び開かれるのを待つことなく新しいサービスワーカーがすぐに有効になり、前のものと置き換わることになります。 開発者は、重要なアップデートを即座に実行するため、この手法が有効であると考え、広く採用されているのです。

キャッシュ処理は、サービスワーカーで頻繁に使用され、オフラインなどの機能を有効にし、パフォーマンスの向上に役立つため、PWA体験の中核をなしています。`ServiceWorkerGlobalScope.caches` プロパティは、異なる キャッシュ[713] にアクセスできるサービスワーカーへ関連付けられた キャッシュストレージオブジェクト[714] を返すものです。サービスワーカーを使用しているデスクトップでは57.41%、モバイルでは57.88%のサイトで使用されていることがわかりました。

図15.7. サービスワーカーを持つモバイルサイトのうち、サービスワーカーキャッシュを利用している割合

キャッシュは信頼性とパフォーマンスの高いWebアプリケーションを可能にするため、その高い使用率は予想外ではありません。これは、開発者がPWAに取り組む主な理由の1つであることが多いのです。

最後に、ナビゲーションプリロード[715] を見てみましょう。これは、サービスワーカーの起動時間と並行してリクエストを行うことで、その状況下でリクエストを遅らせることがないようにするものです。`NavigationPreloadManager` インターフェイスは、この技術を実装するための一連のメソッドを提供します。私たちの分析によると、現在、サービスワーカーを使用しているデスクトップサイトの11.02%とモバイルサイトの9.78%でこの技術が使用されているとのことです。

図15.8. ナビゲーションのプリロードを使用しているモバイルサイトの割合

ナビゲーションのプリロードは、まだすべてのブラウザーで利用可能ではない[716]という事実にもかかわらず、適切なレベルで採用されていると言えます。多くの開発者が恩恵を受ける

713. *https://developer.mozilla.org/docs/Web/API/Cache*
714. *https://developer.mozilla.org/docs/Web/API/CacheStorage*
715. *https://developers.google.com/web/updates/2017/02/navigation-preload*
716. *https://caniuse.com/?search=navigation%20preload%20manager*

ことができる技術であり、プログレッシブエンハンスメント[717]として実装することができるのです。

ウェブアプリマニフェスト

ウェブアプリマニフェスト[718]は、Webアプリケーションに関するメタデータを含むJSONファイルで、Webアプリのマニフェストを公開することが、ユーザーが端末にWebアプリをインストールする「ホーム画面に追加」機能を提供する前提条件の1つとなるため、PWAの主要コンポーネントの1つとなっています。その他の条件としては、HTTPSでサイトを提供すること、アイコンがあること、一部のブラウザ（ChromeやEdgeなど）ではサービスワーカーがあることなどが挙げられます。 ブラウザによってインストールするための基準が異なる[719]ことを考慮してください。

Web App Manifestsに関する使用統計情報をいくつか紹介します。サービスワーカーと一緒に可視化することで、「インストール可能な」Webアプリケーションの潜在的な割合について知ることができます。

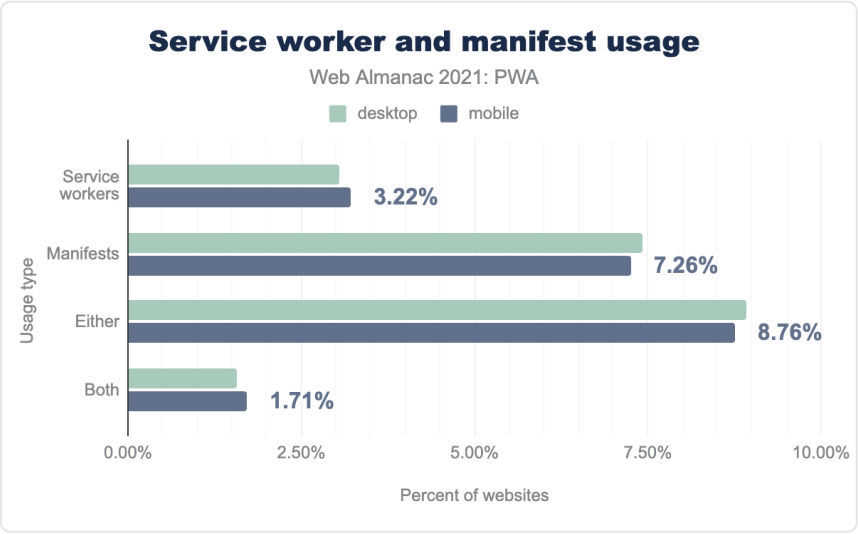


図15.9. サービスワーカーとマニフェストの使用状況。

マニフェストは、サービスワーカーに比べて2倍以上のページで使用されています。その理由のひとつは、一部のプラットフォーム（CMS など）が、サービスワーカーがないサイト

717. https://developer.mozilla.org/docs/Glossary/Progressive_Enhancement
718. https://developer.mozilla.org/docs/Web/Manifest
719. https://web.dev/i18n/ja/installable-manifest/#in-other-browsers

でもマニフェストファイルを自動的に生成するためです。

一方、サービス ワーカーはマニフェストなしで使用できます。たとえば、プッシュ通知、キャッシュ、オフライン機能などをサイトに追加したいが、インストール性には興味がなく、マニフェストを作成しない開発者もいるかもしれません。

上の図では、デスクトップ サイトの1.57%とモバイルサイトの1.71%がサービス ワーカーとマニフェストの両方を備えていることがわかります。これは、「インストール可能な」Web サイトの潜在的な割合の第一近似値です。

Webアプリのマニフェストとサービス ワーカーを持つことに加え、マニフェストのコンテンツは、Webアプリケーションがインストール可能であるために、いくつかの追加の インストール可能基準[720] を満たす必要があります。次にそれぞれの特性を分析する。

マニフェストのプロパティ

次の図は、サービス ワーカーを持つサイトのグループにおける 標準マニフェスト プロパティ[721] の使用状況を示しています。

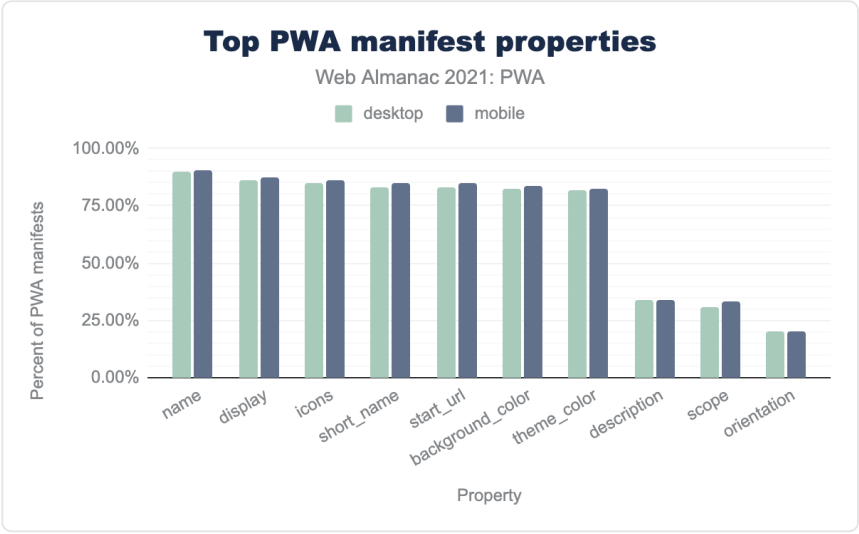


図15.10. PWAマニフェストのトッププロパティ。

このグラフは、Lighthouseのインストール可能なマニフェストの基準[722]と組み合わせると興

720. https://web.dev/installable-manifest/
721. https://w3c.github.io/manifest/#web-application-manifest
722. https://web.dev/i18n/ja/installable-manifest/

味深いものになります。Lighthouse[723] はウェブサイトの品質を分析する人気のツールで、Lighthouse Insightsセクションで見るように、PWAサイトの61.73% がこれらの基準に基づいてインストール可能なマニフェストを有しています。

次に、Lighthouseのインストール可能な要件を、先ほどの表にしたがって1つずつ分析します。

- `name` または `short_name` です。`name` プロパティは90%のサイトに存在し、`short_name` はデスクトップとモバイルのサイトのそれぞれ83.08%と84.69%に表示されています。これらのプロパティの使用率が高いのは、どちらも重要な属性であるためです。`name` はユーザーのホーム画面に表示されますが、長すぎたり画面内のスペースが小さすぎたりすると、代わりに `short_name` を表示することがあります。

- `icon` です。このプロパティは、デスクトップサイトの84.69%、モバイルサイトの86.11%に表示されています。アイコンは、ホーム画面、OSのタスクスイッチャーなど、さまざまな場所で使用されています。このことからも、その採用率の高さがうかがえます。

- `start_url`。このプロパティは、デスクトップサイトの82.84%、モバイルサイトの84.66%に存在します。これは、ユーザーがウェブアプリケーションを起動したときに、どのURLが開かれるかを示すもので、PWAにとってもう1つの重要なプロパティです。

- `display` です。このプロパティは、デスクトップサイトの86.49%およびモバイルサイトの87.67%で宣言されています。これは、ウェブサイトの表示モードを示すために使用されます。表示されていない場合、デフォルト値は `browser` で、これは従来のブラウザのタブであるため、ほとんどのPWAはこれを宣言して、代わりに `standalone` モードで開くべきであると示しています。スタンドアローンモードで開く機能は、「アプリらしい」体験を生み出すのに役立つものの1つです。

- `prefer_related_applications` です。このプロパティはデスクトップサイトの6.87%、モバイルサイトの7.66%で表示されますが、このリストの他のプロパティと比較すると低い割合のように思われます。この理由は、Lighthouseがこのプロパティの存在を必須としておらず、`true` という値で設定することを推奨しているに過ぎないからです。

次に、一連の値を定義することができるプロパティについて深く掘り下げます。どれがもとも広く使われているのかを理解するために。

723. https://developers.google.com/web/tools/lighthouse

トップマニフェストアイコンサイズ



図15.11. PWAマニフェストのトップアイコンサイズ。

圧倒的に人気のあるアイコンサイズは、以下の通りです。192x192と512x512は、Lighthouseが推奨するサイズ[724]です。実際には、開発者もさまざまなサイズを用意し、さまざまなデバイスの画面で見栄えがよくなるようにしています。

724. https://web.dev/add-manifest/#icons

トップマニフェストの表示値

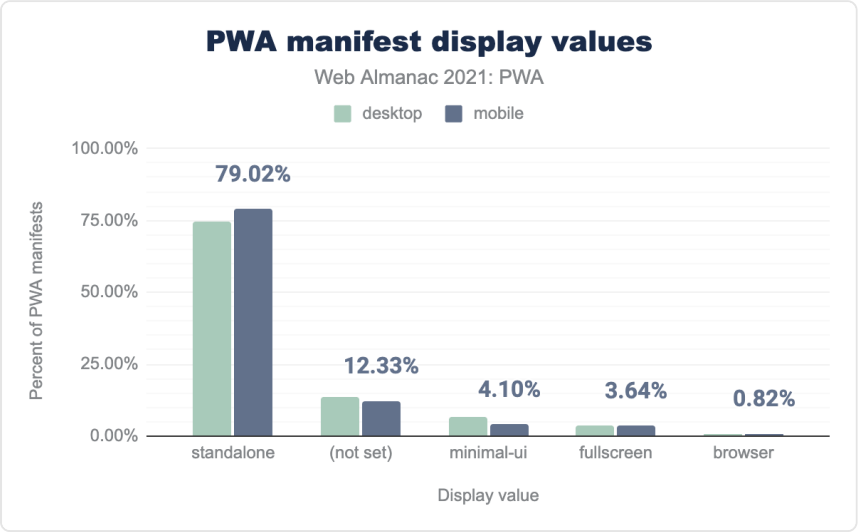


図15.12. PWAマニフェストの表示値。

displayプロパティは、開発者が好むウェブサイトのモードを決定する。 `standalone` モードは、インストールされたPWAをブラウザのUI要素なしで開き、「アプリのように感じる」ようにするものです。このグラフは、サービスワーカーとマニフェストを持つほとんどのサイトがこの値を使用していることを示しています。デスクトップで74.83%、モバイルでは79.02%です。

ネイティブを好むマニフェスト

最後に、 `prefer_related_applications` を分析します。このプロパティの値が `true` に設定されている場合、ブラウザはウェブアプリの代わりに関連するアプリケーションの1つをインストールするよう提案するかもしれません。

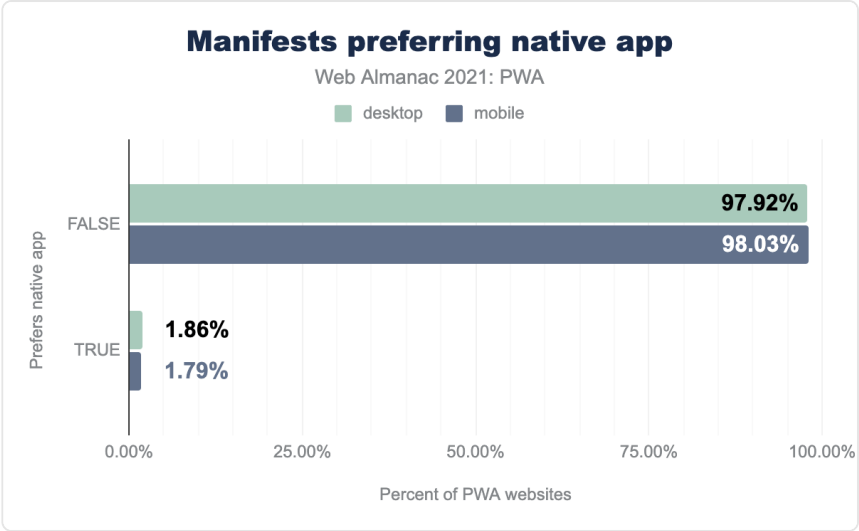


図15.13. ネイティブアプリを優先するマニフェスト。

`prefer_related_applications`は、デスクトップサイトの6.87%、モバイルサイトの7.66%にのみ表示されます。このグラフから、このプロパティを定義したデスクトップサイトの97.92%とモバイルサイトの93.03%が`false`という値を持っていることがわかります。これは、ほとんどのPWA開発者が、ネイティブアプリよりもPWAを提供することを好んでいることを示しています。

PWA開発者の大多数がPWA体験をネイティブアプリケーションに推奨することを好むという事実にもかかわらず、一部の有名なPWA（Twitterなど）は依然としてPWA体験よりもネイティブアプリケーションを推奨することを好むようです。これは、これらの体験を構築するチームの好みによるものか、あるいは特定のビジネスニーズ（ウェブに何らかのAPIがない）によるものかもしれません。

***備考:** 開発者は、設定時にこの決定を静的に行うのではなく、より動的なヒューリスティック[725]を作成して、たとえば、ユーザーの行動やその他の特性(デバイス、接続、場所など)に基づいて体験を促進することも可能です。*

マニフェストのトップカテゴリー

昨年のPWA章では、マニフェストカテゴリ[726]に関するセクションを設け、マニフェストカテゴリ[727]のプロパティに基づいて業界ごとのPWAのパーセンテージを示しました。

725. https://web.dev/define-install-strategy/
726. https://almanac.httparchive.org/ja/2020/pwa#トップマニフェストのカテゴリー
727. https://developer.mozilla.org/docs/Web/Manifest/categories

今年は、このプロパティの使用率が非常に低い（このプロパティが設定されているサイトは1%未満）ため、各カテゴリーのPWAの数を決定するために、このプロパティへ依存しないことにしました。

PWAを使用しているカテゴリーや産業に関するデータがないため、外部の情報源に頼っています。Mobstedは最近、独自のPWAの利用状況に関する分析[728]を発表し、業界別のPWAの割合などを分析しています。

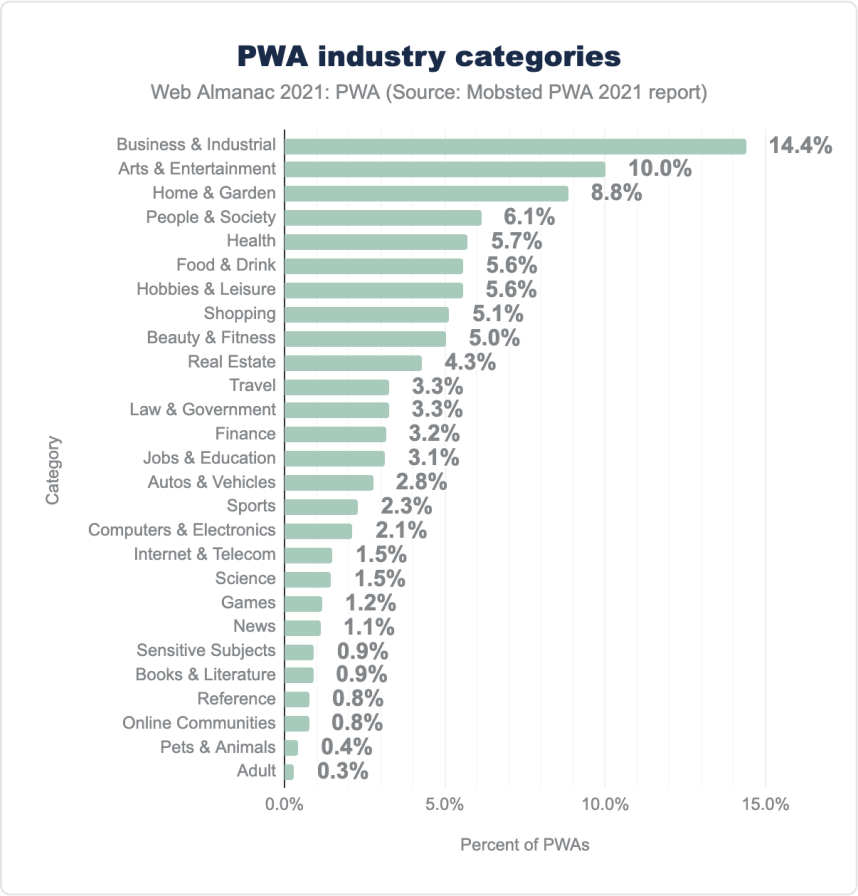


図15.14. PWAの産業カテゴリー（提供元: モブステッドPWA2021レポート[729]）。

モブステッドの分析によると、「ビジネス＆インダストリアル」「アート＆エンターテインメント」「ホーム＆ガーデン」が上位を占めています。これは、昨年のウェブマニフェスト

728. https://mobsted.com/world_state_of_pwa_2021
729. https://mobsted.com/world_state_of_pwa_2021

「カテゴリー」プロパティの分析[730]で、「ショッピング」「ビジネス」「エンターテインメント」が上位3つの値だったことと相関があるようです。

ライトハウスの考察

manifest propertiesセクションで、LighthouseがWebアプリマニフェストファイルに対して持つ インストール可能要件[731] について述べました。Lighthouseは、PWAを構成する他の側面についてもチェックを提供します。方法論に記載されているように、HTTP Archiveは現在、モバイルクロールの一部としてのみLighthouseテストを実行していることに留意してください。

以下のグラフは、各基準をクリアしたサイトの割合を示しています。「PWAサイト」にはサービスワーカーとマニフェストを持つサイトの統計が、「全サイト」には全トータルのサイトのデータが含まれています。

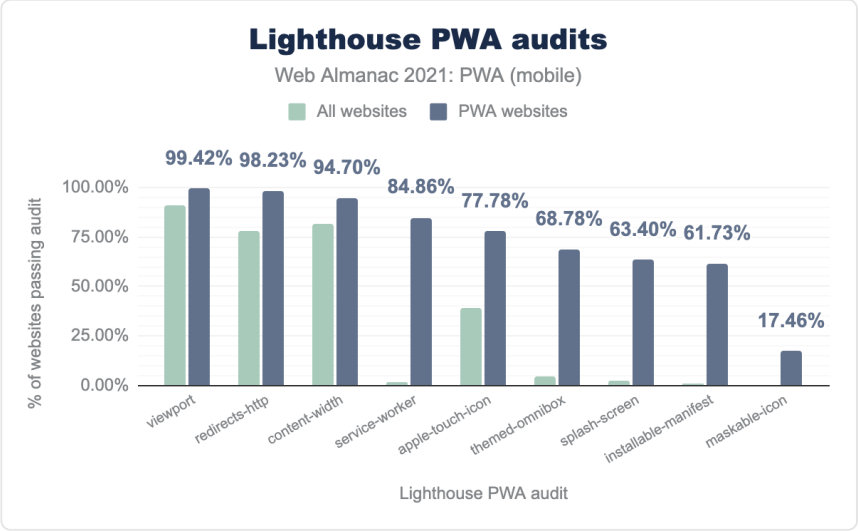


図15.15. ライトハウスPWA監査。

予想通り、この表からPWAとして特定したサイト群（サービスワーカーとマニフェストを持つサイト）は、Lighthouseの各PWA監査に合格する傾向が、あることがわかります。PWAに特化していない監査（たとえば、ビューポートの設定やHTTPからHTTPSへのリダイレクトなど）は、すべてのサイトで高いスコアを獲得していますがPWAに特化した監査では明確な違いがあり、これらは本当にPWAサイトによってのみ使用されています。

730. *https://almanac.httparchive.org/ja/2020/pwa#トップマニフェストのカテゴリー*
731. *https://web.dev/i18n/ja/installable-manifest/*

マスカブル アイコン[732]は、他のPWAオーディションと比較して、PWAサイトでも合格率が低いというのは興味深い点です。マスク可能なアイコンを使用すると、Androidデバイスのアイコンのルック＆フィールを向上させ、割り当てられた形状全体を埋め尽くすようにできます（アイコンのレスポンシブ機能のようなものです）。この機能はオプションで、インストール可能なエクスペリエンスを提供するPWAにとって、ほとんどが興味深いものです。他のPWA機能（オフラインなど）とは異なり、PWAでないサイトが興味を持つことはほとんどないでしょう。

Lighthouseは、これらすべての監査の「合格率」に基づいて、PWAスコア[733]も提供しています。次のグラフは、先に分析した2つのグループ間で、結果のスコアを比較したものです。

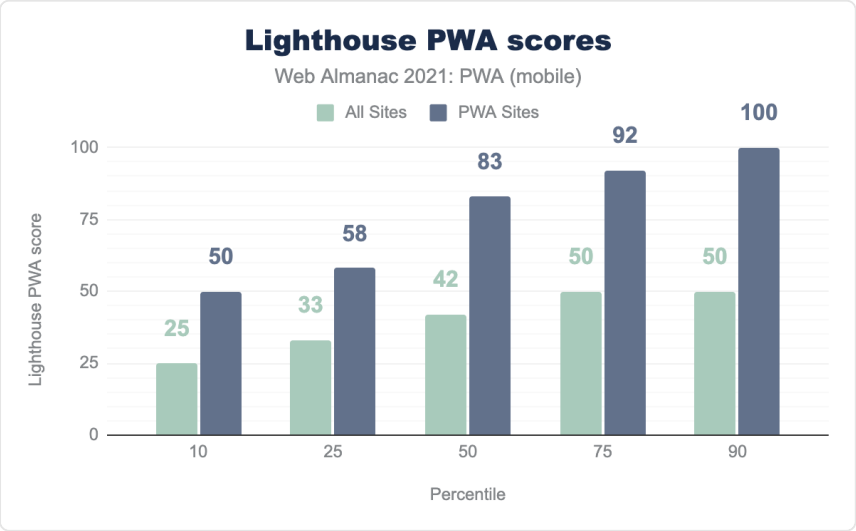


図15.16. Lighthouse PWAのスコア。

以下は、その考察です。

- 中央値は「PWAサイト」が83点であるのに対し、「全サイト」は42点となっています。
- 上位の「PWAサイト」では、少なくとも10%がPWAの最高得点（100点）を獲得していることがわかります。「すべてのサイト」を見ると、75パーセンタイルと90パーセンタイルは、せいぜい50にしか達しません。
- グラフの下端を見ると、「PWAサイト」の90%がLighthouseのPWAスコア50以

732. https://web.dev/i18n/ja/maskable-icon/
733. https://web.dev/i18n/ja/lighthouse-pwa/

上を獲得しています（全サイトでは25）。

「PWAサイト」は「すべてのサイト」よりもPWA固有の要件をより多くクリアする傾向が当然あるため、今回も両グループの差は予想通りです。いずれにせよ、PWAサイトの中央値83というスコアは、PWA開発者のかなりの部分がベストプラクティスに沿っていることを示唆しています。

サービスワーカーライブラリ

サービスワーカーは、ライブラリを使用して、一般的なタスクや機能、ベストプラクティスを実現できます（キャッシュ技術やプッシュ通知の実装など）。もっとも一般的なのはimportScripts()[734]を使う方法で、これはワーカーにJavaScriptライブラリをインポートする方法です。他のケースでは、ビルドツールがビルド時にライブラリのコードをサービスワーカーに直接注入することもできます。

すべてのライブラリがワーカーのコンテキストで使用できるわけではないことを考慮に入れてください。WorkerはWindow[735]、したがってDocument[736]オブジェクトにアクセスできず、ブラウザAPIへのアクセスも制限されています。そのため、サービスワーカーライブラリは、このような文脈で使用されることをとくに想定して設計されています。

このセクションでは、さまざまなサービスワーカーライブラリの人気度を分析します。

人気のインポートスクリプト

以下の表は、`importScripts()`でインポートした各種ライブラリの使用率を示しています。

734. *https://developer.mozilla.org/docs/Web/API/WorkerGlobalScope/importScripts*
735. *https://developer.mozilla.org/docs/Web/API/Window*
736. *https://developer.mozilla.org/docs/Web/API/Document*

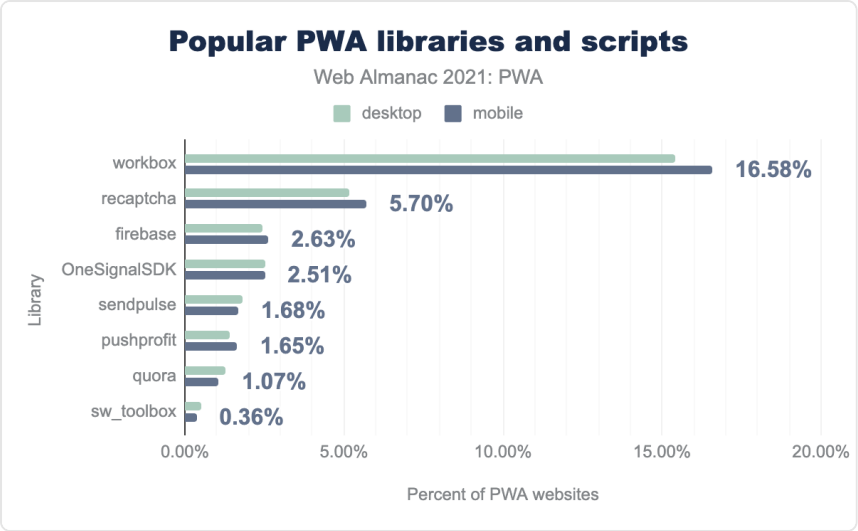


図15.17. 人気のPWAライブラリやスクリプトを紹介。

Workboxは依然としてもっとも人気のあるライブラリで、サービスワーカーがいるデスクトップサイトの15.43% とモバイルサイトの16.58% で使用されていますが、これはWorkbox採用全般の代理と解釈されるかもしれません。次のセクションでは、より全体的で正確なアプローチで採用を測定します。

また、Workboxの前身である `sw_toolbox` は、 昨年の使用率がデスクトップで13.92%、モバイルで12.84%[737]でしたが、今年はそれぞれ0.51%と0.36%に低下しているのも重要な点です。これは、 `sw_toolbox` が2019年に非推奨[738]となったことが一因であると考えられます。人気のあるフレームワークやビルドツールの中には、このパッケージを削除するのに時間がかかったものもあるかもしれないので、今年はより明確に採用数の減少が見て取れます。また2020年と比較して、サイトを増やすなどして測定方法が変わったため、この指標はさらに減少し、直接の前年比は難しくなっています。

備考: *`importScripts()` は `WorkerGlobalScope` の API で、Web Workers[739] など他のタイプの Worker コンテキストでも使用できることを考慮に入れておいてください。reCaptcha[740] は、例えば、reCaptcha JavaScript コードを取得する `importScripts()` コールを含むウェブワーカーを使用しているので、2番目に広く使われているライブラリとして表示されているようです。そのため、サービスワーカーのコンテキストで2番目に広く使われているライブラリとして、代わりに Firebase[741] を考慮すべきです。*

737. https://almanac.httparchive.org/ja/2020/pwa#人気のインポートスクリプト
738. https://github.com/GoogleChromeLabs/sw-toolbox/pull/288
739. https://developer.mozilla.org/docs/Web/API/Web_Workers_API/Using_web_workers
740. https://www.google.com/recaptcha/about/
741. https://firebase.google.com/docs/web/setup

Workboxの使用状況

Workbox[742]は、PWA構築のための共通タスクとベストプラクティスをパッケージ化したライブラリのセットです。先ほどのグラフによると、Workboxはサービスワーカーでもっとも人気のあるライブラリです。では、実際どのように使われているのか、詳しく見ていきましょう。

Workbox 5[743] から、Workboxチームは開発者へ、`workbox-sw`（ランタイム）のロードに`importScripts()`を使う代わりに、Workboxランタイムのカスタムバンドル作成を推奨しています。Workboxチームは`workbox-sw`のサポートを継続しますが、現在は新しい手法を推奨しています。実際、ビルドツールのデフォルトはこの方法を好むように変更されています。

それを踏まえて、あらゆるタイプのWorkboxの機能を利用しているサイトを計測したところ、サービスワーカーが利用しているサイトは、上記で述べたよりもはるかに多いことがわかりました。デスクトップ用PWAの33.04%、モバイル用PWAの32.19%です。



図15.18. サービスワーカーがいるモバイルサイトのうち、Workboxライブラリを使用している割合。

742. *https://developers.google.com/web/tools/workbox*
743. *https://github.com/GoogleChrome/workbox/releases/tag/v5.0.0*

Workboxのバージョン

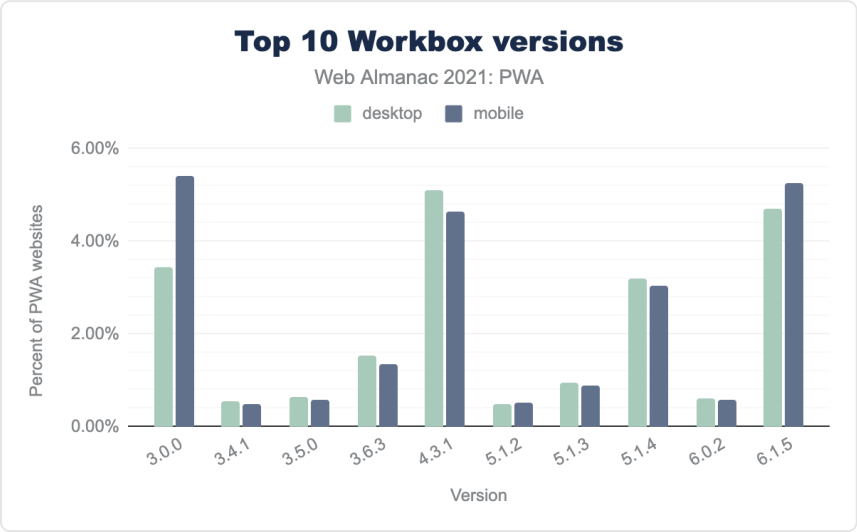


図15.19. Workboxのバージョントップ10。

このグラフから、バージョン 6.1.15[744] の採用率が他と比べてもっとも高いことがわかります。そのバージョンは2021年4月13日にリリースされ、2021年7月にクロールした時点では最新版でした。

当時からさらなるバージョン[745]がリリースされており、チャートで観察される挙動から、発売後すぐにもっとも広く使われるようになると予想されます。

また、今でも広く採用されている古いバージョンも数えるほどしかありません。その理由は、過去に旧バージョンのWorkboxを採用し、提供を続けている人気ツールがあるからだ、すなわち。

- バージョン4.3.1の使用は、ほとんどが create-react-app version 3[746] によってもたらされています。
- バージョン3.0.0も同様に、create-react-app version 2[747]に含まれています。

744. *https://github.com/GoogleChrome/workbox/releases/tag/v6.1.5*
745. *https://github.com/GoogleChrome/workbox/releases*
746. *https://github.com/facebook/create-react-app/blob/v3.4.4/packages/react-scripts/package.json#L82*
747. *https://github.com/facebook/create-react-app/blob/v2.1.8/packages/react-scripts/package.json#L72*

Workboxパッケージ

Workboxライブラリは、特定の機能を含むパッケージまたはモジュールのセット[748]として提供されています。各パッケージは特定のニーズに対応しており、一緒に使用することも、単独で使用することもできます。

次の表は、代表的なパッケージのWorkboxの使用状況を示しています。

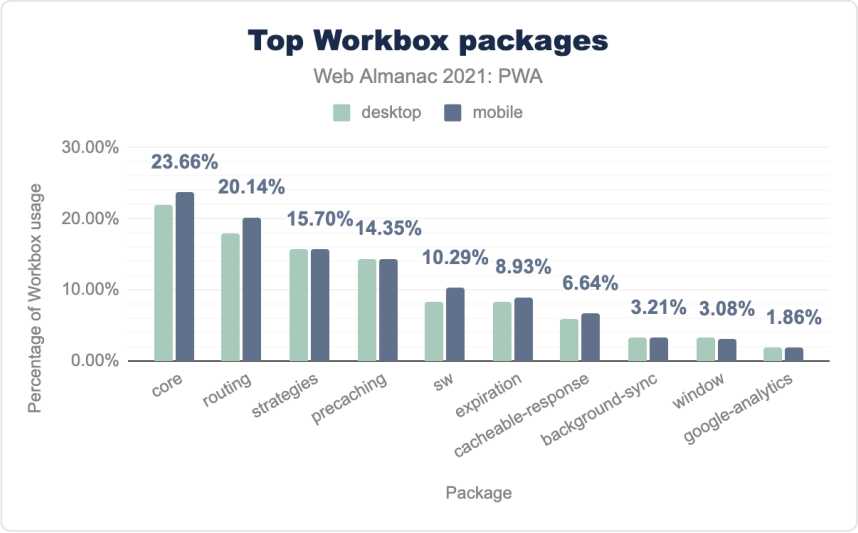


図15.20. Workboxの上位パッケージ。

上のグラフから、次の4つのパッケージがもっとも多く使われていることがわかります。

- Workboxコア[749]: このパッケージには、各Workboxモジュールが依存する共通のコード（たとえば、コンソールと対話するコード、意味のあるエラーを投げるコードなど）が含まれています。そのため、もっとも広く使用されています。
- Workboxルーティング[750]: このパッケージは、リクエストを傍受してさまざまな方法でそれに応答することを可能にします。また、サービスワーカーの内部では非常に一般的なタスクなので、かなり人気があります。
- Workboxプリキャッシング[751]: このパッケージは、サービスワーカーのインストール中に、サイトがいくつかのファイルをキャッシュに保存することを可能にします。このファイル群は通常、PWAの「バージョン」を構成します（ネイティブア

748. https://developers.google.com/web/tools/workbox/modules
749. https://developers.google.com/web/tools/workbox/modules/workbox-core
750. https://developers.google.com/web/tools/workbox/modules/workbox-routing
751. https://developers.google.com/web/tools/workbox/modules/workbox-precaching

プリのバージョンに似ています）。

- Workboxストラテジー[752]: サービスワーカーの“インストール”イベントで行われるプリキャッシングとは異なり、このパッケージは `fetch` イベントを受け取った後にサービスワーカーがどのようにレスポンスを生成するかを決定するランタイムキャッシングストラテジーを可能にします。

Workboxストラテジー

前述のとおり、Workboxはネットワークリクエストに対応するための組み込みストラテジーのセットを提供します。次の図は、もっとも一般的なランタイムキャッシュストラテジーの採用を確認するのに役立ちます。

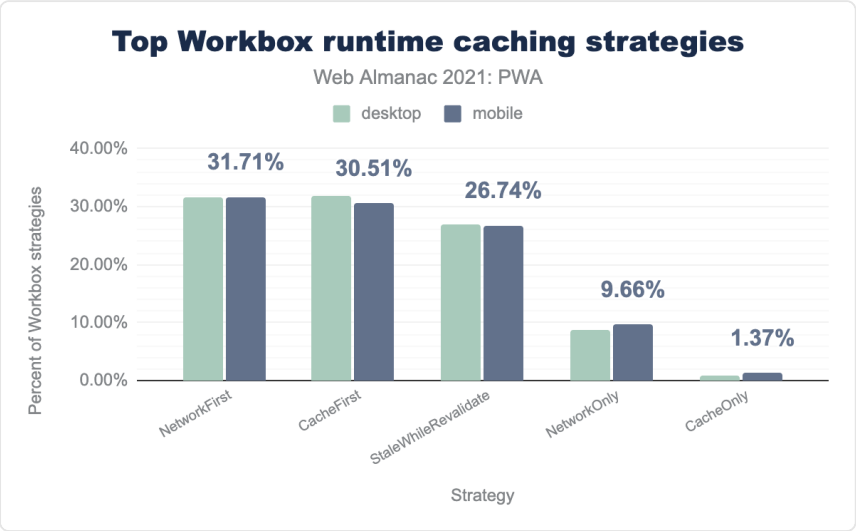


図15.21. Workboxのランタイムキャッシュのトップストラテジー。

`NetworkFirst`、`CacheFirst`、`Stale While Revalidate`が、圧倒的に広く使われています。これらのストラテジーにより、ネットワークとキャッシュをさまざまな方法で組み合わせてリクエストに対応できます。たとえば、もっとも人気のあるランタイムキャッシュストラテジーである `NetworkFirst` は、ネットワークから最新のレスポンスを取得しようとします。もし結果が成功すれば、その結果をキャッシュに格納します。ネットワークが失敗した場合は、キャッシュのレスポンスが使われます。

その他の戦略、たとえば`NetworkOnly`や`CacheOnly`は`fetch()`リクエストをネットワ

752. https://developers.google.com/web/tools/workbox/modules/workbox-strategies

ークかキャッシュに移動して解決し、この2つのオプションを組み合わせないようにします。そのため、PWAの魅力は半減するかもしれませんが、それでも意味のあるユースケースはあります。たとえば、プラグイン[753]と組み合わせることで、機能を拡張することが可能です。

Webプッシュ通知

ウェブプッシュ通知は、ユーザーをPWAに引き留めるためのもっとも強力な方法の1つです。モバイルとデスクトップのユーザーに送ることができ、ウェブアプリがフォアグラウンドにないとき、あるいは（スタンドアロンアプリまたはブラウザタブとして）開いていないときでも受信できます。

ここでは、もっとも人気のある通知関連のAPIについて、使用状況をいくつか紹介します。

ページは Push API[754] の `PushManager` インターフェイス経由で通知を購読します。このインターフェイスは `ServiceWorkerRegistration` の `pushManager` プロパティでアクセスします。デスクトップ用PWAの44.14%、モバイル用PWAの45.09%で使用されています。



図15.22. サービスワーカーを持つモバイルサイトのうち、`pushManager` プロパティの何らかのメソッドが使用された割合

また、サービスワーカーのイベント関連では、図16.4に示すように、プッシュメッセージを受信するための `push` イベントリスナーがデスクトップの43.88%、モバイルの45.44%で使用されています。

サービスワーカーインターフェイスは、通知に関するユーザーインタラクションを処理するために、いくつかのイベントをリッスンすることもできます。図16.4は、`notificationclick`（通知へのクリックを捕捉）がデスクトップの45.64%、モバイルPWAの46.62%で使われていることを示しています。`notificationclose` は使用頻度が低く、デスクトップPWAの5.98%、モバイルPWAの6.34%です。これは、通知の「クリック」よりも、通知の「閉じる」イベントをリッスンすることが意味のあるユースケースが少ないと予想されます。

備考: *サービスワーカーの通知イベント（例：`push`、`notificationclick`）には、さらに `pushManager` プロパティが使われているのが興味深いです。このプロパティは、たとえ*

753. https://developers.google.com/web/tools/workbox/modules/workbox-strategies#using_plugins
754. https://developer.mozilla.org/docs/Web/API/Push_API

ば、Webプッシュ通知の許可を（`pushManager.subscribe`を通じて）要求するために使用されます。この理由の1つは、いくつかのサイトがウェブプッシュを実装し、ある時点でロールバックすることを決定し、そのために許可を要求するコードを排除し、サービスワーカーのコードは変更しないままにしていることかもしれません。

Webプッシュ通知の受理率

通知が有用であるためには、timely, precise, and relevant[755] である必要があります。許可を求めるプロンプトを表示した時点で、ユーザーはそのサービスの価値を理解する必要があります。優れた通知更新は、ユーザーにとって有益で、許可を得た理由に関連するものを提供しなければなりません。

以下の図は、Chrome UXレポート から引用したもので、通知許可プロンプトの受理率を示しています。

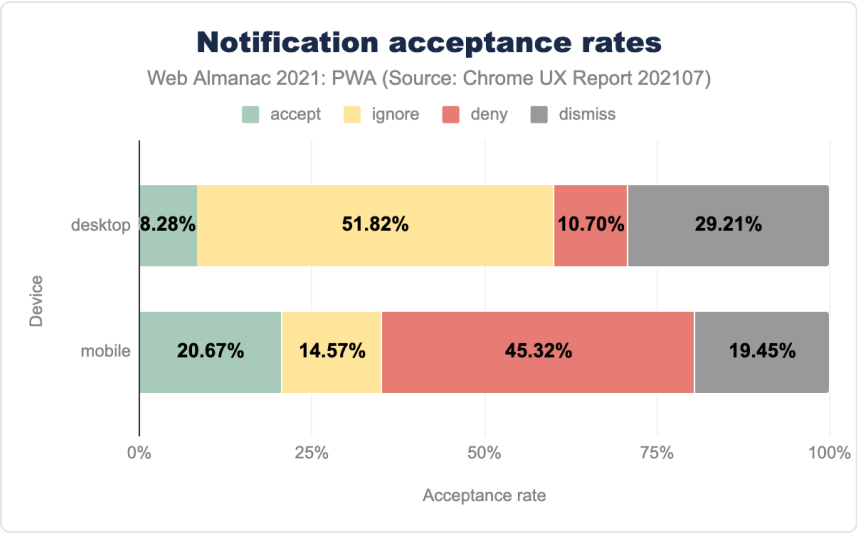


図15.23. 通知の受理率。

モバイルはデスクトップよりも高い承諾率（20.67%対8.28%）を示しています。これは、ユーザーがモバイルの通知をより有用と感じる傾向があることを示唆しています。これには、2つの理由があると考えられます。(1) ユーザーはデスクトップよりも携帯電話の通知に慣れており、モバイルコンテキストでの通知の有用性がより明白であること、(2) 通知プロンプトのモバイルUIは通常より顕著であることです。

また、モバイルはデスクトップに比べて「拒否」率が高く（45.32%対10.70%）、デスクト

755. https://developers.google.com/web/fundamentals/push-notifications

ップユーザーは通知を「無視」する頻度が高い傾向にあります（モバイル19.45%対デスクトップ29.21%）。この理由は、モバイルの登録UIがデスクトップよりも押し付けがましく、ユーザーが通知を受け入れるか拒否するかの判断をする頻度が高いためと考えられます。またデスクトップでは、ユーザーがプロンプトで表示されたタブから離れると、「無視」と記録される場合がありますが、プロンプトの外側をクリックして「無視」するスペースが非常に大きくなっています。

配布方法

PWAの重要な点は、ユーザーがブラウザのURLバーへURLを入力する以上の方法でWebエクスペリエンスへアクセスできるようにすることです。ユーザーは、さまざまな方法でWebアプリをインストールし、ホーム画面のアイコンを使ってアクセスすることもできます。これは、ネイティブアプリのもっとも魅力的な機能の1つであり、PWAもそれを可能にしています。

このインストール可能な体験を配布する方法は、以下の通りです。

- ホーム画面に追加[756]機能により、ユーザーにPWAのインストールを促す。
- Trusted Web Activity (TWA)[757]（現在、Google PlayやMicrosoft Storeなど、あらゆるAndroidアプリストアで利用可能）でパッケージ化し、PWAをApp Storeにアップロードする。

次に、これらの技術に関連する統計データを紹介し、これらのトレンドの使用率と成長率を把握します。

ホーム画面に追加する

これまで、ホーム画面に追加するための前提条件として、サービスワーカーやインストール可能なWebアプリのマニフェストを持っていることを分析してきました。

ブラウザで提供されるインストール体験に加え、開発者は独自のカスタムインストールフローをアプリ内で直接提供できます。

`Window` オブジェクトの `onbeforeinstallprompt` プロパティを使用すると、ユーザーがWebアプリケーションをインストールするように促されようとするときに発生するイベントをドキュメントに取り込むことができます。開発者は、プロンプトを直接表示するか、あるいは、より適切と思われるときに表示するようにプロンプトを延期するかを決定できます。

756. https://developer.mozilla.org/docs/Web/Progressive_web_apps/Add_to_home_screen
757. https://developer.chrome.com/docs/android/trusted-web-activity

我々の分析によると、`beforeinstallprompt`は、サービスワーカーとマニフェストを持つデスクトップサイトの0.48%、モバイルサイトの0.63%で使用されていることが判明しました。

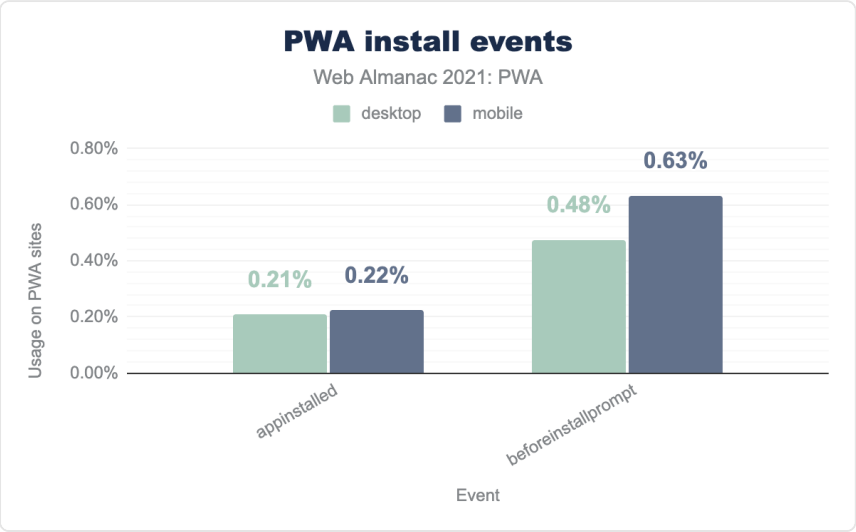


図15.24. PWAのインストールイベント。

`BeforeInstallPromptEvent` APIは すべてのブラウザーでまだ利用できません[758]ので使用率が、比較的低いことが説明されます。それでは、これが示すトラフィックの割合を見てみましょう。

758. *https://caniuse.com/mdn-api_beforeinstallpromptevent*

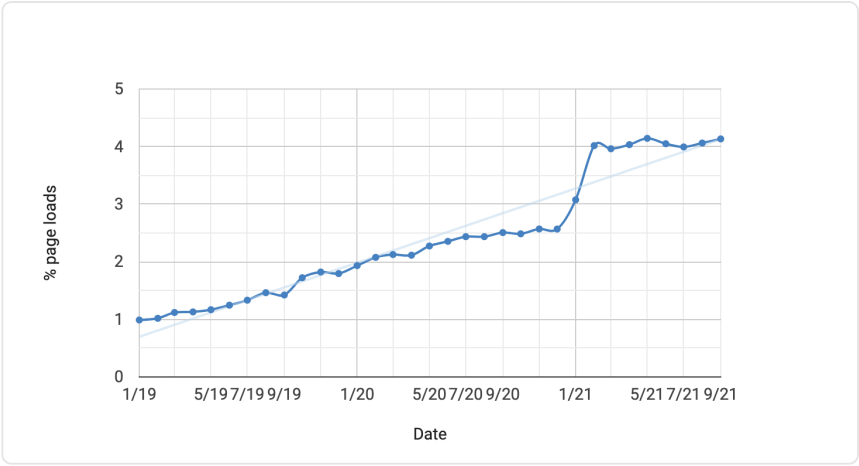


図15.25. あるページで `beforeinstallprompt` を利用したページビューの割合 (提供元: Chrome プラットフォームの状況[759])

Chrome Platform Status[760] によると、この機能を使用しているページロードの割合は4%近いので[761]、一部のトラフィックの多いサイトが、使用している可能性のあることが示唆されます。さらに、昨年と比較して、2.5ポイント採用が伸びていることがわかります。

App Storeでの配信

これまで開発者は、OS固有の言語（AndroidならJavaやKotlin、iOSならObjective-CやSwift）でアプリを作る代わりに、Webベースのモバイルアプリケーションを作り、App Storeにアップロードしてきました。もっとも一般的なアプローチは、Cordova[762] のようなクロスプラットフォームでハイブリッドなソリューションを使用することです。結果として得られるコードは通常、WebView[763]を使用してWebコンテンツをレンダリングしますが、デバイスから機能にアクセスできる一連の非標準APIも提供されています。

WebViewベースのアプリはネイティブアプリと似ているように見えますが、確かにいくつかの注意点があります。WebViewは単なるレンダリングエンジンであるため、ユーザーはフルブラウザとは異なる体験をする可能性があります。最新のブラウザAPIは使用できないかもしれませんし、もっとも重要なのは、WebViewとブラウザの間でCookieを共有できないことです。

TWAを使えば、PWAをネイティブアプリケーションのシェルにパッケージ化し、いくつか

759. https://www.chromestatus.com/metrics/feature/timeline/popularity/1436
760. https://www.chromestatus.com/metrics/feature/timeline/popularity/1436
761. https://www.chromestatus.com/metrics/feature/timeline/popularity/1436
762. https://cordova.apache.org/
763. https://developer.android.com/reference/android/webkit/WebView

のApp Storeにアップロードできます。WebViewベースのソリューションとは異なり、TWAは単なるレンダリングエンジンではなく、フルスクリーンモードで動作する完全なブラウザーとなります。そのため、機能が完全であり、常に最新であり、最新のウェブAPIにアクセスできることを意味します。

開発者はPWAをTWAで直接ネイティブアプリにパッケージ化し、Android Studioを使用[764]できますが、このタスクをはるかに容易にするツールがいくつかあります。次に、そのうちの2つを分析します。PWA BuilderとBubblewrapです。

PWAビルダー

PWAビルダー[765]は、Web開発者がProgressive Web Appsを構築し、Microsoft StoreやGoogle Play Storeなどのアプリストア向けにパッケージ化することを支援するオープンソースプロジェクトです。提供されたURLを確認し、利用可能なマニフェスト、サービスワーカー、およびSSLをチェックすることから始まります。

PWA Builderは3ヶ月のタイムスロットで200kのURLをレビューしました[766]と発見されたのです。

- 75%はマニフェストが検出された
- 11.5%にサービス要員が検出された
- 9.6%はブラウザからインストール可能なPWA（マニフェストとSW、https化）

Bubblewrap

Bubblewrap[767] は、開発者がTWAを使用してPWAを起動するAndroidアプリのプロジェクトを作成、構築、更新できるように設計されたツールおよびライブラリのセットです。

Bubblewrapを使うことで、開発者はAndroidツール（Android Studioなど）周辺の詳細を意識する必要がなく、Web開発者にとっては非常に使い勝手が良いのです。

Bubblewrapの使用統計はありませんが、Bubblewrapに依存していることが知られている注目すべきツールがいくつかあります。たとえば、PWA BuilderやPWA2APK[768]はBubblewrapを搭載しています。

764. *https://developer.chrome.com/docs/android/trusted-web-activity/integration-guide*
765. *https://www.pwabuilder.com/*
766. *https://x.com/pwabuilder/status/1454250060326318082?s=21*
767. *https://github.com/GoogleChromeLabs/bubblewrap*
768. *https://appmaker.xyz/pwa-to-apk*

結論

「Progressive Web Apps」という言葉が生まれてから6年、そのコアテクノロジーの採用は増え続けている。サービスワーカーは間もなくウェブトラフィックの20%を支配するようになり、サイトは毎年、より多くの機能を追加し続けています。

2021年、開発者はウェブアプリケーションを構築・配布するための多様な選択肢を持ち、もっとも一般的な作業を担うことができるツールや、これらの体験をアプリストアにアップロードする簡単な方法を提供することができるようになります。

ウェブは、これまでOS固有の言語だけで作られていたアプリケーションがウェブ技術で開発できることを年々実証し続け、企業はこれらのアプリ的体験をウェブへもたらすために投資を続け[769]ています。

この分析が、お客様のPWAプロジェクトにおいて、より多くの情報に基づいた意思決定を行うための一助となれば幸いです。2022年にこれらのトレンドがどれだけ成長するのか、楽しみです。

著者



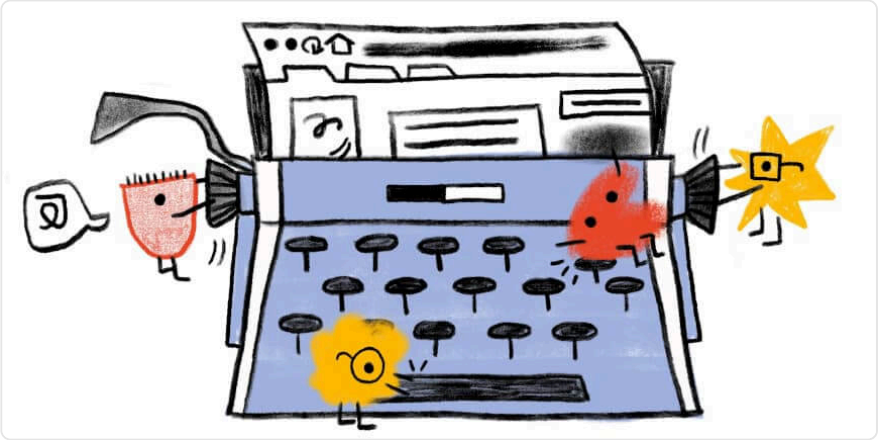
Demian Renzulli

𝕏 @drenzulli demianrenzulli

DemianはGoogleのWeb Ecosystems Consultingチームのメンバーで、アルゼンチンのブエノスアイレスに生まれ、現在はニューヨークを拠点に活動しています。Progressive Web AppsとAdvanced Capabilitiesにフォーカスしている。web.dev[770] でよく執筆している。

769. https://www.theverge.com/2021/10/26/22738125/adobe-photoshop-illustrator-web-announced
770. https://web.dev/authors/demianrenzulli/

部 III 章 16
CMS



***Alon Kochba** によって書かれた。*
Alan Kent、 Andrey Lipattsev、 Chris Sater と John Teague によってレビュ。
Rick Viscomi と Tosin Arasi による分析。
Shaina Hantsis 編集。
Sakae Kotaro によって翻訳された。

序章

この章では、CMSのエコシステムの現状と、ウェブ上でコンテンツを消費し体験する方法についてユーザーの認識を形成する上で、CMSが果たす役割の拡大を理解する一助となることを目指します。私たちの目標は、CMSの一般的な状況や、これらのシステムによって生成されたウェブページの特徴に関連する側面について議論することです。

CMSスペースと、ウェブの現在と未来におけるその役割を理解するための探求において、分析すべき多くの興味深く重要な側面と、答えるべき質問がある。私たちは、CMSプラットフォームの広大さと複雑さを認識し、この分野の主要なプレーヤーに関する深い専門知識と好奇心をもって取り組んでいます。

これらのプラットフォームは、私たちが高速で弾力性のあるウェブという集団的な探求を成功させるために重要な役割を担っています。このことは、この1年でますます明らかになり、今後もそうであることが期待されます。

CMS間のばらつきや、これらのプラットフォーム上で構築されるユーザーコンテンツの種類が異なることを考慮し、これらの比較は慎重に行うことが重要です。

CMSのプラットフォームが多数あるため、採用実績の上位のCMSのみにフォーカスしたセクションもあります。

TLDR; 世界の約半数のサイトがCMSで作成されていることがわかります。もっとも人気のあるCMSのトップ10は、前年比で比較的安定していますが、市場シェアには興味深い変化が見られます。CMSで構築されたサイトのパフォーマンスは、前回チェックしたときから劇的に向上しています。

それでは、分析に入りましょう。

***免責事項:** Alonは*Wix*に勤務し、ウェブパフォーマンスの取り組みを率いていますが、意見は彼個人のものです。*

CMSとは？

コンテンツマネジメントシステム（CMS）とは、個人や組織がコンテンツを作成、管理、公開するためのシステムのことです。とくにWebコンテンツ用のCMSは、Webで消費・体験されるコンテンツの作成・管理・公開を目的としたシステムです。

各CMSは、ユーザーがコンテンツを中心に簡単かつ効果的にウェブサイトを構築するために、幅広いコンテンツ管理機能のサブセットとそれに対応するメカニズムを実装しています。また、CMSは、ユーザーが必要に応じて簡単にコンテンツをアップロードし、管理できるようにすることを目的とした管理機能も提供しています。

CMSのサイト構築支援は、テンプレートにユーザーコンテンツを追加して利用するものから、ユーザーがサイト構造を設計・構築するものまで、その種類と範囲に大きな差があります。

CMSについて考えるとき、ウェブ上でコンテンツを公開するためのプラットフォームを提供するこのようなシステムの実行可能性に役割を果たすすべての構成要素を考慮する必要があります。これらすべての構成要素は、CMSプラットフォームを取り巻くエコシステムを形成し、ホスティングプロバイダー、エクステンション開発者、開発会社、サイトビルダーなどを含みます。したがって、私たちがCMSについて語るとき、通常はプラットフォームそのものと、それを取り巻くエコシステムの両方を指すことになります。

この章でのCMSの定義は、WappalyzerのCMSの定義[771]を使用しています。

771. *https://www.wappalyzer.com/technologies/cms*

CMSの皆様には、このオープンソースプロジェクト[772]に貢献いただき、今後の検出・分類の改善に役立てていただきたいと思います。

Shopify、Magento、Webflow、および他のいくつかのプラットフォームは、WappalyzerでCMSとしてマークされていないため、この章の分析に表示されません。

Eコマースプラットフォームは、非CMSサイトのかなりの部分を占めており、Eコマースの章でカバーされています。たとえば、W3Techs[773]によると、Shopifyはこの1年で大きく成長し、7月にはウェブサイトの3.7%を占めるようになりました。

当社の調査では、200以上の個別のCMSが確認され、これらは1つのCMSにインストールされているものから数百万に及ぶものまでさまざまです。

その中には、オープンソースのもの（WordPressやJoomlaなど）もあれば、独自仕様のもの（WixやSquarespaceなど）もあります。CMSプラットフォームには、「無料」のホスティングプランやセルフホスティングプランで使用できるものもあれば、企業レベルでもより上位のプランで使用できるオプションもあります。

CMSの空間は全体として、CMSエコシステムの複雑な連合宇宙であり、すべてが分離していると同時に絡み合っているのです。

CMSの採用

この作業を通じて、私たちの分析は、デスクトップとモバイルのウェブサイトを調べています。調べたURLの大半は両方のデータセットに含まれていますが、中にはデスクトップまたはモバイルデバイスからしかアクセスされないURLもあります。このためデータに小さな乖離が、生じる可能性があり、そのためデスクトップとモバイルの結果を別々に見ています。

772. https://github.com/AliasIO/wappalyzer
773. https://w3techs.com/technologies/history_overview/content_management/all/q

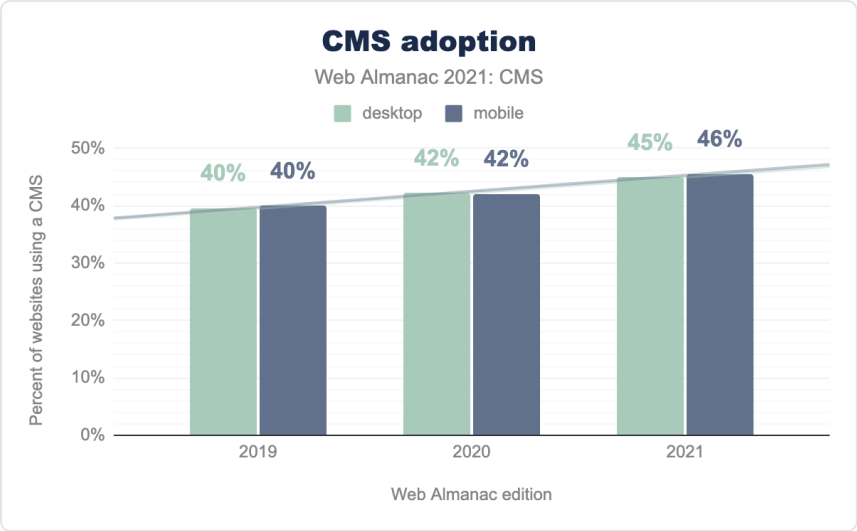


図16.1. CMS導入前年比。

2021年7月時点で、公共サイトの45%以上がCMSプラットフォームを採用しており、2020年から[774]7%以上の成長を示しています。その内訳は、デスクトップが2019年の42%から45%、モバイルが2020年の42%から46%となっています。

この数字をW3Techs[775]など、よく使われる別のデータセットと比較してみると、2021年7月時点でCMSを使って作成されているWebサイトの割合は64.6%と2020年7月の59.2%を上回り、9%以上増加していると報告されていて興味深いです。

我々の分析とW3Techsの分析との乖離は、調査方法の違い、そしてCMSとは何かという定義の違いによって説明できる。

W3Techsの定義は以下の通りです。「*コンテンツ管理システムとは、Webサイトのコンテンツを作成・管理するためのアプリケーションです。このカテゴリには、Wiki、ブログエンジン、ディスカッションボード、静的サイトジェネレータ、ウェブサイトエディタ、またはウェブサイトのコンテンツを提供するあらゆるタイプのソフトウェアとして分類されるシステムも含め、このようなシステムをすべて含みます*」

前述の通り、WappalyzerはCMSの定義をより厳しくしており、W3Techsのレポートに登場する主要なCMSは除外されている。

方法論のページで、私たちの方法を詳しく説明しています。

774. *https://almanac.httparchive.org/ja/2020/cms#CMSの採用*
775. *https://w3techs.com/technologies/history_overview/content_management/all/q*

地域別のCMS導入状況

CMSプラットフォームは、国によって多少の差はありますが、世界中で広く利用されています。

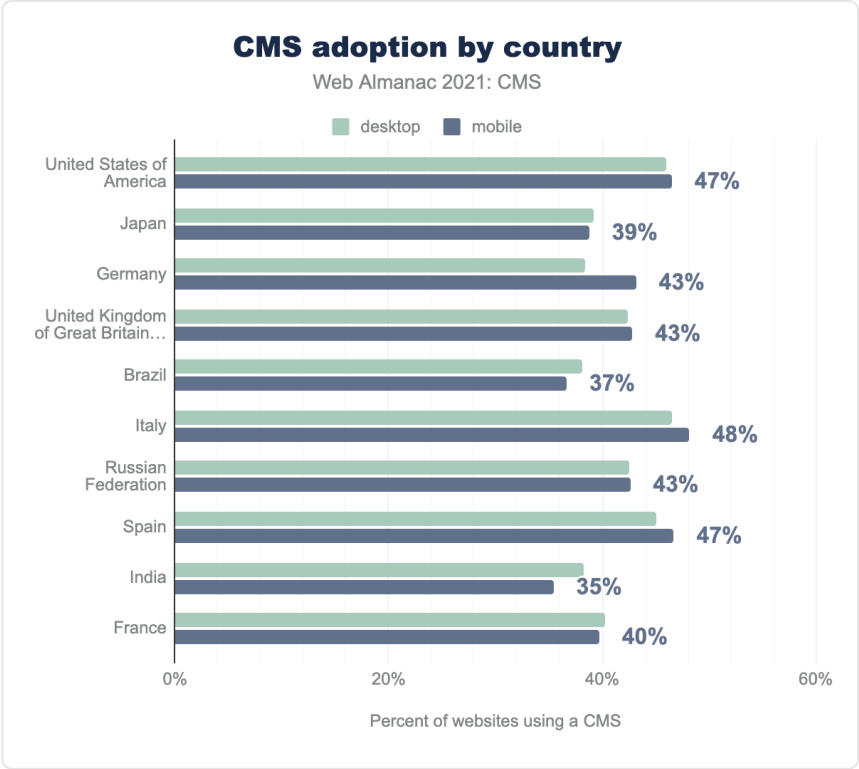


図16.2. 国別のCMS導入状況。

ウェブサイト数が多い地域の中で、CMSの採用率がもっとも高いのは米国、イタリア、スペインで、ユーザーが訪問したモバイルサイトの46%〜47%がCMSで構築されています。インドとブラジルはもっとも低く、それぞれ35%と37%にとどまっています。

また、このデータを世界中のサブリージョン[776]に分割し、もっとも人気のある地域順に並べることで、マクロトレンドの特定をより容易にできます。

776. https://github.com/GoogleChrome/CrUX/blob/main/utils/countries.json

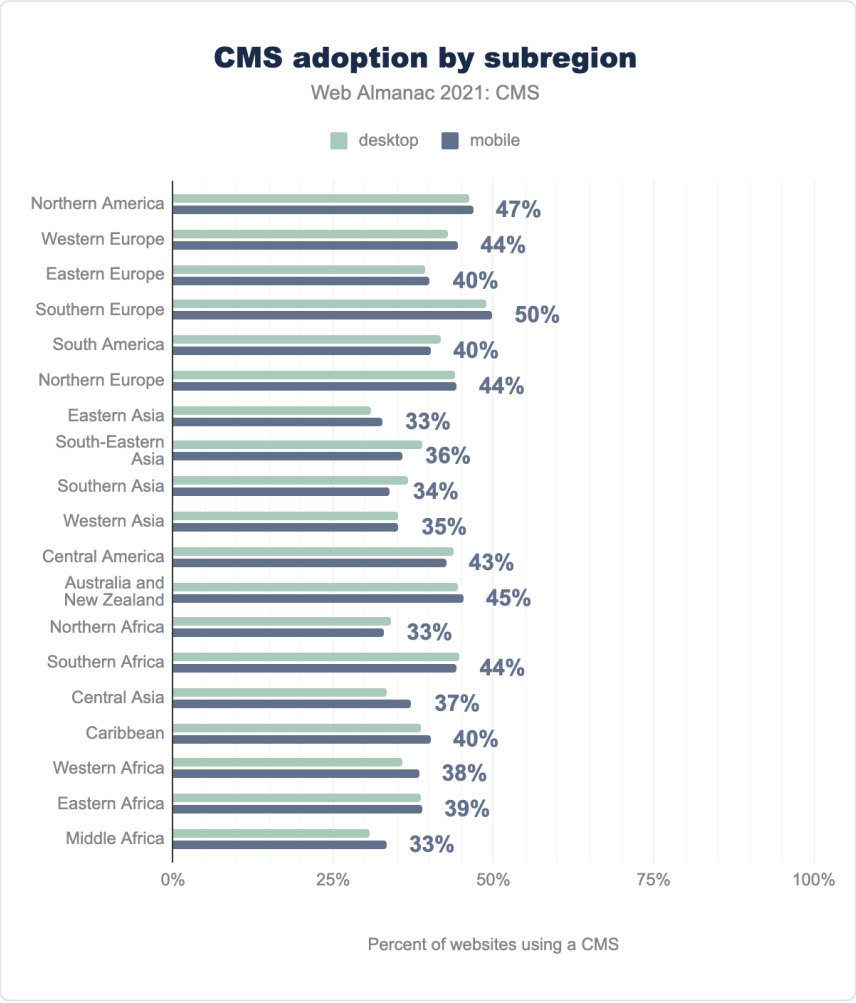


図16.3. サブリージョン別のCMS導入状況。

導入がもっとも進んでいるのは南欧で、半数のサイトがCMSを使用しています。一方、もっとも進んでいないのは東アジアで、データセット中のサイトの3分の1しかCMSを使用していません。

ランク別CMS採用状況

また、サイトの推定ランク別にCMSの採用状況を調査した。

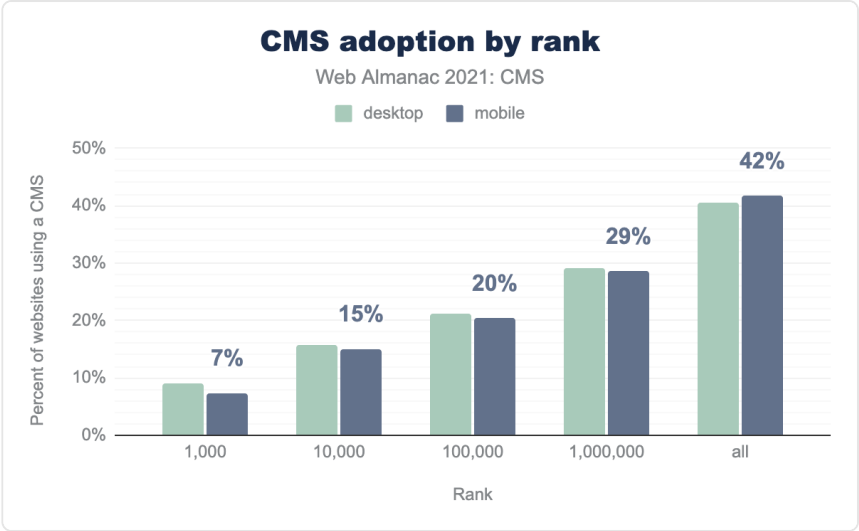


図16.4. CMSのランク別採用状況。

CMSは、分析対象の全データセットの42%であるのに対し、上位1,000のモバイルウェブサイトの7%にしか過ぎません。これは小規模な企業やウェブサイトでは、使いやすさからCMSを使用する傾向があり、上位のウェブサイトは、プロのウェブ開発者による独自のソリューションで構築される傾向があることから説明できます。CMSプラットフォームの利用が増え続けている中、今後、CMSプラットフォームも上位サイトの採用率を高めていくことができるのか、注目したいところです。

上位のCMS

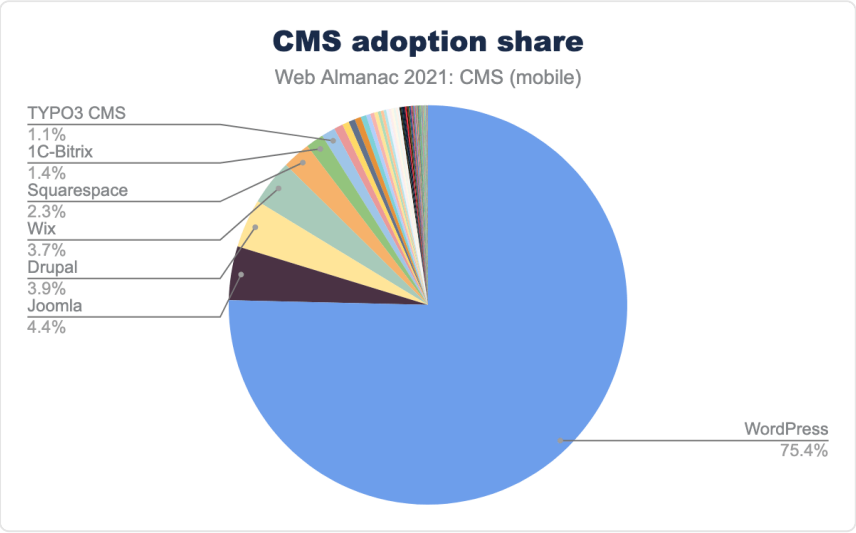


図16.5. CMSの採用シェア。

CMSを利用している全ウェブサイトの中で、相対的に大きなシェアを占めているのがWordPressのサイトで75%以上の導入率、次いでJoomla、Drupal、Wix、Squarespaceの順となっています。

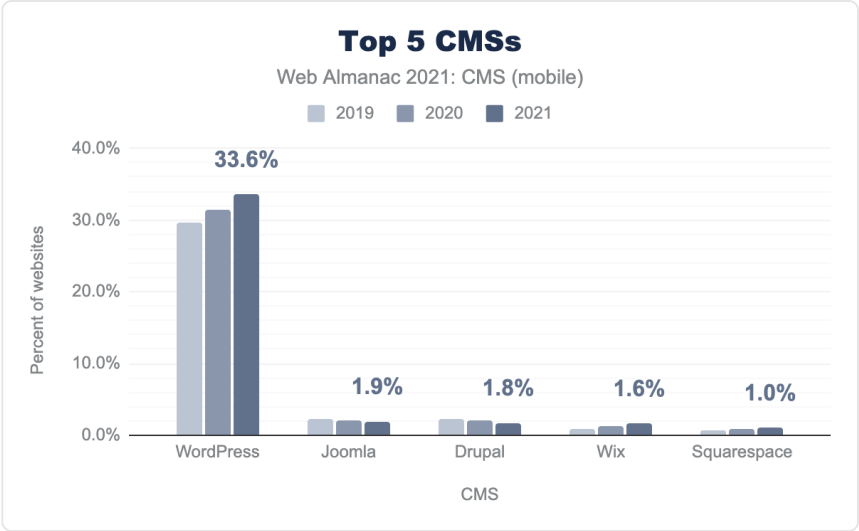


図16.6. CMSの前年比トップ5。

全ウェブサイトにおけるCMSの採用状況を調べたところ、218種類のCMSプラットフォームのうち、利用率が1%を超えているのは5つのプラットフォームのみでした。

もっともよく使われているプラットフォームであるWordPressは、このうち33.6%が使用しており、2020年の31.4%から7%増加し、総導入数は増加しています。

割合で言うと、JoomlaとDrupalの採用は減少しています。Joomlaのサイトは昨年の2.1%から1.9%に減少し（9.5%減）、Drupalは2%から1.8%に減少しました（10%減）。絶対的な普及率は、測定したサイトの数では増加しましたが、CMS全体の使用率および当社のデータセット（増え続けています！）に占める割合では、減少しています。

Wixの採用率は1.2%から1.6%（33%増）、Squarespaceは0.9%から1%（11%増）に増加しました。

CMSプラットフォームで構築されたこれらのサイトの採用を、ランクの大きさ[777]で調べてみると、プラットフォーム間で興味深い分布が、あることがわかります。

777. *https://developers.google.com/web/updates/2021/03/crux-rank-magnitude*

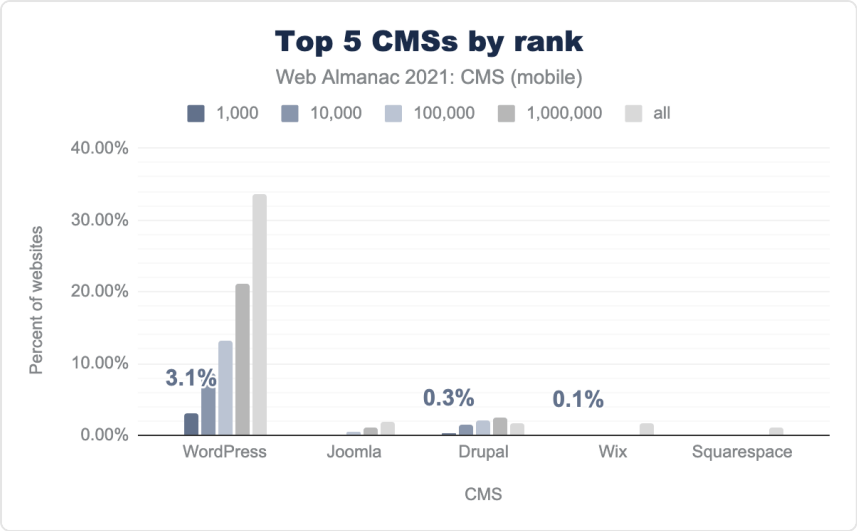


図16.7. CMSのランキングトップ5です。

上位1Kのモバイルサイトの3.1%がWordPressで構築されており、全体の33.6%がWordPressで構築されています。Drupalはランキングの中位（10K〜1M）内で高い採用率を維持しており、WixとSquarespaceのサイトはほとんどが上位1Mサイト外にランクインしています。

CMSのユーザー体験

CMSの重要な点は、そのプラットフォーム上に構築されたサイトを訪れるユーザーに対して、どのようなユーザー体験を提供するかということです。私たちは、Chrome User Experience Report[778] (CrUX)が提供するRUMや、Lighthouseを用いた合成テストを通じて、これらの経験を検証しようと試みています。

コアWeb・バイタル

2021年は、ウェブパフォーマンスにとって素晴らしい年でした。コアWeb・バイタル[779]への注目が高まり、多くのプラットフォームがユーザー体験とロード時間の改善に注力するよう正しい方向へ誘導されました。さらに重要なのは、ユーザーに適切なツールとガイダンスを提供し、ウェブサイトのパフォーマンスを監視して改善することです。その結果、多くのプラットフォームから大幅なパフォーマンスの改善が見られました。これらのプラットフォー

778. *https://developers.google.com/web/tools/chrome-user-experience-report*
779. *https://web.dev/articles/vitals#core-web-vitals*

ムは進化を続け、ウェブ全体のユーザー体験を徐々に良くしており、これは私たち全員にとって大きな収穫です。

コアWeb・バイタルの技術レポート[780]では、このデータを掘り下げ、月単位で更新される各技術の進捗を確認することが可能です。

このセクションでは、Web Almanac全体で提示されるデータに一貫した時間枠を提供するため、2021年7月のデータに焦点を当て、ユーザーがCMS搭載のWebページをどのように体験しているかについての理解に光を当てることができる、Chromeユーザー体験レポートが提供する重要な3要素を検証しています。

- 最大のコンテントフルペイント (LCP)
- 最初の入力までの遅延 (FID)
- 累積レイアウトシフト (CLS)

これらの指標は、優れたウェブ・ユーザー体験を示す中核的な要素を網羅することを目的としています。パフォーマンスの章でより詳しく説明しますが、ここではとくにCMSの観点からこれらの指標を見ることに興味があります。

まず、オリジン数が多い10のCMSプラットフォームを確認し、各プラットフォームのサイトのうち何パーセントが*passing*グレード（上記の各指標の75パーセンタイルが「良好」（緑）の範囲にあること）を持っているかを検証してみましょう。

780. *https://httparchive.org/reports/cwv-tech*

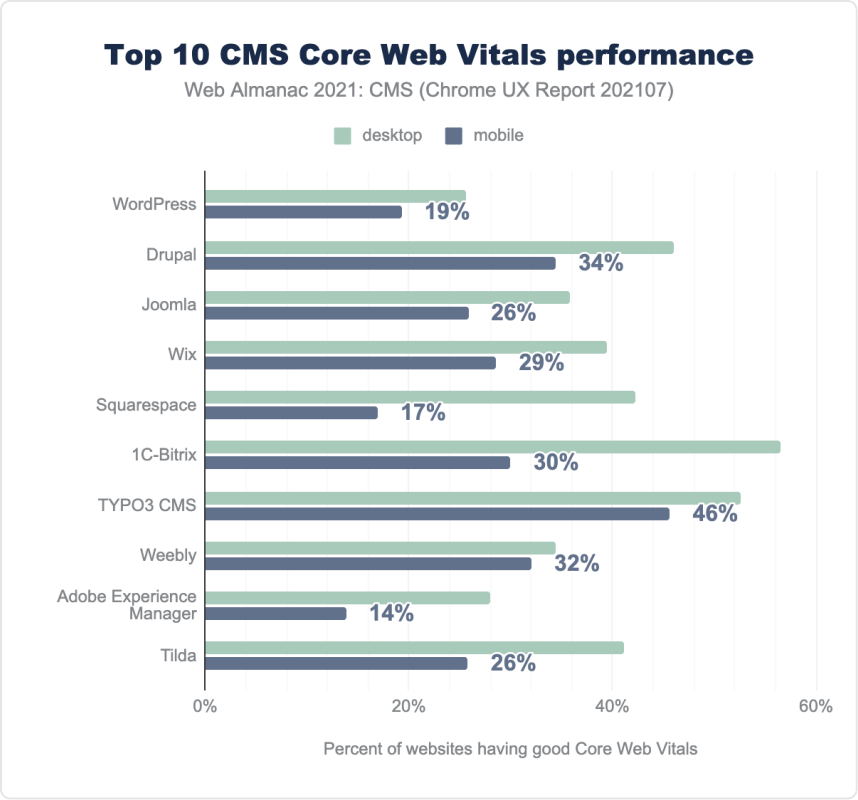


図16.8. CMSのコアとなるウェブバイタルの性能トップ10。

デスクトップからの訪問者は、一般的にモバイルよりもわずかに良いスコアであることがわかります。これは、モバイル機器の弱体化と回線の貧弱さによって説明できます。

また、特定のプラットフォームでモバイルとデスクトップの差が大きいということは、デバイスによってユーザーに提供されるページがかなり異なることを示唆しています。

7月のモバイル端末向けでは、TYPO3 CMS（主にヨーロッパ諸国で使用）が、もっとも合格率が高く、46%のモバイルサイトが3つのCWVすべてに合格しています。WordPress、Squarespace、Adobe Experience Managerは、合格したサイトの割合が20%未満でした。

デスクトップ端末の体験はやや良好で、1C-Bitrix（主にロシアで使用）は、CWVを通過したサイトの割合が56%ともっとも高かった。WordPressは合格率がもっとも低く、26%にとどまりました。

Dudaは、8月に47%のサイトが通過し、昨年から全体的に大きく進歩した点で、栄誉ある賞に値すると思います。彼らは、Wappalyzer[781]の誤った検出に関連して、7月に壊れたデータ収

集のためにこのレポートに含まれておらず、彼らの起源を誤って膨らませ、彼らのCWV割合を減少させたのである。

また、これらのCMSプラットフォームの進捗を、モバイルビューに着目して昨年のデータと比較して評価することもできます。

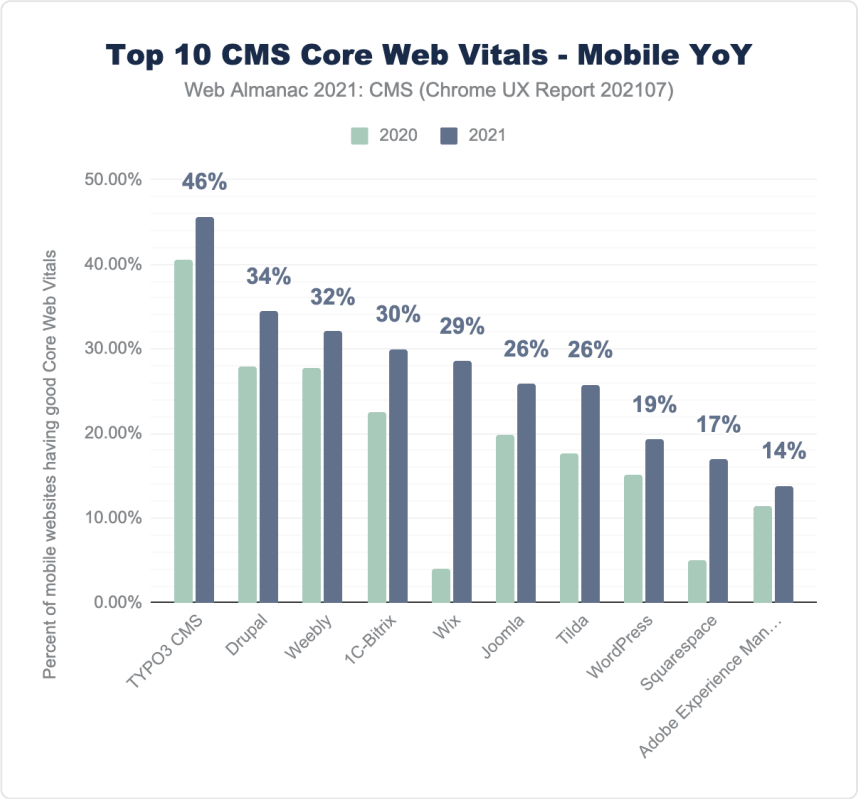


図16.9. トップ10 CMSコアWebバイタルのモバイルビューのパフォーマンス前年比。

これらのCMSはいずれも、2020年8月以降、CWVが良好なオリジンの割合が向上していることが確認できました。WixとSquarespaceがもっとも顕著な進展を見せ、他のCMSとの差を縮めています。

ここでは、3つのコアWeb・バイタルについて、各プラットフォームに改善の余地があるか、またどの指標が昨年からもっとも改善されたかを見ていきましょう。

781. *https://github.com/AliasIO/wappalyzer/pull/4189*

最大のコンテントフルペイント (LCP)

最大のコンテントフルペイント (LCP)は、ページのメインコンテンツが読み込まれ、ユーザーにとって有益なページとなった時点を測定します。これは、ビューポート内に表示される最大の画像またはテキストブロックのレンダリング時間を測定することによって行われます。

「良い」LCPは2.5秒以下とされています。

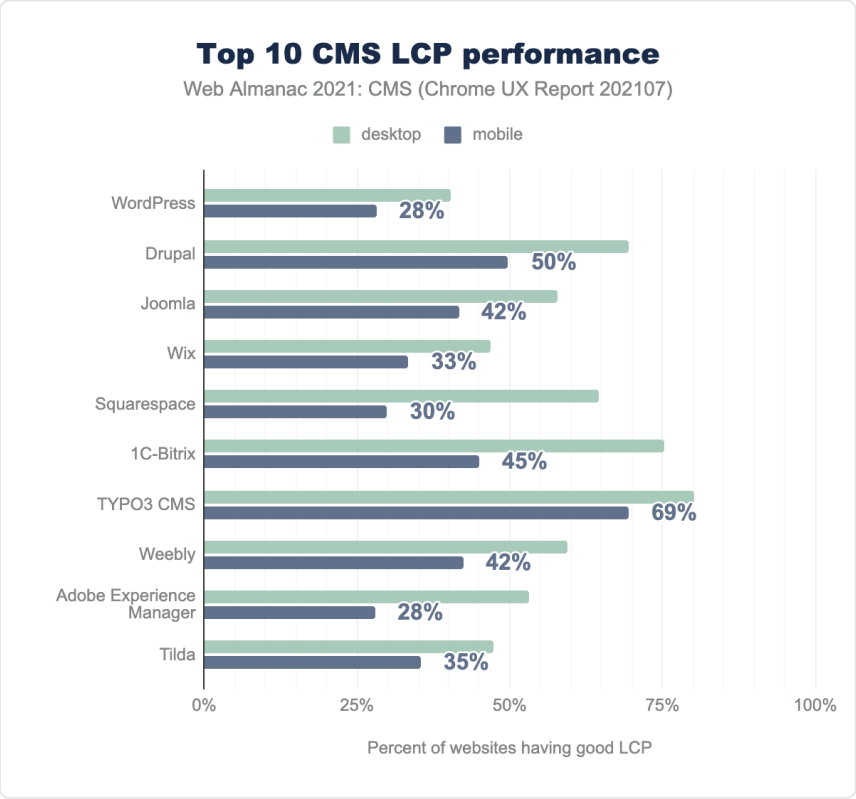


図16.10. CMSのLCP性能トップ10。

TYPO3 CMSはLCPスコアがもっとも高く、69%のオリジンが「良い」LCP体験をしたのに対し、WordPressとAdobe Experience ManagerはLCPスコアがもっとも低く、28%のオリジンしか「良い」LCP体験をしていません。

一般に、ほとんどのプラットフォームがLCPの指標に苦労しているようです。これはおそらく、LCPが画像/フォント/CSSをダウンロードし、適切なHTML要素を表示することに依存し

ていることに関連していると思われます。あらゆる種類のデバイスと接続速度で、これを2.5秒未満で達成することは困難です。LCPのスコアを向上させるには、通常、キャッシュ、プリロード、リソースの優先順位付け、競合する他のリソースの遅延ロードを正しく使用することが必要です。

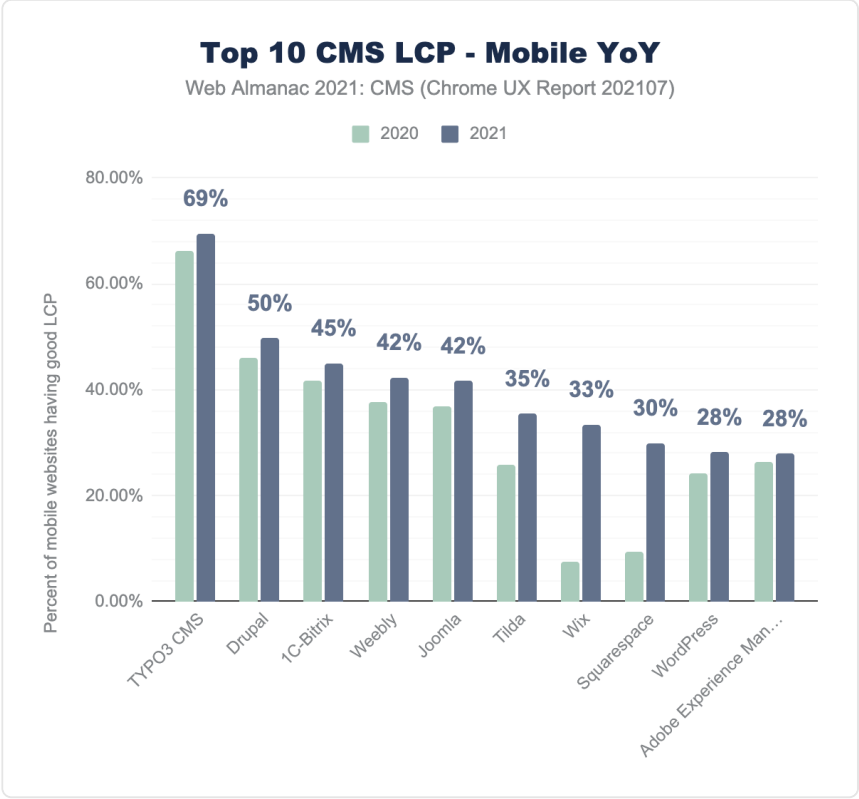


図16.11. CMS上位10社のモバイルビューのLCPパフォーマンス前年比。

すべてのCMSがこの1年でLCPを向上させたことがわかりますが、ほとんどのCMSが小幅な向上でした。もっとも大きく伸びたのは、昨年LCPのスコアが非常に低かったWixとSquarespaceです。Tildaもかなりの進歩があったようです。

最初の入力までの遅延（FID）

最初の入力までの遅延 (FID)は、ユーザーが最初にページとインタラクションを行ったとき（すなわちリンクをクリックしたとき、ボタンをタップしたとき、またはJavaScriptを使用したカスタムコントロールを使用したとき）から、ブラウザがそのインタラクションを処理

できるようになるまでの時間を測定するものです。ユーザーの視点に立った「速い」FIDとは、サイト上でのユーザーの行動から、停滞した体験ではなく、ほとんど即座にフィードバックされることです。

遅延は苦痛であり、ユーザーがサイトを操作しようとしたときに、サイトの他の側面からのロードの干渉と相関している可能性があります。

「良い」FIDは100ミリ秒以下と見なされている。

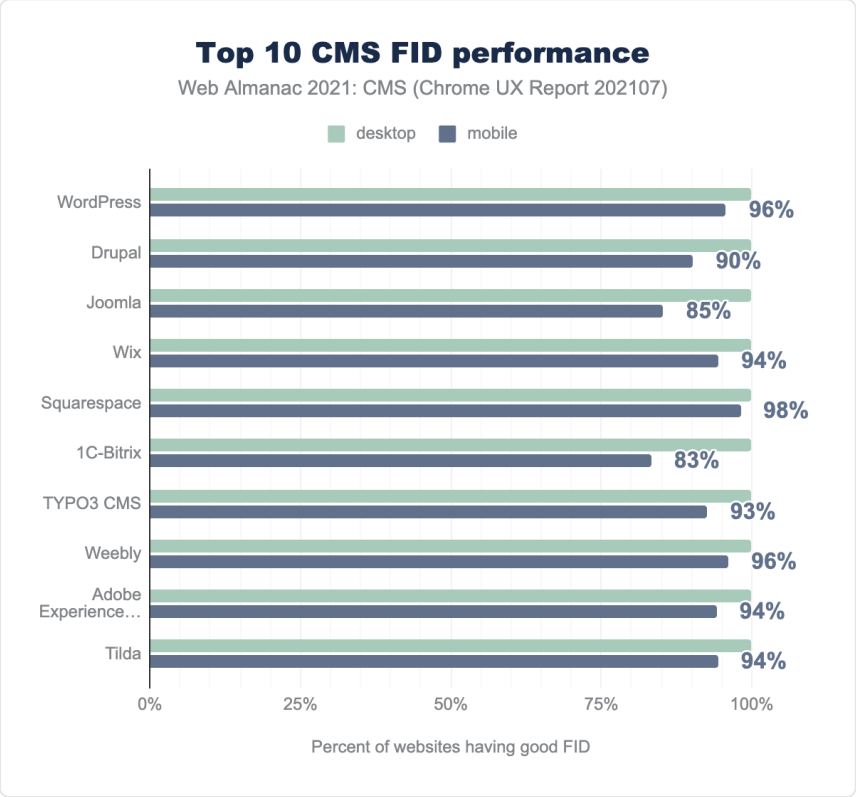


図16.12. CMSのFID性能トップ10。

デスクトップでは、ほとんどのCMSでFIDは非常に良好で、すべてのプラットフォームで100%のスコアを獲得しています。ほとんどのCMSはモバイルでも90%以上の良好なFIDを実現していますが、BitrixとJoomlaは83%と85%のオリジンしか良好なFIDを実現していません。

ほぼすべてのプラットフォームが良好なFIDを提供できていることから、最近ではこの指標

の厳密さについて疑問の声が上がっています。Chromeチームは最近、記事[782]を公開し、将来的により良い応答性の指標を持つための考えを詳しく説明しました。

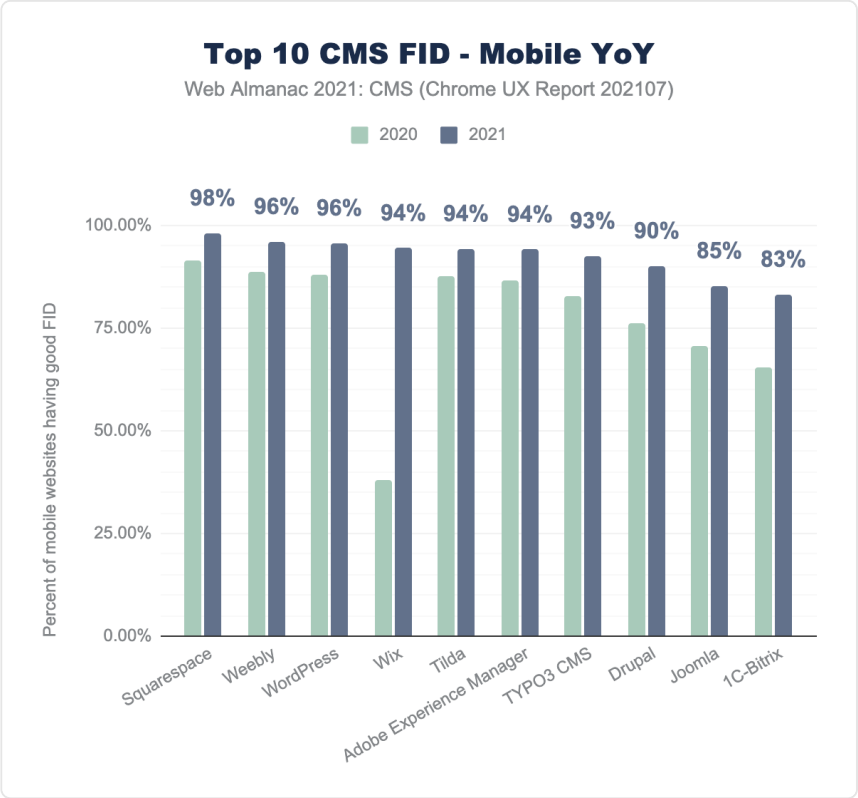


図16.13. トップ10 CMSのモバイルビューのFIDパフォーマンス前年比。

年間データを見ると、これらのCMSはすべて過去1年間にFIDを改善することができました。WixはFIDでもっとも追いつく必要があり、かなり数字を伸ばしました。JoomlaとBitrixは今年もっとも低いFIDスコアでしたが、それでも改善することができました。

累積レイアウトシフト (CLS)

累積レイアウトシフト（CLS）は、ウェブページ上のコンテンツの視覚的安定性を測定するもので、ページの全寿命期間中に発生した。ユーザーとの直接的なインタラクションによらない、予期せぬレイアウトシフトごとに、レイアウトシフトのスコアの最大バーストを計測する。

782. *https://web.dev/responsiveness/*

レイアウトシフトは、あるレンダリングフレームから次のレンダリングフレームへ可視要素の位置が変わるときに発生します。

CLSメトリクスは昨年進化しました[783]。主にセッション ウィンドウズの概念を導入し、長寿命のページやシングルページアプリ (SPA) に対してより公平になりました。

0.1点以下を「良い」、0.25点以上を「悪い」、その間を「要改善」として測定しています。

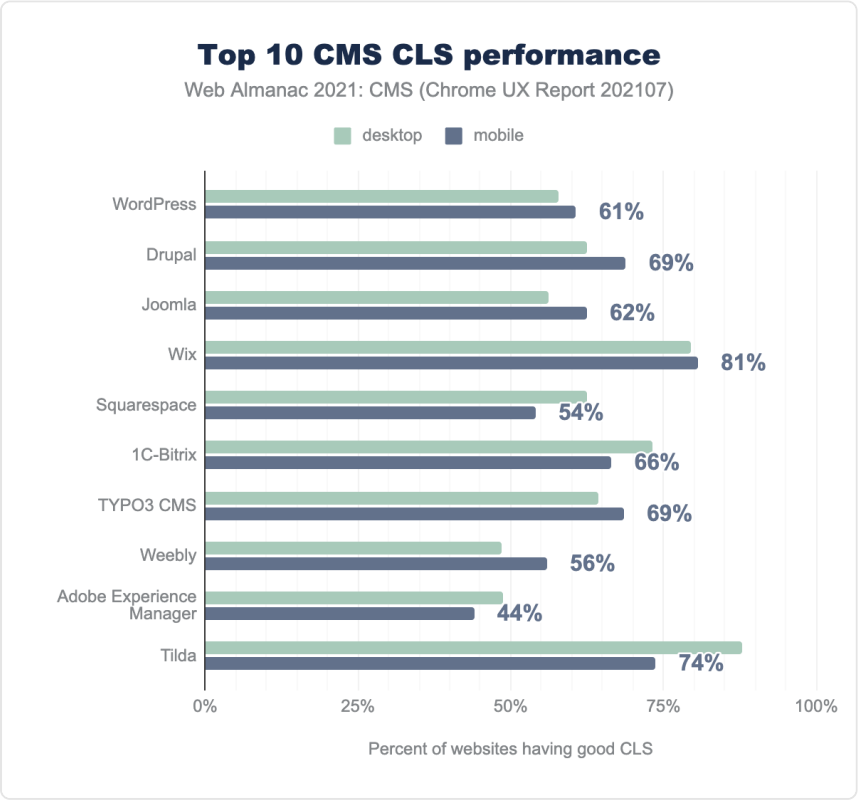


図16.14. CMSのCLS性能トップ10。

WixのCLSスコアはもっとも高く、モバイル端末の81%が「良い」CLSを獲得しています。AdobeエクスペリエンスマネージャーのCLSスコアはもっとも低く、モバイル端末の44%が「良い」CLSを獲得しているに過ぎません。レイアウトのずれは接続速度にかかわらず通常は避けることができるので、すべてのプラットフォームは、レイアウトのずれを最小限に抑

783. https://web.dev/evolving-cls/

える[784]ことによって、これらの数値を改善するよう努力する必要があります。

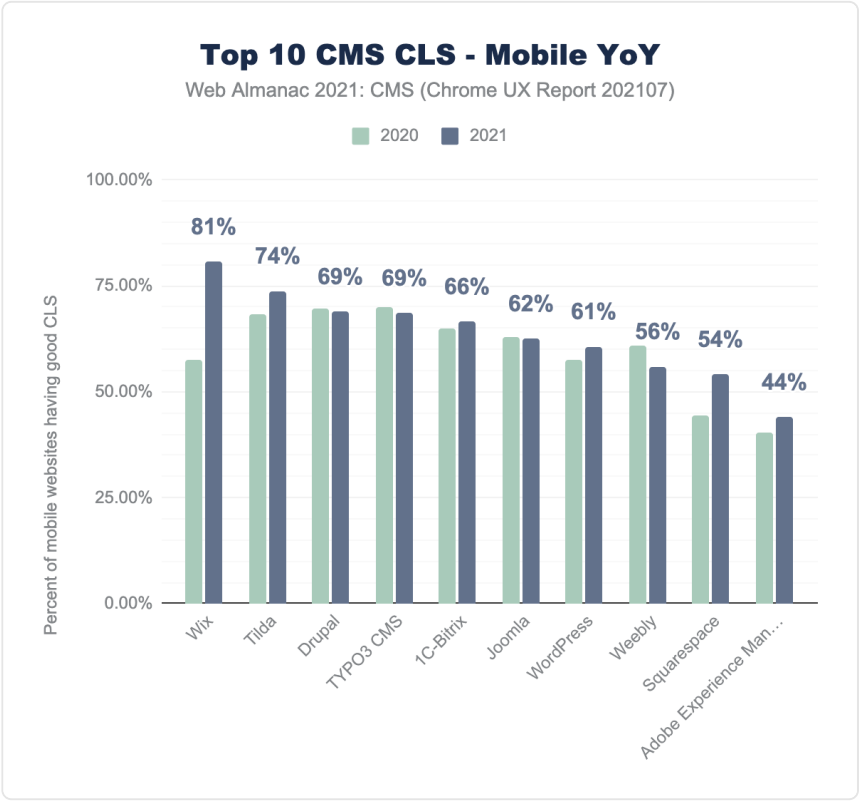


図16.15. トップ10 CMSのCLSのパフォーマンスは、モバイルビューの前年比。

年間のデータを比較すると、ほとんどのCMSが何らかの進歩を遂げ、あるいはウィンドウズCLSメトリックスへの変更によって恩恵を受けたことがわかります。しかし、Weeblyのような特定のCMSは、過去1年間でCLSのスコアが後退していることがわかります。

Lighthouse

Lighthouse[785]は、ウェブページの品質を向上させるためのオープンソースの自動化ツールです。このツールの重要な点は、パフォーマンス、アクセシビリティ、SEO、ベストプラクティスなどの観点からウェブサイトの状態を評価するための一連の監査機能を提供していることです。Lighthouseレポートは、開発者がWebサイトのパフォーマンスを改善するための提

784. https://web.dev/articles/optimize-cls
785. https://developers.google.com/web/tools/lighthouse/

案を得ることができる方法であるラボデータを提供しますが、LighthouseスコアはCrUX[786]が収集した実際のフィールドデータに直接影響を与えるものではありません。Lighthouseとそのラボスコアとフィールドデータ[787]の相関については、こちらをご覧ください。

HTTP Archiveは、すべてのモバイル用ウェブページでLighthouseを実行しており（残念ながらデスクトップ用の結果はありません）、これもCPUが遅い4G接続をエミュレートするためにスロットルされています。

このデータを分析することで、CrUXでは追跡できない指標も含むこれらの合成テストの結果を用いて、CMSのパフォーマンスを別の角度から評価できます。

パフォーマンススコア

Lighthouseのパフォーマンススコア[788]は、複数の指標スコアの加重平均値です。

786. *https://developers.google.com/web/tools/chrome-user-experience-report*
787. *https://web.dev/lab-and-field-data-differences/*
788. *https://web.dev/performance-scoring/*

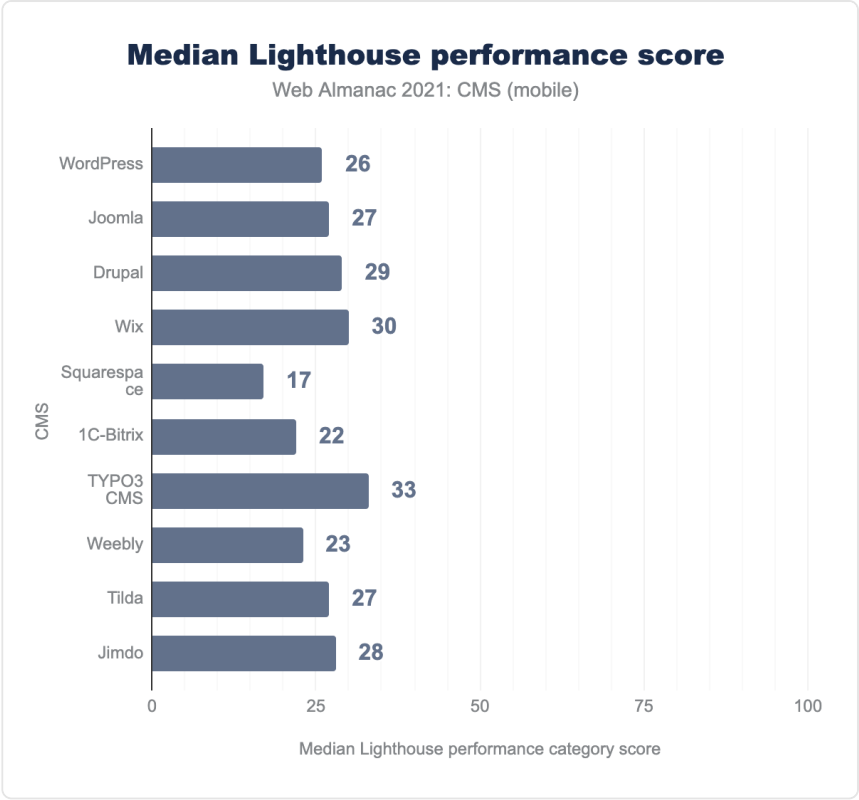


図16.16. CMS上位10社のLighthouseパフォーマンススコア（中央値）。

モバイルの上位プラットフォームのパフォーマンススコアの中央値は、17〜33と低いことがわかります。上で見たように、 これはモバイルフィールド データの悪い結果[789]を直接意味しませんが、とくにローエンドデバイスとLighthouseが、エミュレートしたのと同様のネットワーク接続ではすべてのプラットフォームに改善の余地があることを示唆しています。

SEOスコア

検索エンジン最適化（SEO）とは、検索エンジンで見つけやすくするために、ウェブサイトを改善することです。詳しくはSEOの章で説明しますが、検索エンジンのクローラーにできるだけ多くの情報を提供し、検索エンジンの結果に適切に表示されるようサイトをコーディングすることもそのひとつです。カスタムメイドのWebサイトと比較すると、CMSはSEO対策に優れていることが期待されますが、このカテゴリにおけるLighthouseのスコアは適切に

789. https://philipwalton.com/articles/my-challenge-to-the-web-performance-community/

高いものでした。

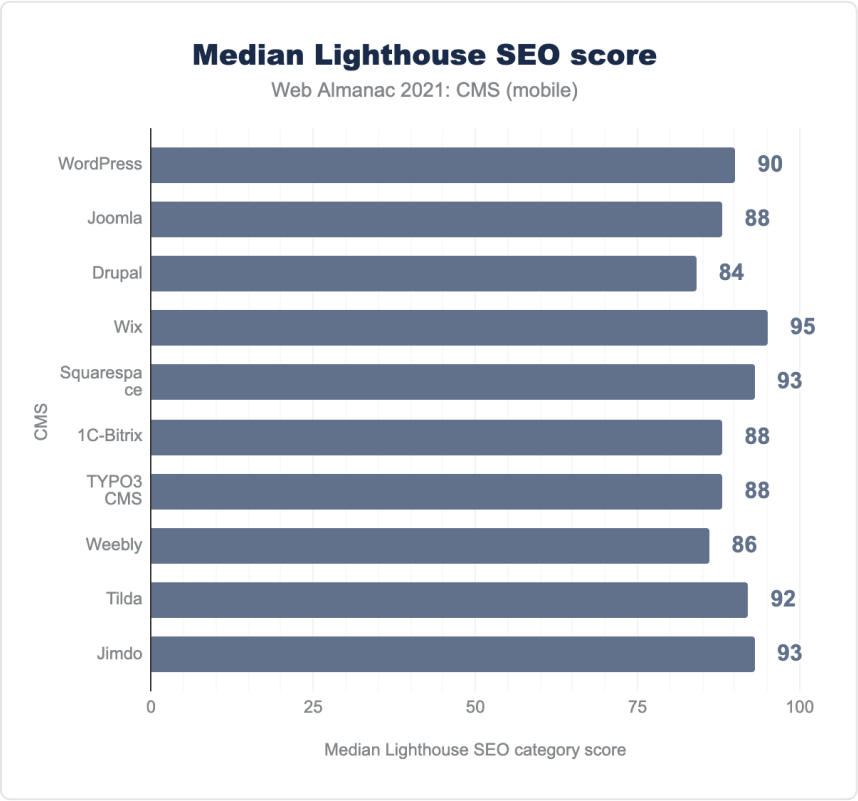


図16.17. CMSトップ10 Lighthouse SEOスコアの中央値。

上位10プラットフォームのSEOスコアの中央値はすべて84を超えており、Drupalのスコアがもっとも低く、Wixのスコアがもっとも高く、中央値は95となっています。

アクセシビリティスコア

アクセシブルWebサイトとは、障がい者の利用できるように設計・開発されたサイトのことです。また、Webアクセシビリティは、低速のインターネット接続環境など、障害のない人々にもメリットがあります。詳しくは、アクセシビリティの章をご覧ください。

Lighthouseはアクセシビリティ監査のセットを提供し、すべての監査の加重平均を返します（各監査の加重方法の詳細については、スコアの詳細[790]を参照してください）。

790. *https://web.dev/accessibility-scoring/*

各アクセシビリティ監査は合格か不合格のどちらかですが、他のLighthouse監査とは異なり、アクセシビリティ監査に部分的に合格してもページにはポイントが加算されません。たとえば、ある要素にはスクリーンリーダーに適した名前がついているが、他の要素にはついていない場合、そのページはスクリーンリーダーに適した名前の監査で0点を取られます。

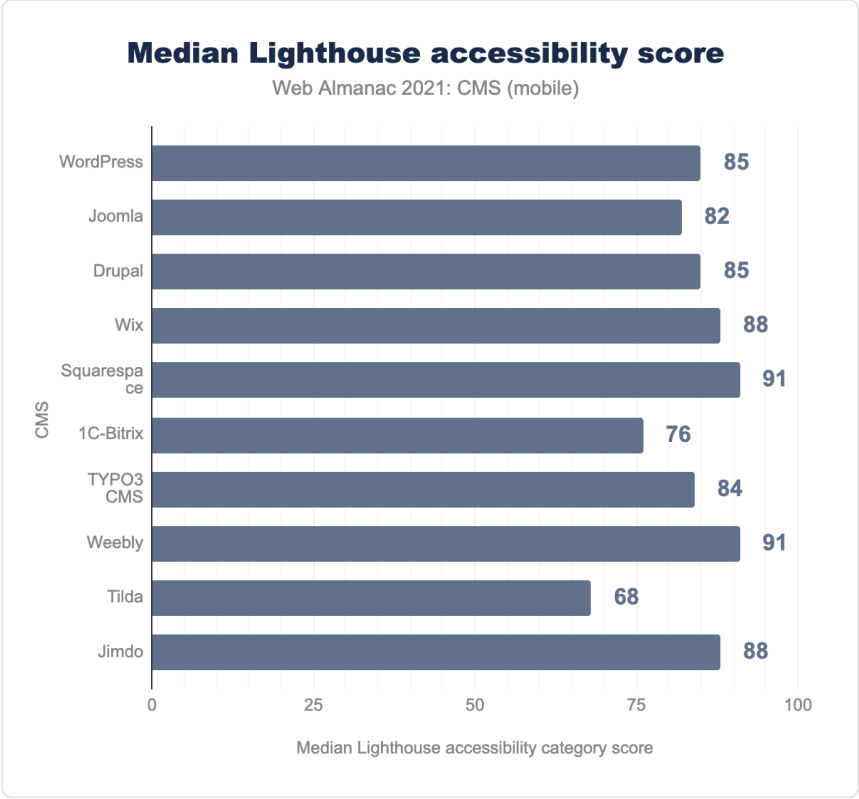


図16.18. CMS上位10社のLighthouseアクセシビリティスコアの中央値。

上位10社のCMSのLighthouseアクセシビリティスコアの中央値は、76から91の間です。SquarespaceとWeeblyのスコアが91ともっとも高く、Tildaはもっとも低いアクセシビリティスコアでした。

ベストプラクティス

Lighthouse ベストプラクティス[791]はHTTPSをサポートしているか、コンソールにエラーが記録されていないかなど、さまざまな異なる指標について、WebページがWebのベストプラク

791. https://web.dev/lighthouse-best-practices/

ティスにしたがっているかを確認しようとするものです。

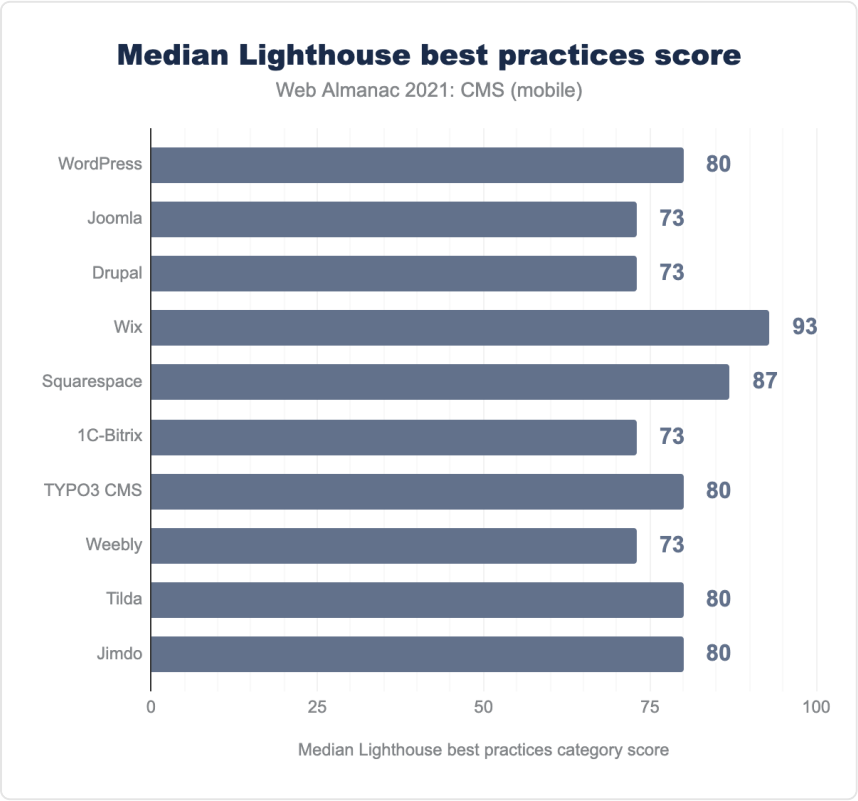


図16.19. CMS上位10社のライトハウスベストプラクティススコア中央値。

Wixはベストプラクティスの中央値が93ともっとも高く、他のトップ10のプラットフォームの多くは73ともっとも低いスコアを共有しています。

リソースの重み

また、HTTP Archiveのデータを使用して、さまざまなプラットフォームで使用されるリソースの重みを分析し、可能性のある機会を明らかにできます。ページの読み込みパフォーマンスは、ダウンロードされたバイト数だけに依存するわけではありません。しかしページの読み込みに必要なバイト数が少なければ、コストや二酸化炭素排出量を削減でき、とくに低速接続の場合はパフォーマンスが、向上する可能性があります。

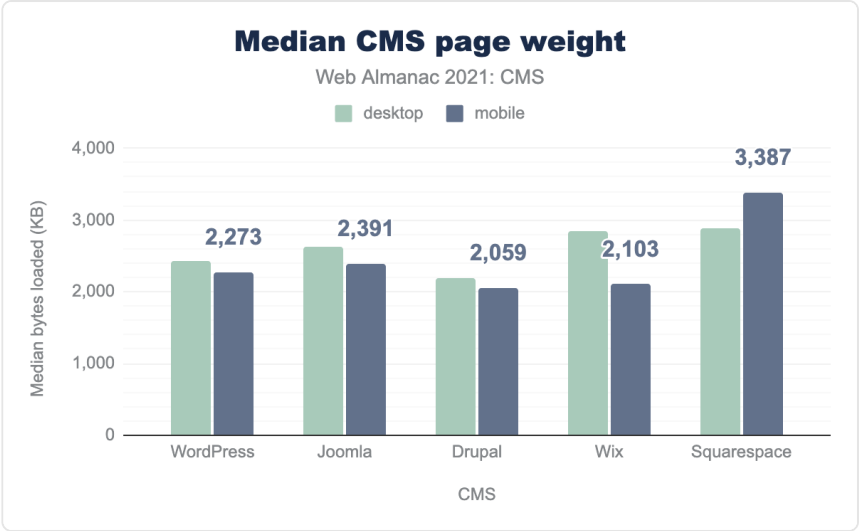


図16.20. CMSのページ重さの中央値トップ5。

上位5つのCMSのページウェイトの中央値は、Squarespaceの3.3MBを除き、ほとんどが2MB程度です。Squarespaceは、デスクトップよりもモバイルでより多くのバイトを提供する唯一のプラットフォームです。

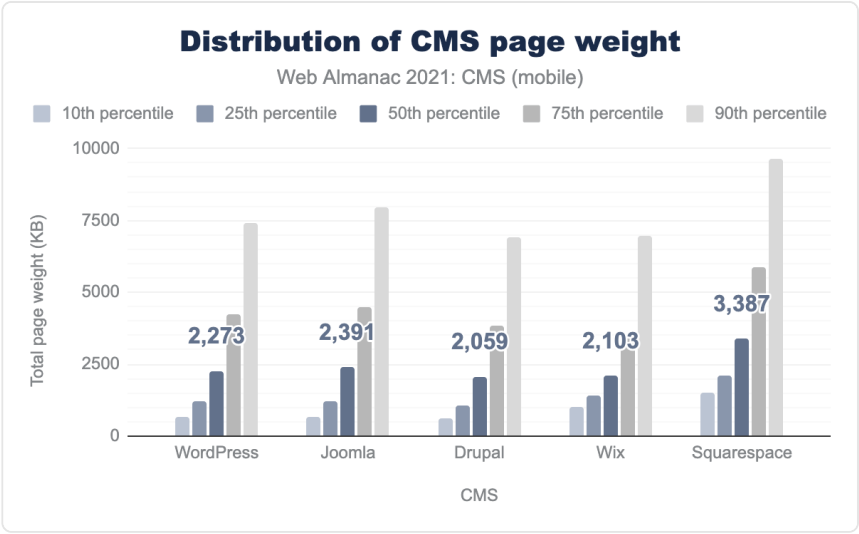


図16.21. CMSのページ重量の中央値トップ5。

各プラットフォームのパーセンタイルにおけるページ重量の分布はかなり大きく、おそらく異なるウェブページ間のユーザーコンテンツの違い、使用されている画像の数、プラグインなどに関連していると思われます。プラットフォームごとに配信されるページがもっとも小さいのはDrupalで、10パーセンタイルの訪問に対して595KBしか送信されません。最大のページはSquarespaceで、90パーセンタイルの訪問で9.6MBが配信されました。

ページ重量の内訳

ページ重量は、使用するリソースの総和です。このリソースサイズの違いを、異なるCMS間で評価することを試みることができます。

画像

通常、もっとも重いリソースである画像は、リソースの重さの大部分を占めている。

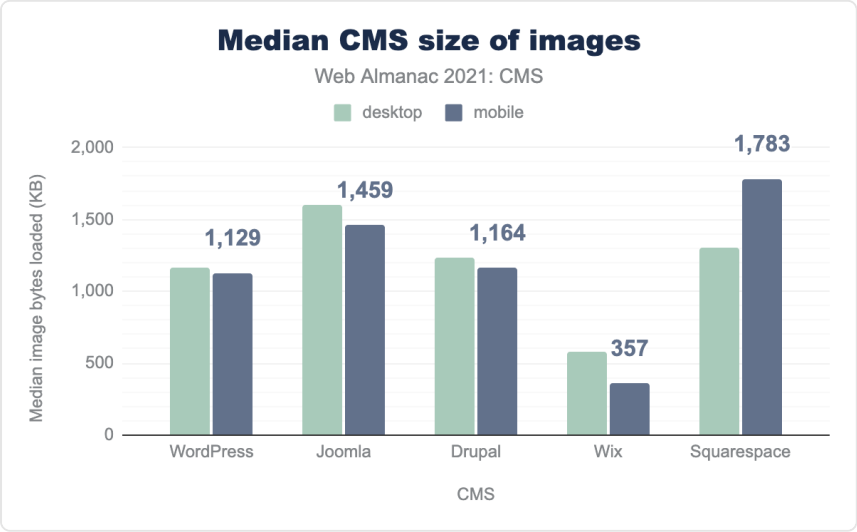


図16.22. 上位5つのCMSの画像重量の中央値。

Wixは、モバイルビューの中央値で357KBと大幅に少ない画像バイトしか配信しておらず、画像圧縮と遅延画像読み込みをうまく利用していることがうかがえます。他の上位5つのプラットフォームはすべて1MB以上の画像を配信しており、Squarespaceは最大の1.7MBを配信しています。

高度な画像フォーマットにより圧縮率が大幅に向上し、リソースの節約とサイトの高速読み込みが可能になりました。WebPは現在、すべての主要なブラウザで一般的にサポートされ

ており、95% サポート[792]されています。さらに新しい画像フォーマットとしてAVIF[793] や、JPEG-XL[794] は、まだ完全ではないが優れた可能性を秘めており、人気が出てきています。

トップクラスのCMSにおける、さまざまな画像フォーマットの使用状況を調べることができます。

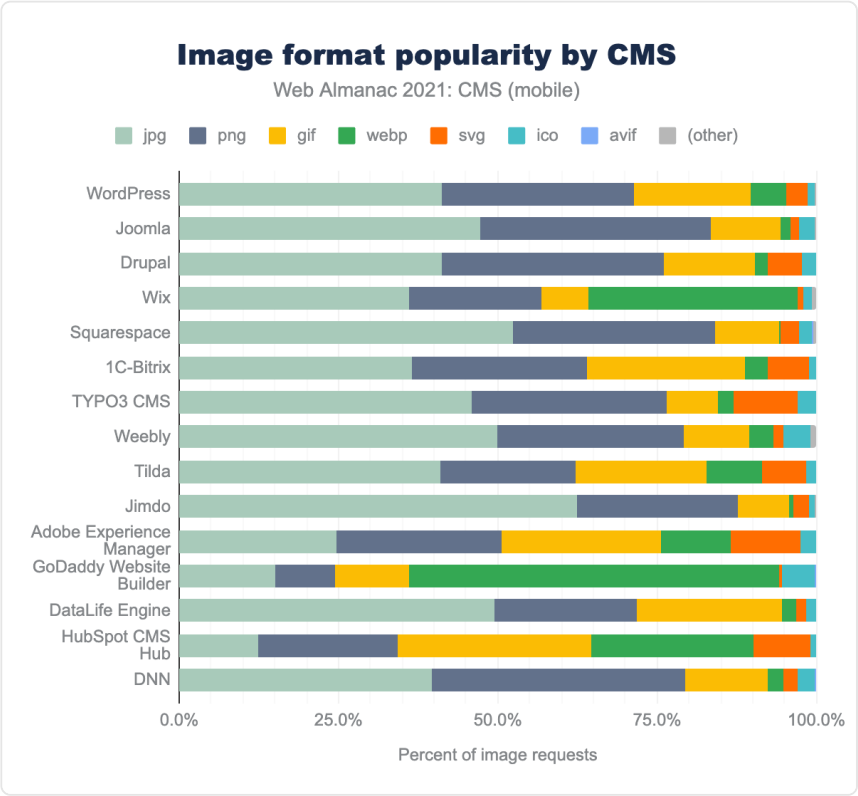


図16.23. CMSの画像フォーマット人気TOP15。

GoDaddy Website BuilderとWixはWebPをもっとも多く使用しており、それぞれ～58%と33%の採用率です。一方、WordPress、Joomla、Drupalはほとんどwebpを提供しておらず、WordPressサイトで提供される画像のうちWebPは～5.7%のみとなっています。AVIFはこれらのプラットフォームではほとんど使用されておらず、すべてのプラットフォームで~0.1% 未満となっています。

WebPのサポートの拡大[795]により、すべてのプラットフォームで、画質を落とさずに適用で

792. *https://caniuse.com/webp*
793. *https://caniuse.com/avif*
794. *https://jpegxl.info/*
795. *https://caniuse.com/webp*

きる古いJPEGやPNG形式の使用を減らすための作業が必要になっているようです。

JavaScript

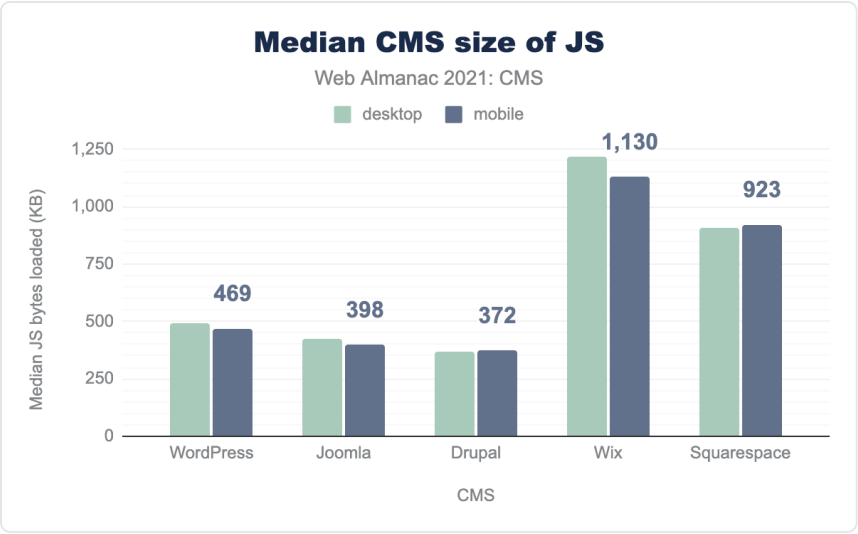


図16.24. CMSのJavaScriptの重さの中央値トップ5。

上位5つのCMSはすべてJavaScriptに依存したページを配信していますが、DrupalのJavaScriptバイト数はモバイルで372KBと少なく、Wixは1.1MB以上ともっとも多いJavaScriptバイトを配信しています。

HTMLドキュメント

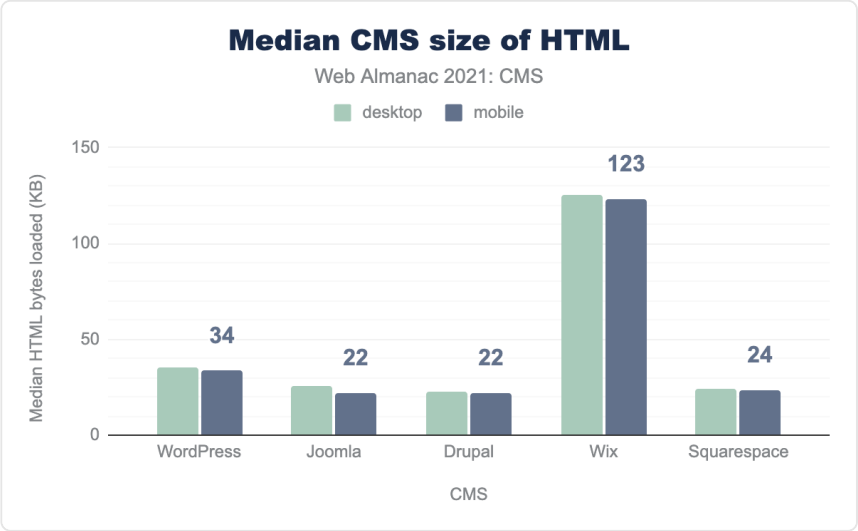


図16.25. CMSのHTML重量の中央値トップ5。

HTMLドキュメントのサイズを調べてみると、Wixが123KBとかなり多くのHTMLを配信している以外は、ほとんどの上位CMSのHTMLサイズの中央値は22KBから34KBであることがわかります。これは、インライン化されたリソースを広範囲に使用していることを示唆しており、さらに改善できる領域であることを示しています。

CSS

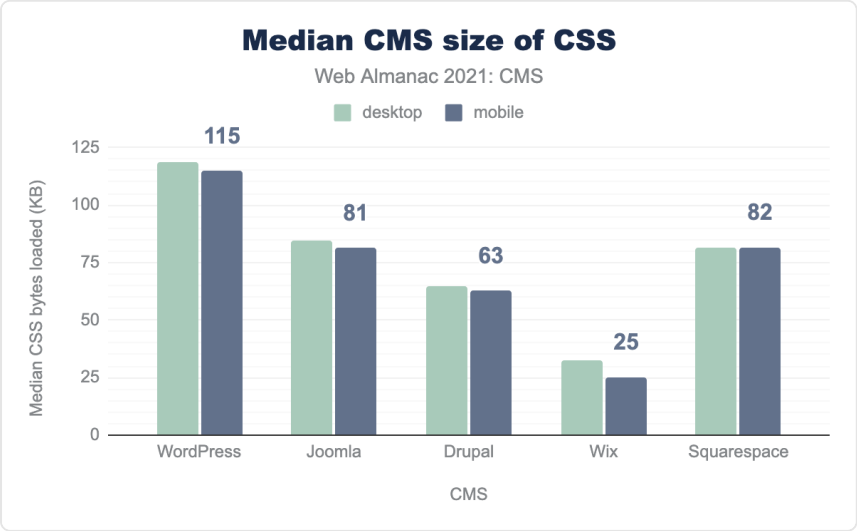


図16.26. CMSのCSS重さの中央値トップ5。

次に、ダウンロードされる明示的なCSSリソースの使用状況を調べます。ここでは、プラットフォーム間で異なる分布が見られ、インライン化アプローチの違いが強調されています。WixはCSSリソースの配信がもっとも少なく、モバイルビューではわずか~25KB、WordPressはもっとも多く、~115KBです。

フォント

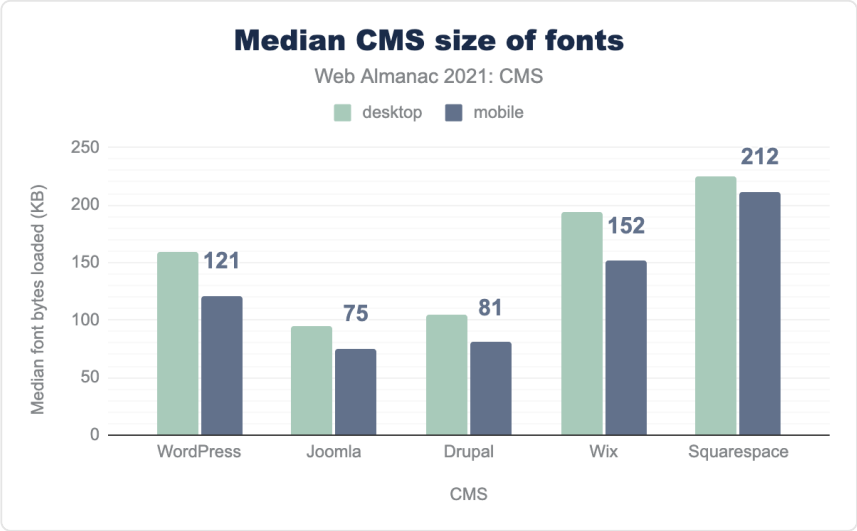


図16.27. CMSの中央値フォントの重さトップ5。

テキストを表示するために、ウェブ開発者は多くの場合、さまざまなフォントを使用することを選択します。Joomlaは、モバイルビューで75KBともっとも少ないフォントバイトを提供し、Squarespaceは212KBともっとも多いフォントバイトを提供しています。

WordPress専用

WordPressは現在もっともよく使われているCMSで、CMSで構築されたサイトのほぼ4分の3がWordPressを使用しているので、さらに議論する必要があります。

WordPressは、2003年から続くオープンソースのプロジェクトです。WordPressで構築された多くのサイトでは、さまざまなテーマやプラグイン、時にはElementorやDiviなどのページビルダーを利用しています。

WordPressコミュニティは、CMSを維持し、カスタムサービスや製品（テーマやプラグイン）を通じて追加機能の必要条件を提供します。このコミュニティは、比較的少数の人々がCMSそのものと、WordPressをほとんどの種類のウェブサイトに対応できるほど強力で柔軟なものにする追加機能の両方を保守しているため、非常に大きな影響を及ぼしています。この柔軟性は、市場シェアを説明する上で重要ですが、WordPressベースのサイトのパフォーマンスに関する議論を複雑にしています。

WordPressコミュニティの貢献者は最近、パフォーマンスの現状を認め、パフォーマンス専門のコア チームを作るという 提案[796] で、平均的なWordPressサイトの現在のパフォーマンスを改善できると期待されています。

採用情報

まず、データセットに含まれるすべてのサイトについて、地域別のWordPressの導入状況を調査しました。

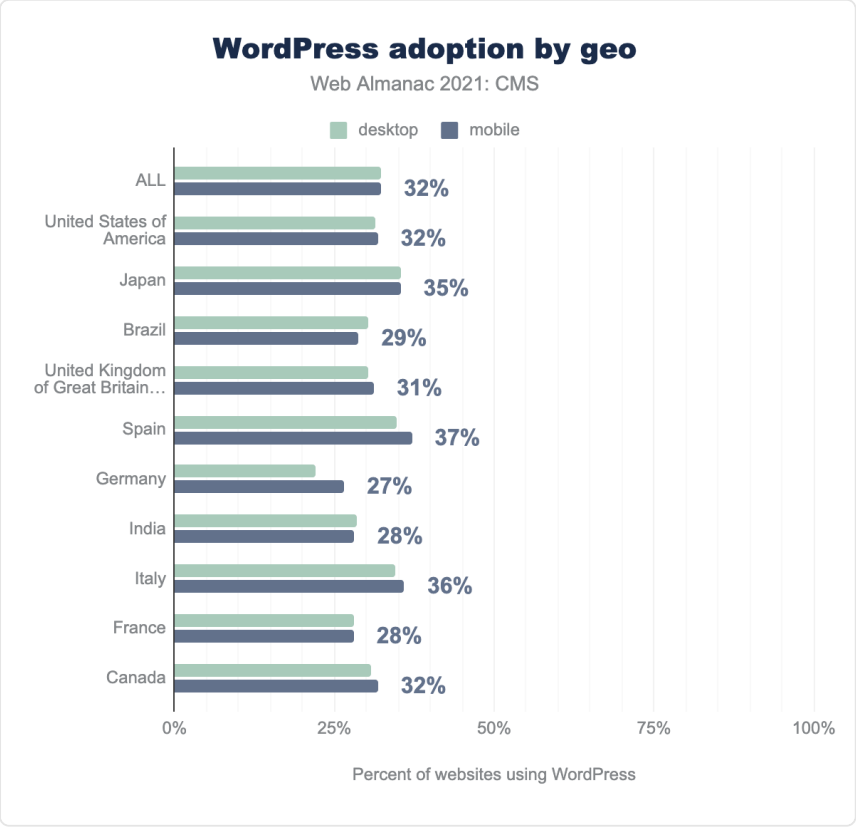


図16.28. WordPressの国別採用状況。

データセットに含まれるサイトの数が多い上位10カ国では、WordPressの導入率は27% 以上でした。スペインはこれらの国の中でもっともWordPressの普及率が高く、モバイルページの37%がWordPressを使用しているのに対し、ドイツでは28%しかWordPressが使用されて

796. https://make.wordpress.org/core/2021/10/12/proposal-for-a-performance-team/

いませんでした。

地域別CWV合格実績

次に、コアWeb・バイタルを通過したWordPressのオリジン量ですが、今回は地域別にモバイルデバイスの内訳を見てみましょう。

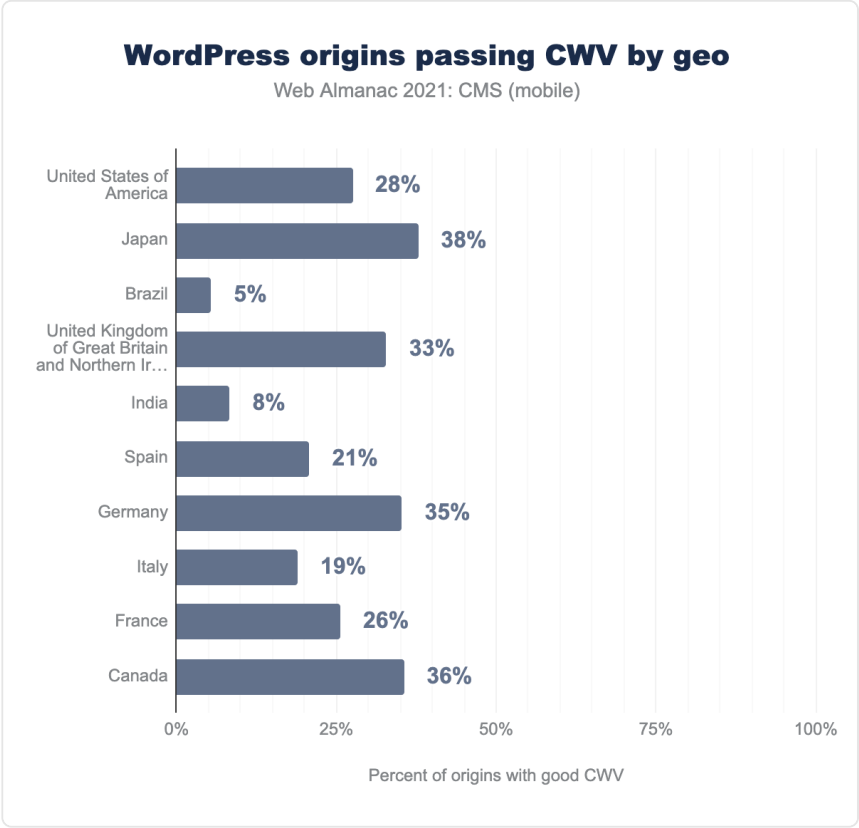


図16.29. CWVを通過するWordPressの起源を地域別に紹介。

すべての国でカウントされたオリジンのうち、WordPressは19%で合格していましたが、WordPressサイトは国によって合格率が大きく異なります。日本では、38%のサイトがモバイル訪問者のための良いCWVを持っていますが、ブラジルでは、わずか5%のサイトが良いCWVを持っています。

これは、コアWeb・バイタルの非常に興味深い見方を露呈しており、異なるプラットフォームのCWVを比較する際の地理的な偏りを示唆しています。CMSが特定の国でしか存在しな

い場合、総計の割合を比較することは公平な比較とは言えません。

デバイスの性能が低く、接続速度が遅い国々を含め、世界中で非常に多くの採用実績を持つWordPressは場合によってはこの比較に苦しむかもしれませんが、おそらくすべての地域で改善の余地があると思われます。一方、CMSはターゲットとする地域で最高の体験を提供するよう努力すべきであり、それは時に、より厳しい条件下でも十分に機能するようサイトを高速化することを意味します。

プラグイン

WordPressサイトが外部リソースをどのように利用しているかを調査し、プラグインやテーマに含まれるリソースと、WordPressコアに同梱されるリソース（wp-includes）に分けました。

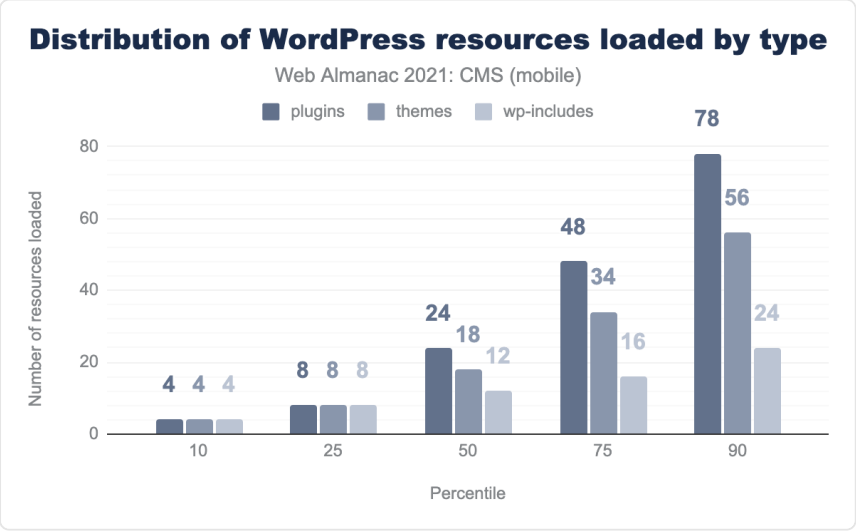


図16.30. WordPressのリソースの種類別負荷分布。

モバイルWordPressページの中央値では、`/plugins/`パスに24個のリソース、`/themes/`パスに18個のリソース、`/wp-includes/`パスに12個のリソースがロードされていることがわかります。90パーセンタイルでは、78個のプラグインリソース、56個のテーマ、24個のwp-includesと、膨大な量のリソースがリクエストされていることがわかります。

WordPressの拡張エコシステムは並外れた柔軟性を提供し、その高い普及率に大きく寄与していると思われます。しかし、多くのプラグインが存在し、多くのリソースに依存しているため、多くの場合、性能に悪影響を及ぼしているように見えます。

結論

CMSプラットフォームは成長を続けており、年々ユビキタスになってきています。とくに、より多くの人々や企業がオンラインプレゼンスを確立する中で、インターネット上で簡単にコンテンツを作成し、消費するために不可欠な存在となっています。

コアWeb・バイタルの導入は、パフォーマンスデータの可視化の進展とともに、ウェブ全体のウェブパフォーマンスへの注目を生み出しました。これらの洞察が、ウェブの現状をよりよく理解し、最終的にウェブをより良い場所にするため役立つことを期待しています。

CMSは素晴らしい仕事をしておりインフラの強化に努め、進化する新しい標準を試し統合しベストプラクティスに従うことで、ウェブ上のユーザー体験を大規模にさらに改善する大きなチャンスを持っています。

一方、コアWebバイタルはまだまだ進化しています。

私たちは、上記の ベターレスポンスメトリクス[797]に向けた考えについて言及しました。さらに、サイト内のページ間の移動をより適切に追跡し、シングルページ・アプリケーション（SPA）とマルチページ・アプリケーション（MPA）[798] のアーキテクチャの違いを考慮する必要があります。

これからも突き進んでいきましょう。

著者



Alon Kochba

@alonkochba alonkochba alonkochba

Alon KochbaはWixのソフトウェア開発者であり、パフォーマンスに関する取り組みを統括しています。バックエンド出身で、ネットワーキングの豊富な経験を持ち、ウェブを大規模に高速化することを楽しんでいます。

797. https://web.dev/responsiveness/
798. https://web.dev/articles/vitals-spa-faq

部 III 章 17
Eコマース



Tom Robertshaw によって書かれた。
Rockey Nebhwani、 Alan Kent、 Manuel Garcia と Fili Wiese によってレビュ。
Rajiv Ramnath による分析。
Shaina Hantsis 編集。
Sakae Kotaro によって翻訳された。

序章

この章では、Web上のeコマースの状況を確認します。eコマースサイトとは、物理的またはデジタルな製品を販売する「オンラインストア」です。オンラインストアを構築する場合、いくつかのタイプから選択できます。

- **ソフトウエア・アズ・ア・サービス (SaaS)** Shopifyのようなプラットフォームは、オンラインストアを開設し、管理するために必要な技術的知識を最小限に抑えることができます。これは、コードベースへのアクセスを制限し、ホスティングを心配する必要をなくすことによって実現されています。
- **プラットフォーム・アズ・ア・サービス (PaaS)** Adobe Commerce (Magento)などのプラットフォームは、最適化されたテクノロジースタックとホスティング環境を提供しながらも、コードベースへのフルアクセスを可能にします。

- **セルフホスティング** WooCommerceなどのプラットフォーム
- また、CommerceToolsのような「APIアズ・ア・サービス」である**ヘッドレス**プラットフォームも存在します。同社は、eコマースのバックエンドをSaaSとして提供し、小売業者はフロントエンド体験の構築とホスティングを担当します。

プラットフォームは、これらのカテゴリのうちの1つ以上に分類される場合があることに注意してください。たとえば、Shopwareには、SaaS、PaaS、セルフホスティングのオプションがあります。

プラットフォーム検出

私たちは、Wappalyzer[799] というオープンソースのツールを使って、Webサイトが使用している技術を検出しました。コンテンツ管理システム、eコマースプラットフォーム、JavaScriptのフレームワークやライブラリなどを検出できます。

今回の分析では、以下のいずれかに該当する場合は、eコマースサイトであると判断しました。

- 既知のeコマース・プラットフォームの使用（制限事項参照）
- オンラインストアを示唆する技術の使用（例：Google Analytics Enhanced Ecommerce[800]）。

方法論について詳しく説明しています。

制限事項

この方法論には、その精度に影響を与えるいくつかの限界があります。

まず、ECサイトを認識する能力に限界があります。

- Wappalyzerは、eコマースプラットフォームを検出したのでしょう。
- PayPalのような支払いプロセッサの検出は、ウェブサイトをeコマースとみなすには不十分でした。これは、オンラインショップではないオンライン決済を受け付けるサイト（例：B2B SaaS）が存在するためです。
- eコマースプラットフォームがウェブサイトのサブディレクトリ内にホストされ

799. *https://github.com/AliasIO/wappalyzer/*
800. *https://developers.google.com/tag-manager/enhanced-ecommerce*

ている場合、ホームページのみが分析対象となるため、検出することはできません。

- ヘッドレス実装は、使用中のプラットフォームを検出する能力を低下させます。eコマースプラットフォームを検出する主な方法の1つは、共通のHTMLまたはJavaScriptコンポーネントを認識することです。そのため、ECプラットフォームのフロントエンドを使用しないヘッドレスウェブサイトは、ECとして検出することが難しくなります。

次に、指標や解説の正確性は、以下のような制約にも影響される可能性があります。

- このような傾向は、検出精度の変化に影響されたものであり、業界の動向を完全に反映したものではない可能性があります。たとえば、あるeコマース・プラットフォームは、検出方法が改善されたため、より普及したように見えるかもしれません。
- すべてのウェブサイトのリクエストは、米国から行われました。ウェブサイトが地理的な位置に基づいてより適切なウェブサイトにリダイレクトされる場合、最終的な位置が分析されます。
- クロールしたサイトは、Chromeブラウザーのユーザーが訪問するウェブサイトに偏りがあるChrome UX Reportからのものです。

Eコマースプラットフォーム

我々の分析では、モバイルとデスクトップのウェブサイトを考慮しました。これらのサイトは、Chromeユーザーが積極的に訪問しているサイトです。詳しくは、方法論をご覧ください。訪問したウェブサイトのほとんどは両方の結果セットに含まれていますが、中には片方だけに含まれているものもあります。モバイルとデスクトップの統計情報を共有することが多いです。変動が少ない場合は、どちらか一方のみを表示することもあります。この場合、とくに断りのない限り、モバイルの指標のみが表示されます。

モバイルの分析では、750万サイトから回答を得て、そのうち150万サイト（19.5%）が何らかの形でeコマース機能を有していることがわかりました。同様にデスクトップ分析では、630万サイトから回答を得て、130万サイト（20.2%）が、eコマースであることが判明しています。

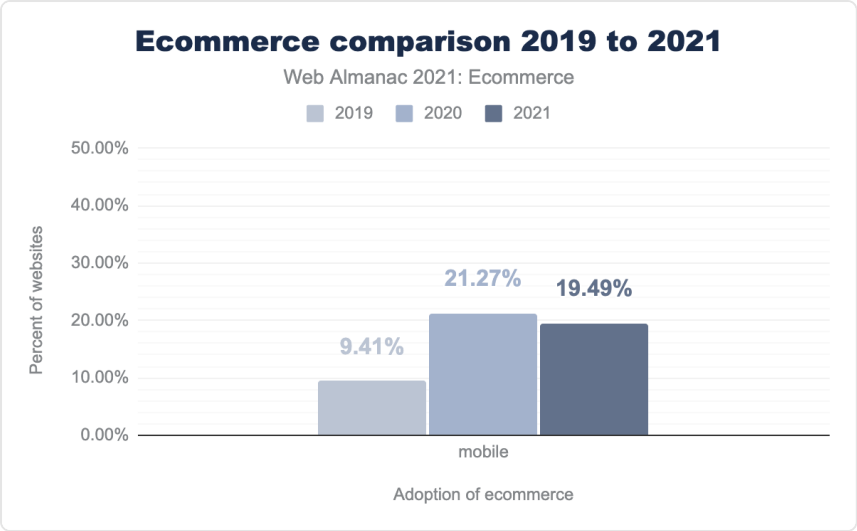


図17.1. Eコマース比較2019年〜2021年。

eコマースサイトの全体のシェアは、21.3%（デスクトップは21.7%）であった昨年のレポートに比べ、モバイルでは1.8%縮小した（デスクトップは1.6%）。一方、eコマースサイトの数は依然として増加しており、昨年と比較して今年はデスクトップで4.5%増加（モバイルでは8.3%）しています。しかし、この増加は、Chromeユーザーが訪問するサイトのリスト全体の増加には追いつきませんでした。

モバイルサイトの9.45%がeコマースだった2019年の結果[801]と比較すると、ここ1年の変化は僅少ですが、ここ2年は劇的かつ持続的に増加していることが分かります。

しかし、これをCOVID-19に反応して電子商取引が拡大した証拠と考えるべきではないでしょう。昨年も報告[802]しましたが、この増加は、eコマース・プラットフォームを検出する能力が向上したことによるものです。プラットフォームカバレッジの拡大から、Google Analytics Enhanced Ecommerceの存在など、サイトがeコマースであることを示す二次的なシグナルも使用するようになりました。

トップeコマース・プラットフォーム

当社の分析では、215のEコマース・プラットフォームが検出され、昨年の145と比較して48%の増加となっています。しかし、デスクトップとモバイルの両方で0.1%以上利用されているプラットフォームは、わずか10にとどまっています。

801. *https://almanac.httparchive.org/ja/2019/ecommerce#プラットフォーム検出*
802. *https://almanac.httparchive.org/ja/2020/ecommerce#Eコマースプラットフォーム*

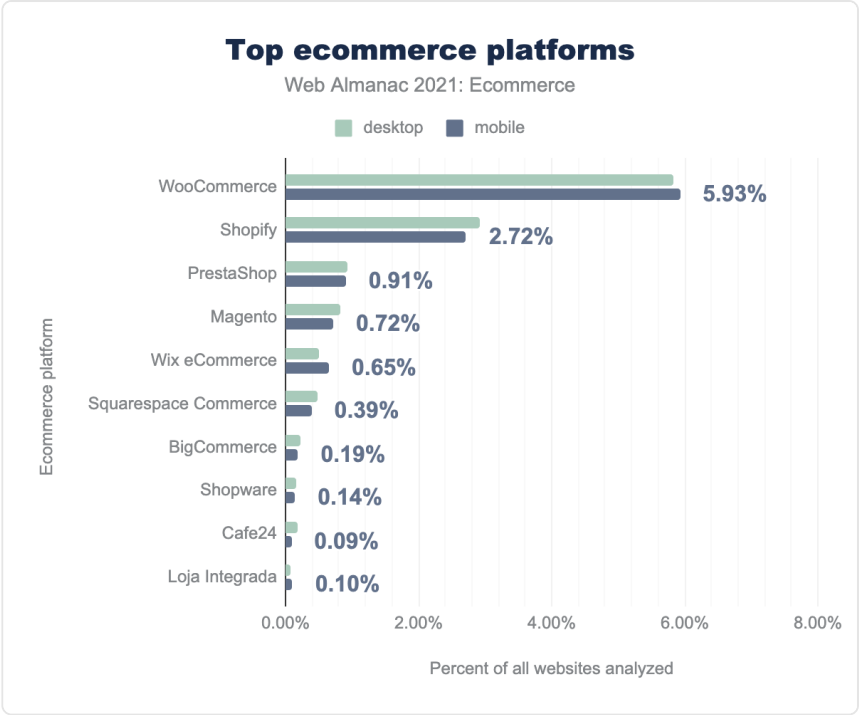


図17.2. トップクラスのeコマース・プラットフォーム

WooCommerce[803]は、WordPress[804]のプラグインで、全ウェブサイトの約6%が使用しているもっとも普及しているeコマース・プラットフォームです。これは、モバイルでのeコマース市場の30%に相当します。

Shopify[805]はSaaS型ソリューションで、WooCommerceの約半分のサイト数があり、2番目に普及しているソリューションです。モバイルでのeコマース市場のシェアは14%です。PrestaShop[806]はオープンソースのプラットフォームで、WooCommerceの6分の1程度の普及率で、3番目に使われているプラットフォームです。

上位10社のうち4社は、オープンソースやセルフホスティングのエディションを持っています。WooCommerce、PrestaShop、Magento[807]、Shopware[808]の4種類。プラットフォームのバージョンの違いを検知しないため、MagentoやShopwareのオープンソース版と商用版の区別がつきません。

803. https://woocommerce.com/
804. https://wordpress.org/
805. https://shopify.com/
806. https://www.prestashop.com/
807. https://magento.com/
808. https://www.shopware.com/

10のプラットフォームのうち6つがSaaSである（またはSaaS版がある）。Shopify、Wix eCommerce[809]、Squarespace Commerce[810]、BigCommerce[811]、ShopwareとLoja Integrada[812]。

注：2021年7月のHTTP Archiveデータに、OpenCart[813]のサイト数が過小に報告される問題[814]が発生しました。9月の結果では、10,801のOpenCartサイトが検出されたことは認めるに値します。もし7月に同数のOpenCartサイトが検出されていたとしたら、人気の点ではBigCommerceとShopwareの間に位置することになります。

ウェブサイトの人気順で上位のeコマース・プラットフォーム

今年は、Chromeユーザーエクスペリエンスレポート[815]が各ウェブサイトの人気ランキングを提供しました。これにより、市場のさまざまなセグメントにおける人気度によって、上位のeコマース プラットフォームを分類することができました。”All“は、モバイル向けプロファイリング750万サイト、デスクトップ向け630万サイトすべてを指します。

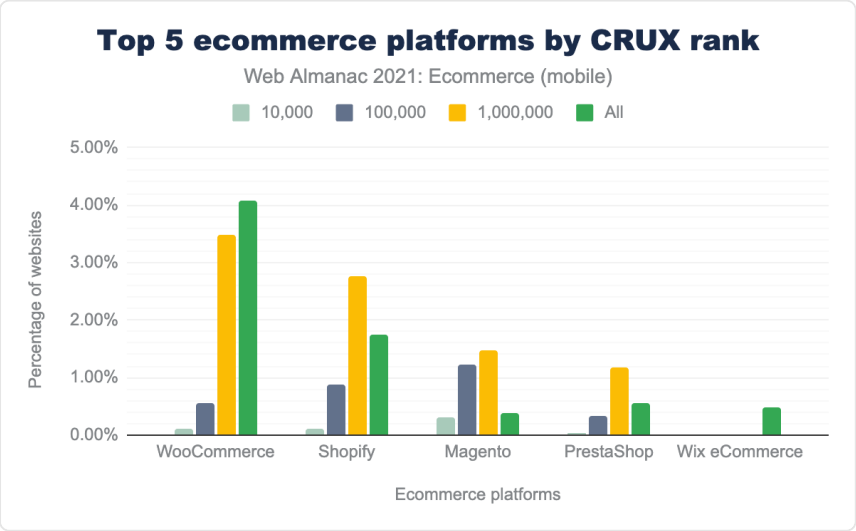


図17.3. CRUXランクによるeコマースプラットフォームのシェアトップ5

ウェブサイトがランク付けされることで、市場のさまざまなセグメントでプラットフォームの人気がどのように変化するかを観察できます。

809. https://www.wix.com/ecommerce/website
810. https://www.squarespace.com/ecommerce-website
811. https://www.bigcommerce.com/
812. https://lojaintegrada.com.br/
813. https://www.opencart.com/
814. https://github.com/HTTPArchive/httparchive.org/issues/414
815. https://developers.google.com/web/tools/chrome-user-experience-report/

- WooCommerceは、全体でもっとも普及しているECプラットフォームで、上位100万位以内に入っています。
- Shopifyは、分析したすべてのサイトと比較して、上位100万位以内（割合として）のサイトにおいてより人気があります。
- 上位10,000サイトのうち、Magentoは5つの中でもっとも普及している。
- WixのeコマースサイトIは、上位10万位以内に確認されませんでした。100万位以内では、モバイルの164サイトのみ確認されました。WixのEコマースサイトのほぼ全体が、100万位以下のサイトであることがわかります。

上位100万サイト

この結果を見るもう1つの方法は、各階層のランキングの中でもっとも普及しているプラットフォームを考えることです。たとえば、上位10,000サイトと上位100万サイトでは、異なる傾向が見られると予想されます。

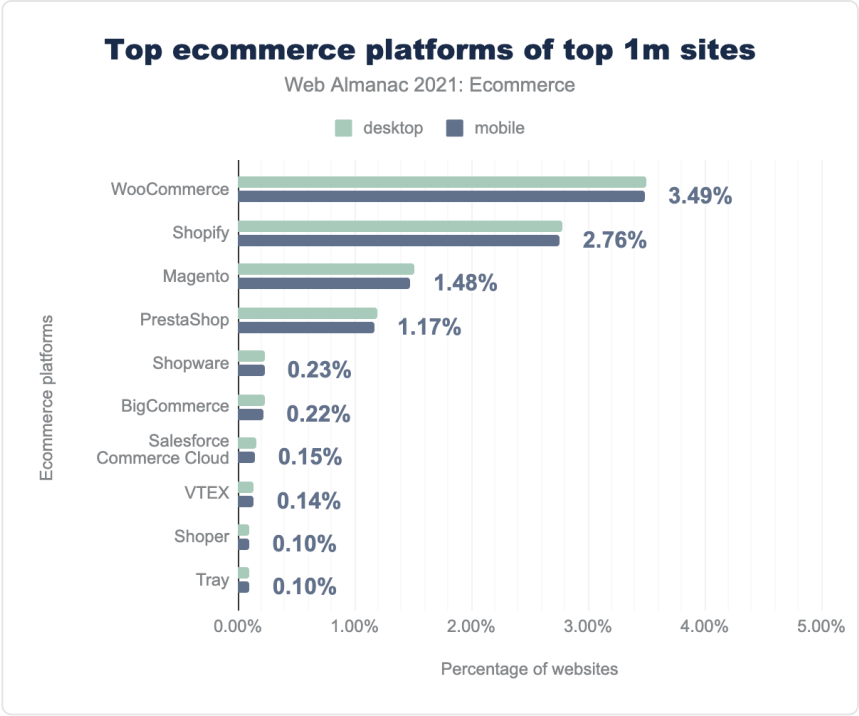


図17.4. 100万サイトのトップeコマース・プラットフォーム

上位100万サイトでは、WooCommerceとShopifyがそれぞれモバイルでのリクエストの3.49%と2.76%を占め、依然として主要なプラットフォームとなっています。しかし、分析したすべてのサイトと比較すると、両者の差はかなり小さくなっています。モバイルでの全サイトリクエストのうち、WooCommerceはShopifyの2倍以上であったのに対し、上位100万サイトでは25%増にとどまっています。

また、MagentoがPrestaShopを抑えて3位となったことも確認できます。Wix eCommerceとSquarespace eCommerceは上位7つのプラットフォームから外れています。代わりに、Shopware、BigCommerce、そしてSalesforce Commerce[816]が先行していることが分かります。

上位10万サイト



図17.5. 上位10万サイトのトップeコマース・プラットフォーム

CrUXランクによる上位10万サイトを考慮すると、その様相はかなり大きく変化します。MagentoはモバイルサイトのIn 1.21%を占め、もっとも普及しているeコマースプラットフォー

816. *https://www.salesforce.com/uk/products/commerce-cloud/overview/*

ムベンダーとなりました。Shopifyは2位（0.88%）を維持し、Salesforce Commerce Cloudは3位（0.63%）となっています。SAP Commerce Cloud[817] はリーダーボードを6位に上げ、この分野でのエンタープライズ・プラットフォームの競争力が高まっていることを示しています。

上位10,000サイト

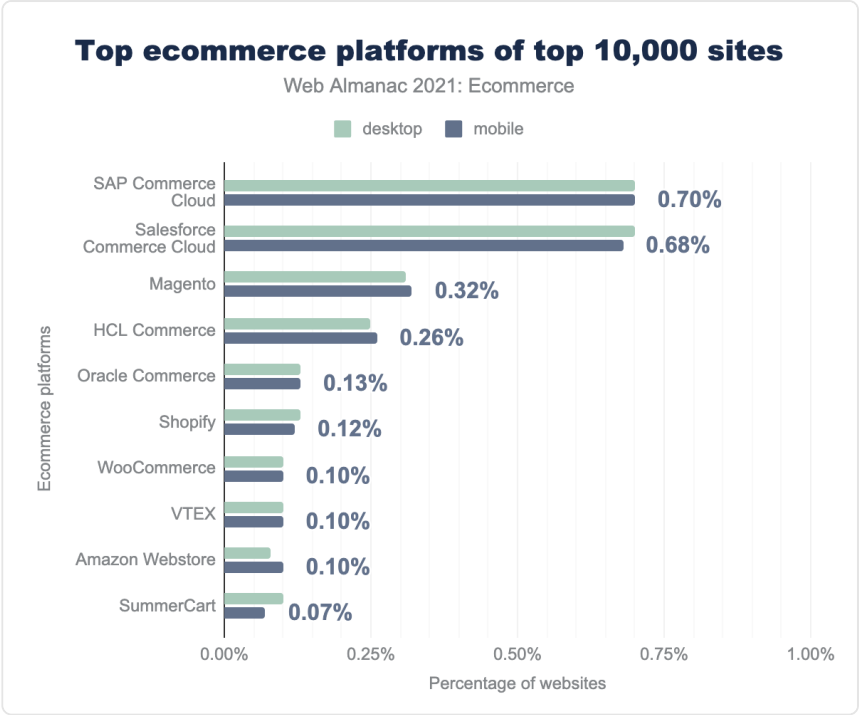


図17.6. 上位10,000サイトのトップeコマース・プラットフォーム

上位10,000サイトにおけるeコマースプラットフォームを搭載したサイトのシェアは、明らかに小さくなっています。

Salesforce Commerce CloudとSAP Commerceは、同程度の数のeコマースサイト（モバイルではそれぞれ0.70%と0.68%）をリードし、その力を発揮しています。

リーダーボードを見下ろすと、この分野ではほとんどサプライズはありません。上位2位から大きく離れているのはMagento（Adobe製品）で、上位10,000サイトでのシェアは0.32%

817. *https://www.sap.com/uk/products/commerce-cloud.html*

です。それに続くのが、HCL Commerce[818]（以前はIBM WebSphere Commerceとして知られていた）、Oracle Commerce[819]です。これらのプラットフォームは、いずれも大企業に適していると一般的に考えられています。

COVID-19の影響

年度をまたいで発見されたECサイトの総数を比較することは困難です。前述したように、ECサイトかどうかを検出する能力が大幅に向上したためです。Google Analytics Enhancedコマースインテグレーションのような二次的なシグナルの使用によるところもあります。

そこで、代わりに昨年のレポートでは、少数のプラットフォームに焦点を当て、その利用状況がどのように変化したかを確認しました。2020年前半の初期の兆候として、ShopifyとWooCommerceの利用が測定可能で顕著に増加していることが分かりました。Magentoなどの他のプラットフォームが同じような伸びを見せなかったのに対し、2020年1月から2020年7月にかけて20%台の伸びを見せました。これらのプラットフォームは参入コストの低さや使いやすさで知られていますがMagentoは、そうではありません。

2021年に向けて、世界中の人々や企業は適応を続けています。2020年の米国における電子商取引は、商務省の報告書[820]によると、32.4%の収益成長率を示しました。英国では、Office of National Statistics報告[821]が46%の伸びを示しています。

818. *https://www.hcltechsw.com/commerce*
819. *https://www.oracle.com/uk/cx/ecommerce/*
820. *https://www.digitalcommerce360.com/article/coronavirus-impact-online-retail/*
821. *https://internetretailing.net/industry/industry/ecommerce-grew-by-46-in-2020---its-strongest-growth-for-more-than-a-decade--but-overall-retail-sales-fell-by-a-record-19-ons-22603*

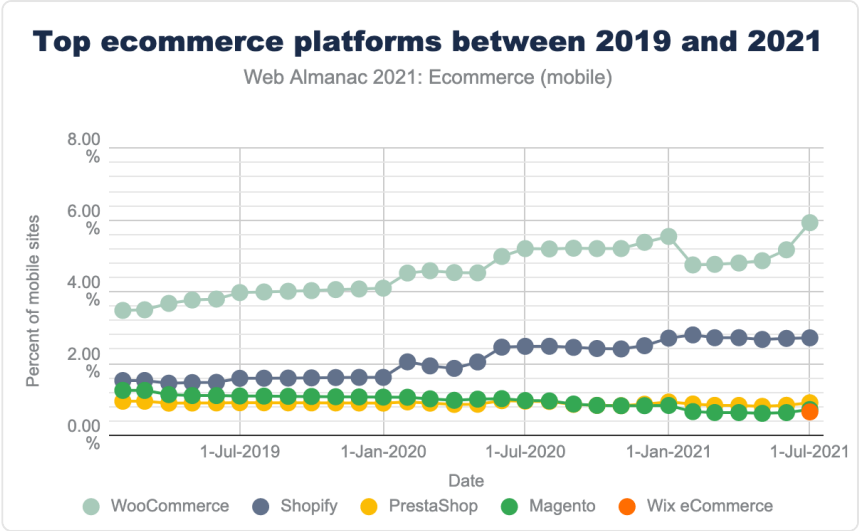


図17.7. Eコマースプラットフォームの成長、Covid-19の影響

また、2019年2月から2021年7月までの間、月単位で結果を見ることもできる。ただし、結論を出す前に、プラットフォームの検出問題がシェアの変化の原因になっていることがあることに注意しなければならない。具体的な問題としては、2021年2月から6月にかけてのWooCommerceの市場シェアの低下があり、これはバグ[822]と認定されました。）

それを考慮すると、やはりモバイルでは注意すべきかもしれません。

- WooCommerceは3.48%から5.93%に成長した。この成長の大部分は、欧米諸国が実施したCOVID-19規制の直後に発生したものです。
- Shopifyの成長率は2020年に大きく上昇し、同年に1.61%から2.50%に成長しました。しかし、この成長率は持続していない。
- また、この間、以前はShopifyと競合していたMagentoがPrestaShopを下回るようになったことがわかる。全サイトのシェア1.25%から0.72%に移行。

筆者の視点では、中小企業がECチャネルを追加する初動が早かったと思います。これは、WooCommerceやShopifyといった費用対効果が高く使いやすいプラットフォームの利用により、2020年前半にほぼ達成されました。

しかし、報告されたオンライン収益の増加の大部分は、すでにeコマースに対応していた企業が恩恵を受けたものであると予想されます。

822. https://github.com/HTTPArchive/almanac.httparchive.org/issues/1843

Eコマースでのユーザー体験

eコマースサイトの目的は、収益を上げることです。この目的を達成するために、企業は複数の戦略を採用します。たとえば、幅広い購買行動を考慮した機能豊富な体験を提供することが挙げられます。また、ウェブサイトを可能な限り高速化することも必要でしょう。この2つの戦略が目的に対してどのように作用するかは明らかですが、同時に互いに作用することもあります。

後ほど、機能豊富な体験を実現するためのツール＆タクティクスをご紹介します。

まず、サイトの技術的な品質とパフォーマンスを評価します。どちらか一方を決定的に評価できる単一の指標やツールはないため、複数の指標を用いました。

- Google Lighthouse
- Chrome UXレポートから見るコアウェブバイタル
- WebPageTest

Lighthouse

ウェブページの技術的な品質を測定する方法の1つに、Google Lighthouse[823]を使用する方法があります。lighthouseテストでは、5つのカテゴリーごとに100点満点のスコアが提示されます。下図は、リクエストされたすべてのEコマースサイトの各カテゴリーのスコアの中央値を示しています。

823. *https://developers.google.com/web/tools/lighthouse/*

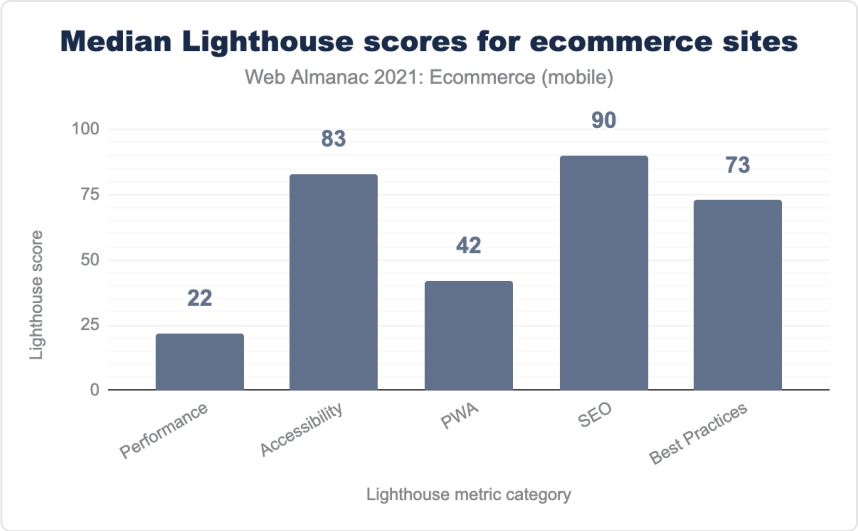


図17.8. EコマースサイトのLighthouseスコア中央値

ここでもっとも重要なポイントは、eコマースサイトがパフォーマンスに関して良い灯台のスコアを獲得するのに苦労していることです。これは、このカテゴリーで良いスコアを獲得するためには、より大きなレベルの努力が必要なためと思われます。

Lighthouseのプラットフォーム別スコア

Lighthouseのスコアをeコマースプラットフォームのベンダー別に分類すると、比較的ばらつきが少なくなっています。このことは、各ECプラットフォームが、これらの分野において、すぐに使える同様の機能を提供していることを示唆しています。

パフォーマンス

パフォーマンスは、システムの創発的な特性であり、新機能のように実装できるようなものではありません。新しい機能を追加するように実装すればよいというものではありません。単純に考えれば、機能を増やせば増やすほど遅くなるということです。

同時に、サイトが高速だとコンバージョン率の向上につながることは、もはや常識です。では、なぜEコマースサイトのパフォーマンススコアがこれほど低いのでしょうか？その理由の1つは、サイトスピードや会話率の統計が、eコマースビジネスが直面する意思決定を考慮せずに常に提供されていることかもしれません。毎年、収益の拡大が求められると、収穫逓減の法則でも、コンバージョン率の向上はスピードの向上だけでは対応できなくなります。これは、eコマース体験に対する消費者の高い要求と相まって、より多くの機能が優先され

る状況になっています。

さらに、機能を含めるかどうかの決定には、多くの場合、より多くのニュアンスがあります。たとえば、ライブチャットウィジェットの利点は、パフォーマンスへの影響を上回るかどうか？答えは、文脈によって変わるのでしょうか？それが遅延ロードされていることを確認するために、開発者がそれをインストールするのを待つべきでしょうか、それとも、Googleタグマネージャーを使用するだけでしょうか？他のもののためにその開発時間を使用しないことの機会費用はいくらですか？

パフォーマンスを別の見方をすれば、コモンズの悲劇パラダイム[824]に苦しむ共有リソースであるということです。プロジェクトのスタート時には最高レベルにあり、時間の経過とともに、消費する権利を持つさまざまなステークホルダーからの要求で枯渇します。

最良の結果を得られるのは、サイトのスピードとユーザー体験のバランスを取ることができる企業でしょう。そのような企業は、最初のページロードにおける機能の影響を最小限に抑えつつ、優れたユーザー体験を提供できます。

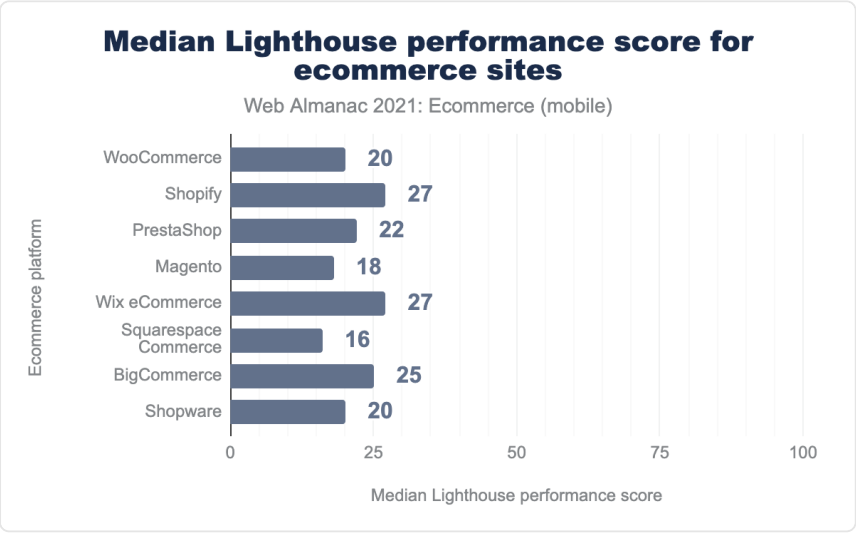


図17.9. LighthouseのECサイトパフォーマンススコアの中央値

プラットフォーム間でもっとも差があったのは、パフォーマンススコアでした。ShopifyとWix eCommerceはもっともパフォーマンスが高く、モバイルでのライトハウスパフォーマンススコアの中央値は27/100でした。もっとも低いスコアは、Loja Integradaの6/100、Squarespace Commerceの16/100、そしてMagentoの18/100でした。繰り返しになりますが、これらはすべて悪いスコアです。

824. https://www.investopedia.com/terms/t/tragedy-of-the-commons.asp

Shopifyは最近、すべての新しいマーケットプレイスのテーマで、Lighthouseの平均パフォーマンススコア60/100を達成するという要件を追加[825]したことを高く評価しています。このことが、今後の分析結果にどのような影響を与えるか、興味深いところです。

アクセシビリティ

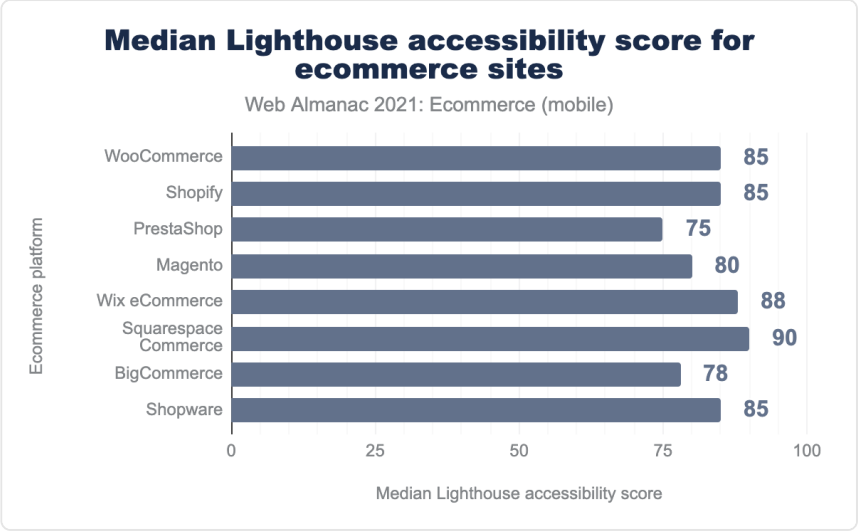


図17.10. EコマースサイトのLighthouseアクセシビリティスコアの中央値

上位8つのプラットフォームは、アクセシビリティの中央値で非常によく似たスコアを出しています。また、アクセシビリティに関する法律や認知度が向上するにつれて、さらに改善されることが期待されます。

プラットフォームが標準テーマのアクセシビリティを向上させることで改善されるかもしれません。たとえば、BigCommerceでは、デフォルトのテーマを更新[826]して、ウェブサイトコンテンツのアクセシビリティ[827]ガイドライン（かWCAG）2.1 Level AA標準に適合させています。

また、プラットフォームは、より広範なアプリやテーマのコミュニティに対して、高水準の技術的品質を提供するように促すことができます。Shopify[828]は、新しいマーケットプレイスのテーマに対して、Lighthouseアクセシビリティスコアの最低要件を発表しました。

ウェブ全体のアクセシビリティ・スコアに関するより詳細な調査については、アクセシビリ

825. *https://shopify.dev/themes/store/requirements*
826. *https://support.bigcommerce.com/s/blog-article/aAn4O0000000CdJDSA0/improvements-to-accessibility-coming-in-cornerstone-52?language=en_US*
827. *https://www.w3.org/WAI/standards-guidelines/wcag/#intro*
828. *https://www.shopify.com/partners/blog/theme-store-accessibility-requirements*

ティの章をお読みください。

PWA

すべてのEC事業者にとって、PWA対応は優先順位が低いようです。その理由は2つ考えられるかもしれません。

- ホーム画面に追加するようなPWAの機能が消費者に採用されているかどうかの調査はほとんど行われていません。
- iOSのSafariはPush Notification APIやホーム画面にPWAを追加する機能をサポートしていない。iOSの市場シェアが大きいため、PWAへの投資の見返りが少なくなっています。

ベストプラクティス

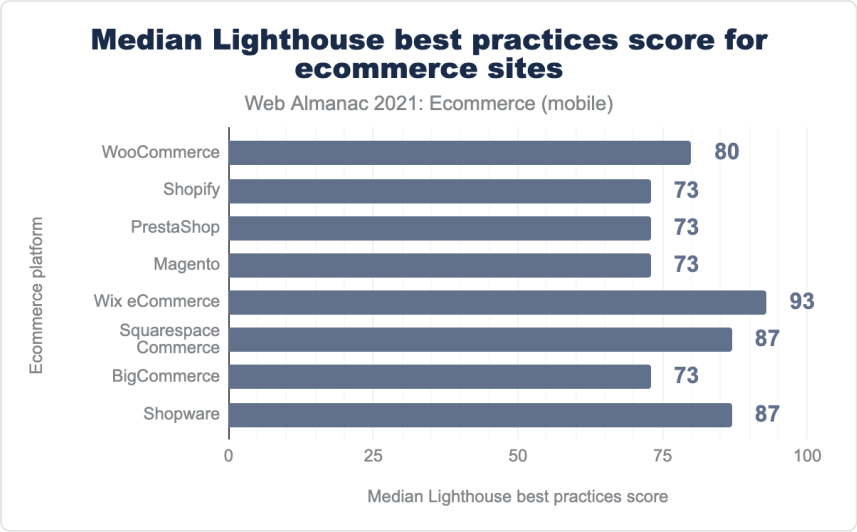


図17.11. ライトハウスのベストプラクティスのスコア（中央値）（eコマースサイトの場合

Wix Ecommerceは、ライトハウスのベストプラクティススコアの中央値で93/100ともっとも高いスコアを獲得しています。小規模ビジネスに特化しているため、平均してよりシンプルなユーザー体験を提供している可能性がありますが、これほど高いスコアを獲得したことは印象的です。

コアウェブ・バイタル

2020年、Googleはコアウェブ・バイタル（CWV）という名称で、ウェブサイトの所有者と開発者が、優れたユーザー体験のために重要な3つのパフォーマンス指標に集中できるよう取り組みを開始しました。これらの指標は次のとおりです。

最大のコンテントフルペイント[829] (LCP)

- *ローディング*のパフォーマンスを測定します。良いユーザー体験を提供するために、LCPはページが最初にロードされ始めてから2.5秒以内に発生する必要があります。

最初の入力までの遅延[830] (FID)

- *インタラクティビティ*を測定します。良いユーザー体験を提供するために、ページのFIDは100ミリ秒以下であるべきです。

累積レイアウトシフト[831] (CLS)

- *視覚的*な安定性を測定します。良いユーザー体験を提供するために、ページのCLSは0.1以下を維持する必要があります。

コアウェブバイタルがGoogleの検索アルゴリズムにおけるランキング要因[832]となったことで、eコマース事業者からの注目度が高まっています。

Chromeユーザー体験レポートは、実際のユーザーからこれらの指標を収集できます。そのため、制御された環境でページロードをシミュレートする従来の「ラボ」テストと比較して、より正確な結果が得られると考えられます。

このセクションでは、LCP、FIP、CLSの3つの指標すべてで「良い」の基準値に達しているサイトをレビューします。

829. **https://web.dev/articles/lcp**
830. **https://web.dev/articles/fid**
831. **https://web.dev/articles/cls**
832. https://developers.google.com/search/blog/2020/05/evaluating-page-experience

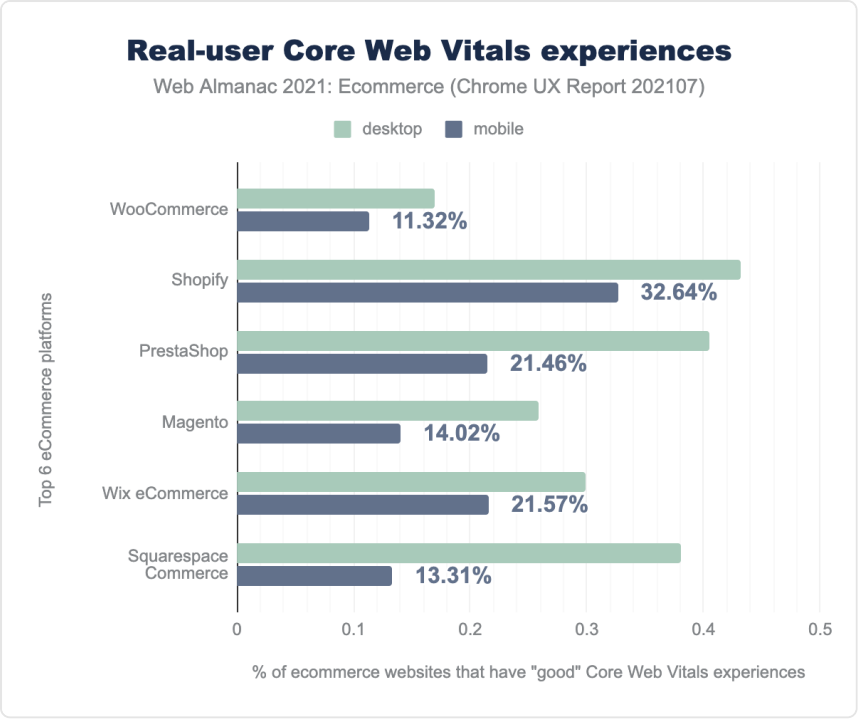


図17.12. コアウェブバイタルのリアルユーザー体験記

CWVによる「良い」体験を実現しているサイトの割合をプラットフォーム別に見ると、Shopifyがモバイルで32.64%ともっとも高いパフォーマンスを示していることがわかります。一方、WooCommerceのモバイルサイトでは、11.32%しか「良い」体験を実現していません。

パフォーマンスの章にある結果を見て、より広いウェブと比較できます。デスクトップでは41%、モバイルでは29%のサイトが「良い」CWV体験を達成していることがわかりました。このレンズで見ると、Shopifyストアは平均して、モバイルサイトに基づく平均的なサイトよりも良いパフォーマンスを示し、WooCommerceサイトは悪いと言うことができます。ただし、これは因果関係ではなく、相関関係であることを指摘することが重要です。

昨年と比較すると、すべてのプラットフォームでCWVスコアの中央値が向上していることがわかります。もっともパフォーマンスが向上したのは、Shopifyのサイトです。モバイルサイトの21.24%が良いCWV体験をしていたのが、32.64%に増加しました。

最後に、良いCWV体験を実現しているサイトの割合は、プラットフォームがSaaSかセルフホスティングかとは相関していないことを指摘しておきます。

次のセクションでは、各CWV指標を独立して検討し、各プラットフォームでサイトパフォーマンスの低下の最大の要因となっているものが何かについて確認します。

最大のコンテントフルペイント(LCP)

まず、最大のコンテントフルペイント[833]がありますが、これはメインページのコンテンツがロードされるまでの時間、ページが使えるまでにかかる時間の代理として使用するものです。

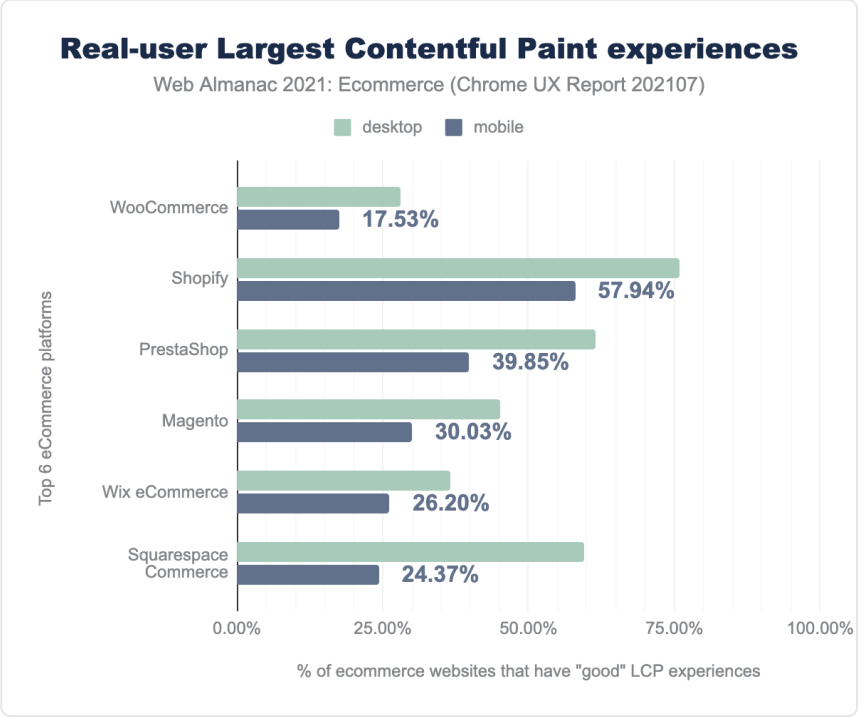


図17.13. 実ユーザーによる最大のコンテントフルペイント体験

モバイル向けShopifyサイトの57.94%が良好なLCP体験を達成し、Shopifyが再びトップeコマースプラットフォームの座を獲得しました。WooCommerceを使用しているサイトは、わずか17.53% が良好な体験を実現しており、もっとも悪い結果となりました。とくにこの指標は、WooCommerceのCWV総合スコアの低さにもっとも貢献しているように思われます。

ウェブ全体では、モバイルサイトの45%が良好なLCP体験を実現していることが、「パフォ

833. https://web.dev/i18n/ja/lcp/

ーマンス」の章で明らかにされています。もっとも普及している上位6つのeコマース・プラットフォームのうち、Shopifyだけが、モバイルでリクエストされた全サイトの平均を上回る結果を達成しました。

CWVの3つの指標のうち、ホスティングの設定は主にLCPスコアにのみ影響します。したがって、この時点では、一般的にセルフホスティングされているプラットフォームと、ベンダーによってインフラストラクチャが管理・最適化されているSaaSプラットフォームを比較することに価値があります。SaaSとしてのShopifyは、他のプラットフォームをリードしていることがわかります。しかし、他の2つのSaaSプラットフォーム、Wix eCommerceとSquarespace Commerceは、人気のあるセルフホスティングのプラットフォームMagentoとPrestaShopと比較してモバイルでのパフォーマンスが、悪いことがわかります。

最初の入力までの遅延(FID)

2つ目の指標である 最初の入力までの遅延[834] は、ウェブサイトの訪問者がリンクやボタンをクリックするなどしてサイトとやり取りしたときに、ブラウザがどれだけの作業をしなければならないかを測定するものです。これは、サイトの応答性、またはユーザー入力への反応が遅いかどうかの代用品と見なすことができます。

834. https://web.dev/i18n/ja/fid/

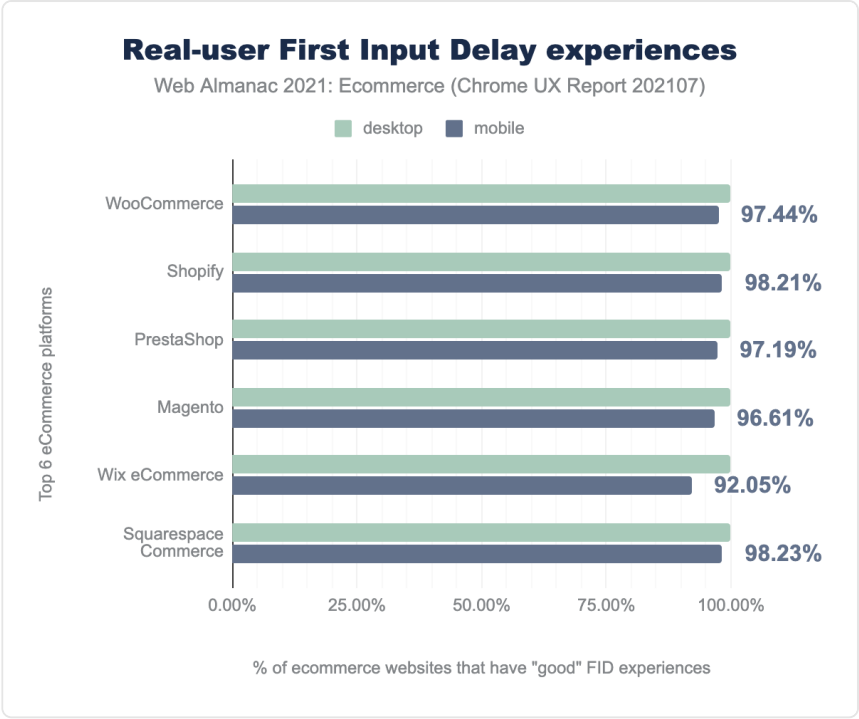


図17.14. 実ユーザーの最初の入力までの遅延体験

上位のすべてのeコマース・プラットフォームのサイトが、この指標で良好な結果を残しています。デスクトップでは、調査対象となったほとんどのECプラットフォームが100%良好なFID体験を達成しています。モバイルでは、一部体験の悪さが目立ち始めていますが、大半は良好なFID体験を達成しています。Shopify（98.21%）とSquarespace Commerce（98%）は上位のeコマースプラットフォームの中でもっとも高いパフォーマンスを示し、WooCommerce、PrestaShop、Magentoは98%にわずかに及ばない程度に留まっています。

Wix eCommerceは、一般的に良好なパフォーマンスを示しているプラットフォームですが、FIDは同社のウェブサイトの92.05%のみが良好なFID体験を有しており、不得意な分野の1つとなっています。

とはいえ、6つとも非ECサイトよりパフォーマンスが高い。パフォーマンスの章では、モバイルの全サイトの90%が良好な最初の入力までの遅延体験を実現していることがわかりました。

累積レイアウトシフト(CLS))

3つのCWVメトリクスの最後のものは、累積レイアウトシフト[835]です。これは、ページ上のアイテムが「動き回る」量の測定値です。たとえば、新しい画像が表示されて、読んでいたテキストやクリックしようとしていたボタンが別の場所に押されるなどです。

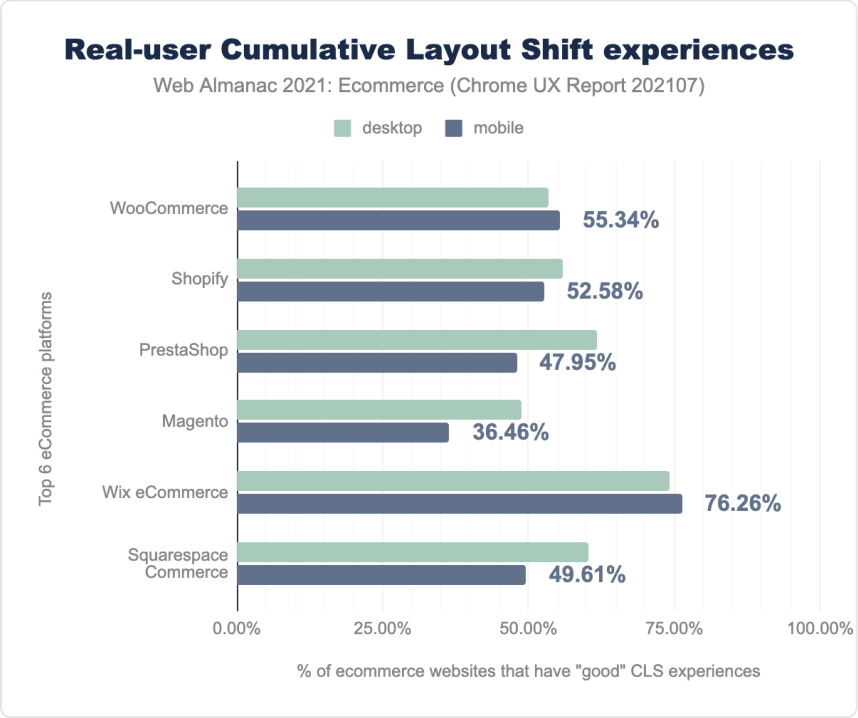


図17.15. 実ユーザーの累積レイアウトシフト体験

上位プラットフォームのうち、Wix eCommerceは76.26%のモバイルサイトがCumulative Layout Shift Experienceを達成し、すべてのプラットフォームを凌駕しています。一方、Magentoのサイト（36.46%）では、良い体験をした訪問者はその半分以下でした。

これらのeコマースサイトの指標をより広いウェブと比較すると、上位のeコマースプラットフォームのパフォーマンスが、若干悪いことがわかります。パフォーマンスの章では、62%のサイト（モバイルおよびデスクトップ）が良好なCLS体験をしていることがわかりました。

835. https://web.dev/i18n/ja/cls/

ページ解剖学

サイトのパフォーマンスの理由を理解する場合、最初に調べるのは、ページの重さ（ダウンロードに必要なキロバイト数）と、ページの読み込みに必要なリクエストの数です。

ページのリクエスト

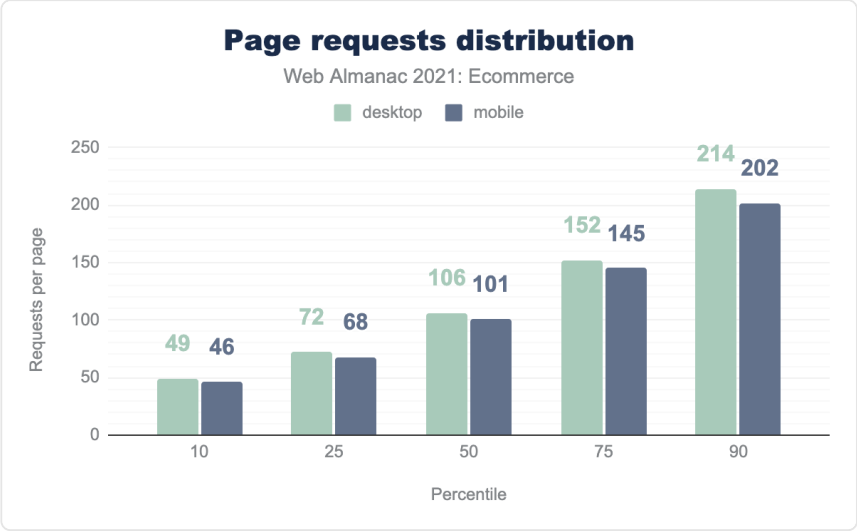


図17.16. ページが配信を要求する。

全ECサイトの50パーセンタイルでは、モバイルでのホームページのリクエスト数は101件でした。これは、昨年見られた98件のリクエストと非常に似た数字です。ページあたりのリクエスト数は、昨年と比較すると、すべてのパーセンタイルで非常によく似ています。

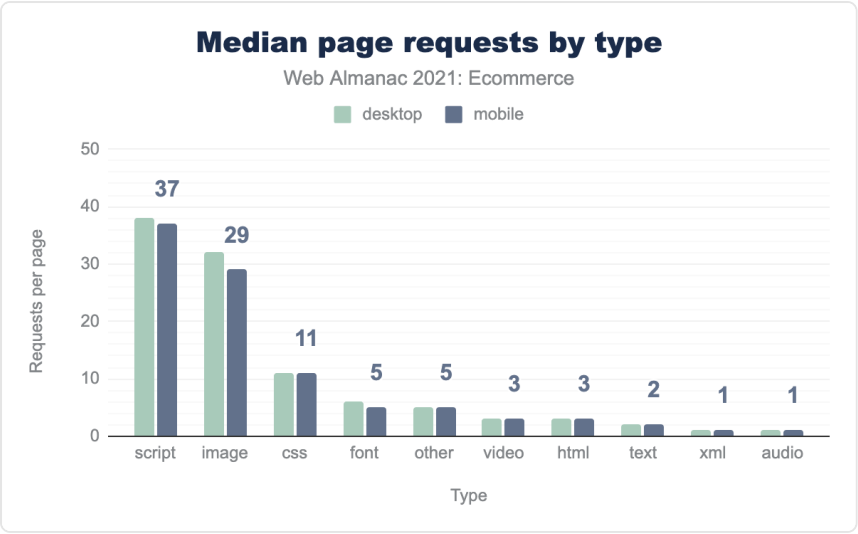


図17.17. ページリクエストのタイプ別中央値。

これらのリクエストを種類別に分類すると、JavaScriptがもっとも普及しているリソースであり、平均的なeコマース・モバイルのホームページで37件のリクエストが、あることがわかります。これは、1ページあたり30件のJavaScriptリクエストがあった昨年から23%増加したことになります。以前は画像がもっともリクエストされるリソースで、モバイルページあたり34リクエストでしたが、これは29リクエストとわずかに減少しています。

ページの重さ

サイトのページ重量には、すべてのHTML、CSS、JavaScript、JSON、XML、画像、音声、動画が含まれます。

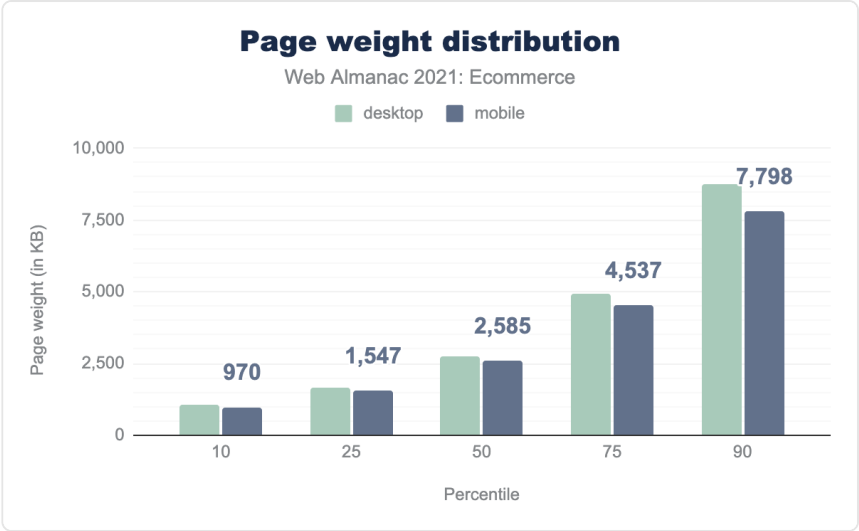


図17.18. ページの重量配分。

eコマースホームページのページ重量の中央値は、モバイルでは2.5MBでした。この数値は昨年の結果と同じであるため、平均してホームページは重くなっていない（軽くなっている）ことがわかります。

もっとも重いサイト（90パーセンタイル）は2020年の結果より4%重いので、ワースト1位は若干悪くなっています。

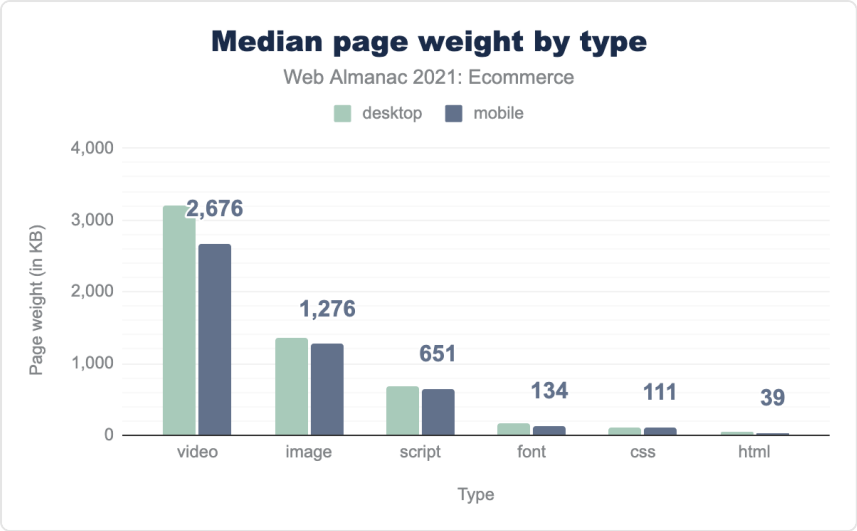


図17.19. タイプ別ページキロバイトの中央値。

この理由を理解するために、リソースの種類別のページの重さを見てみましょう。モバイルサイトでは、動画が2.6MBともっとも重いリソースであり、画像（1.2MB）、JavaScript（0.6MB）がそれに続いています。昨年と比較すると、動画の読み込みMB数が24%増加していることがわかります。一方、他のすべてのリソースタイプのMBは安定しています。

このことから、もっとも重いサイトは、すぐに全体のページウエイトをかなり大きくすることができる動画を使用しているサイトである可能性があります。2020年と2021年の間でページウェイトの中央値に変化がないことから、動画を使用しているサイトの数は変わっていないが、動画を使用しているサイトのうち、より多く使用していることが示唆されます。この分野でのさらなる研究の機会としては、動画のウェイトを増加させた原因を調べることでしょう。動画の数が増えたのか、動画の時間が長くなったのか、品質が上がったのか？

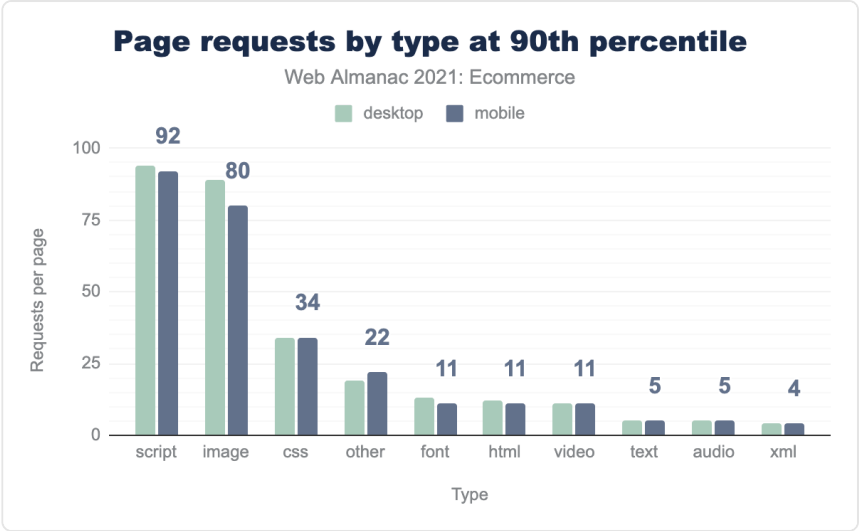


図17.20. 90パーセンタイルでのタイプ別ページ要求数。

もっとも重いページ（モバイルで17MB）を持つサイトは、中央値（4.8MB）よりはるかに重いことがわかりました。とくに90パーセンタイルでタイプ別のページウェイトを見て、50パーセンタイルと比較すると、すべてのリソースタイプのウェイトが増加していることが分かります。

90パーセンタイルでページの重さにもっとも貢献しているのは、引き続き動画の9MBと画像（5.6MB）です。もっとも重いeコマースホームページが、動画と画像を大量に使用しているページであることは、まったく驚くことではありません。このページはコンテンツが、多いことが多く、これらのリソースタイプはブランドを伝えるのにもっとも効果的な方法だからです。動画と画像は購買体験の重要な要素であり続けますが、筆者の見解では、他のページタイプでは、これらの極端な表現はあまり見られないと思われます。

HTMLペイロードサイズ

HTMLペイロードは、ドキュメントレスポンスのサイズです。HTMLの他に、インラインのJavaScriptやCSSが含まれることもあります。

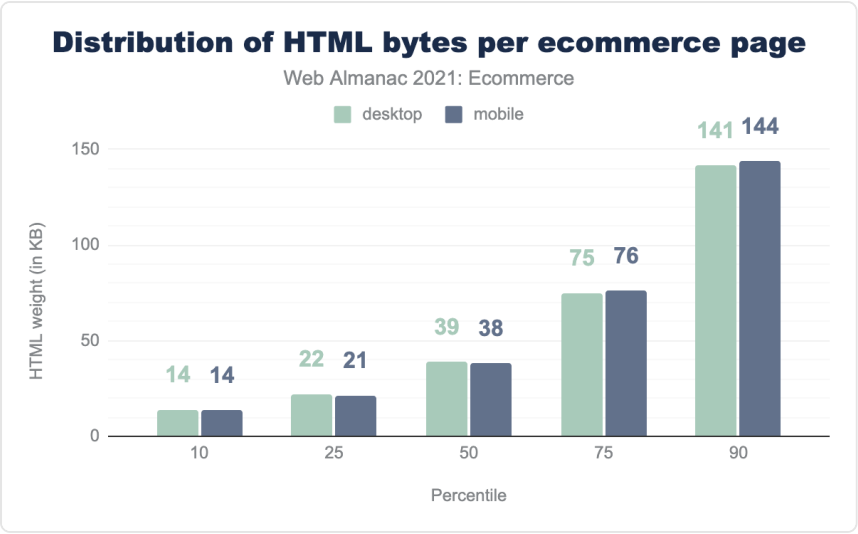


図17.21. EコマースページごとのHTMLバイト数分布

HTMLのペイロードの中央値は、モバイルで38KB、デスクトップで39KBでした。一方、90パーセンタイルでは、モバイルで144KB、デスクトップで141KBと、ペイロードが約4倍になっています。

ペイロードサイズは、モバイルとデスクトップの両方でほぼ一定であり、サイトが両方のデバイスタイプにほぼ同じHTMLを配信していることが示唆されました。

画像

画像は、2番目に要求の多いリソースタイプであり、ページの重量を増加させる要因としても2番目に大きいものです。

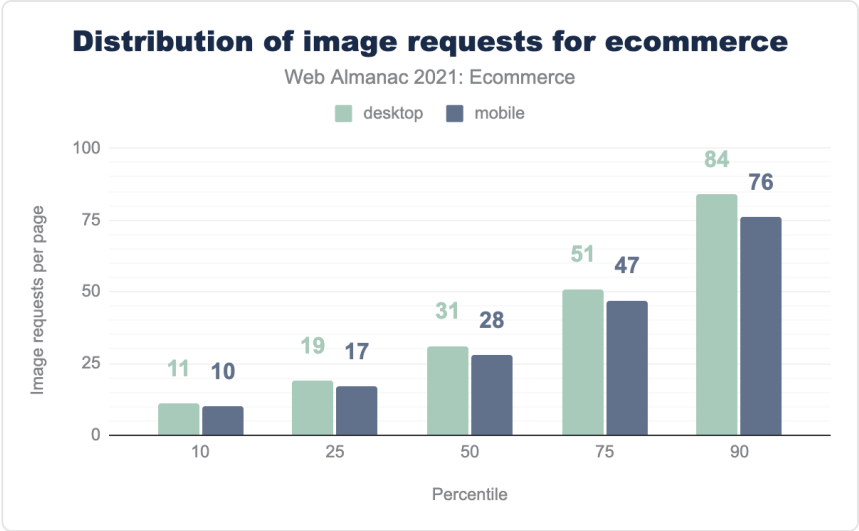


図17.22. eコマース向け画像リクエストの配信

モバイルのホームページで要求される画像数の中央値は28枚であるのに対し、デスクトップでは31枚であることがわかります。10%のサイトがモバイルで76枚の画像を読み込んでいますが、これは昨年の91枚という最高値から減少しています。

全体として、要求される画像枚数が10〜20%削減されています。明確な答えを出すのは難しいのですが、遅延読み込み属性[836]の採用が、増えたことが原因かもしれません。テスト中はスクロールやサイトとのインタラクションが行われないため、遅延ロードされたアセットは測定に加味されません。JavaScriptチャプターによる分析では、17%のサイトがこの属性を使用していることがわかり、この説に一定の説得力が、あることがわかりました。

836. https://web.dev/browser-level-image-lazy-loading/

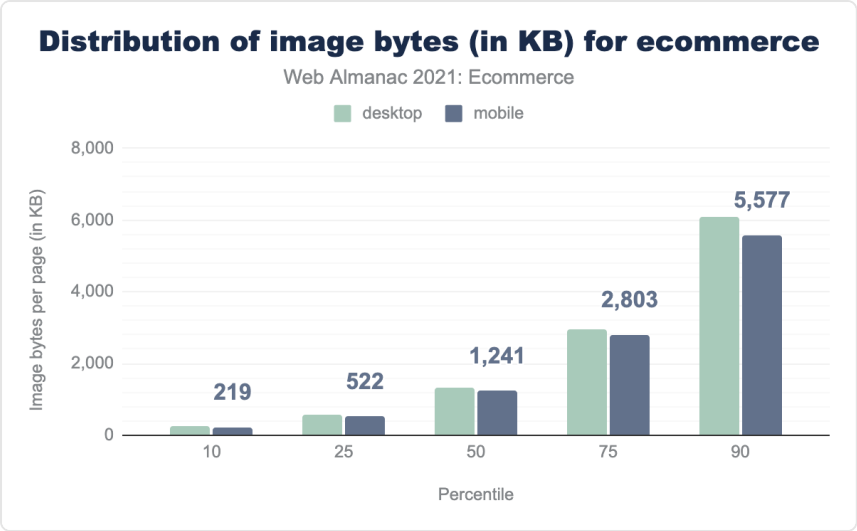


図17.23. eコマース向け画像バイトの配布

画像を数ではなく重みで考慮すると、ページ重量の貢献度の中央値は1.2MB（モバイル）です。90パーセンタイルでは、5.4MBに上昇します。

2020年の分析と比較すると、全体的にECサイトホームページの画像の比重は非常に似ています。

画像のリクエスト数が若干減っていることから、各画像の平均的な重みが若干増えているのでしょう。

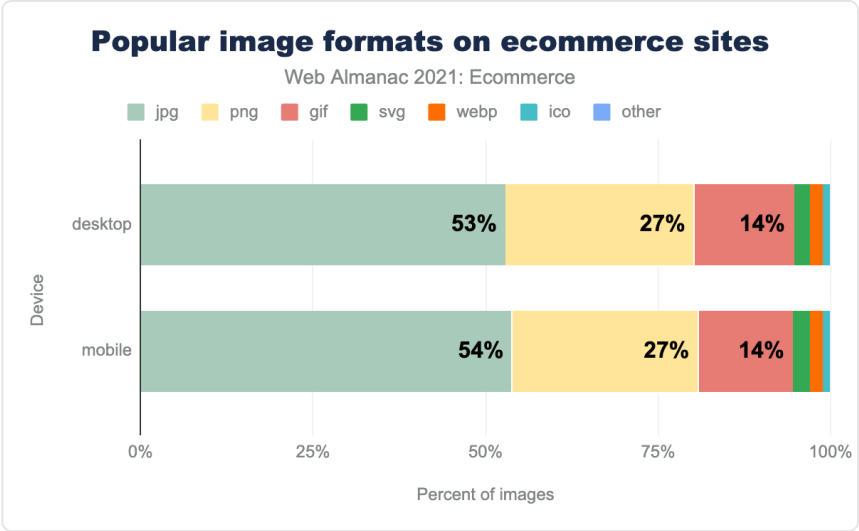


図17.24. Eコマースサイトで人気の画像フォーマット

画像サービスやCDNの中には、WebPをサポートするプラットフォームには、接尾辞が `.jpg` *や* `.png` *のURLでも(JPEGやPNGではなく)WebPを自動的に配信するものがあることに注意してください。たとえば、* `IMG_20190113_113201.jpg` *は、ChromeではWebPの画像を返します。しかし、HTTP Archiveが画像フォーマットを検出する方法は、まずMIMEタイプのキーワードを確認し、次にファイル拡張子にフォールバックします。つまり、上記のようなURLを持つ画像のフォーマットは、HTTP ArchiveがユーザーエージェントとしてサポートしているWebPが与えられることになります。*

もっとも普及している画像形式はJPGで、モバイルでは54%がこの形式でした。これは、画像の50%がJPGであった昨年に比べ、8%増加しています。

画像の27%はPNGで、これは昨年と同様の割合です。その他の画像の種類はほぼ同じですが、モバイルではGIFが17%から14%に減少しています。

残念ながら、WebPへの対応はまだまだ少ないのが現状です。これは、よりファイルサイズの効率的なフォーマットであり、すべてのモダンブラウザ[837]でサポートされているにもかかわらず。

サードパーティーからの要望

Eコマースプラットフォームやサイトでは、しばしばサードパーティのコンテンツが利用されます。サードパーティウェブプロジェクトを使用して、第三者の利用を検知しています。

837. https://caniuse.com/webp

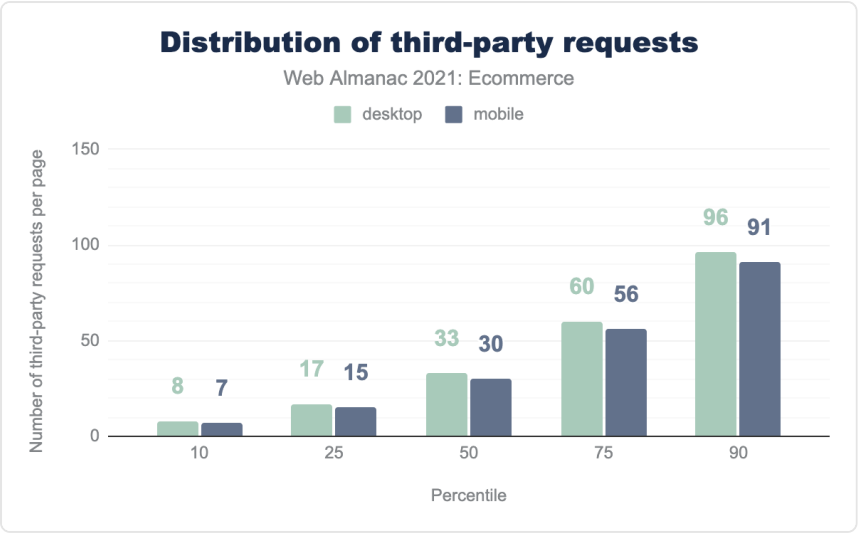


図17.25. サードパーティーからの要求の配布

モバイルのEコマースサイトの中央値は、サードパーティに30件のリクエストを行いました。昨年の分析ではサードパーティへのリクエストが増加していましたが、今年はほぼ全体的にほとんど変化がなく、静観しています。上位10%のページが、モバイルでは98から91に、デスクトップでは103から96に、サードパーティへのリクエスト数を減らしているというわずかな変化があります。

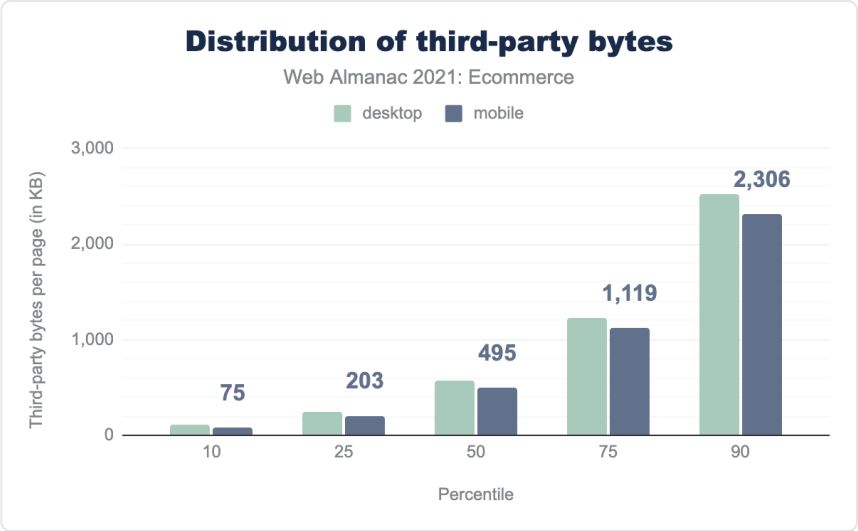


図17.26. サードパーティーバイトの分布

サードパーティコンテンツの比重も、昨年の分析とほぼ同じです。50パーセンタイルのサイトでは、495KBのサードパーティ製コンテンツを要求しています。下位10%は75KBを要求しているのに対し、上位10%は2306KBを要求しています。

ツール

サイトのパフォーマンスや品質の分析に加え、当社の方法論では、eコマースサイトで使用されているその他の技術も検証することが可能です。これにより、採用されたeコマース戦略（例：国際化）や、典型的な開発手法（例：使用したJavaScriptライブラリ）についての知見を得ることができます。

JavaScriptフレームワーク＆ライブラリ

JavaScriptの使用は、とくにコア製品がブラックボックスであるSaaSプラットフォームにおいて、コマース体験をカスタマイズするための一般的な方法です。

今年、ECサイトで使用されるJavaScriptの量が著しく増加したわけではありませんが、どのフレームワークやライブラリがもっともよく使用されているかを調べてみました。これにより、JavaScriptが何を実現するために使われているのかを知ることができるかもしれません。

残念ながら、eコマースにおけるヘッドレスフロントエンドの実装の普及について言及することはできません。この手法の限界の1つは、ヘッドレスの場合、eコマースプラットフォームの典型的なマーカーがもはや存在しないため、サイトがeコマースであることを検出するのがより困難であることです。この時点で、分析はより弱い二次的なシグナルへ頼ることになります。

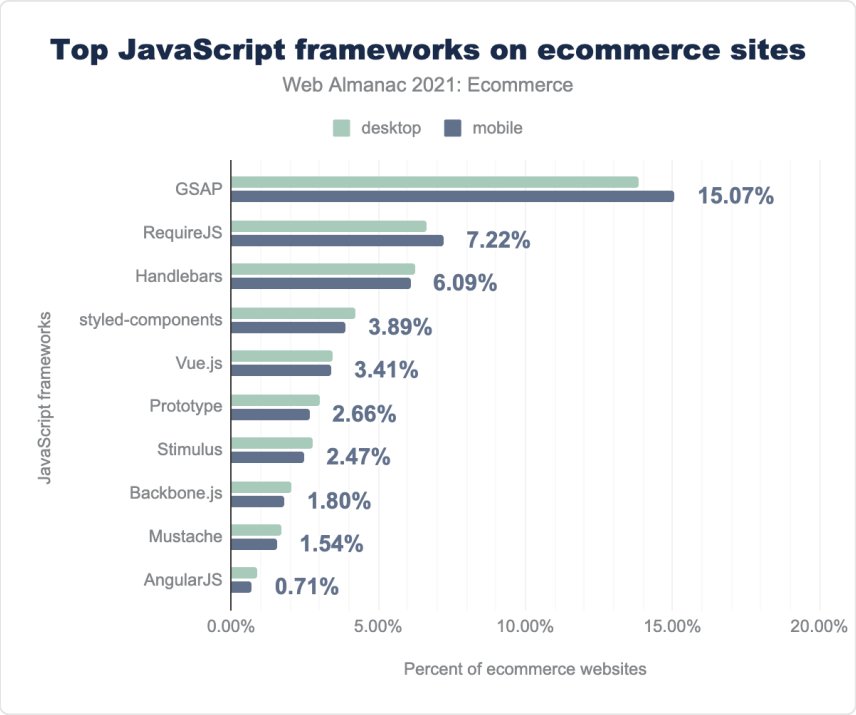


図17.27. eコマースサイトで人気のJavaScriptフレームワーク

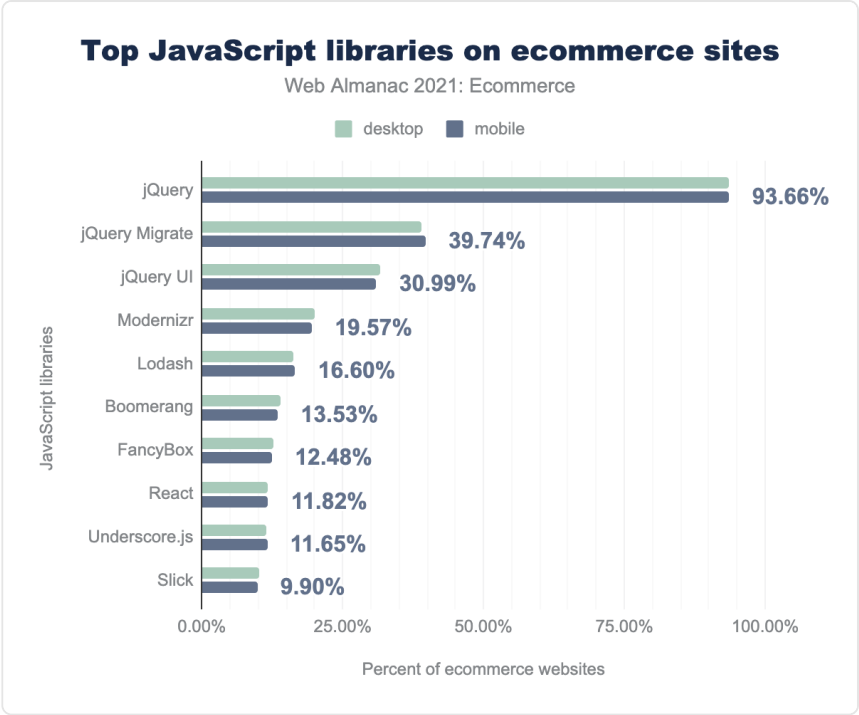


図17.28. eコマースサイトで人気のJavaScriptライブラリ

jQuery[838]は、いまだもっとも普及しているライブラリであることがわかります。その終焉の報道は大いに誇張されている。調査対象となったeコマースサイトの93.66%がまだ使用していました。人気のあるeコマースベンダーの多くは、デフォルトのフロントエンドの一部としてjQueryを提供しています。さらに、プラットフォームはアプリやプラグインのエコシステムによって生かされており、追加機能を購入できます。これらのソリューションも、コスト効率よく機能を提供するためにjQueryを定期的に使用しています。

とくに、GSAP[839] (GreenSock Animation Platform) は、モバイルでリクエストされたeコマースウェブサイトの15%に含まれています。これは、人気のあるLightboxライブラリであるFancybox[840] (12.48%) や、カルーセル作成に使われるライブラリ Slick[841] (9.90%) より多いのです。

制限のセクションで、すべてのリクエストはホームページに対して行われるため、結果が歪むことを認識しました。つまり、この分析では、Slickがさらに人気を博した可能性のある製品詳細ページのメディアギャラリーに使用されたライブラリは検出されないということで

838. https://jquery.com/
839. https://greensock.com/gsap/
840. https://fancyapps.com/docs/ui/fancybox/
841. http://kenwheeler.github.io/slick/

す。

分析学

eコマースの優れた点の1つは、サイトを訪問した人のうち何人がコンバージョンしたかによって、自社の業績がどの程度かを測定できることです。理論的には、すべての変更、新しい価格設定、新しい機能は分析によって客観的に評価できます。

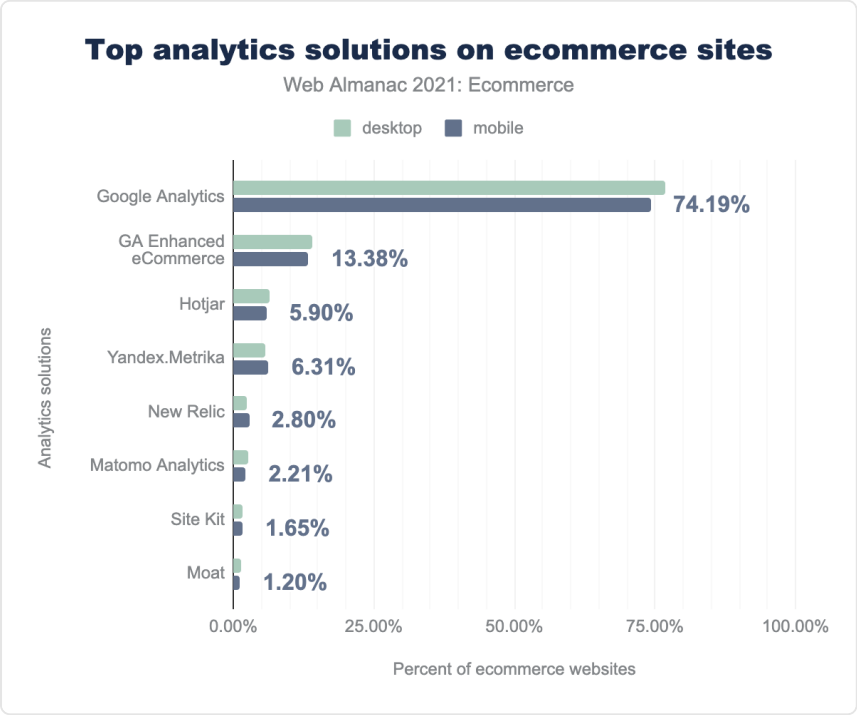


図17.29. Eコマースサイトのトップ分析ソリューション

Google Analytics[842]は、74.19%のWebサイト（モバイル）で見られる、もっとも普及している解析ツールです。興味深いことに、モバイルのリクエストでは13.38%、デスクトップのリクエストでは13.99%しか、拡張eコマース[843]の利用を指摘していません。しかし、主に強化されたeコマース機能は、商品一覧ページ、商品詳細ページ、カート、チェックアウトまでのeコマースジャーニーをトラッキングするものであるため、この割合が高まらないのは調査がホームページに限定されていることが原因かもしれません。

842. *https://marketingplatform.google.com/about/analytics/*
843. *https://support.google.com/analytics/answer/6014872?hl=en#zippy=%2Cin-this-article*

タグマネージャー

これらのツールは、コアウェブサイトプラットフォームの導入（あるいは開発者の関与）なしにJavaScriptの変更をサイトに加えることができるため、eコマースチームやマーケティングチームに新機能を立ち上げるためのサイクルタイムを短縮することを可能にします。

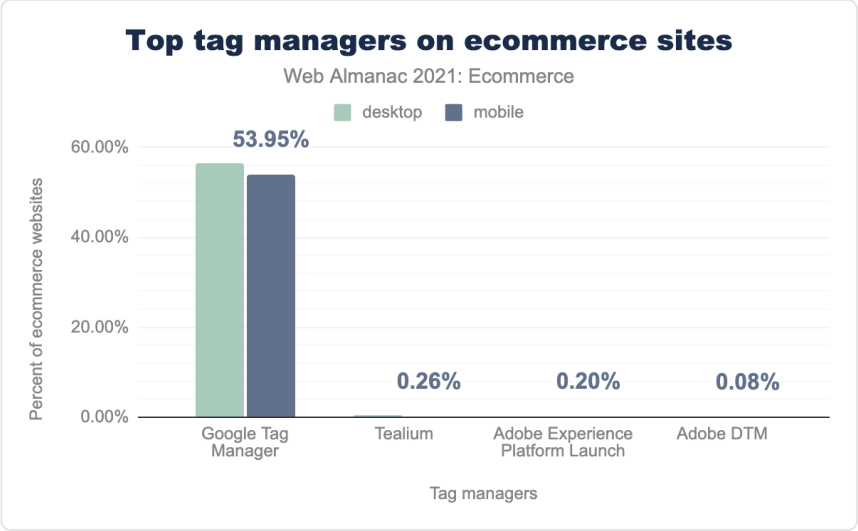


図17.30. ECサイトにおけるトップタグマネージャー

Google Tag Manager[844]は、デスクトップで56.39%、モバイルで53.95%の利用率で圧倒的に市場をリードしています。2位と3位は、Tealium[845]（モバイル0.26%）とAdobe Experience Platform Launch[846]（モバイル0.20%）となっています。

A/Bテスト

アナリティクスと同様に、A/Bテストソリューションを導入することで、仮説を検証できます。新機能に対するフィードバックの仕組みを提供することは、どの戦略が有効で、どれがもはや投資されるべきかを理解する唯一の方法です。

844. https://marketingplatform.google.com/intl/en_uk/about/tag-manager/
845. https://tealium.com/
846. https://business.adobe.com/uk/products/experience-platform/launch.html

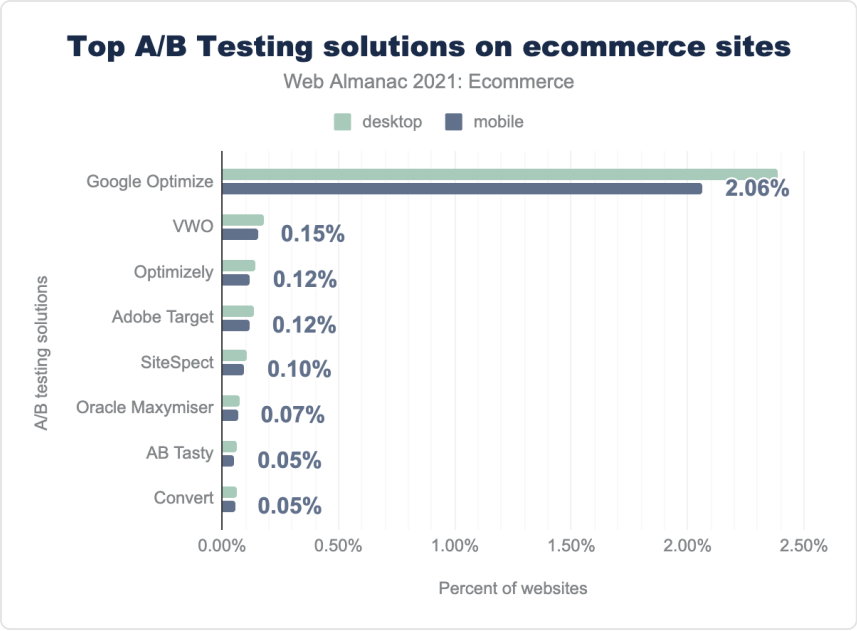


図17.31. Eコマースサイトにおけるトップ A/B テストソリューション

Google Optimize[847]は、モバイルECサイトの2.06%で使用されている、もっとも普及しているA/Bテストツールです。VWO[848]は2番目に多いソリューションでしたが、Google Optimizeと比較して10分の1以下のサイト数で見られました（モバイルでは0.15%）。

当然のことですが、残念なことに、調査時点で大半のeコマースサイトはA/Bテストを実施していませんでした。

Webプッシュ通知

訪問者の許可を得れば、*Push API*を使って、ECサイトが開いていない時でもプッシュ通知を送ることができるようになります。

Chromeのユーザー体験レポートを使って、ECサイトによるWebプッシュ通知の導入状況を調べてみました。実際のユーザーデータから作成されているため、プッシュ許可リクエストの承認率も確認できます。このデータがどのように取得され、どのような指標が利用できるかの詳細については、このGoogleの記事[849]を参照してください。

847. *https://marketingplatform.google.com/about/optimize/*
848. *https://vwo.com/*
849. *https://developers.google.com/web/updates/2020/02/notification-permission-data-in-crux*

0.43%

図17.32. Webプッシュ通知（モバイル）を利用しているECサイトの割合。

モバイルで0.43%（デスクトップでは0.48%）のホームページのみがWeb Push APIの利用を要求しています。とくにiOSのSafariはPush Notifications APIをサポートしていませんが、他のブラウザではまだ広く採用されています。注文の更新など、eコマースにおける適切なタイミングでプッシュ通知による体験を段階的に向上させる良い機会がまだあることを示唆しています。

さらに、モバイルサイトの0.69%（デスクトップでは0.68%）がPush通知の送信許可を求めていた昨年から、その使用率は目に見えて減少しています。

利用率の低さを「認知度が低いから」と言い訳することもできるかもしれません。しかし、利用率の低下は別の傾向を示しており、3分の1以上のサイトがプッシュ通知を利用しなくなりました。これはプッシュ通知の受理率が、低いことが原因かもしれません。

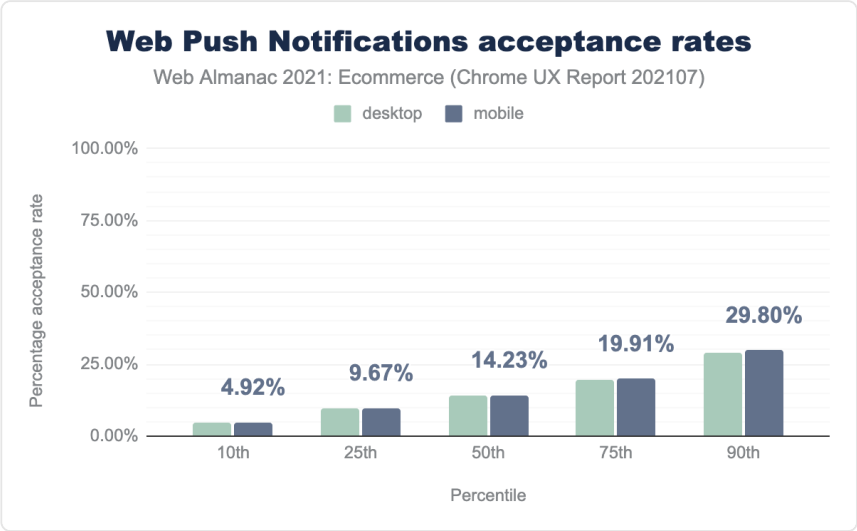


図17.33. Webプッシュ通知受付率

プッシュ通知の受付率は、昨年の結果とほぼ同じです。プッシュ通知リクエストの受付率の中央値は、モバイルで14.23%でした。残念なことに、年をまたいで傾向があるとすれば、それは下向きです。90パーセンタイルでは、昨年は36.9%のプッシュ要求が受け入れられたのに対して、今年はモバイルで29.80%でした。

なぜ、これほどまでに取り込みが進まないのか、筆者は複数の示唆を与えることができます。

- リクエストのタイミングが悪い（例：最初のページロード時）、または
- 十分な動機付けがなされる前に行われた場合、たとえば、通知を受けることの利点について何の説明もなく行われた場合、または
- もっと単純に、訪問者がウェブベースのプッシュ通知にまだ慣れていないだけかもしれません。

アクセシビリティオーバーレイ

Webサイトをアクセシブルにすることは、決して後回しにすべきではない。しかし、ウェブサイトをよりアクセシブルにすることを謳う技術は増えてきています。アクセシビリティオーバーレイは、サイトに自動的なアクセシビリティの修正を適用しようとするJavaScriptです。これらは通常、アクセシビリティの専門家には推奨[850]されません[851]。

0.77%

図17.34. アクセシビリティオーバーレイがあるECサイトの割合（モバイル）

今回の調査では、サードパーティ製のアクセシビリティツールをホームページで公開しているウェブサイトは1%未満でした。

このようなツールの詳細については、アクセシビリティの章に記載されています。

AMP

0.61%

図17.35. ECサイト（モバイル）でのAMPの利用状況。

Googleが提供するAMPは、最新情報を素早く提供するメディア業界ではよく使われていますが、eコマースではなかなか普及が進んでいません。今年、AMP対応を宣言したウェブサ

850. *https://overlayfactsheet.com/*
851. *https://www.a11yproject.com/posts/2021-03-08-should-i-use-an-accessibility-overlay/*

イトやAMPリソースにリンクしているウェブサイトは0.7%未満であると報告されました。

同意の管理

6.85%

図17.36. ECサイト（モバイル）でのサードパーティ製同意管理ソリューションの利用状況。

EUのCookieポリシーとGDPRにより、要求されるマーケティングパーミッションの複雑さが増しています。今年、モバイルのECサイトの6.85%が、法令に従った同意の収集を容易にするため、サードパーティの同意管理アプリを導入していることがわかりました（デスクトップでは6.52%）。

コンテンツセキュリティポリシー

お客様が機密情報を共有することが予想されるサイトでは、システムに入り込んだ不正なコードがないことを確信できることがさらに重要です。コンテンツセキュリティポリシー（CSP）は、ホワイトリストに載っていない第三者のウェブサイトへのリクエストを監視したり、ブロックしたりする手法です。

多くのセキュリティポリシーと同様に、このような管理方法は、サードパーティのコードをサイトに素早く追加することを主目的とするタグマネージャーなどのツールを使って素早く行動したいeコマースビジネスの敵対者と見なすことができます。筆者の体験では、CPSを管理するためのオーバーヘッドが原因で、ほとんど利用されていません。

23.28%

図17.37. コンテンツセキュリティポリシーを使用しているモバイルeコマースページの割合。

最初に読んだとき、デスクトップで25.02%、モバイルページでは23.28%のリクエストでコンテンツセキュリティポリシーが利用されていることに驚かされました。しかし、一部のeコマースプラットフォームベンダーは、緩やかなコンテンツセキュリティポリシーをそのまま提供しています。たとえば、Shopifyのサイトでは、すべてのリクエストがHTTPSであることを保証するだけでなく、iframe内にサイトを読み込むことをブロックするポリシーが用意されています。さらに調査を進めないと、サードパーティの資産を管理する手段として

CSPを使用しているeコマースサイトの数を特定することはできません。ポリシーの変更を実施する前にテストすることを目的としたCSPの「レポートのみ」モードを使用しているサイトが0.70%しかないことを考えると、ごく少数である可能性があります。

国際化

成功するeコマース・ビジネスの重要な成長戦略は、新しい国への進出です。そのためには、ローカライズされた言語版のサイトを提供する必要があります。

今年の分析では、`hreflang`ヘッダーとリンクタグを探し、どれだけのサイトがそれらを使用しているかを調べました。これらのタグはもっとも普及しているプラットフォーム（例：WooCommerce、Shopify、Magento）ではすぐに利用できないため、何らかの存在が示唆されています。

`hreflang`属性は、そのページが対象としている言語を伝えるために使用されます。たとえば、イギリスを対象とする英語は`en-gb`で、米国を対象とする英語は`en-us`であるのに対し、特定の国に絞り込むこともできます。



図17.38. Eコマースサイトで使用される`hreflang`リンクのトップページ

その結果、英語のhreflangを指定するリクエストは、デスクトップでは8.81%、モバイルのECサイトでは8.07%であることが確認されました。次にもっとも普及している言語は、ドイツ語（モバイルでは3.28%）、フランス語（2.82%）、スペイン語（2.66%）の順となりました。

このデータから、さらなる調査なしにあまり多くの結論を導き出すことは困難です。しかし、eコマース事業者が言語別のサイトバリエーションを提供することは、まだ珍しいと言えるでしょう。その中でも、西ヨーロッパ諸国で使用されている1つまたは複数の言語のサポートを宣言しているケースがもっとも多いようです。筆者の体験では、イギリス、フランス、ドイツ、スペイン、イタリアはそれぞれ地理的に近いため、国際化は魅力的な成長戦略であると言えます。

eコマース・ウェブサイトの国際化能力をよりよく理解するために、ここでさらに調査を行うことができます。たとえば、宣言された `hreflang` 属性の平均数を調べることで、マルチリージョンサポートの幅を決定することができるかもしれません。

`hreflang` の使用とCRUXメトリクスから得られるランキングデータを相互参照することで、企業がマルチリージョンサポートに投資する時期の傾向を明らかにできます。

結論

2020年の第2四半期から第3四半期にかけて、eコマース機能を持つサイトの割合に測定可能な増加が見られました。この増加率は2021年まで維持されていません。実際、モバイルでのECサイトの割合は21.27%から19.49%に減少しており、ECがウェブ全体と同じペースで成長していないことが示唆されています。

WooCommerceとShopifyはもっとも普及しているeコマース・プラットフォームです。また、パンデミックに対応した成長の割合がもっとも大きいのもこの2つです。

今回はじめて、ウェブサイト人気ランキングのデータを用いて分析を行いました。これにより、事業規模別にeコマースプラットフォームの人気度を検証することができました。とくに、10万サイトの中でもっとも普及しているのはMagentoです。次いでShopify、Salesforce Commerce Cloudとなっています。

最後に、サイトのパフォーマンスについてですが、コアウェブ・バイタルはGoogleの検索エンジンのランキング要因になったため、昨年から業界で盛んに議論されています。上位5つのプラットフォームのほとんどで、モバイルで良好なCWVを達成したサイトが10〜20%増加しました。Shopifyのサイトでは、平均33%ともっとも高い割合で良好なCWVを体験しています。昨年からの改善にもかかわらず、eコマースサイトは、コアウェブ・バイタルのすべてのプラットフォームにおいて、依然として非常に低いパフォーマンスとなっています。

今後の解析の可能性

この方法の限界の1つは、ホームページのみをテストすることです。たとえば、支払いや配送のプロバイダーは、チェックアウトプロセスの間だけ表示されます。このような場合、チェックアウトプロセスのこの段階に到達するため必要なステップを考えると、実現は困難と思われます。

また、トップページのみの評価は、サイトパフォーマンスの分析に影響を及ぼします。商品一覧ページと商品詳細ページが、スピードの最適化においてより重要であることは間違いないでしょう。サイトごとに複数のページを取得することは調査中[852]で、Web Almanacの将来のエディションで利用できるようになるかもしれません。

Wappalyzerは2,700以上の人気のあるウェブテクノロジーを追跡しており、すでに素晴らしい分析機会を提供してくれています。しかし、とくにeコマースにおいては、非常にロングテールの技術が存在します。現時点では、パーソナライゼーションツールのトップ、レビューアプリのトップ、カート放棄のトップなど、eコマースにおける技術のカテゴリーをレビューすることは、十分なカバレッジがないため、現実的ではありません。これは、検出可能な技術の数が多いことと、1サイトにつき1ページしか要求しないことが一因です。

今後、Wappalyzerを対応する技術が増えれば、技術の利用状況、パフォーマンス、ウェブサイトのCrUXランクに相関性があるかどうかを調べる、さらなる分析ができるようになるかもしれません。

著者



Tom Robertshaw

@bobbyshaw bobbyshaw tomrobertshaw https://www.space48.com

Tomは、野心的な小売業者のためのeコマース代理店であるSpace 48[853]のイノベーション・ディレクターです。彼は、Ordnance Survey、Betty's & Taylors of Harrogate、Smythsonなどのブランドで10年以上eコマースの経験を積んできました。現在は、BigCommerceでマーチャント向けのアプリ群を立ち上げるためのイニシアチブをリードしています。

852. https://github.com/HTTPArchive/httparchive.org/issues/400
853. https://www.space48.com

部 III 章 18
Jamstack



***Artem Denysov** によって書かれた。*
Alba Silvente Fuentes、 Thom Krupa と Barry Pollard によってレビュ。
Artem Denysov、 Barry Pollard と Rick Viscomi による分析。
Barry Pollard と Shaina Hantsis 編集。
Sakae Kotaro によって翻訳された。

序章

> *Jamstackは、よりシンプルな開発者体験、より優れたパフォーマンス、より低いコスト、より高いスケーラビリティを提供することで、Web構築に関する考え方に革命を起こしました。*
>
> *— Jamstack.wtf*[854]

Jamstackは、JavaScript、API、Markupアーキテクチャに基づいています。この3つの基盤は分離されており、Jamstackのサイトは純粋にマークアップを使って構築できます。純粋なHTMLを使うのも「ある意味」Jamstackですが、スケールさせるのは本当に難しいです。幸いなことに、静的サイトジェネレーター（SSG）の巨大なエコシステムが存在します。

854. *https://jamstack.wtf/*

JavaScriptベースのSSG。

- Next.js
- Gatsby
- Nuxt.js
- その他

伝統的なもの。

- Eleventy
- Hugo
- Jekyll
- Hexo
- その他

そして、これら以外にも多くのSSGがあります[855]。必要に応じて、純粋なHTMLとJavaScriptの良さに変換したサイトを構築できます。

より複雑なサイトでは、データを構造化する必要があります。ヘッドレスCMS[856]を使って、API経由でデータを保存・管理する方法がいくつかあります。

さらに、Jamstackのサイトでは、フォーム送信やユーザー入力処理などのサーバーインタラクションをサポートする必要があります。Netlifyのようなサービスは、このニーズに応えるためサーバーレス機能[857]をサポートしています。

この章の目的は、Jamstackで使用されている主なSSGが何であるかを特定し、Jamstack技術の採用状況を年ごとに見ていくことです。世界中にどのように分布しているのか、Jamstackサイトのパフォーマンスレベル、そしてどのように成長しているのかを調べました。また、Jamstackサイトのさまざまなプロバイダーのデータも調査しました。さらに、Jamstackサイトで使用されているリソースの結果と、それがユーザー体験に与える影響についても調査しました。

この章を読むにあたって、いくつかのデータの免責事項について触れておきます。

1. 検出されたSSGのHTTP Archiveデータは、Wappalyzerの技術に基づいており、いくつかの制約があります。Eleventyなどの特定のSSGで構築されたサイトかどう

855. https://jamstack.org/generators/
856. https://jamstack.org/headless-cms/
857. https://www.netlify.com/products/functions/

かを検出することはできません。また、Next.jsの静的レンダリング[858]とサーバーサイドレンダリング[859]で生成したサイトかどうかを判別することができません。
2. 今回の分析では、ヘッドレスCMSに関連する情報は得られなかったので、こちらも取り上げないことにします。
3. SSGを使用して構築されたサイト数の上位5つのSSGを使用して、SSGデータを可視化しています。

詳細は、方法論に記載されています。

SSGの採用

SSGの採用は、一般的に前年比2倍で伸びています。2019年はモバイルサイト0.4%、デスクトップサイト0.3%にとどまりました。2020年にはモバイルで0.6%、デスクトップサイトで0.7%とほぼ倍増しました。2021年にはモバイルの1.1%、デスクトップサイトの0.9%と再び成長しています。その技術の傾向を裏付けている。たとえば、今年VercelはシリーズCラウンドで1億200万ドルを調達し[860]、さらにラウンドDで1億5000万ドル[861]を投資して、Next.jsなどの最新技術でよりよいWebを構築しています。Jamstack指向のCDNプロバイダーNetlifyが、シリーズD[862]で1億500万ドルの投資を行いました。したがって、来年はJamstackの採用数が、さらに増加することが予想されます。

858. *https://nextjs.org/docs/basic-features/pages#static-generation-recommended*
859. *https://nextjs.org/docs/basic-features/pages#server-side-rendering*
860. *https://vercel.com/blog/series-c-102m-continue-building-the-next-web*
861. *https://vercel.com/blog/vercel-funding-series-d-and-valuation*
862. *https://www.netlify.com/press/netlify-raises-usd105-million-to-transform-development-for-the-modern-web*

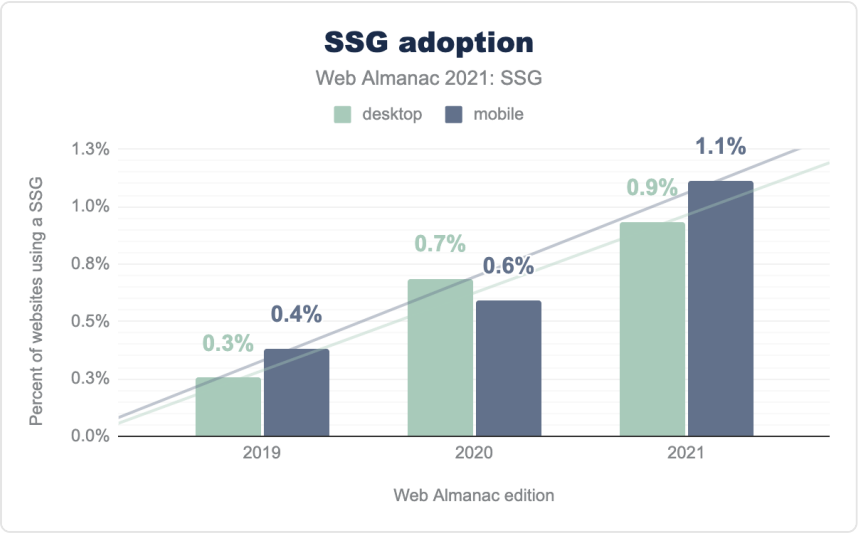


図18.1. SSG採用の前年比。

2020年はデスクトップが2.76倍、モバイルは1.5倍にとどまりました。2021年、SSGが構築したサイトのモバイル可用性は2020年よりもずっと良くなり、今年は2020年の1.9倍になっています。

もっとも普及しているSSGはどれか

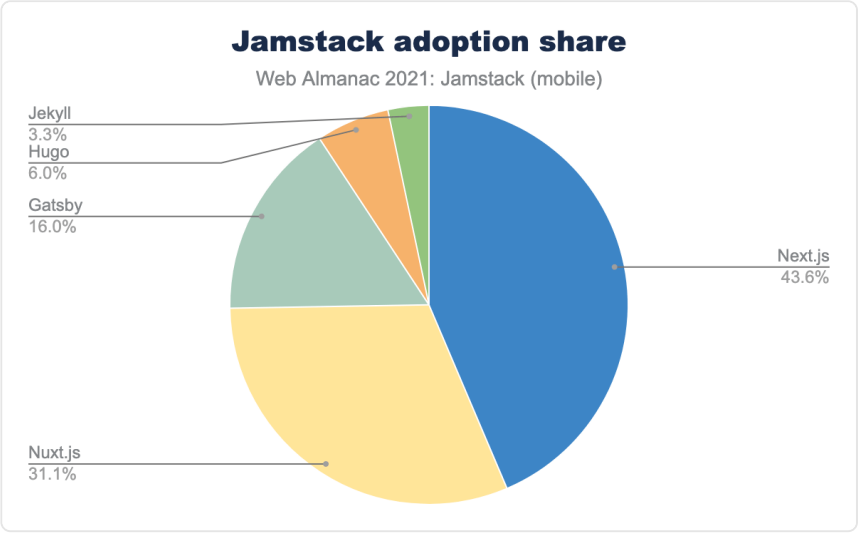


図18.2. SSGの採用シェア。

まずは、もっとも普及しているSSGを把握することからはじめましょう。Next.jsはJamstackサイトの43.6%をカバーしています。2位はNuxt.jsで31.1%、3位はGatsbyで16.0%、4位はHugoで6.0%となっています。

なお、本章の初出時においては、NuxtとNextのサイト数が誤って多く計上されていたため、数値が異なっています。本章の他の数値も含め、上記の数値は修正されています。

上位3つのSSGは、すべてJavaScriptベースです。Next.jsとGatsbyは、React.js[863]をコアとして、その上に独自の機能を追加することで補完しています。Nuxt.jsは、Vue.js[864]をベースにしています。これらの人気のあるフロントエンドフレームワークは、巨大なエコシステムを備えているため、開発が非常に容易になります。Node.js[865]は、従来使われているブラウザだけでなく、サーバ上でもJavaScriptを実行できるようにし、開発者が1つの言語に固執することを可能にしています。そのため、プログラミング言語Go[866]をベースにしたHugoや、Ruby[867]ベースのJekyllと比べて、サーバの観点からもこれらのSSGの採用が容易になるのだそうです。

今回は、WebサイトにおけるSSGの採用率について見ていきます。

863. https://reactjs.org/
864. https://vuejs.org
865. https://nodejs.org/en/
866. https://go.dev/
867. https://go.dev/

ランク別採用状況

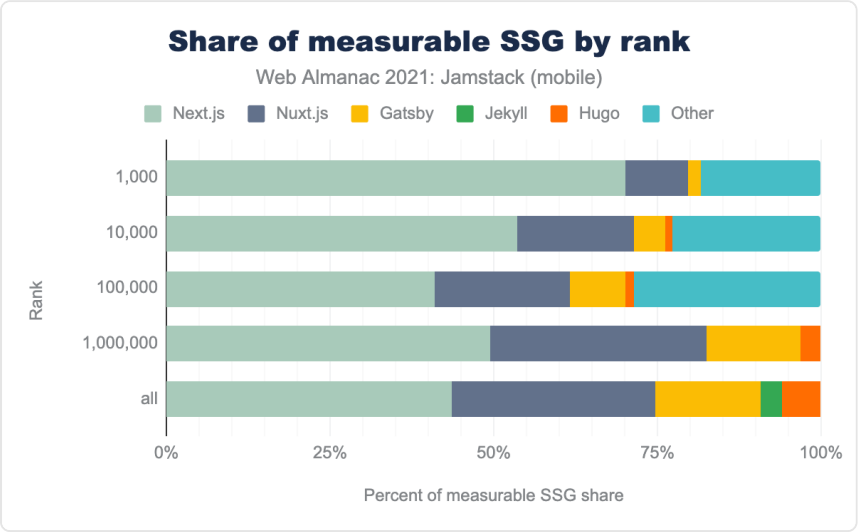


図18.3. SSGのランク別採用シェア。

Next.jsは、すべてのランクで依然として人気のあるSSGですが、とくに上位10k位までのSSGに人気があります。

地域別採用状況

このセクションでは、Jamstackの地理的な採用を取り上げ、国や地域ごとの分布を探ります。

国別採用状況

SSGは世界中で盛んに利用されています。下図は、サイト数のもっとも多い上位10カ国を示したものです。

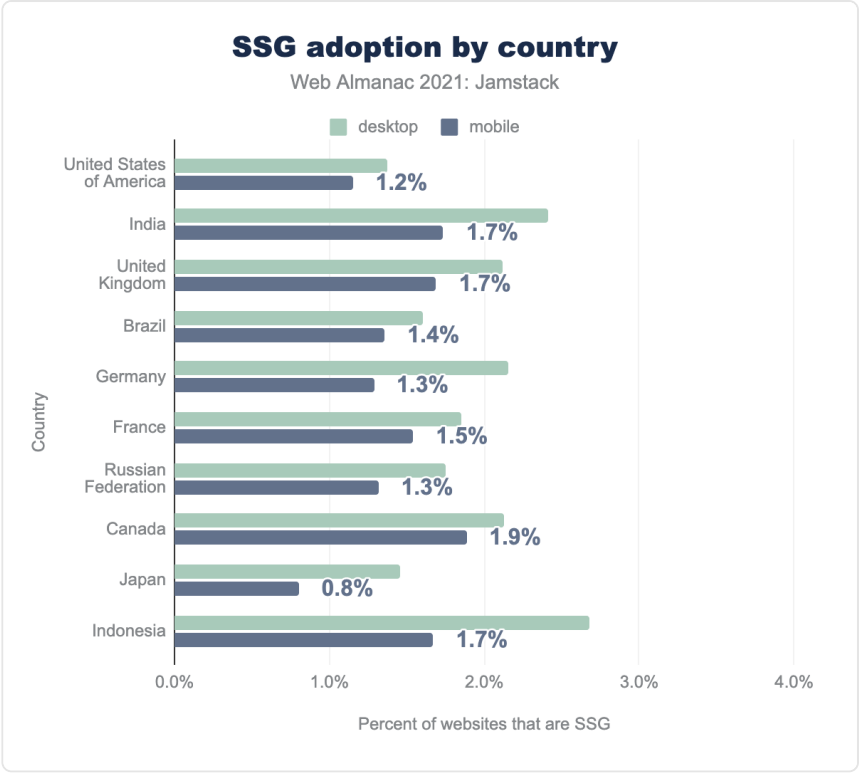


図18.4. 国別のSSG採用状況。

米国では、全サイトの1.2〜1.4%のページ（デスクトップ用で約22kページ、モバイル用で約16kページ）がSSGで作成されています。インドはデスクトップ用で6kページ、モバイル用で7kページと少ないが、全ページの1.7%がJamstackの技術でカバーされている。3位はイギリスで、こちらも1.7%のページがある。

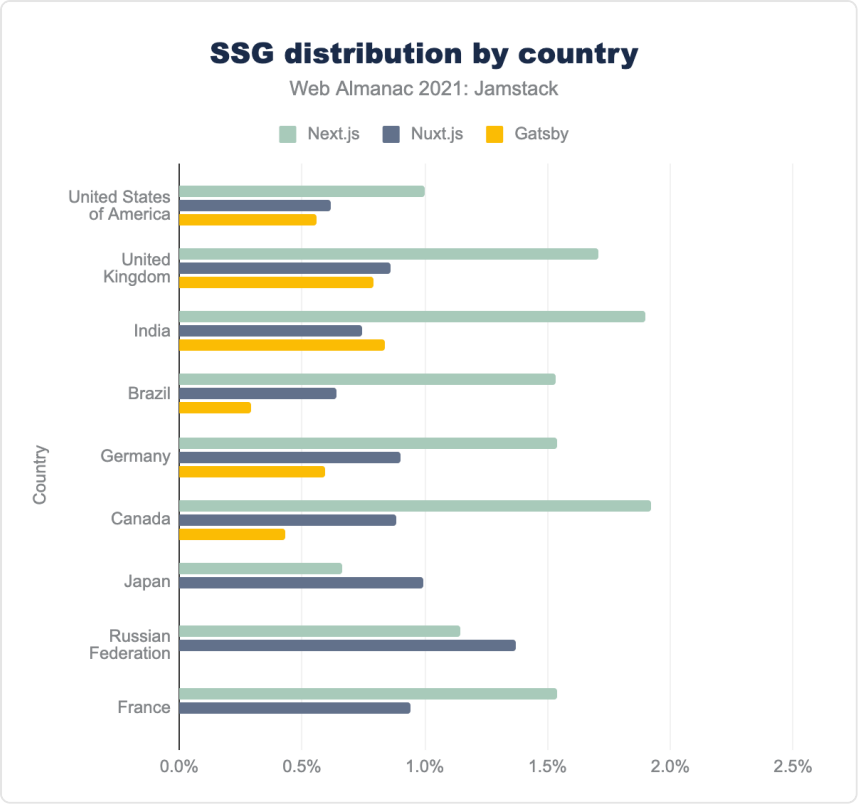


図18.5. 国別のSSG分布。

アメリカはNuxt.jsやGatsbyに比べ、Next.jsの採用が多い。ほぼすべての国で同じような傾向です。ほとんどの国で、Next.jsは好ましい選択です。興味深いことに、GatsbyはJamstackの技術を使用している上位10カ国のうち3カ国のデータがありませんが、そのうちの2カ国は日本とロシア連邦でNuxt.jsがより好まれているようです。

地域別採用状況

また、地域別の普及率も調べました。

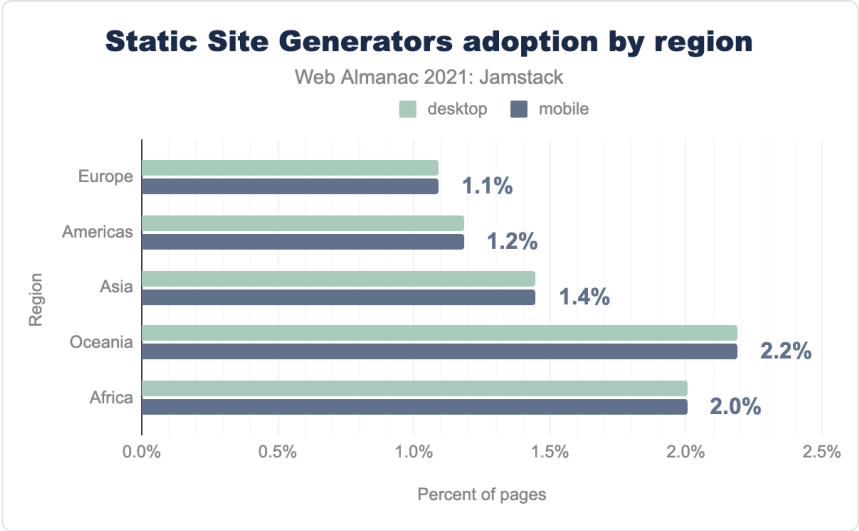


図18.6. SSGの地域別分布。

欧州のサイト数は、デスクトップが2万3千サイト、モバイルが2万6千サイトで、同地域の全ウェブサイトの1.1%にあたる。アメリカ大陸では、デスクトップが2万6千サイト、モバイルが2万4千サイト（全体の1.2%）です。アジアはデスクトップが2万1000件、モバイルが2万2000件とほぼ同数で、Jamstackの導入率が高い地域のトップで1.45%。オセアニアとアフリカは、全体の数は少ないが、Jamstackの採用率は高い。オセアニアは2.19%、アフリカでは2%です。サイト全体の導入率は1.1%です。

サブリージョン別採用状況

さらに小地域別に分類して、さらなる傾向を観察できます。

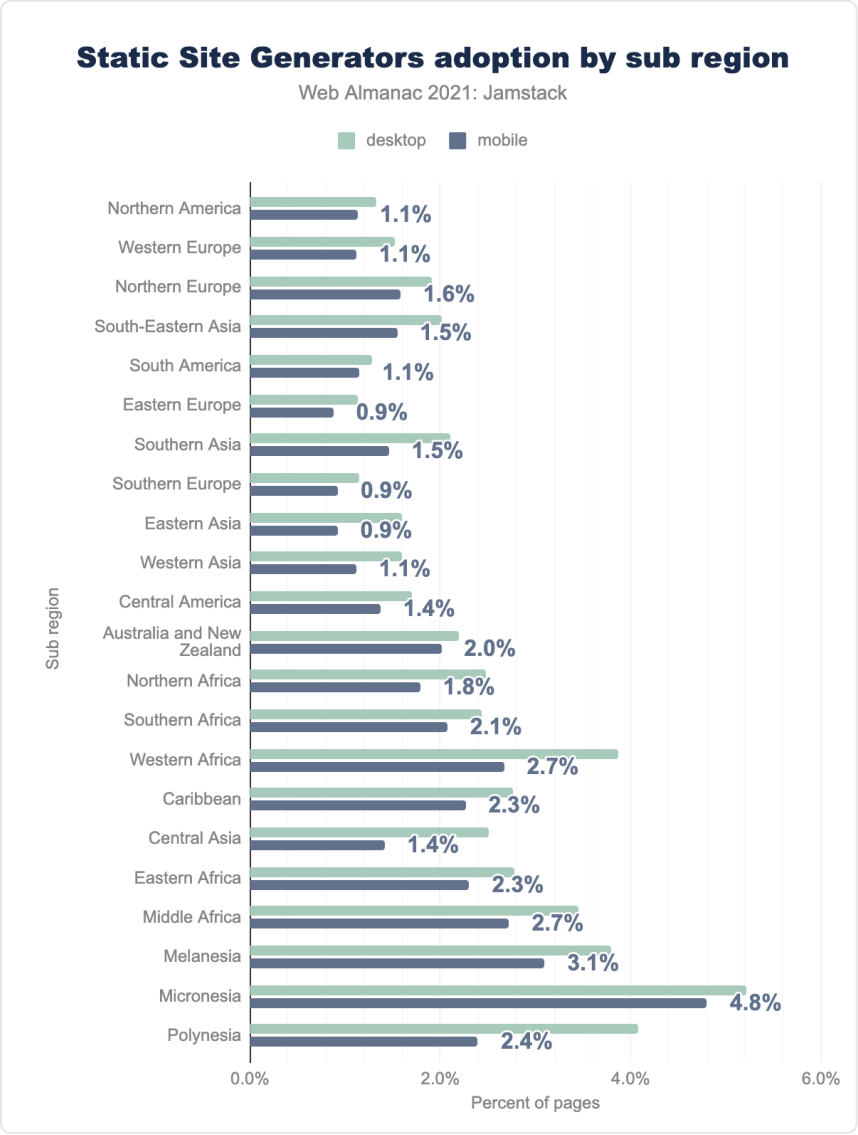


図18.7. サブリージョン別のSSG分布。

このリストはSSGサイトの総数で並べられ、その地域の全サイトに対する割合で表示されます。SSGを開発した企業の多くがアメリカにあるため、北アメリカがトップであることは驚くことでありません。しかし、全サイトに占める割合は低く、Jamstackを採用したサイトはわずか1.1%です。しかし驚くべきことに、西ヨーロッパは2位で、リストの下位にあるいくつかのサブ地域と比較して、同様の低い割合の採用となっています。

尾部も素晴らしい結果を示している。たとえば、ミクロネシアでは4.8%というように、サイト数の少ないサブリージョンほど、より広い範囲で技術を導入している。

CDNプロバイダー間でのSSGの配布

国によってSSGがどのように採用されているかを説明しましたが、ここではCDNプロバイダーによってどのSSGがもっとも普及しているかを分析してみましょう。

SSGでもっとも普及しているCDNプロバイダーは7社です。

- Netlify
- Vercel
- Cloudflare
- AWS
- Azure
- Akamai
- GitHub

Jamstack CDNサービスは、単なるネットワーク配信のためのものではありません。開発者がJamstackサイトを簡単にデプロイし、管理できるようにするための多くの機能を提供している。たとえばNetlifyは、開発者がコードを更新するだけで、継続的なデプロイプロセスが管理されるように、サービスの範囲内でサイトをデプロイする機能を簡単に使用できるよう提供している。Jamstack CDNは、*サーバレス機能*、*A/Bテスト*など、他にも多くの機能を提供します。

一方、Cloudflare、Akamai、AWSは純粋にコンテンツを配信するためだけに使われているわけではなく、保護サービスやDNSバランシングなども提供することができる。しかし、Cloudflare、Akamai、AWSがどのように利用されているかは検知できないため、Jamstackのイネーブラーとして見た場合、結果は誤検出となる可能性があります。「Jamstack」の部分はオリジンサーバーで処理されるため、実際にはこれらのサービスでは処理されない可能性があります。

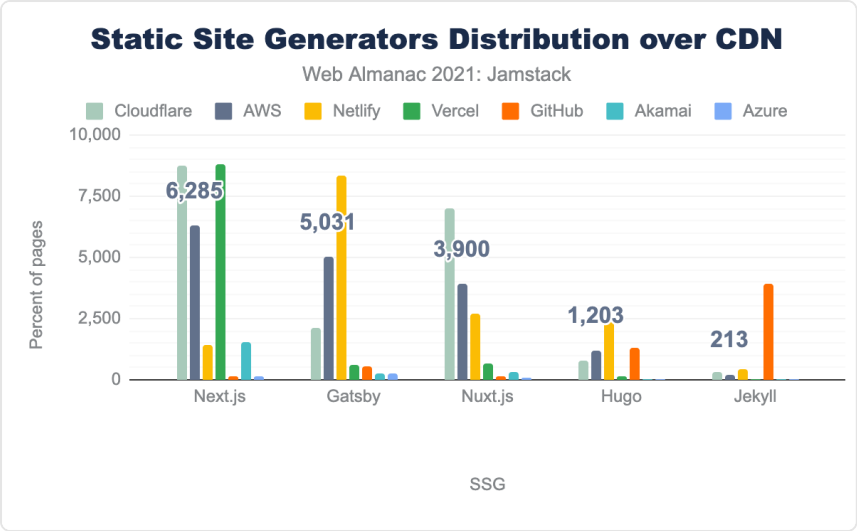


図18.8. CDNを利用したSSG配信。

Next.js、がもっとも普及しており、そのほとんどがCloudflare、Vercel、AWSで提供されています。Gatsbyのサイトのほとんどは、Netlify、AWS、Cloudflareを使用しています。Nuxt.jsのサイトは、Cloudflare、AWS、Netlifyによるサービスを好んで利用しています。HugoはほとんどNetlifyを使用しており、JekyllはGitHubで主に使用されていることは驚くことでありません。

次のグラフでは、人気のあるCDNについて、利用されるCDNの相対的な割合を示しています。

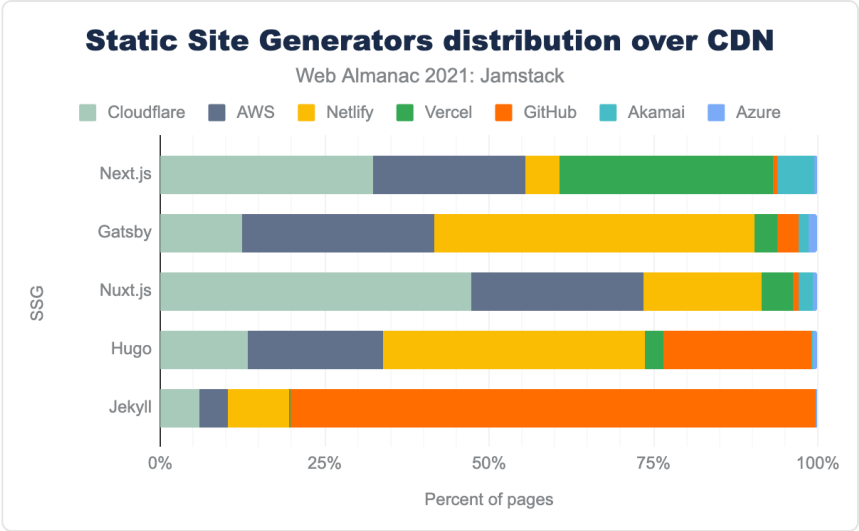


図18.9. CDNを利用したSSG配信。

Next.jsは、そのほとんどがVercel（Next.jsを開発した会社）によって提供されています。NetlifyやVercelのようなJamstackに特化したサービスとは対照的に、AWSのような一般的なCDNはJamstackサイトに大きな割合でサービスを提供していないことがわかる。

GitHubをCDNプロバイダーとするのは珍しいかもしれませんが、*GitHub*ページにより、ユーザーはJekyll SSGで構築したgithub.ioサブドメインにサイトをデプロイすることができるようになります。

ユーザー体験とパフォーマンス

今回の分析では、Jamstackの技術を採用した1.1%のサイトがどのようなユーザー体験をしているのかを探りました。Lighthouseとコアウェブ・バイタルの結果に注目しました。

Lighthouse

Lighthouseのスコアは、すべて弊社クロールによる模擬テストデータです。そのため、モバイルデータプロバイダーや実際に使用するデバイスによって、実際の結果が影響を受ける可能性があります。

パフォーマンススコア

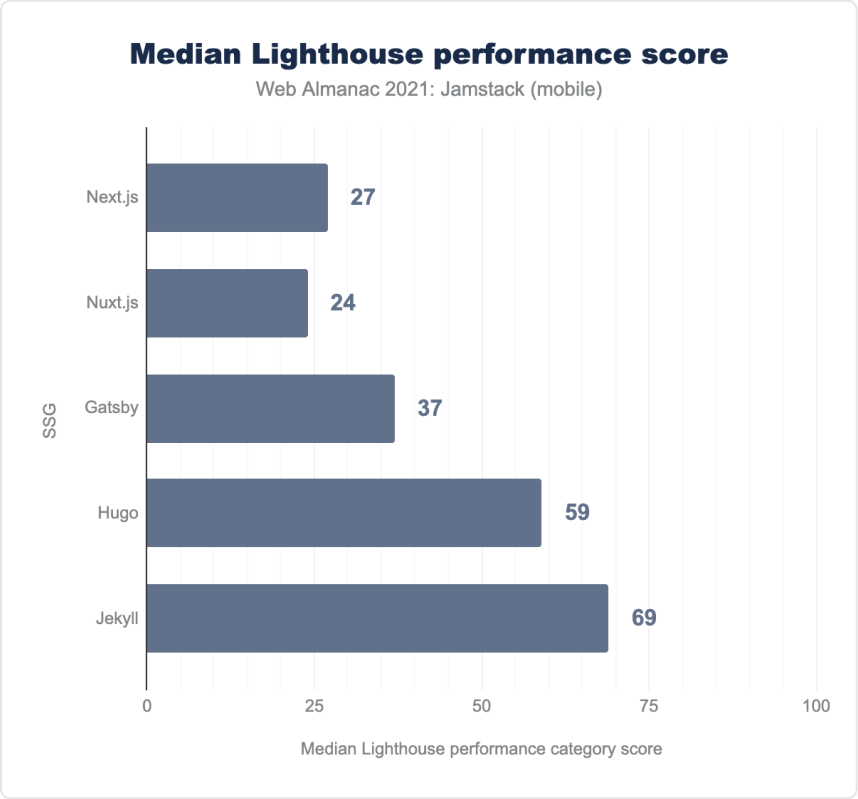


図18.10. Lighthouseのパフォーマンススコアの中央値。

モバイルの全SSGのパフォーマンススコアの中央値はさまざまです。人気上位3つのSSGは、40点を超えることができません。ランキング上位のサイトで使用されていること、ユーザーが世界中に分散していることから、さまざまなデバイスやネットワークで使用されていることが推測されます。Next.js画像コンポーネント[868]のように、すぐに使える改善でパフォーマンスを向上させることが期待できます。

とくにJekyllは70点近くを獲得しており、SSGエリアのマストドンとしては素晴らしい結果だと思います。Lighthouseパフォーマンス監査[869]の詳細については、このスコアに含まれる指標を正確に理解するためにご覧ください。

868. *https://nextjs.org/docs/basic-features/image-optimization*
869. *https://web.dev/i18n/ja/lighthouse-performance/*

アクセシビリティスコア

Lighthouseでは、アクセシビリティを測定するための監査[870]も行っており、こちらの方が良い結果を出しているようです。

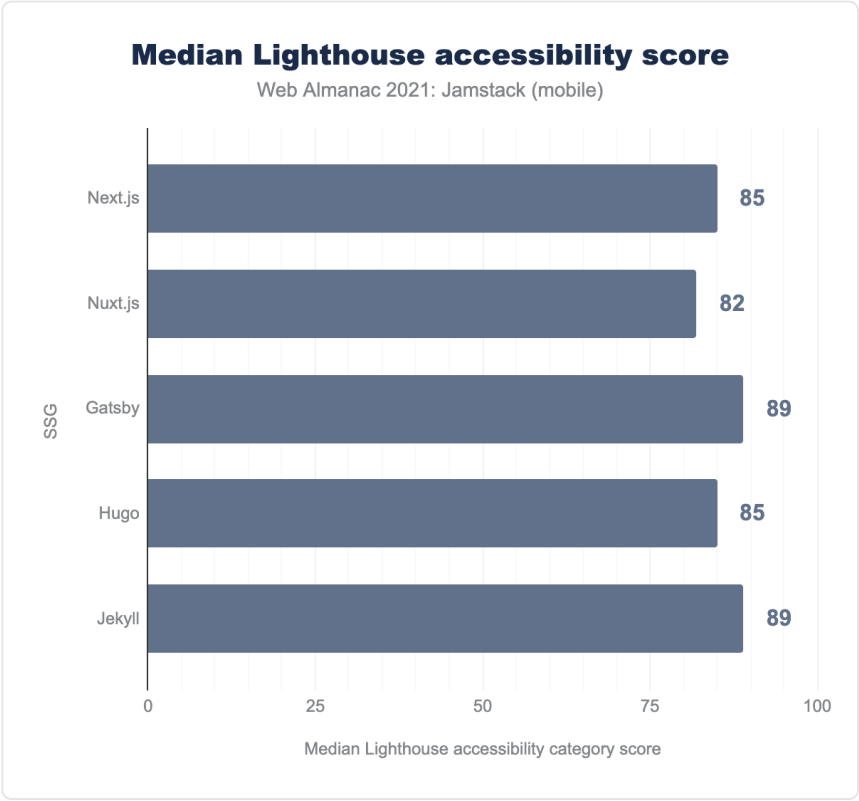


図18.11. Lighthouseのアクセシビリティスコアの中央値。

自動化されたアクセシビリティ・チェックでチェックできる内容には限界がありますが、それでもこれは好ましい兆候です。このテーマについては、アクセシビリティの章をお読みください。

870. https://web.dev/i18n/ja/lighthouse-accessibility/

SEOスコア

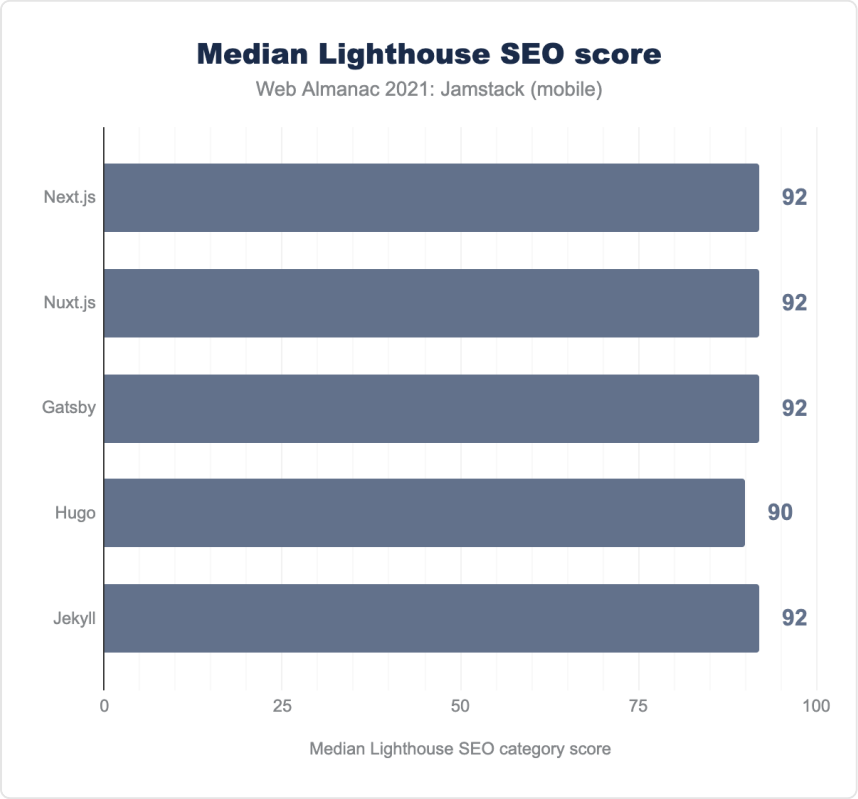


図18.12. LighthouseのSEOスコアの中央値。

同様に、Jamstackのサイトはすべて90から92の素晴らしいSEOスコアを提供しています。静的なコンテンツを使用することは、常にデフォルトでSEOフレンドリーな技術でした。さらに、SSGは検索エンジンのためにサイトを最適化するための追加機能を備えています。

つまりLighthouseの結果は全般的に良好ですが、SSGのメインターゲットはパフォーマンスとPWAであり、これらのカテゴリでは開発者の体験を改善するための作業が必要で、その結果、サイトのパフォーマンスが改善されるということです。

コアウェブ・バイタル

コアウェブ・バイタル[871] (CWV) は、ウェブ上で優れたユーザー体験を提供するために不可欠な品質シグナルの統一ガイダンスを提供するためのイニシアチブです。CWV自体は、3つのパフォーマンス指標を使用しています。

- 最大のコンテントフルペイント (LCP) - ページのメインコンテンツと推定される部分のロードタイムを計測します。
- 最初の入力までの遅延 (FID) - 相互作用の遅延を測定するものです。
- 累積レイアウトシフト (CLS) - ページが読み込まれ、ユーザーがコンテンツを読むときにコンテンツが移動しないように、視覚的な安定性を測定するものです。

私たちは、これらの値の実際のユーザーデータを集めている*Chrome UX体験レポート*（CrUX）を使用したので、Lighthouseが提供するラボベースのパフォーマンス指標よりも、実際のユーザー体験をよりよく測定することができるのです。

SSGのデータを分析しましたが、これはSSGがどのように配信されているかも反映しています。上記で見たように、いくつかのサイトは異なるCDNで多かれ少なかれ使用されており、そのためパフォーマンスに良い（または悪い！）影響を与える可能性があるので、そのデータも見ています。

SSGの総合評価では、Jamstackサイトの基本的なパフォーマンス レベルを理解できます。CWV評価には、すべてのメトリクスでCWVの良いスコアを持つページロードの75%パーセンタイルのデータが含まれています。

871. *https://web.dev/i18n/ja/learn-web-vitals/*

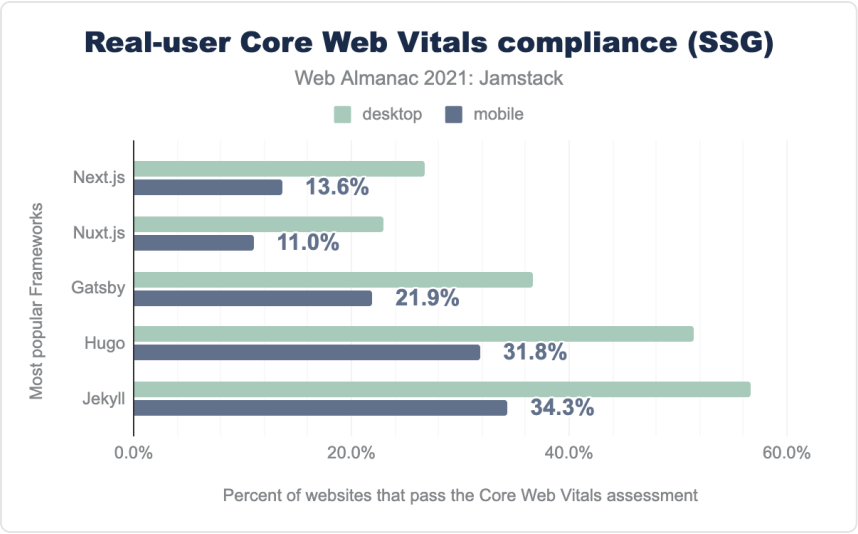


図18.13. 実ユーザーによるコアウェブバイタルへの対応。

モバイルでの結果を見ると、JekyllとHugoは全サイトの34.3%と31.8%が良いスコアを獲得し、SSGを上回る結果となりました。Gatsbyは21.9%で3位ですが、JavaScriptベースのSSGの中では1番です。Next.js 13.6%、Nuxt.js 11.0%と、パフォーマンスの良いページが、多いのが特徴です。

最大のコンテントフルペイント

最大のコンテントフルペイント[872](LCP) メトリクスは、ページの読み込みを最初に開始したときからの相対時間で、ビューポート内で表示される最大の画像またはテキスト ブロックのレンダリング時間を報告します。

872. *https://web.dev/articles/lcp*

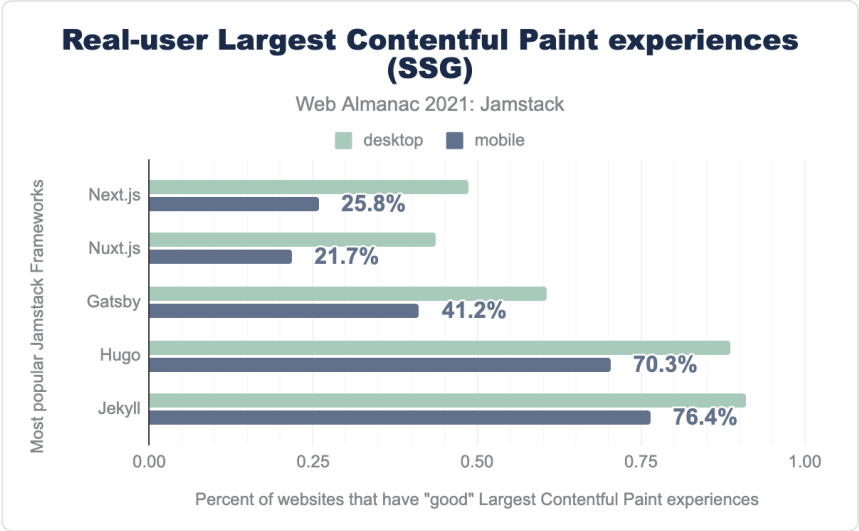


図18.14. 実ユーザーによるコアウェブバイタルLCP。

上では、同じ結果が良好なLCP体験を持つサイトの割合で承認されていることがわかります。もっとも良い結果はJekyllとHugoで、モバイルサイトの76.4%と70.3%が2.5秒以下の「良い」LCPを獲得しています。JavaScriptベースのSSG（Gatsby、Next.js、Nuxt.js）は、より悪い結果となっています。

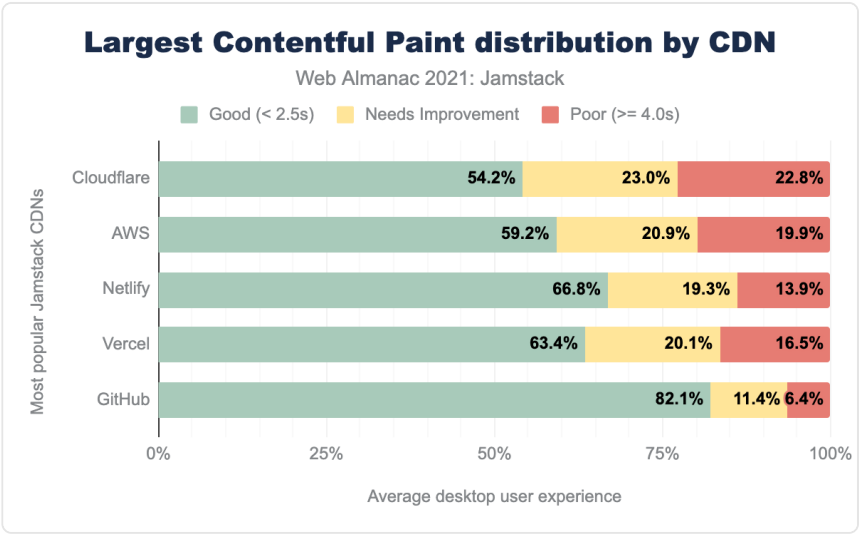


図18.15. CDN向けLCP配信。

GitHubはCDNレベルで測定した場合、トップとなり、おそらくここでホストされているサイトがよりシンプルであることを反映している。次に、Jamstack向けのCDNであるNetlifyが66.8%のサイトでLCPが良好であり、Vercelが63.4%、AWSで59.2%、Cloudflareは54.2%で続いています。

最初の入力までの遅延

最初の入力までの遅延[873] (FID) はユーザーがページと最初に対話したとき（すなわち、リンクをクリックしたとき、ボタンをタップしたとき、またはJavaScriptを使用したカスタムコントロールを使用したとき）から、その対話に対応してブラウザがイベント ハンドラーの処理を実際に開始できるまでの時間を測ります。

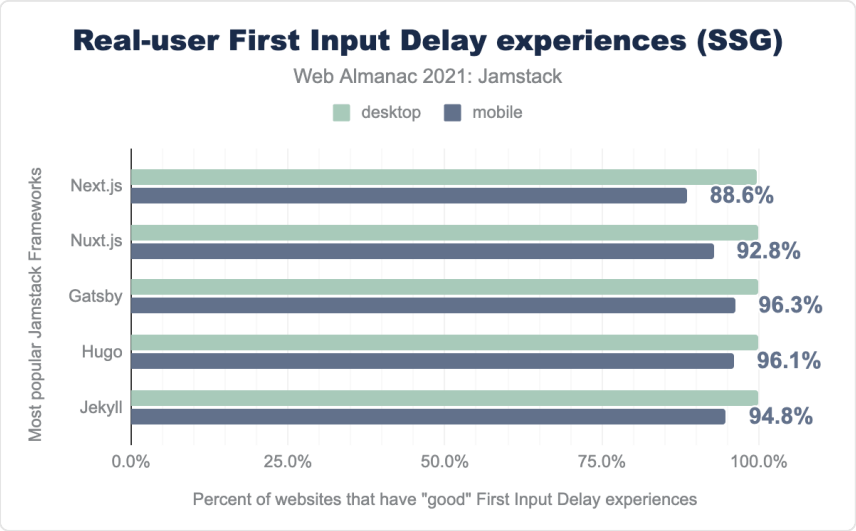


図18.16. リアルユーザーのコアウェブバイタルFID。

実際のユーザー体験では、すべてのSSGが異なるSSG間で素晴らしいFIDの結果を示しています。

873. https://web.dev/i18n/ja/fid/

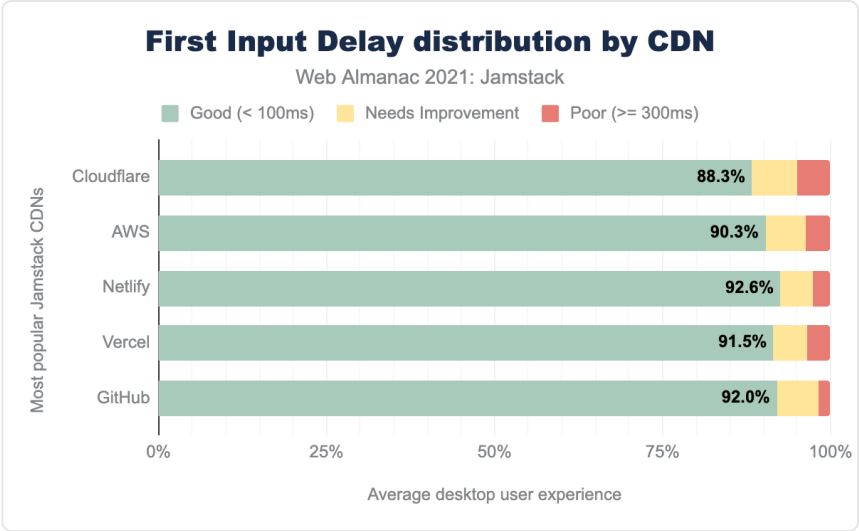


図18.17. CDN向けのFID配信。

すべてのCDNがJamstackのサイトを88%以上の良好なFIDで配信しているが、CloudflareとAWSのサイトはJamstack向けのCDNよりも若干悪い結果となっているのは興味深いところだ。

累積レイアウトシフト

累積レイアウトシフト[874] (CLF) は、ページの全寿命の間に発生する予期せぬレイアウトシフトごとで、*レイアウトシフトのスコア*の最大のバーストを測定するものです。

874. *https://web.dev/i18n/ja/cls/*

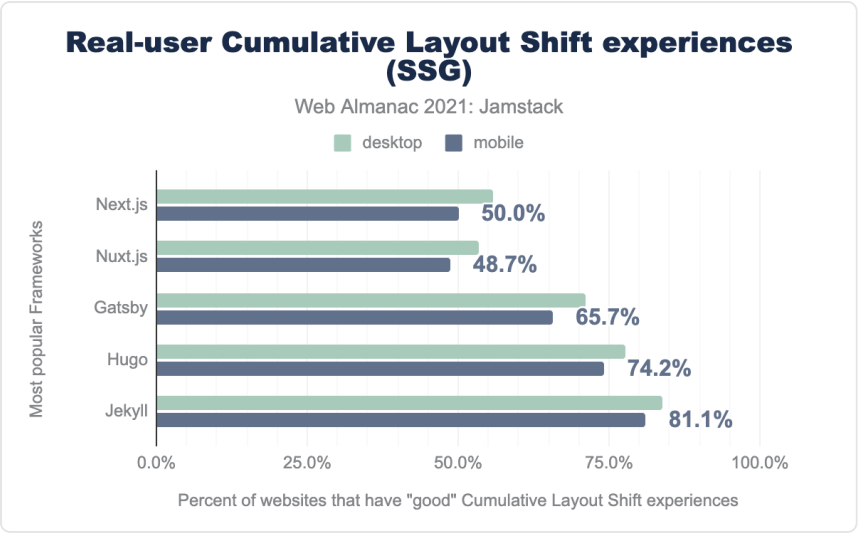


図18.18. 実ユーザーによるコアウェブバイタルCLS。

ここでもJekyllは素晴らしいパフォーマンスを発揮しています。モバイルの81.1%が良い結果です。次いでHugoが74.2%、Gatsbyが65.7%、Next.jsが50.0%、Nuxt.jsが48.7%と後塵を拝しています。

CDNについては、前回と同じ結果です。GitHub、Netlify、Vercelです。

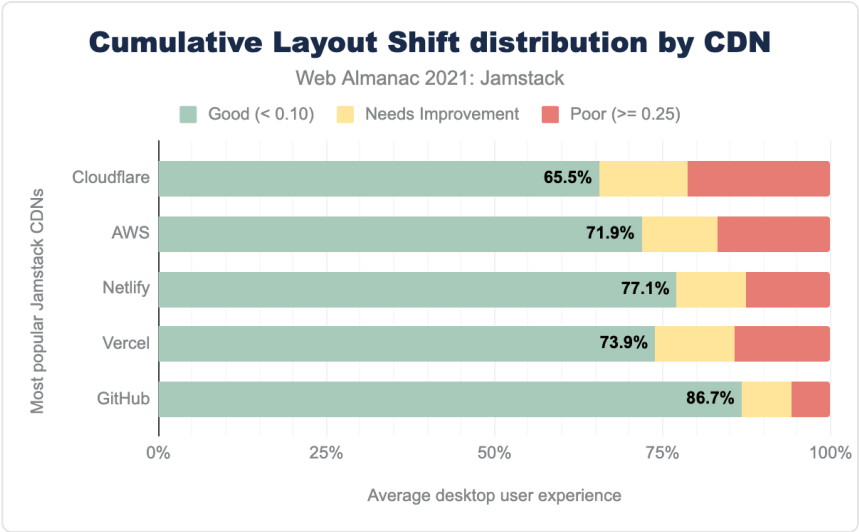


図18.19. CDN向けCLS配信。

一般的にCWVの結果は、Lighthouseの結果を反映しています。HugeとJekyllは、より良い実際のユーザーのパフォーマンスデータを持っています。我々は、これらのSSGを使用して構築されたどのように複雑なサイトを検出することはできません。我々はNext.js、Nuxt.js、Gatsbyのような最新のSSGで多くのJavaScriptが配信され、画像を含むより多くのデータをレンダリングされていることを確認できます。そのため、パフォーマンスに影響が出るのです。それでも、GitHubとJekyllの間に興味深い相関があり、同時に素晴らしい結果を示しています。

リソース

ここでは、上位5つのSSGのリソースウェイトを調査し、パフォーマンスへの影響度を把握します。結果は中央値で表示しています。

リソースの重量

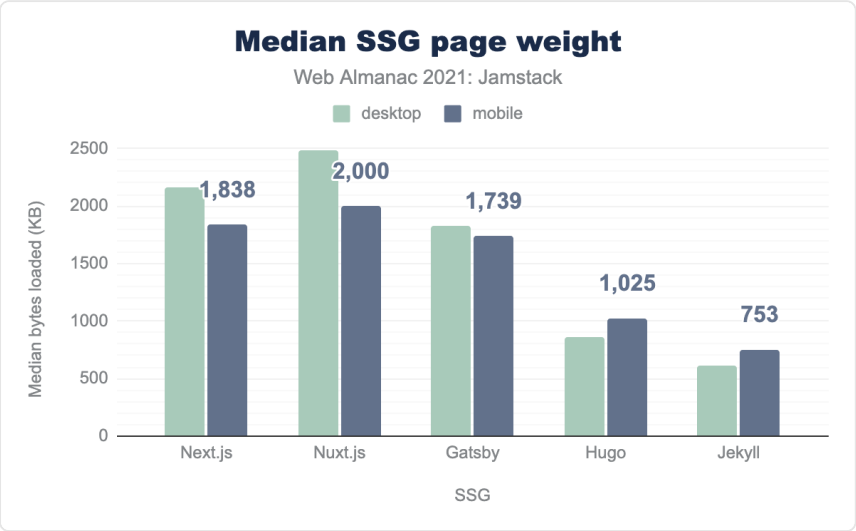


図18.20. ページの重さの中央値。

JavaScriptベースのSSGは、HugoやJekyllの2倍近いリソース量になります。トップはNuxt.jsで〜2MB、次いでNext.jsでほぼ1.8MB、Gatsbyで1.7MBとなっています。

前述したように、JavaScriptベースのSSGはJavaScriptのフレームワークをそのまま含んでいます。それは開発を容易にしますが、より多くの責任を必要とします。JavaScriptのエコシステムは、さまざまな目的のために、サイトにどんどんライブラリを追加することを容易にし、バンドルサイズを大きくすることにつながります。

JavaScript

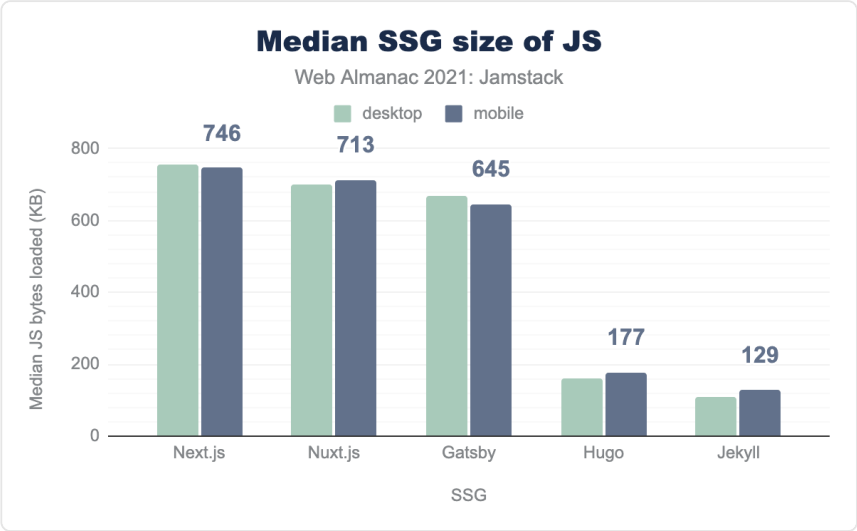


図18.21. JavaScriptの重量の中央値。

リソースの大きな塊はJavaScript用です。ここでもJavaScriptベースのSSGは他と比べてかなり大きく、非JavaScriptベースのSSGが約150KBであるのに対して、約700KBとなっています。これは驚くべきことでありませんが、このような形で実際の違いを見ることができるのは興味深いことです。Next.jsベースのサイトは、他のサイトよりも多くのJavaScriptを使用しています。一方、HugoとJekyllの開発者は、より責任を持ってJavaScriptを使用し、そのバンドルをタイトに保っているようです。そのためのもう1つの理由は、サイトの複雑さかもしれません。HugoとJekyllのサイトは、上位のランキングサイトにあまり表示されないので、彼らは、たとえば上位のランキングサイトに頻繁に表示されるNext.jsのサイトよりもシンプルなユースケースを持っているかもしれません。

SSGの中でどのようなサードパーティライブラリが使用されているかを分析しました。SSGで使用されている他のライブラリやフレームワークを把握するため、ReactとVueは除外しています。

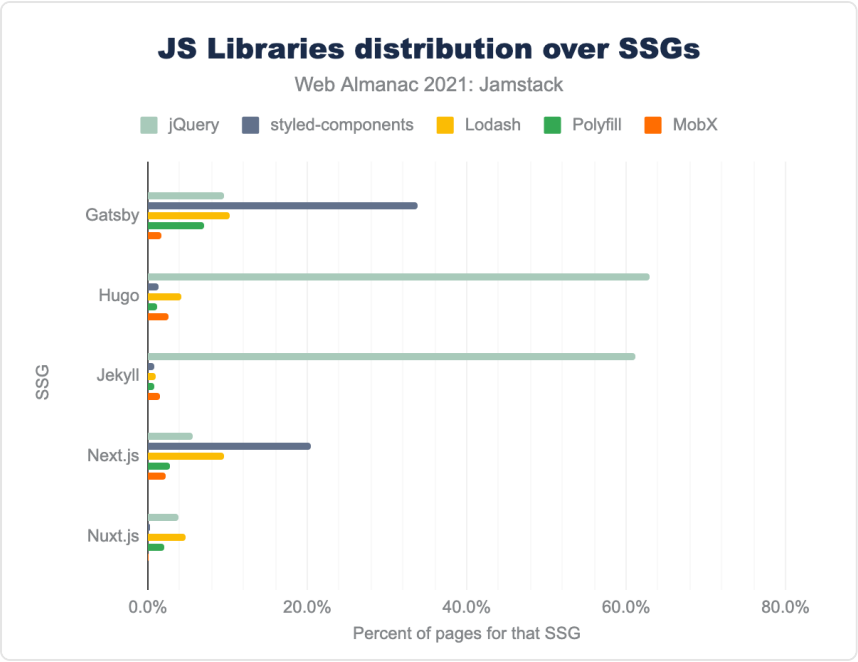


図18.22. SSG上でのJavaScriptサードパーティ配布。

大きな驚きだったのは、jQueryです。HugoやJekyllベースのサイトで使われている（60%以上）のは驚きではありませんでしたが、ReactやVueベースのサイト内で使われているのは予想外でした！Next.js、Many Nuxt.js、GatsbyのサイトでもjQueryが使われています。Next.js、Many Nuxt.js、GatsbyのサイトもjQueryを使用しています。

サードパーティライブラリのうち、Styled-componentsはNext.jsで20%、Gatsbyでは34%使用されています。Nuxt.jsのサイトでは、ほとんど使われていないようです。

Lodashは多用されており、Gatsbyでは10%まですべてのSSGに存在した。

CSS

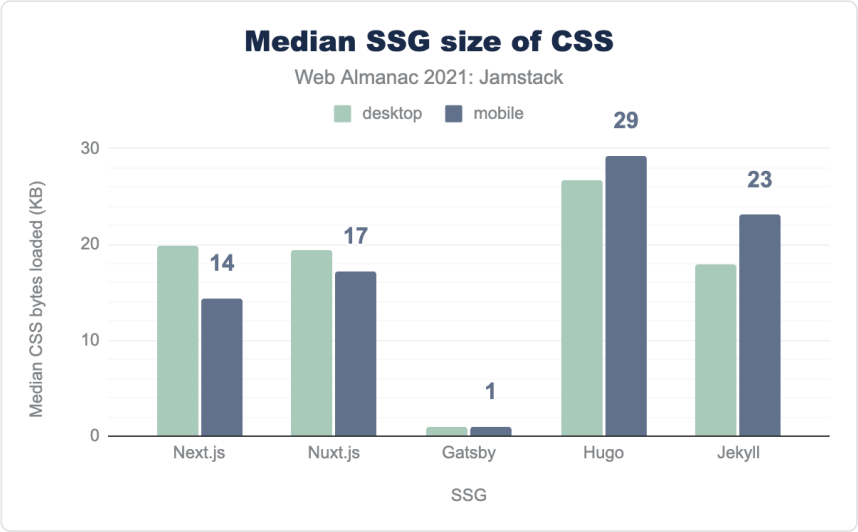


図18.23. CSS重量の中央値。

一方、CSSはHugoやJekyllよりも若干重くなっています。 styled-componentsの利点は、繰り返しのないきれいなCSSであるため、これらのJavaScript SSGのCSSのサイズが小さくなるのは、このためかもしれません。もうひとつの仮説は、旧式のSSGはCSSを使ってインタラクションやアニメーションを処理するために旧式の方法を用いているということです。JavaScriptベースのSSGは一般にJavaScriptを多く使うので、CSSで実装可能な機能を置き換えるために使われることが多いのかもしれません。

画像

画像の重みの配分が違う。SSGグループ間の相関はない。

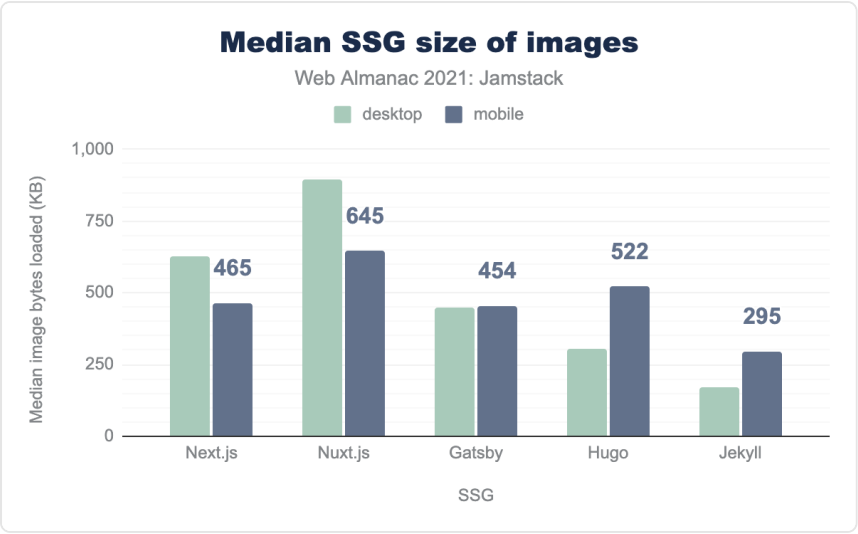


図18.24. 画像の重みの中央値。

Nuxt.jsは645KBともっとも高い値を示しています。次がHugoで522KB。Next.jsは465KB、Gatsbyは545KBとほぼ同じ。Jekyllは295KBともっとも低い値になっています。

画像フォーマット採用

画像は、優れたユーザー体験（UX）のボトルネックの1つです。サイズが大きければ、ユーザーは画像が届くまで長い時間待たされることになります。レイアウトがずれるなどの問題にもつながります。

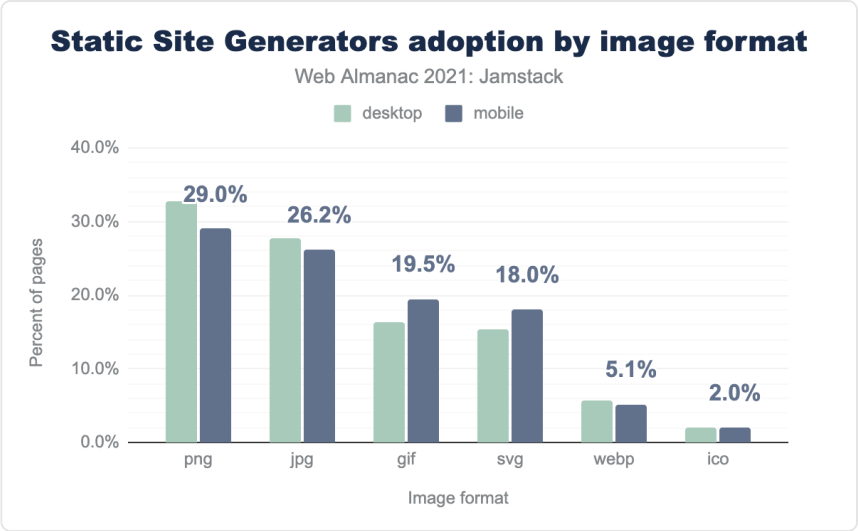


図18.25. 画像フォーマットの採用。

新しい世代の画像フォーマットの1つであるWebP[875]は、Jamstackサイトでの使用率が5.1%となっています。WebPが3%しかなかった昨年の結果[876]と比較すると、1年前より良くなっていると言えるでしょう。

それでももっとも使われているのはPNGで29.0%、JPEGは26.2%、GIFが19.5%、そしてSVGは18.0%のWebページで使われているそうです。

資料からわかること

このリソース重量の分析から、Next.js、Nuxt.js、Gatsbyのパフォーマンスは、巨大なリソースのために苦労していると思われることが確認されました。2MBのページウェイトと700KBのJavaScriptは、とくに平均的なモバイルデバイスと低速のネットワークにおいて、パフォーマンススコアに確実に影響を与えるでしょう。Next.jsとGatsbyのサイトでは、styled-componentsを多用することも、パフォーマンスが低下する原因かもしれません[877]。ポジティブなシグナルは、次世代画像フォーマットの画像採用が進んでいることであり、これは長期的にはエンドユーザーのUXを向上させるはずです。

875. *https://developers.google.com/speed/webp/*
876. *https://almanac.httparchive.org/ja/2020/jamstack#画像フォーマット*
877. *https://pustelto.com/blog/css-vs-css-in-js-perf/*

結論

ヘッドレスCMSや、いくつかの有名なSSG（EleventyやNext.jsの検出モード）を含めることができないという制限はあるものの、ここでは興味深い結論を導き出すために多くのデータを分析できます。Jamstackのトレンドは年々高まっており、現在では全ウェブサイトの1%以上がJamstackベースとなっています。

Next.jsは、Jamstackで計測可能なサイトの約40%をカバーしていることが分かっています。また、トレンドだけでなく、上位1,000サイトの3.8%で利用されており、Nuxt.jsやGatsbyといった他の人気SSGがそれに続いています。これらはいずれも参入して数年の比較的新しいプレイヤーですが、ランキング上位のサイトでもよく利用されており、その地位を確固たるものにしています。

SSGは世界中で利用されており、このモデルの創業企業が拠点とする国に限定されるものではありません。実際、Jamstackの技術を採用し、最大5%のサイトが急成長しているのは、シリコンバレーのハイテク拠点からもっとも遠い地域であるようです。

他のウェブサイトと同様にJamstackサイトの良好なパフォーマンスを維持するには、ベストプラクティスの知識と経験豊富な開発者レベルが必要ですがSSGはその辺りを改善するために、すぐに使えるソリューションに取り組むことで、これを改善できます。データを楽しみながら、Jamstackを試していただけると幸いです。

著者



Artem Denysov

@denar90_ denar90

Artem Denysov は、ソフトウェアエンジニア、オープンソースの貢献者、Mozillians のメンバー、講演者、そして執筆者です。Webperfとツールで、開発者とユーザーの生活をより快適にします。Stackbit[878] で、開発者が Jamstack ウェブサイトを簡単に構築できるようにするために働いています。Twitter[879] と Linkedin[880] でご覧いただけます。

878. *https://stackbit.com*
879. *https://x.com/denar90_*
880. *https://www.linkedin.com/in/denar90/*

部 IV 章 19
ページの重さ



***John Teague** によって書かれた。*
Sia Karamalegos と Rebecca Holmlund によってレビュ。
Jess Peck による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

私のようなウェブパフォーマンスジャンキーでない限り、ウェブページの重さは、切手をなめるのと同じくらいエキサイティングなことです。しかし、なぜページの重さが重要なだけでなくクリエイター、ホスティングプロバイダー、そして消費者に影響を与えるもっとも重要な要素であると言えるのか、私は全力であなたを説得しようと思っています。そのために、ページの重さがウェブサイトやウェブアプリケーションのパフォーマンスにどのように影響するか、ページの重さがユーザー体験にどのように影響するか、そして私たちがウェブページの重さを減らすことができるいくつかの方法を実際のデータを使って説明します。

過去10年間で、平均的なWebページの重量[881]は、平均約484キロバイトから2,205キロバイトへと、実に356%も増加しました。この増加は、需要と供給の関数として説明できます。画像、ビデオ、オーディオ、フォント、データの収集と処理、WebサイトやWebアプリケーシ

881. https://httparchive.org/reports/page-weight

ョンの分析、監視、警告機能などの接続サービスの利用の増加に合わせて、コンピュータプロセッサの高速化、データ転送、データの保存と提供の方法のすべてが進歩したのです。

何千ドルもするハイエンドのスマートフォンやデスクトップパソコン、ノートパソコンを所有し、高価な高速インターネットプロバイダーや5Gデータプランに接続していれば、すべてがうまくいくように思えます。しかし、インターネット接続が不安定な低速の3Gや4Gのデータプランを利用することになると、そのようなインターネットユーザーの階級に属することの喜びは失われていきます。多くのインターネットユーザーにとって、完全に読み込まれないかもしれないページを待つことは、インターネットの約束を破り、緊急時[882]に人命を危険にさらすことさえあるのです。

データセンターとそれが提供する機器に電力を供給するために、多くのエネルギーが使用されています。私たちはファイルのペイロードをより小さくし、ペイロードの転送をより速く、より効率的にすることで、全体のエネルギー需要の削減に貢献できます。

Googleは現在、コアウェブ・バイタルを良好に達成できていないウェブサイトに対して、検索ランキングのペナルティを課しています。その成否を評価する指標のひとつが「ページ重量」だ。あなたが興味を持っている場合は、Googleのページスピードインサイト[883]とGoogleの測定[884]を使ってあなたのサイトをテストできます。どちらも、重いWebページに起因するパフォーマンスやユーザー体験の問題を解決する方法について、貴重な知見を提供しています。

ウェブページをより軽く、より速くするためには、ページ重量とは何かということを理解することが大切です。そこで、さらに深く掘り下げてみましょう。

ページ重量とは何ですか？

ページ重量は、特定のウェブページの総バイト数を表します。ウェブページは、ウェブブラウザでレンダリングして見ることができる特定の要素や資産で構成されており、以下のようなものがあります。

- ページそのものを構成するHTMLのことです。
- ページに埋め込まれた画像やその他のメディア（ビデオ、オーディオなど）。
- ページのスタイル設定に使用される カスケーディング・スタイル・シート (CSS) です。
- JavaScriptでインタラクティブ性を持たせる。

882. *https://www.nbcnews.com/tech/tech-news/verizon-admits-throttling-data-calif-firefighters-amid-blaze-n902991*
883. *https://pagespeed.web.dev/*
884. *https://web.dev/measure/*

- 上記の1つ以上を含むサードパーティ リソース。

これらのリソースはそれぞれ、重量（バイトサイズ）、送信、処理、およびWebブラウザでのレンダリングで必要な計算リソースに相当するコストがあります。ある点では同じようなコスト（ストレージと転送）ですが、あるリソースタイプのCPUコストは、他の点よりコストが、高い場合があります。

ウェブページのリソースを管理し、要求されたときに利用するというプロセスは、過去数十年の間に急速に変化してきました。この変化の一部は、ウェブページのリソースをより効率的に、要求されたときに素早く送信できるようすることが前提となっています。ここでは、リソースのページウエイトがもたらす3つの影響について見ていきましょう。

ストレージ

ページのリソースは、要求された時すぐに取り出せるよう保存しておく必要があります。画像、動画、CSS、JavaScript、フォントなどのファイル資産は、サーバー、ローカルデバイス、メモリーなど、複数の場所に保存されます。各ファイルのサイズは数バイトから数メガバイトに及ぶため、複数の場所でコストに影響を与えることになります。サーバーのストレージコストは比較的安いように見えますが、デバイスのストレージは限られているため、キャッシュやメモリーから資産が削除され、ダウンロード数が増え、コストがかかることになります。

多くの人は、最適化されていないこれらのタイプの資産がページの読み込みパフォーマンスに与える悪影響を理解しておらず、またほとんど注意を払っていません。今日のウェブサイトをレビューしていると、4メガバイトを超える画像や、その何倍もの大きさの動画ファイルが埋め込まれているのを日常的に見かけます。

幸い、圧縮、メディア用ファイルフォーマット、CDNへのコンテンツのオフロードなど、保存するファイルのサイズを大幅に削減できるオプションや最適化もあり、多くの場合ほとんどコストなしでウェブページを軽量化することが可能です。

トランスミッション

ユーザーがHTTPでWebページを要求すると、そのページが必要とするすべてのファイルが要求されます。ファイルの場所を特定し、要求元のデバイスに送り返し、すべてがうまくいけば要求元のブラウザがペイロードを受け取り、要求元のユーザーの画面上に大きなウェブページの一部として処理しレンダリングします。ファイルのサイズによってリソースの転送が完了するまでの時間が決まり、最終的に結果のレンダリングに影響を与えるため、送信プロセスではページの重さが重要になります。

ページ重量が大きいことの悪影響は、*レイテンシー*と*帯域幅*の制約によります。レイテンシ

ーはリクエストがファイルを保存しているサーバーに接続し、それらのファイルの転送処理を開始するまでの時間を測定し、帯域幅は、リソースをダウンロードするのにかかる時間を測定します。大量のファイルが要求された場合、どのような技術であっても、一定時間内に処理・転送できる量には限界があります。私は、170以上のファイルを要求するWordPressサイトを監査したことがありますが、これは、高い待ち時間から始まるひどいページ読み込みパフォーマンスを保証するものです。

たとえば特定のファイル要求を圧縮してまとめる、HTTP/2や新しいHTTP/3プロトコルを使用する、最新のブラウザの機能である特定のファイルへの事前接続と事前ロードを使用してプロセス全体を高速化するなど多くの最適化によって転送/読み込み時間を改善できますが、最終的にはページの重量が影響を及ぼします。パフォーマンスの章では、ページの読み込み性能に影響を与えるさまざまな要素を取り上げています。

レンダリング

ウェブブラウザは、最終的にユーザーに代わってリソースへのリクエストを行うソフトウェアです（したがって、*ユーザーエージェント*という用語があります）。リクエストの結果は、ブラウザのレンダリングエンジンに渡され、ユーザーがリクエストしたウェブページを処理し、再作成します。ページの総量が多いほど、ブラウザエンジンが処理しなければならない量が多くなり、ブラウザの画面に表示するまでに時間がかかることは、容易に推測できます。

多くのファイル。とくに大きなメディアや複雑なスクリプトを取得、読み込み、処理し最終的にブラウザでレンダリングしてからコンテンツを利用できるようにするとページの読み込みに時間がかかりすぎて、ユーザーがページを放棄する可能性が高くなるのです。

また大きなペイロードは、ユーザーのスマートフォンやコンピューターで利用可能なクライアント側のリソースを圧迫し、デバイスをストールさせたり、クラッシュさせることさえあります。高速のケーブル・インターネット・サービスやハイエンド・デバイス向けの5Gデータ・プランに加入している幸運なユーザーは、こうした問題をめったに体験することはありません。しかし、多くのインターネットユーザーは、そのようなレベルのインターネットサービスやデバイスを利用することができません。

資産

昨年の章[885]で説明したように、Webページで使用されるアセットの種類は昔からあまり変わっていませんが、注目すべき例外があります。

885. *https://almanac.httparchive.org/ja/2020/page-weight#資産*

画像

静的ファイルは、それ自体で存在し、ウェブページを構築してレンダリングするためのリソースとして使用されます。画像、ビデオ、オーディオ、フォントファイルは、すべて静的アセットの例です。画像は、平均的なWebページの重量の大部分を占めるので、ここでは画像を例にとって説明します。

PNGやJPEGなどの画像フォーマットは、すべてのブラウザーで広くサポートされています。最近の画像フォーマットでは、WebPやAVIFが、より小さなファイルサイズでより高い品質を提供し、人気を博しています。WebPはほとんどのブラウザでサポートされていますが、AVIFは新しく、あまりサポートされていません。`<picture>` タグを使えば、JPEGやPNGのフォールバックを提供しながら、最新の画像フォーマットを使用できます。メディアの章では、この点についてより詳しく説明しています。画像のサイズや圧縮を適切に行わないと、パフォーマンスが低下します。

備考: 最適化し、さまざまな画像サイズ形式を比較できるオンライン サービスが必要な場合、Google の Squoosh[886] アプリケーションより優れたソースはないと私は考えています。同様に、Jake Archibald[887]'s SVGOMG[888] は SVG の最適化にはもってこいです。

JavaScriptの普及について一言

JavaScriptは動的なWebサイトを作成するために使用する素晴らしいツールですが、無防備に使用すると深刻なパフォーマンスの問題が発生し、ユーザーにとって恐ろしい体験になることがあります。過去数十年の間に、複雑なJavaScriptのWebフレームワークやライブラリの使用が急増し、膨大な量のJavaScriptがページ全体の重量に占める割合が大きくなっています。JavaScriptの中には、サイトのサイズを急増させ、深刻なパフォーマンスのボトルネックになるものもあります。中には、サイトが不安定になったり、使えなくなったりするほどひどいものもあります。スクリプトがブロックされると、ユーザーが操作するのに十分なページ資産のレンダリングが完了する前に、送信、処理、実行される必要があります。これは、ユーザーに混乱とフラストレーションを与え、サイトを放棄させる原因となります。

サイトが停止する場合、十中八九、JavaScriptのブロックが原因で、スマートフォンの処理リソースが不足するか、メモリが原因となっています。JavaScriptを賢明かつ専門的に使用することで、素晴らしいユーザー体験を生み出すことができます。しかし、このことを忘れないでください。JavaScriptはクライアント側で実行されます。JavaScriptはクライアントサイドで実行され、クライアントコンピューターのリソースを使ってスクリプトを処理・実行します。もう一度言いますが、誰もが最新のGoogle PixelやAppleのスマートフォンへ釘付けになっているわけではないのです。JavaScriptの章には、この問題についての豊富な情報が含まれています。

886. https://squoosh.app/
887. https://x.com/jaffathecake
888. https://jakearchibald.github.io/svgomg/

サードパーティサービス

ページの重さは、ウェブページから呼び出される外部サービスによっても影響を受けます。これらのサービスには、CDN、分析、チャットボット、フォーム、その他のデータ収集・処理方法などがあります。これは、ページの重さを肥大化させる原因として、もっとも急速に増加している問題領域の1つだと私は考えています。これらのサードパーティーのサービスの多くは、時代遅れでお粗末なJavaScriptやクエリー技術を使用しており、実行には必要以上に時間がかかる上、サイトオーナーは、サードパーティーがページの読み込みに与える影響をほとんど制御できません。あるサービスがページの読み込み性能にどのような影響を与えるかについて問い合わせることは、非常に重要であることは言うまでもありません。また、その影響をテストすることも重要です。

キャッシング

キャッシュは、リソースを迅速に提供し、再ダウンロードのコストを回避することを可能にするものです。キャッシュは、ユーザーのブラウザだけでなく、サーバーにも存在します。最適化された資産をキャッシュすることで、ページ重量とページ読み込み時間が劇的に減少します。これは、資産がすぐに利用可能になり、リクエストプロセス全体を実行する必要がなくなるためです。ページ全体の重量を減らすことはできませんが、その影響を軽減することは可能です。

数字で見るページウエイト

デスクトップとモバイルの両方でページの重さを見ると、これらのデバイスの機能が、異なることが多いにもかかわらず、両者の差は概して小さいです。

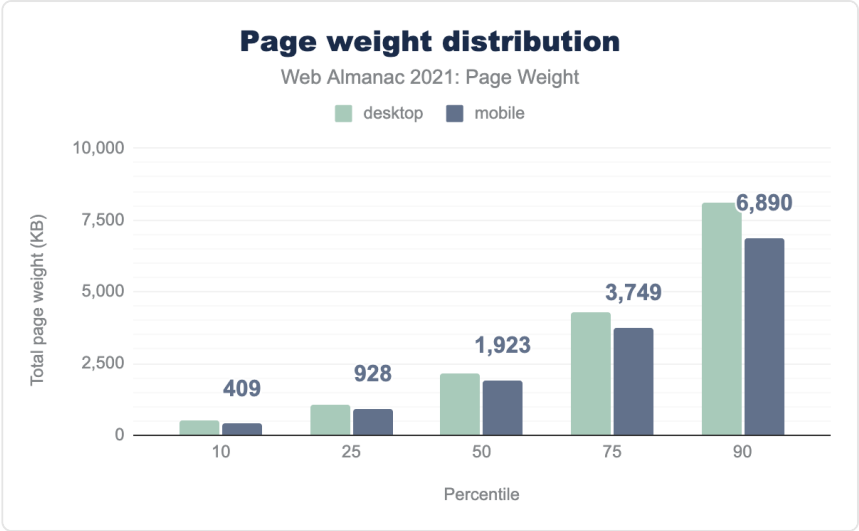


図19.1. 1ページあたりの総バイト数の分布。

モバイルでは6.9MB、デスクトップでは8.1MBと90%台のページ重量に迫っています。

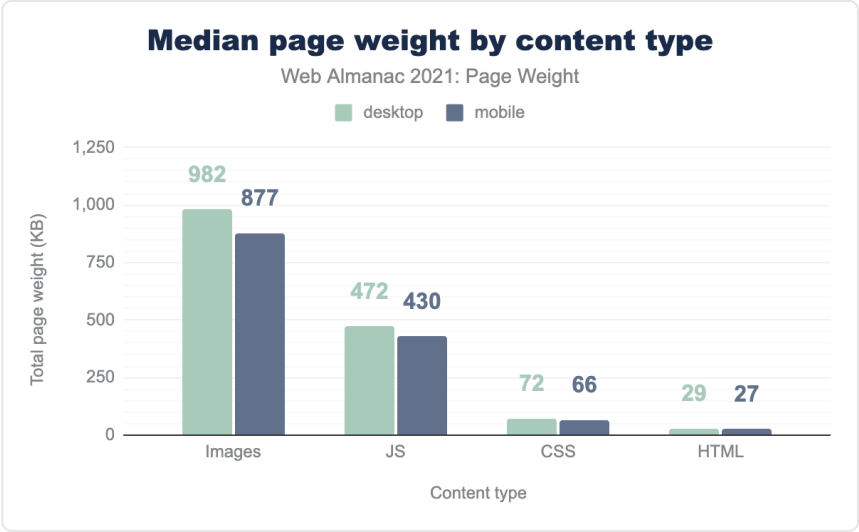


図19.2. コンテンツタイプ別の1ページあたりのバイト数の中央値。

中央値で見ると、画像がもっとも大きなリソースであることに変わりはなく、次いでJavaScriptとなっています。

経年変化をみてみよう。

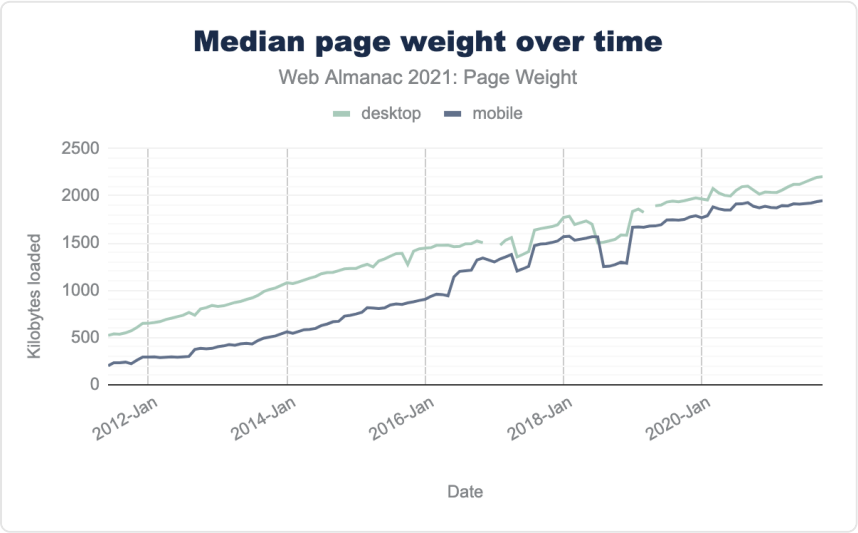


図19.3. ページの重みの時間的な中央値。

ページ重量の増加傾向は、これ以上ないほど明確です。私たちは上昇気流に乗り、その勢いが衰える気配はありません。

要望事項

この章ですでに説明したように、リソースのサイズと同様に、リクエスト数もページの読み込み性能に悪影響を与える可能性があるため、ページの重みのもう1つの指標となります。

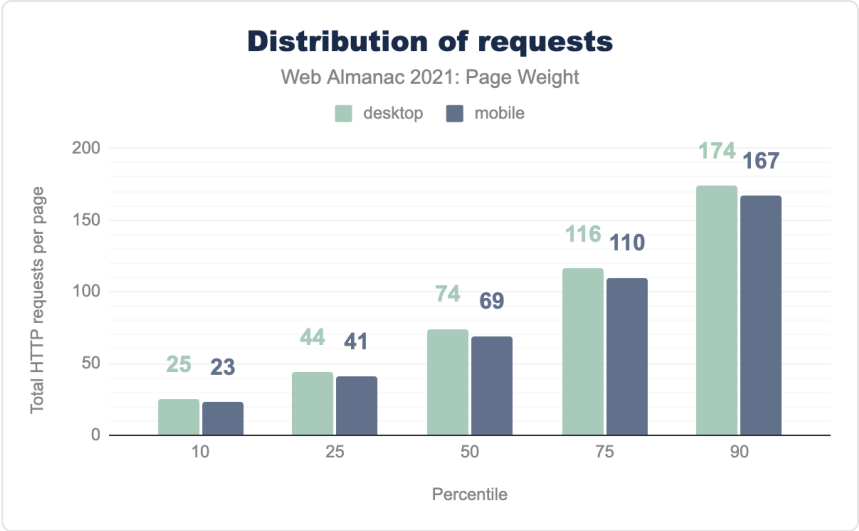


図19.4. ページごとのリクエストの分布。

リクエストの分布を見ると、デスクトップとモバイルの差は大きくなく、デスクトップがリードしていることがわかります。

今年と昨年の現在の結果の差は、実際には、ほとんどのパーセンタイルにおいて、GETリクエストの平均数がわずかに減少していることを示しています。このまま減少傾向が続くことを期待しましょう。

デスクトップでのリクエストの中央値は昨年[889] (74)と同じですが、ページの重さは増加しています（141kb）。

889. *https://almanac.httparchive.org/ja/2020/page-weight#リクエスト*

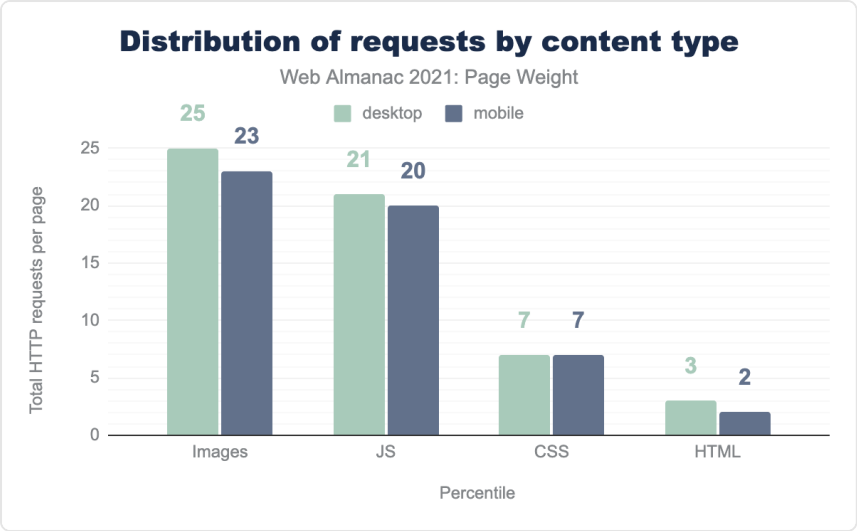


図19.5. コンテンツタイプ別のリクエスト数の中央値。

画像は、昨年に引き続きもっとも多いリクエスト数ですが、JavaScriptとの差はわずかながら縮まっています。画像は、2年間で4件減少しています。おそらく、シンプルなHTML属性でネイティブに利用できるようなったため、レイジーローディング[890]が増えた結果ではないでしょうか？

890. *https://developer.mozilla.org/docs/Web/Performance/Lazy_loading*

ファイルフォーマット

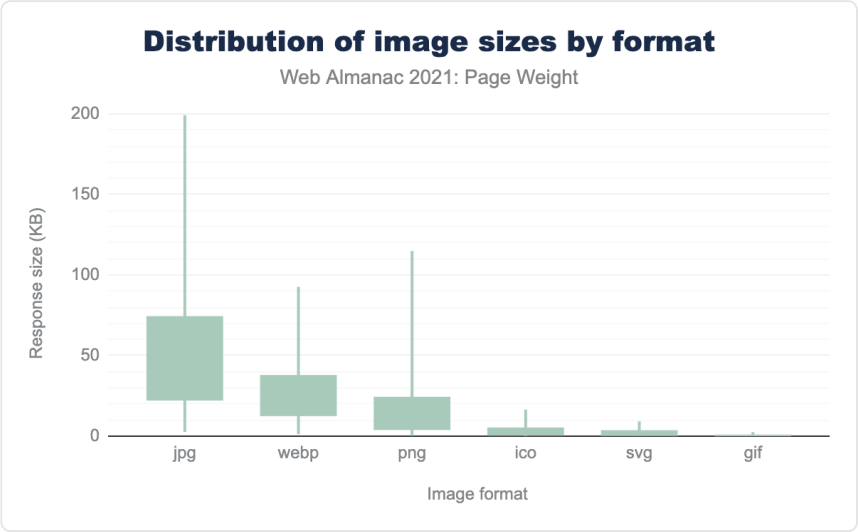


図19.6. フォーマット別の画像サイズ分布。

ウェブページの重量のうち、画像が大きな割合を占めていることが分かります。上の図は、画像の重量の上位のソースと重量の配分を示しています。上位3つ。JPG、WebP、PNGです。昨年と比較すると、すべての主要なブラウザでようやくサポートされるようになったWebPの利用が増加していることがわかります。PNGは、アイコンやロゴなどのユースケースで依然として人気があります。

画像バイト

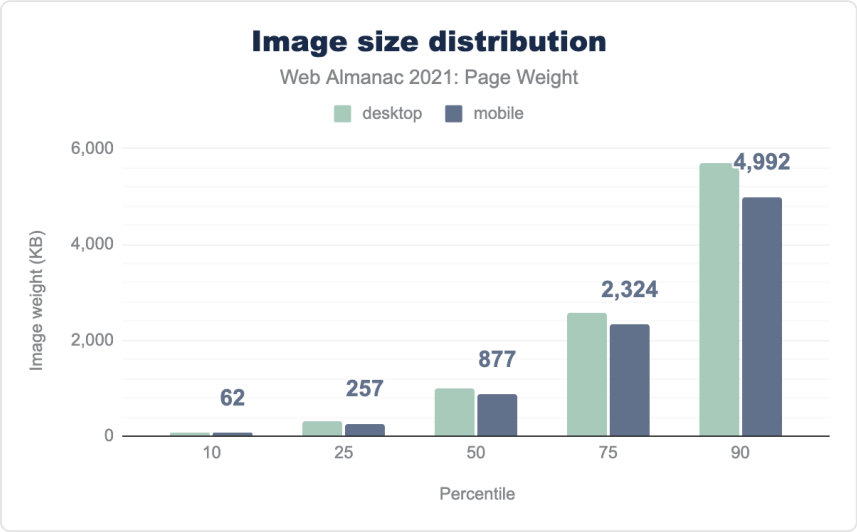


図19.7. 1ページあたりの画像応答サイズの分布。

画像の総バイト数を見ると、この指標は前年とほとんど変わらない[891]ことがわかります。この理由のひとつは、コンテンツ配信ネットワーク（CDN）が提供する画像数の増加です。CDNは、サーバーへアップロードされた画像に強力な最適化を適用するため、新しい画像の増加を抑制できます。

結論

ウェブページを軽量化することの重要性は？ページ全体の重さはページの読み込み速度に影響し、ページの読み込み速度はユーザー体験に影響します。GoogleのWeb Vitalsプログラムは、ユーザー体験、とくにモバイルユーザーの体験を重視しており、Google検索ランキングに直接影響を与えます。つまり、ウェブページをできるだけ軽くすることは、現実的なインセンティブであり、現実的な結果でもあるのです。

しかし、検索ランキングへの影響は、ページロードを軽くするような直接的な圧力につながるのでしょうか？AmazonのようなWebの巨人はどうでしょうか？大人気のウェブサイトがページの重さを気にする動機はあるのでしょうか？おそらく。アマゾンはページの資産やサービスのサイズを小さくすることで、それらのページを提供するために必要な費用を削減したいと考えるかもしれませんし、ユーザーが超高速スマートフォンを購入したり、5Gデー

891. *https://almanac.httparchive.org/ja/2020/page-weight#ファイル形式*

タネットワークや高速ケーブルプロバイダーを利用できないような新興市場に進出したいと考えるかもしれません。時間が解決してくれるでしょう。

著者



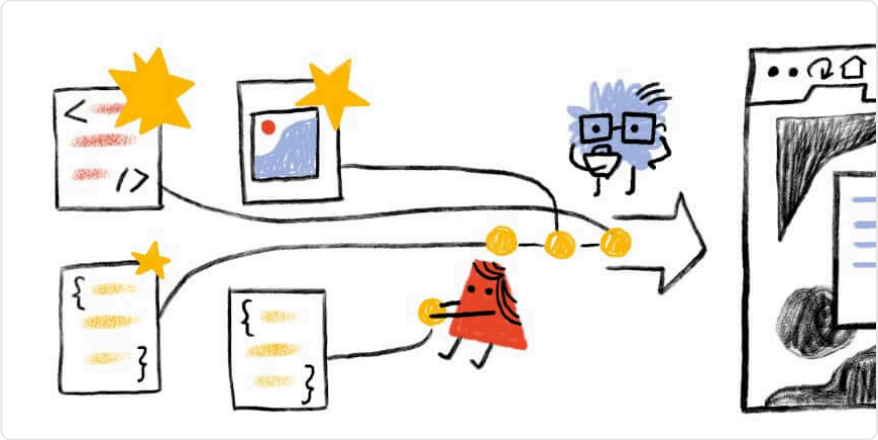
John Teague

@jtteag logicalphase https://gemservers.com

現在、JohnはGoogleクラウドプラットフォーム[892]のシニア開発者兼アーキテクトとして働いています。ウェブ開発者として、ウェブパフォーマンスとブラウザ標準の活用に焦点を当てた技術的な旅を開始しました。彼は、フリーランスのWordPress[893]開発者として、またいくつかのマネージドホスティングプロバイダーでアーキテクトやエンジニアとして、これらの原則を適用しました。オープンなウェブスタンダードと持続可能なウェブのベストプラクティスを強く信奉している。そのため、John は Google の Lit[894] プロジェクトを含むいくつかのオープン ソース プロジェクトに参加しており、ウェブコンポーネント[895] やその他のパフォーマンスに基づくソリューションなど、新しい Web 技術の強力な擁護者でもあります。

892. *https://cloud.google.com*
893. *https://wordpress.org*
894. *https://lit.dev/*
895. *https://developer.mozilla.org/docs/Web/Web_Components*

部 IV 章 20
リソースヒント



***Kevin Farrugia** によって書かれた。*
Sia Karamalegos、 *Barry Pollard*、 *Andy Davies*、 *Samar Panda* と *Weston Ruter* によってレビュ。
Nitin Pasumarthy による分析。
Rick Viscomi 編集。
Sakae Kotaro によって翻訳された。

序章

リソースヒントは、ウェブサイトのパフォーマンスを向上させるために使用できる、ブラウザへの指示です。この指示によりブラウザが、取得および処理する必要があるオリジンまたはリソースの優先順位を決定するのを支援できます。

ここでは、リソースヒントの実装方法、よくある落とし穴、そしてリソースヒントをできるだけ効果的に使うためにできることを詳しく見ていきましょう。

Linkディレクティブ

もっとも広く採用されているリソースヒントは、Linkディレクティブの `rel` 属性で実装されています。それらは `dns-prefetch`、 `preconnect`、 `prefetch`、 `prerender` と `preload` です。

これらは、次の2つの方法で実装できます。

HTML要素

```
<link rel="dns-prefetch" href="https://example.com">
```

HTTPヘッダー

```
Link: <https://example.com>; rel=dns-prefetch
```

また、JavaScriptを使用することで、HTML要素を動的に注入することも可能です。

```
const link = document.createElement("link");
link.rel="prefetch";
link.href="https://example.com";
document.head.appendChild(link);
```

HTTPヘッダーの採用率は、ドキュメントマークアップの一部としてリソースヒントを実装するよりも著しく低く、分析したページの1.5%未満しかHTTPヘッダーでリソースヒントを実装していません。これは、サーバ側でHTTPヘッダーを追加するのに比べ、HTMLソースから簡単に追加・変更できることに起因すると思われます。

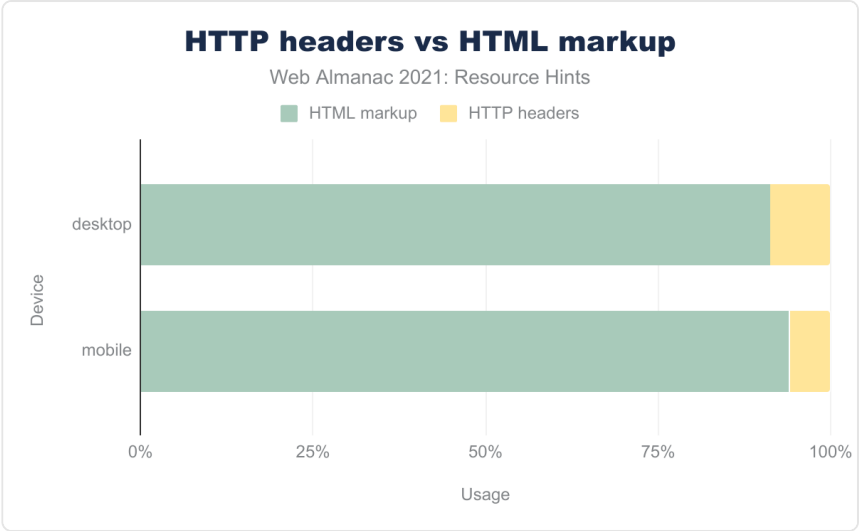


図20.1. HTTPヘッダーやHTMLマークアップとしてのリソースヒントの普及。

私たちの現在の 方法論 では、QuickLink[896] を通じて追加されるような、ユーザー インタラクションに続いて追加されるリソース ヒントを確実に測定することは不可能ですが、この特定のライブラリはコアウェブ・バイタル・テクノロジー・レポート[897]によると分析したページの0.1%以下でしか表示されませんでした。

HTTPヘッダーを用いたリソースヒントの採用は、`<link>` HTML要素に対する採用よりも著しく少ないことを考慮し、本章の残りの部分では、HTML要素を用いたリソースヒントの使用状況の分析に焦点を当てることにする。

リソースヒントの種類

現在、ほとんどのブラウザでサポートされているリソースヒントのリンク関係は5つあります。`dns-prefetch`, `preconnect`, `prefetch`, `prerender`, `preload` です。

`dns-prefetch`

```
<link rel="dns-prefetch" href="https://example.com/">
```

896. https://github.com/GoogleChromeLabs/quicklink
897. https://datastudio.google.com/s/uMbv5CQfW4Q

`dns-prefetch` ヒントは、ドメイン名を解決するための初期リクエストを開始します。これはクロスオリジンドメインのDNS検索にのみ有効で、 `preconnect` とペアで使用することができる。Chromeは現在、最大で64[898]の同時実行中のDNS要求をサポートしていますが、昨年までの6から増加し、他のブラウザではまだ厳しい制限を受けています。たとえば、Firefoxでは 8[899] に制限されています。

`preconnect`

```
<link rel="preconnect" href="https://example.com/">
```

`preconnect` ヒントは `dns-prefetch` と同様の動作をしますが、DNSルックアップに加えて、HTTPSで提供される場合はTLSハンドシェイクとともに接続を確立します。`dns-prefetch` の代わりに `preconnect` を使用すると、より高いパフォーマンスを得ることができます。しかし、証明書は通常3KB以上あり、他のリソースの帯域と競合してしまうため、控えめに使用する必要があります。また、重要なリソースに必要のないコネクションを開いてCPU時間を浪費することも避けたいものです。もしコネクションが短時間（たとえばChromeでは10秒）使用されない場合、ブラウザによって自動的に閉じられ、 `preconnect` の努力がムダになることを覚えておいてください。

`prefetch`

```
<link rel="prefetch" href="/library.js" as="script">
```

`prefetch` ヒントを使うと、次のナビゲーションでリソースが必要になるかもしれないことをブラウザに勧めることができます。ブラウザはリソースの優先順位の低いリクエストを開始し、必要なときにキャッシュから取得されるため、ユーザー体験を向上させることができるかもしれません。リソースは `prefetch` で事前に取得できますが、ユーザーがリソースを必要とするページに移動するまで、前処理や実行は行われません。

`prerender`

```
<link rel="prerender" href="https://example.com/page-2/">
```

898. https://source.chromium.org/chromium/chromium/src/+/fdf9418d23d434e0f7134da67dc41b0fe8268e91:net/dns/host_resolver_manager.cc;l=416
899. https://github.com/mozilla/gecko-dev/blob/master/netwerk/dns/nsHostResolver.h#L48

`prerender` ヒントを使用すると、ページをバックグラウンドでレンダリングし、ユーザがそのページに移動した場合のロード時間を改善できます。リソースを要求するだけでなく、ブラウザは前処理を行い、サブリソースを取得・実行できます。`prerender` は、ユーザーがプリレンダリングされたページにナビゲートしない場合、ムダに終わる可能性があります。仕様に反して、Chromeは `prerender` ヒントを NoStateプリフェッチ[900] として扱い、このリスクを軽減しています。完全なプリレンダーとは異なり、JavaScriptの実行やページの一部のレンダリングは事前に行わず、リソースを事前に取得するのみです。

`preload`

ほとんどのモダンブラウザは `preload` ヒントもサポート[901]していますし、程度の差[902]はありますが `modulepreload` というヒントもサポートしています。`preload` 命令は、ページの読み込みに必要なリソースの初期フェッチを開始し、フォントファイルやスタイルシートで参照される画像など、発見が遅れたリソースにもっともよく使用されます。リソースのプリロードは、リソースの優先順位を上げるために使われることがあり、開発者はHTMLの解析中に発見されたとしても、最大のコンテントフルペイント[903] (LCP) 画像の読み込みを優先させることができます。

`modulepreload` は `preload` に特化した代替手段で、動作は似ていますが、使用できるのは モジュールスクリプト[904] に限定されています。

900. *https://developers.google.com/web/updates/2018/07/nostate-prefetch*
901. *https://caniuse.com/link-rel-preload*
902. *https://caniuse.com/link-rel-modulepreload*
903. *https://web.dev/articles/lcp*
904. *https://html.spec.whatwg.org/multipage/webappapis.html#module-script*

採用実績と傾向

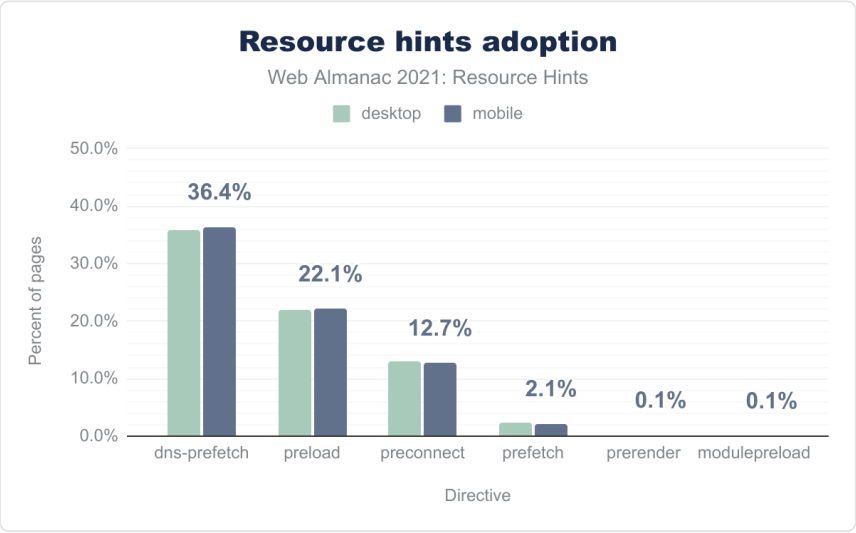


図20.2. リンクのrel属性の採用。

もっとも多く利用されているリソースヒントは `dns-prefetch` (モバイルでは36.4%)です。というのは、2009年[905]に導入されたことを考えると、当然といえば当然なのですが。HTTPSの普及に伴い、そのドメインに接続することが確実な場合は、多くの場合、 `preconnect` に置き換える必要があります（モバイルでは12.7%）。 `preload` ヒントは2016年[906]にChromeではじめて登場した比較的新しいものですが、2番目に広く採用されているリソースヒント（モバイルでは22.1%）で、前年比で一定の成長を見せていることが、この指示の重要性と柔軟性を裏付けています。

上のグラフにあるように、モバイルとデスクトップでの普及率はほぼ同じです。

905. *https://caniuse.com/link-rel-dns-prefetch*
906. *https://groups.google.com/a/chromium.org/g/blink-dev/c/_nu6HlbNQfo/m/XzaLNb1bBgAJ?pli=1*

ランク別

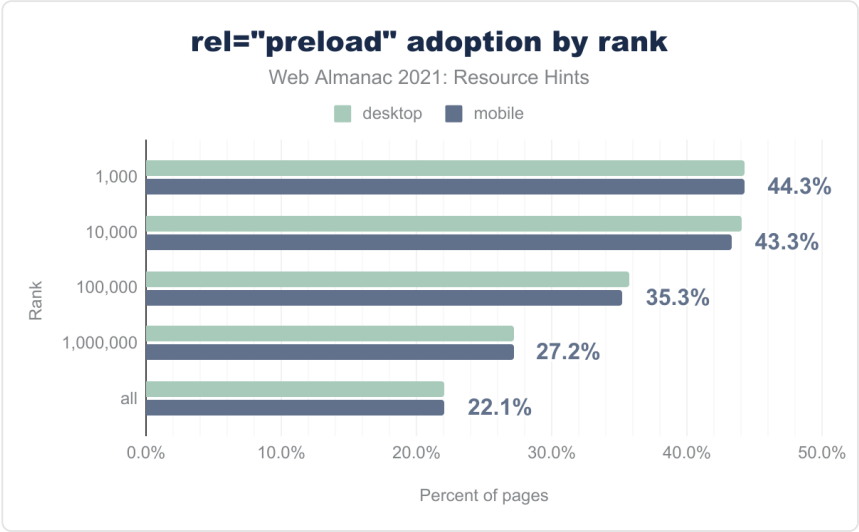


図20.3. `rel="preload"` のCrUXランク別採用状況。

ランクでデータを分割すると、採用率が顕著に変化し、`preload` ヒントはデータセット全体の22.1%から、上位1000サイトでは44.3%の採用率でトップに立っていることが観察されます。

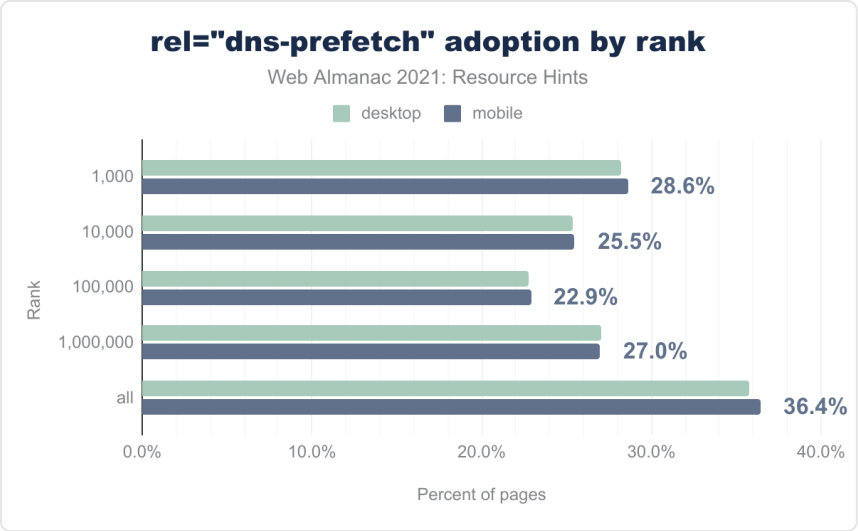


図20.4. `rel="dns-prefetch"` の採用状況をCrUXランクで区分したもの。

上位1,000サイトと全体の導入状況を比較すると、`dns-prefetch` が唯一の導入減少を示すリソースヒントであることがわかります。

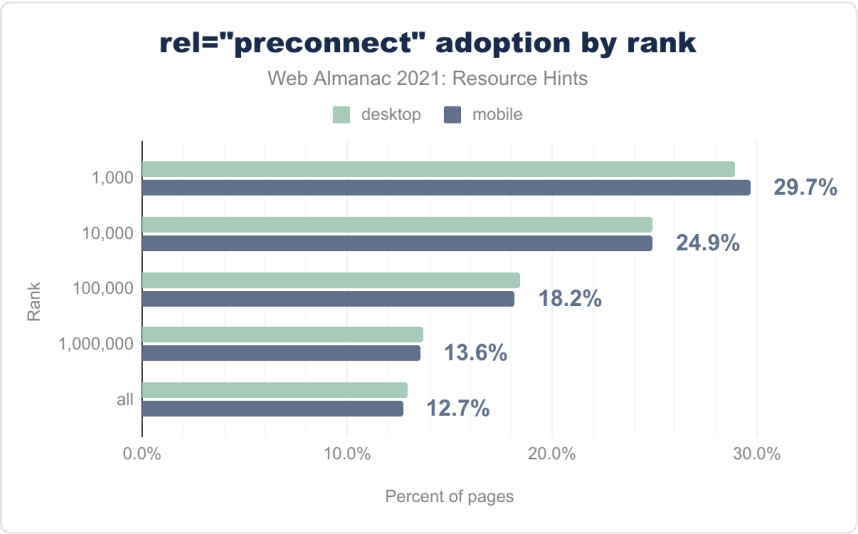


図20.5. `rel="preconnect"` のCrUXランク別採用状況。

この減少に対抗するため、上位1,000ページでは `preconnect` ヒントの採用が増加し、その

性能向上と幅広いサポートを活用しています。今後、他のインターネットサイトがこれに追随することで、`preconnect`の採用が増加し続けるものと思われます。

使用方法

リソースヒントは、正しく使用すれば非常に効果的です。ブラウザから開発者に責任を移すことで、重要なレンダリングパスに必要なリソースを優先し、ロード時間やユーザー体験を向上させることができます。

ランク	preload	prefetch	preconnect	prerender	dns-prefetch	modulepreload
1,000	3	2	4	0	4	1
10,000	3	1	4	1	3	1
100,000	2	2	3	1	3	1
1,000,000	2	2	2	1	2	1
all	2	2	1	1	2	1

図20.6. 1ページあたりのリソースヒントの数のランク別中央値。

リソースヒントを使用しているサイトのうち、上位1,000サイトの中央値をコーパス全体と比較すると、上位のサイトほどページあたりのリソースヒントの数が、多いことがわかる。上位1,000サイトで合計0回しか出現していない`prerender`は、異なるパターンを観察する唯一のヒントです。

コアウェブバイタルとの相関

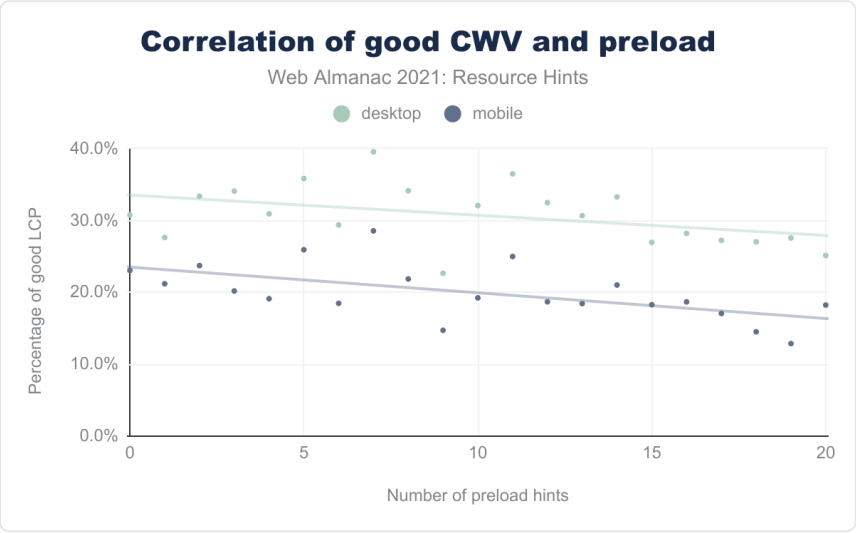


図20.7. 良好なCWVスコアと `rel="preload"` ヒントの数との相関関係

CrUXデータセットにおけるページの コアウェブ・バイタル[907] のスコアとプリロード リソースヒントの使用を組み合わせることにより、リンク要素の数とCWVで良い評価を得たページの割合の間に負の相関関係を観察することができました。プリロードヒントが少ないページは、良い評価を受けやすいと言えます。

907. https://web.dev/i18n/ja/vitals/

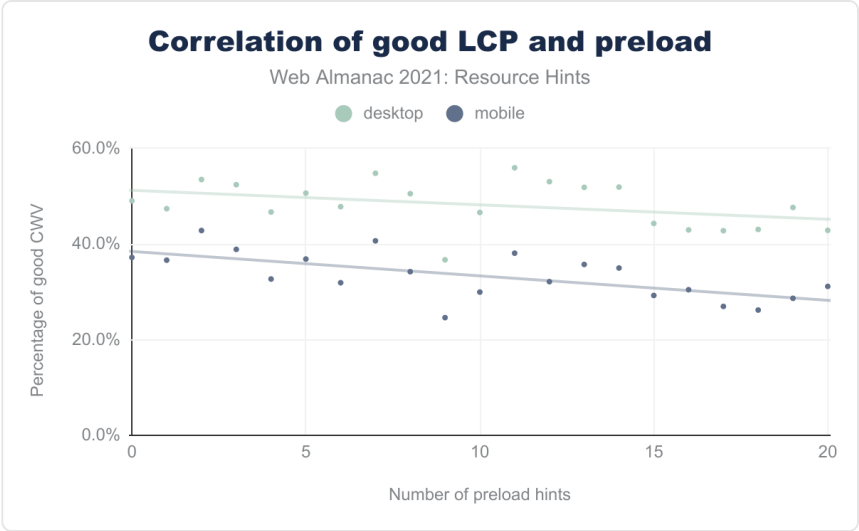


図20.8. 良いLCPスコアと `rel="preload"` ヒントの数との相関関係

この現象は、ページのLCPにも見られ、多くの場合、開発者はLCP要素のレンダリングに必要のないリソースを優先させ、その結果、ユーザー体験を低下させていることがわかります。

これは、プリロードヒントがあるとページが遅くなることを証明するものではありませんが、多くのヒントを持つことは、パフォーマンスが低下することと相関しています。しかし、多くの場合、プリロードされるリソースの数は少なくし、リソースの優先順位付けは可能な限りブラウザに委ねるべきです。

注: ヒントの数だけでなく、プリロードされる各リソースのサイズもウェブサイトのパフォーマンスに影響を及ぼします。上図では、プリロードされた各リソースのサイズは考慮されていません。

`rel="preload"`

とはいえ、より多くのウェブサイトが `preload` を採用することが予想されるため、preloadリソースヒントをよりよく見て、なぜそれが効果的でありながら同時に誤用されやすいのかを理解しましょう。

`as` 属性

`as` 属性は、 `rel="preload"` (または `rel="prefetch"`) を使用して、ダウンロードするリソースの種類を指定するときに指定する必要があります。正しい `as` 属性を適用することで、ブラウザはより正確にリソースの優先順位を決定できます。たとえば、 `preload as="script"` は低または中の優先順位になり、 `preload as="style"` は内部リクエストの優先順位が *最高* になります。 `as` 属性は、今後のリクエストに備えてリソースをキャッシュし、正しい コンテンツセキュリティポリシー[908] を適用するために必要です。

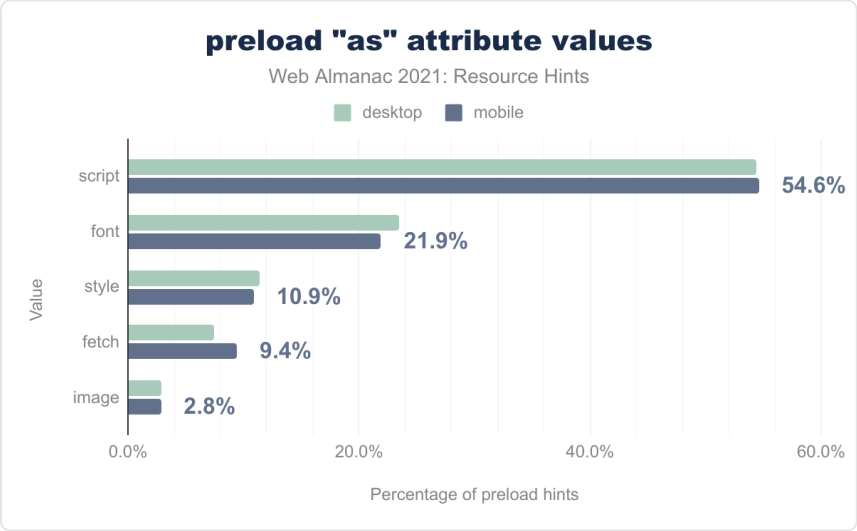


図20.9. `rel="preload" as` の属性値です。

`script`

`script` はかなりの差をつけてもっとも一般的な値です。 `<script>` 要素は最初のHTML文書に埋め込まれるため、通常は早期に発見されますが、 `<body>` タグを閉じる前に `<script>` 要素を配置するのが一般的なやり方となっています。HTMLは順次解析されるため、DOMがダウンロードされ解析された後にスクリプトが発見されることになり、JavaScriptフレームワークに依存するWebサイトが増えたことで、JavaScriptを早期に読み込む必要性が高まっています。しかしJavaScriptのリソースは、画像やスタイルシートなど、HTML文書内で発見された他のリソースよりも優先されるため、ユーザー体験が、損なわれる可能性があります。

908. *https://developer.mozilla.org/docs/Web/HTTP/CSP*

`font`

2番目によくプリロードされるリソースは `font` です。ブラウザはレイアウトフェーズの後に、そのフォントがページ上に表示されることが分かってはじめてフォントファイルをダウンロードするので、これは発見が遅れたリソースと言えます。

`style`

スタイルシートは通常ドキュメントの `<head>` に埋め込まれ、ドキュメントのパース中、早期に発見されます。さらに、スタイルシートはレンダーブロッキングリソースであるため、リクエスト優先度が *最高* に割り当てられています。これにより、スタイルシートのプリロードは不要になるはずですが、リクエストの再優先が必要な場合もあります。Google Chromeのバグ[909]（Chrome95で修正）は、プリロードスキャナーが検出した、CSSファイルなどの優先度の高いリソースよりも、プリロードしたリソースを優先して使用します。スタイルシートをプリロードすると、その優先順位は *最高* に戻されます。スタイルシートがプリロードされるもう1つの例は、HTMLドキュメントから直接ダウンロードされない場合です。たとえば、非同期CSS[910]ハックは `onload` イベントを使って、重要ではないCSSによるページのレンダーブロックを回避するものです。

`fetch`

プリロードは、JSONレスポンスやストリームなど、ページのレンダリングに重要なデータを取得するためのリクエストを開始するために使用できます。

`image`

CSSの `background-image` のように、画像が最初のHTMLに含まれていない場合、画像をプリロードすることでLCPスコアを向上させることが、できる場合があります。

`crossorigin` 属性

`crossorigin` 属性は、要求されたリソースを取得する際にオリジン間リソース共有[911] (CORS)を使用しなければならないかどうかを示すために使用されます。これはあらゆるリソースタイプに適用できますが、フォントファイルは常にCORSを使用して要求されるべきなので、もっとも一般的に関連付けられます。

909. *https://bugs.chromium.org/p/chromium/issues/detail?id=629420*
910. *https://www.filamentgroup.com/lab/async-css.html*
911. *https://developer.mozilla.org/docs/Web/HTTP/CORS*

値	デスクトップ	モバイル
未設定	66.6%	65.9%
クロスオリジン(または同等)	14.5%	13.5%
使用許可証	< 0.1%	< 0.1%

図20.10. `rel="preload" crossorigin` 属性の値。

`anonymous`

値が指定されていない場合のデフォルト値は `anonymous` で、この値では資格情報フラグが `same-origin` に設定されます。CORSで保護されたリソースをダウンロードする際に必要です。また、フォント ファイルをダウンロードする際の 必須条件[912]でもあります、たとえそれが同じオリジンであってもです！プリロードされたリソースに対する最終的なリクエストがCORSを使用しているときに `crossorigin` 属性を省略すると、プリロードキャッシュでマッチしないため、重複したリクエストになってしまいます。

`use-credentials`

たとえば、クッキー、クライアント証明書、 `Authorization` ヘッダーなどの使用により、認証を必要とするクロスオリジンリソースを要求する場合。 `crossorigin="use-credentials"` 属性を設定すると、このデータをリクエストに含め、サーバーがリクエストに応答して、リソースをプリロードできるようにします。これは0.1%の使用率で一般的なシナリオではありませんが、ページのコンテンツが認証状態に依存している場合、ログイン状態を取得するために早期フェッチリクエストを開始するために使用できます。

`media` 属性

`rel="preload"` の機能として、メディアクエリを `media` 属性で指定できますが、この属性を使っているプリロードは全体の4%未満です。 `media` 属性はメディアクエリを受け付け、メディアタイプやビューポート幅のような特定のブラウザ機能をターゲットにできます。たとえば、 `media` 属性を使えば、ビューポートが狭いデバイスには低解像度の画像を、ビューポートが大きいデバイスにはフルサイズの画像をあらかじめ読み込ませることができます。

`media` 属性に加えて、 `<link>` 要素は `imagesrcset` と `imagesizes` 属性をサポートし

912. https://drafts.csswg.org/css-fonts/#font-fetching-requirements

ています。これは、 `<img>` 要素の `srcset` と `sizes` 属性に対応するものです。これらの属性を使用すると、画像に使用するのと同じリソースの選択基準を使用できます。残念ながら、その採用率は非常に低く（1%未満）、Safariのサポート[913]がないためと思われます。

注: `media` 属性は、仕様にあるようにすべての `<link>` 要素で利用できるわけではなく、`rel="preload"` にのみ利用できます。

悪い習慣

`rel="preload"` の多用途性により、preloadヒントの実装方法を規定する明確なルールはありませんが、失敗から多くを学び、それを回避する方法を理解することは可能です。

未使用のプリロード

Webサイトのパフォーマンスとプリロードヒントの数には、負の相関があることをすでに見てきました。この関係には、2つの要因が影響している可能性があります。

- 不適切なプリロード
- 未使用のプリロード

不正なプリロードとは、ブラウザが優先するはずの他のリソースよりも重要でないリソースをプリロードした場合を指します。ヒントを表示した場合と表示しない場合のA/Bテストが必要なため、不正なプリロードの程度を測定することはできません。

未使用のプリロードは、ページを読み込んでから数秒以内に必要のないリソースをプリロードした場合に発生します。



図20.11. 最初の3秒間で未使用のプリロードヒントのパーセンテージ。

このような場合、すぐに必要でない、あるいはまったく必要でないファイルやリソースをダウンロードし、優先的に使用するようブラウザに指示しているため、プリロードヒントはウェブサイトのパフォーマンスを低下させることになります。リソースヒントは定期的なメンテナンスが必要であり、プロセスを自動化すると、このような問題が、発生する可能性があります。

913. *https://caniuse.com/mdn-html_elements_link_imagesizes*

不正確な `crossorigin` 属性

正しい `crossorigin` 属性を含めずにCORS対応のリソースをプリロードしようとすると、同じリソースを2回ダウンロードすることになります。最終的なリクエストでもCORSを使用する場合は、 `<link>` 要素に `crossorigin` 属性が必要です。これは、同じオリジンでフォントファイルをセルフホストしている場合でも、フォントファイルは常にCORSが有効であるとして扱われるため、フォントファイルをリクエストする場合にも当てはまります。

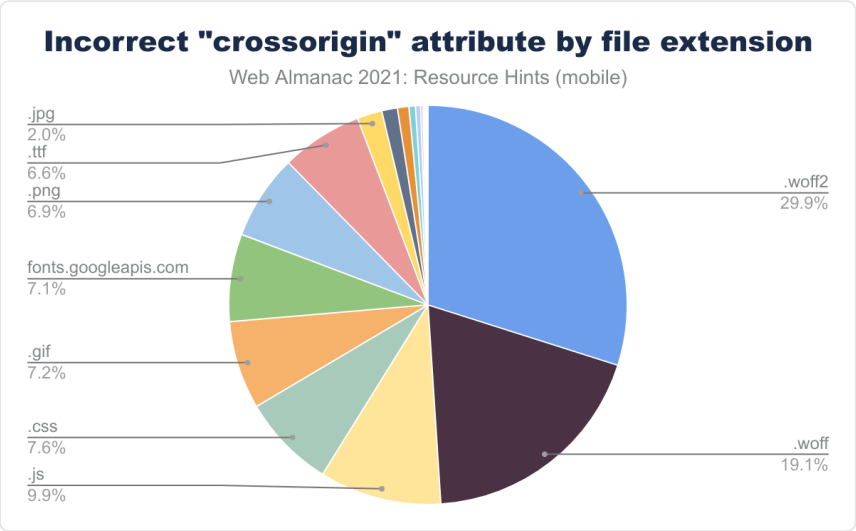


図20.12. モバイル端末において、ファイル拡張子ごとに区分したクロスオリジン値の不正確さの割合。

`rel="preload"` ヒントの `crossorigin` 属性が見つからないか不正確な場合の半数以上(63.6%)は、フォントファイルのプリロードに関連しており、データセット全体で合計14,818インスタンスになります。

無効な `as` 属性

`as` 属性はリソースをプリロードする際に重要な役割を果たします。これを間違えると、同じリソースを2回ダウンロードすることになるかもしれません。ほとんどのブラウザでは、認識できない `as` 属性を指定すると、プリロードが無視されます。サポートされている値は `audio`、 `document`、 `embed`、 `fetch`、 `font`、 `image`、 `object`、 `script`、 `style`、 `track`、 `worker` および `video` です。

認識できない値は17,861件あり、もっとも多い誤りは完全に省略することです。一方、値と

してもっとも多い無効なものは `other` と `stylesheet` です（正しい値は `style` です）。

図20.13. ページで `"style"` の代わりに `as="stylesheet"` が誤って使用されていた。

正しくない `as` 属性値（たとえば `script` の代わりに `style` を使用するような認識できない値）を使用すると、ブラウザはプリロードキャッシュに保存されているリソースと一致しないため、ファイルのダウンロードを重複して行うことになります。

注: `video` は仕様に含まれていますが、どのブラウザでもサポートされていないため、無効な値として扱われ、無視されるでしょう。

未使用のフォントファイル

フォントファイルをプリロードするページの5%以上が、必要以上のフォントファイルをプリロードしています。フォントファイルをプリロードする場合、`preload` をサポートしているすべてのブラウザは `.woff2` もサポートしています。つまり、`.woff2` のフォントファイルが利用可能であると仮定すると、`.woff` を含む古い形式のフォントをプリロードする必要はありません。

サードパーティ

リソースヒントを使用して、ファーストパーティとサードパーティの両方に接続したり、リソースをダウンロードしたりできます。`dns-prefetch` と `preconnect` はサブドメインを含む異なるオリジンへの接続時にのみ有効ですが、`preload` と `prefetch` は同じオリジンのリソースとサードパーティーがホストするリソースの両方に使用できます。

サードパーティーのリソースにどのリソースヒントを使用するかを検討する際には、アプリケーションのロード体験における各サードパーティーの優先順位と役割、およびコストが正当化されるかどうかを評価する必要があります。

自社のコンテンツよりも第三者のリソースを優先させることは、潜在的に警告のサインですが、これが推奨される場合もあります。たとえばEUでは 一般データ保護規則[914] によって要求されているCookie通知スクリプトを見ると、これらは通常 `dns-prefetch` または `preconnect` ヒントを伴っており、ユーザー体験に非常に邪魔である上、パーソナライズした広告の配信など、一部のサイト機能にとって必要不可欠なものだからです。

914. https://ja.wikipedia.org/wiki/EU%E4%B8%80%E8%88%AC%E3%83%87%E3%83%BC%E3%82%BF%E4%BF%9D%E8%AD%B7%E8%A6%8F%E5%89%87

ホスト	dns-prefetch	preconnect	preload	合計
adservice.google.com	0.2%	0.5%	35.7%	36.4%
fonts.gstatic.com	0.9%	24.0%	0.6%	25.5%
fonts.googleapis.com	14.0%	4.5%	2.7%	21.2%
s.w.org	19.7%	0.2%	-	19.9%
cdn.shopify.com	-	1.7%	9.6%	11.2%
siteassets.parastorage.com	-	-	5.9%	5.9%
www.google-analytics.com	1.2%	3.9%	0.2%	5.3%
www.googletagmanager.com	1.9%	2.7%	0.2%	4.8%
static.parastorage.com	-	-	4.7%	4.7%
ajax.googleapis.com	2.2%	1.6%	0.3%	4.1%
www.google.com	2.7%	1.0%	0.1%	3.8%
images.squarespace-cdn.com	-	3.5%	-	3.5%
cdnjs.cloudflare.com	1.6%	1.0%	0.4%	2.9%
monorail-edge.shopifysvc.com	2.0%	0.8%	-	2.8%
fonts.shopifycdn.com	-	1.1%	1.0%	2.1%

図20.14. モバイル端末でリソースヒントを使用しているもっとも普及しているサードパーティ接続。

上の表を分析すると、`preload` ヒントを含む全ページの36.7%がadservice.google.comでホストされているリソースをプリロードしています。s.w.orgホストは `dns-prefetch` でもっとも普及しているドメインで、WordPressサイト（バージョン4.6以降）で、ブラウザがネイティブの絵文字をサポートしていないと検出された場合にTwemoji CDNからのSVG画像を読み込むために使用されています。Google Fonts関連のサービスでは、`fonts.gstatic.com` と `fonts.googleapis.com` が `preconnect` ディレクティブを使用するホストとしてもっとも普及しているようです。



図20.15. Google Fontsの説明では、fonts.gstatic.comとfonts.googleapis.comへ事前接続するようになっています。（出典: Google Fonts[915]）

Google Fontsは現在、fonts.gstatic.com originとfonts.googleapis.comの両方に `preconnect` するように指示します。これは通常、発見が遅れたリソースの影響を相殺する良い習慣です。

サードパーティのあり方について詳しく知りたい方は、サードパーティの章をご覧ください。

ネイティブ遅延ローディング

遅延ローディングとは、リソース（この場合は画像やiframe）のダウンロードを、それが必要になるまで、あるいはビューポート内で見えるようになるまで延期する技術のことを指します。ネイティブ遅延ローディングは、これを処理するためにJavaScriptライブラリを使用するのではなく、HTMLで `loading="lazy"` 属性を使用して指定できることを指します。画像とiframeのネイティブ遅延ローディングは2019年に標準化され、それ以来、その採用は、とくに画像について飛躍的に伸びています。

画像の `loading="lazy"` は、ほとんどの主要なブラウザでサポートされています。Safariでは、進行中[916]と表示され、フラグの後ろに表示されて利用できますが、まだデフォルトでは有効ではありません。

915. https://fonts.google.com/
916. https://bugs.webkit.org/show_bug.cgi?id=200764

iframeの遅延ロードはChromeでサポートされ、Safariでは再びフラグの背後にありますが、Firefoxではまだサポートされていません[917]。

`loading` 属性をサポートしていないブラウザは、この属性を無視しますから、不要な副作用なしに安全に追加できます。JavaScriptベースの代替品、たとえばlazysizes[918]はまだ使えるかもしれませんが、ブラウザのフルサポートが間近であることを考えると、この段階でプロジェクトに追加する価値はないかもしれません。

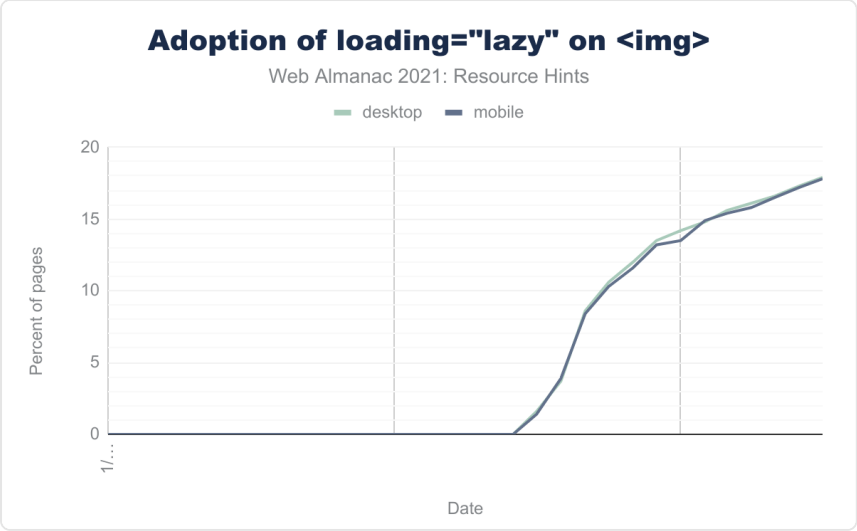


図20.16. `img` 要素に `loading="lazy"` 属性が設定されているページの割合です。

また、`loading="lazy"` を使用しているページの割合は、2020年の4.2%から、分析時点では17.8%に増加しています。なんと423%もの成長率です。この急成長は驚異的であり、その背景には2つの重要な要素があると思われます。クロスブラウザーに対応したページへの追加が容易であること、そしてこれらのウェブサイトを支えているフレームワークや技術です。WordPress 5.5ではlazy-loading imagesがデフォルトの実装となり[919]、`loading="lazy"` の採用率は急上昇し、WordPressサイトでは、ネイティブ画像の遅延ロードを使用するページ全体の 84%[920] が構成されるようになったそうです。

917. *https://bugzilla.mozilla.org/show_bug.cgi?id=1622090*
918. *https://github.com/aFarkas/lazysizes*
919. *https://make.wordpress.org/core/2020/07/14/lazy-loading-images-in-5-5/*
920. *https://web.dev/articles/lcp-lazy-loading*

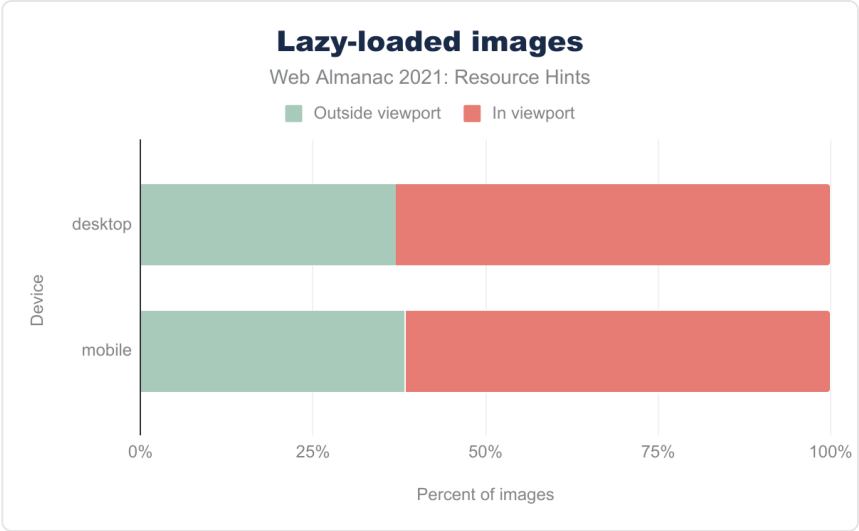


図20.17. 初期ビューポートにある `loading="lazy"` を持つ `img` 要素の割合です。

モバイルで遅延ロードされた画像の61.5%、デスクトップで遅延ロードされた画像の63.1%は、実際には初期ビューポート内にあり、遅延ロードされるべきではない画像です。遅延ロードを使用したページのロード時間に関する研究[921]によると、遅延ロードを使用したページはLCP性能が、悪化する傾向があり、おそらく遅延ロード属性を使いすぎたことが原因であると指摘されています。これは、遅延ロードされるべきではないLCP要素でますます顕著になります。もしあなたが `loading="lazy"` を使っているなら、遅延ロードされた画像が折り目の下にあるかどうか、さらに決定的なことは、LCP要素が遅延ロードされていないかどうかを確認する必要があります。LCPイメージのレイジーローディングがコアウェブ・バイタルに与える影響については、パフォーマンスの章で詳しく説明しています。



図20.18. `iframe` 要素に `loading="lazy"` 属性が設定されているページの割合です。

少なくとも1つのiframeを含むページの可能性は、画像を含むページよりもはるかに低く、iframeを含むページのわずか2.6%しかネイティブ遅延ローディングを利用していません。iframeは、スクリプトや画像を含むより多くのリソースをダウンロードするためにさらなるリクエストを開始する可能性があるため、iframeを遅延ロードすることの利点は潜在的に重要です。これは、YouTubeやTwitterの埋め込みなどの埋め込みを使用する場合にとくに当て

921. *https://web.dev/articles/lcp-lazy-loading*

はまります。同様に、画像の読み込み方法を決定する場合、iframeが初期ビューポート内に表示されているかどうかを確認する必要があります。もし表示されていない場合は、通常、`<iframe>` 要素に `loading="lazy"` を追加することで、初期読み込みを減らし、パフォーマンスを向上させることができるので安全です。

HTTP/2サーバープッシュ

HTTP/2は *サーバープッシュ* と呼ばれる技術をサポートしており、クライアントがリクエストすると予想されるリソースを先取りしてプッシュします。サーバーはクライアントにリソースを要求するよう通知する代わりにリソースをプッシュするので、キャッシュ管理が複雑になり、場合によっては、プッシュされたリソースがHTMLの配信を遅らせることさえあります。

残念ながら、HTTP/2プッシュは期待外れでした。ブラウザがすでに持っている、あるいはブラウザが要求するリソースよりも重要度の低いリソースを過剰にプッシュするリスクと比較して、約束したパフォーマンスの向上を提供するという証拠がほとんどないのです。

つまり技術は広く普及しているにもかかわらず、これらの障害を克服するために、普及率が非常に低く、採用率は1%未満にとどまっているのです。Chromeはまた、削除の意向[922]を提出しました。これは103の初期ヒント（次に説明）のテスト可能な実装が利用可能になるまで一時停止されます。ChromeはHTTP/3でのServer Pushもサポートしていません[923]。

HTTP、HTTP/2、HTTP/3については、HTTPの章に詳しく書かれています。

未来

新しい `rel` ディレクティブを追加する提案はありませんが、Chromeの優先順位付けバグ[924]など、現在のリソースヒントのセットに対するブラウザベンダーの改善は、良い影響を与えると期待されます。ヒントの採用が進み、 `preload` の用途が本来の目的である発見が遅れたリソースにシフトしていくことが予想されます。

さらに、初期ヒント103と優先順位付けのヒントという2つの提案も近日中に公開される予定で、すでにChromeでは実験的なサポートが提供されています。

922. *https://lists.w3.org/Archives/Public/ietf-http-wg/2019JulSep/0078.html*
923. *https://github.com/httpwg/http2-spec/issues/786#issuecomment-724371629*
924. *https://bugs.chromium.org/p/chromium/issues/detail?id=629420*

初期のヒント103

Chrome 95では、`preload`と`preconnect`に対して、初期ヒント103[925]を実験的にサポートしました。初期ヒントは、メインレスポンスが提供される前にブラウザがリソースをプリロードすることを可能にし、リクエストが送信されてからサーバーからのレスポンスが送信されるまでのブラウザ上のアイドル時間を利用します。初期ヒント103を使用する場合、サーバーは、実際のドキュメント応答を処理している間に、HTTPヘッダーメソッドを使用してプリロードされるリソースの詳細を示す「情報」レスポンスステータスを直ちに送信します。こうすることで、ブラウザはHTMLが届く前でも、重要なリソースに対してプリロードのリクエストを開始することができ、ドキュメントのマークアップに`<link>`要素を使用する場合よりもずっと早くプリロードのリクエストを開始することができるようになるのです。初期ヒント103は、HTTP/2サーバープッシュで発生する困難のほとんどを克服しています。

優先順位付けのヒント

優先度ヒントは、ページ内のリソースの相対的な重要度をブラウザに通知し、重要なリソースを優先してコアウェブバイタルを向上させることを目的としています。優先度ヒントは、`<img>`や`<script>`などのリソースに`importance`属性を追加することで、ドキュメントのマークアップを通じて有効になります。`importance`属性は`high`, `low`, `auto`の列挙を受け付け、これをリソースのタイプと組み合わせることで、ブラウザはヒューリスティックに基づいて最適なフェッチ優先度を割り当てることができるようになります。優先ヒントは、Chrome 96でオリジン・トライアル[926]として利用できます。

結論

この1年間でリソースヒントの採用は拡大し、開発者がこれらのAPIを活用してリソースの優先順位を決め、ユーザーの体験を向上させることから、今後も拡大すると予想されています。同時に、ブラウザベンダーは、これらのディレクティブを調整し、その役割と効果を進化させています。

リソースヒントは、ユーザーにとっての利点を評価しなければ、諸刃の剣になりかねません。プリロード要求のほぼ4分の1は未使用であり、プリロードヒントの数はロード時間の遅さと相関しています。

資源のヒントは、レーシングカーのエンジンの微調整に似ている。遅いエンジンを速くすることはできないし、調整しすぎると壊れてしまう。でも、ちょっとした工夫で、その性能を最大限に引き出すことができる。

925. *https://datatracker.ietf.org/doc/html/rfc8297*
926. *https://developer.chrome.com/blog/origin-trials*

もう一度言いますが、リソースヒントには、「すべてが重要なら、何も重要ではない」という考え方があります。リソースヒントは賢く使い、使い過ぎないようにしましょう。

著者



Kevin Farrugia

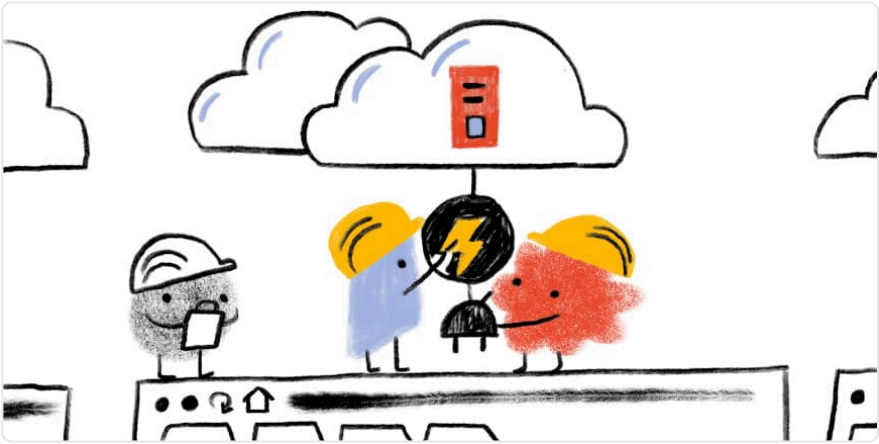
@imkevdev @https://webperf.social/@kevinfarrugia kevinfarrugia imkevdev

https://imkev.dev

Kevin Farrugiaは、ウェブパフォーマンスとソフトウェアアーキテクチャのコンサルタントです。ブログはimkev.dev[927] でご覧いただけます。

927. *https://imkev.dev*

部 IV 章 21
CDN



Navaneeth Krishna によって書かれた。
Julia Yang と Shilpa Raghunathan によってレビュ。
Paul Calvano による分析。
Julia Yang と Shaina Hantsis 編集。
Sakae Kotaro によって翻訳された。

序章

コンテンツ配信ネットワーク（CDN）は、データセンターに設置されたプロキシサーバーの地理的な分散ネットワークです。CDNの目的は、ウェブコンテンツに高い可用性とパフォーマンスを提供することです。CDNは、コンテンツをエンドユーザーの近くに配信することで、これを実現します。

CDNは20年以上前から存在しています。オンラインビデオの消費、オンラインショッピング、COVID-19によるビデオ会議の増加により、インターネットトラフィックが急激に増加しているため、CDNは以前にも増して必要とされています。CDNは、このようなインターネットトラフィックの増加にもかかわらず、高い可用性と優れたウェブパフォーマンスを保証しています。

初期の頃は、CDNはプロキシサーバーの単純なネットワークでした。

1. コンテンツ（HTML、画像、スタイルシート、JavaScript、動画など）をキャッシュする。
2. エンドユーザーがコンテンツにアクセスする際のネットワークホップを減らすことができる
3. ウェブプロパティをホストするデータセンターからTCPコネクションの終了をオフロードする

主にウェブ所有者がページのロード時間を改善し、これらのウェブプロパティをホストするインフラからトラフィックをオフロードするのに役立っています。

時を経て、CDNプロバイダーが提供するサービスは、キャッシュや帯域/接続のオフロードにとどまらず、進化を遂げています。現在、彼らは以下のような追加サービスを提供しています。

- クラウドホスティング型Webアプリケーションファイアウォール
- ボットマネジメントソリューション
- クリーンパイプソリューション（データセンターのスクラビング）
- サーバーレス・コンピューティングの提供
- 画像・映像管理ソリューションなど

このように、最近のウェブオーナーは、多くの選択肢の中から選ぶことができるのです。CDNが提供する新しいサービスは、アプリケーションの延長線上にあり、アプリケーション開発のライフサイクルと密接に統合する必要があるため、これは圧倒的で複雑なものとなり得ます。

Webアプリケーションのロジックやワークフローをエンドユーザーに近づけることは、Webオーナーにとってメリットがあります。これにより、HTTP/HTTPSリクエストにかかる往復の時間と帯域幅が不要になります。また、オリジンのスケーラビリティ要件も、ほぼ瞬時に処理できます。この副次的効果として、インターネットサービスプロバイダー（ISP）もスケーラビリティ管理の恩恵を受け、インフラストラクチャーの能力を向上させることができます。

このリクエストの減少により、インターネットのバックボーンへの負荷が軽減されます（インターネットのミドルマイルを読む[928]）。また、インターネットのラストワンマイル内のインターネット負荷のより多くを管理するのに役立ちます。このように、CDNは、ウェブ所有者がコンテンツ配信のパフォーマンス、信頼性、スケーラビリティを向上させることができるため、インターネットの状況において多面的な役割を担っているのです。

928. *https://en.wikipedia.org/wiki/Middle_mile*

注意事項・免責事項

観察研究と同様、測定可能な範囲と影響には限界があります。Web Almanacのために収集したCDNの使用に関する統計は、使用中の適用可能なテクノロジーに重点を置いており、特定のCDNベンダーのパフォーマンスや有効性を測定することを意図したものではありません。このため、どのCDNベンダーにも偏ることはありませんが、より一般的な結果であることを意味します。

これらは、私たちのテスト方法論の限界です。

- **ネットワーク遅延のシミュレーション:** トラフィックを合成的に形成する専用のネットワーク接続を使用しています。

- **単一の地理的な場所:** テストは単一のデータセンターから実行され、多くのCDNベンダーの地理的分散をテストすることはできません。

- **キャッシュの有効性:** 各CDNは独自の技術を使用しており、セキュリティ上の理由からキャッシュの性能を公開していないものが多くあります。

- **ローカライゼーションと国際化:** 地理的分布と同様に、言語や地域固有のドメインの影響も、これらのテストでは不透明です。

- **CDN検出:** これは主にDNSの解決とHTTPヘッダーによって行われます。ほとんどのCDNは、ユーザーを最適なデータセンターにマッピングするためDNS CNAMEを使用します。しかし、一部のCDNはエニーキャストIPを使用したり、DNSチェーンを隠す委任ドメインから直接A+AAAA応答を使用したりします。また、ウェブサイトが複数のCDNを使用してベンダー間のバランスを取っている場合もあり、この場合はクローラーのシングルリクエスト・パスからは見えません。

これらはすべて、私たちの測定に影響を与えるものです。

もっとも重要なことは、これらの結果はサイトごとの特定の機能（たとえば、TLS、HTTP/2など）の利用状況を反映していますが、実際のトラフィック利用状況を反映していないことです。YouTubeは “www.example.com “よりも人気がありますが、利用率を比較すると、どちらも同じ値として表示されます。

これを踏まえて、CDNの文脈では意図的に測定していない統計データをいくつか紹介します。

1. 先頭バイトまでの時間 (TTFB)
2. CDN往復時間
3. コアウェブ・バイタル
4. キャッシュヒットとキャッシュミスのパフォーマンスなど。

これらのうちいくつかはHTTP Archiveデータセットで、その他はCrUXデータセットで測定できますが我々の方法論の限界と、いくつかのサイトが複数のCDNを使用していることから測定が困難で、その結果誤った帰属をする可能性があります。これらの理由から、この章ではこれらの統計を測定しないことにしました。

CDN採用

Webページのコンテンツは、大きく3つに分けられます。

1. ベースとなるHTMLページ (たとえば、`www.example.com`)
2. サブドメインに埋め込まれたファーストパーティコンテンツ（たとえば、`images.example.com`、`css.example.com`など。）
3. サーとパーティーコンテンツ（Google Analytics、広告など。）

CDNは、設立当初から、画像、スタイルシート、JavaScript、フォントなどの埋め込みコンテンツを配信するための最適なソリューションとして利用されてきました。この種のコンテンツは頻繁に変更されないので、CDNのプロキシサーバーへキャッシュするのに適しています。

CDN技術の進化に伴い、インターネット上に非キャッシュ資産のための高速道路が設置されました。これにより、オリジンへのTCP接続に比べ、メインのWebページやAPIを確実かつ高速に、配信できるようになりました。

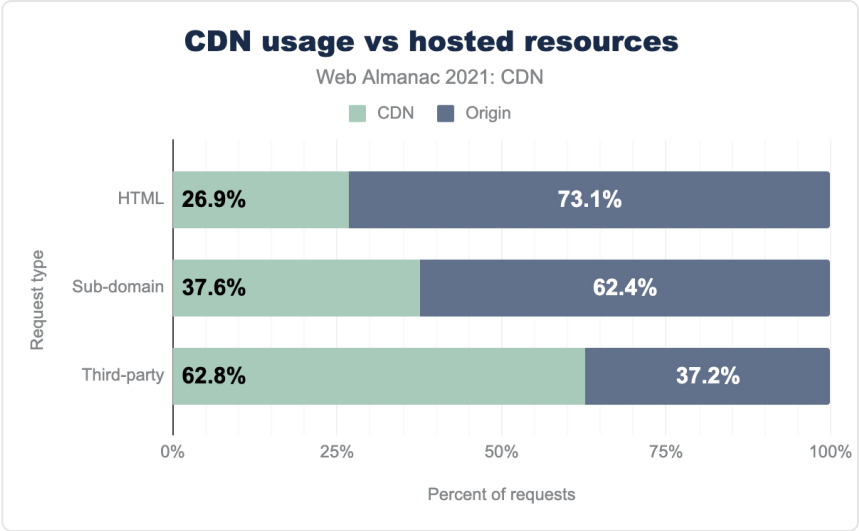


図21.1. CDNの使用とホスティングされたリソースの比較。

その影響は、2019年版[929]の同データと比較すると、上のグラフのようにわかります（注、2020年版Web AlmanacにはCDNの章がありませんでした）。CDNを利用しているサイトの傾向が、2019年から2021年にかけて7%改善されたのは良いことです。これは、より多くの業界がCDNを活用して、安定したコンテンツ配信時間の恩恵を受け、インターネット上の混雑の影響を最小限に抑えていることを示しています。

サードパーティコンテンツを見ると、CDN採用がマイナス成長となっています。2019年の章[930]と比較すると、CDNを利用しているドメインが3%減少していることが分かります。サードパーティドメインは、SaaSベンダーが分析、広告、レスポンシブページなどのために使用しています。SaaSベンダーのサービスにCDNを利用することは、SaaSベンダーにとって利益となります。SaaSベンダーのコンテンツは複数のウェブ所有者によって利用され、そのコンテンツは地域を超えてエンドユーザーによってアクセスされるため、ビジネスとパフォーマンスの両方の観点からCDNが必要なのです。このことは、サードパーティのコンテンツがCDNをもっとも高く採用していることが明らかなグラフに表れています。

しかし、なぜサードパーティドメインのCDN Adoptionがこのようにネガティブな成長を見せるのでしょうか。

その理由として考えられるのは、以下の通りです。

- HTTP/2プロトコルは、最適なパフォーマンスを得るために、ウェブオーナーが

929. https://almanac.httparchive.org/ja/2019/cdn
930. https://almanac.httparchive.org/ja/2019/cdn

複数のドメインを使用するのではなく、ドメインを統合することを要求している

- サードパーティーのコンテンツがページの総重量に占める割合も年々増加しており（詳細はサードパーティーの章を参照）、ウェブオーナーのページロード時間に対する懸念が高まっている。
- ウェブオーナーの要求に合わせたサードパーティ製スクリプトのカスタマイズ/パーソナライズ

このような変化により、SaaSベンダーはウェブオーナーに「セルフホスティング」オプションを提供するようになりました。これにより、ベンダーのドメインではなく、ファーストパーティのドメイン上で配信されるコンテンツが増えることになります。この場合、CDNを経由してコンテンツを配信するか、ホスティングインフラから直接配信するかは、ウェブオーナー次第です。

さまざまな種類のコンテンツでCDN導入を観測していますが、以下ではこのデータを別の視点から見ていきます。

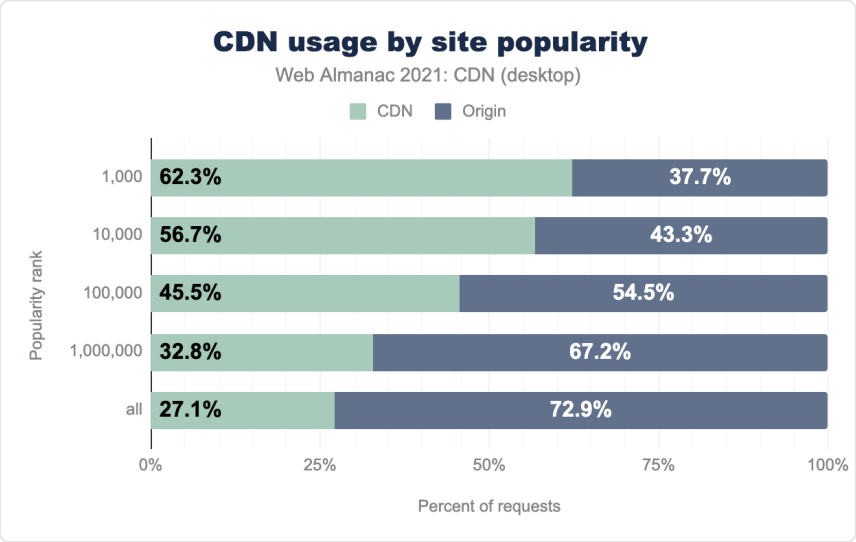


図21.2. サイトの人気度によるCDN利用状況（デスクトップ）。

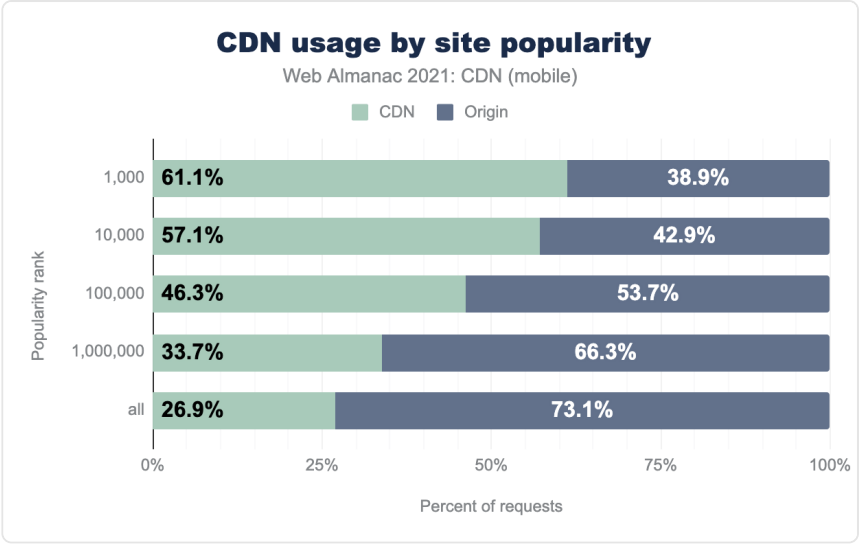


図21.3. サイトの人気度別のCDN利用状況（モバイル）。

Webサイトのウェブ上での人気度に基づいてランキング（GoogleのChrome UX Reportから引用）し、CDNの利用状況を調べたところ、上位1,000位までのサイトはもっともCDNの利用率が、高いことがわかりました。上位のウェブサイトは、GoogleやAmazonのような大企業が所有しており、今日我々が目にするインターネットトラフィックの多くに貢献しています。したがって、これらの名前が次のセクションのトップCDNプロバイダーのリストに掲載されているのは当然のことなのです。これは、CDNが大規模に運用され、必要に応じてさらに拡張できる能力を持つことでもたらされる利点についての事実も裏付けています。

61.1%

図21.4. モバイルサイト上位1,000サイトのうち、CDNを利用している割合。

CDN採用率は、上位10万サイトを見ると50%を切りますが、それ以上では減少率が鈍化します。全データセット（デスクトップで620万サイト、モバイルで750万サイト）では、これらのウェブサイトの27%がCDNを使用しています。この割合を実際の数字に置き換えると、200万ものモバイルサイトがCDNを利用していることになります。このように考えてみると、それほど小さな数字ではありません。

しかし、CDNの利点（キャッシュやTCP接続のオフロードなど）がウェブプロパティのエンドユーザー数とともに増加することを考えると、人気のないウェブサイトエンドでのCDN採用の割合が減少することは理にかなっていると言えるでしょう。ウェブプロパティのエンド

ユーザーのトラフィックが一定規模以下になると、CDNの費用対効果の計算がうまくいかなくなり、ウェブコンテンツをオリジンから直接配信したほうが、よくなる可能性があるのです。

トップCDNプロバイダー

CDNプロバイダーは、大きく2つのセグメントに分類されます。

1. 汎用CDN（Akamai、Cloudflare、Fastlyなど）。
2. 目的別CDN（Netlify、WordPressなど）

Generic CDNは、マスマーケットの要求に応えます。その提供するものは以下の通りです。

- Webサイト配信
- モバイルアプリAPI配信
- ビデオストリーミング
- サーバーレスコンピュートの提供
- Webセキュリティの提供など

これは、より多くの業界にアピールするものであり、データにも反映されています。HTMLとファーストパーティサブドメインのトラフィックは、ジェネリックCDNが大部分を占めています。

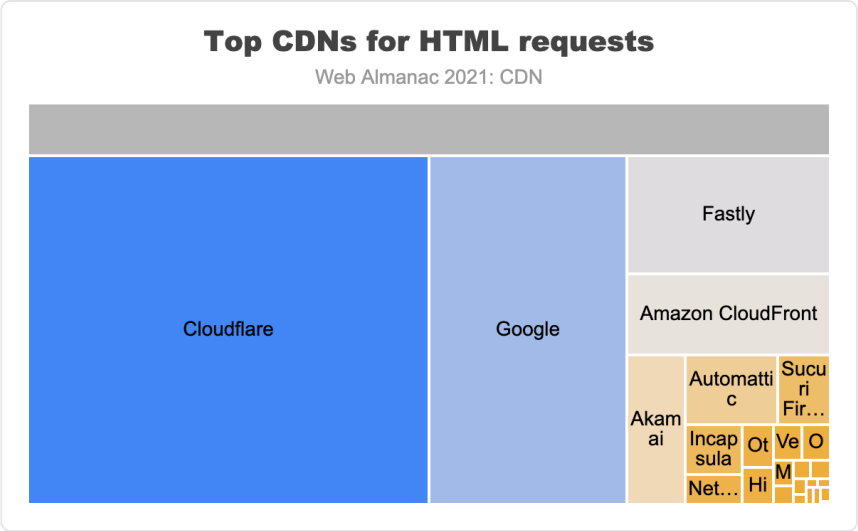


図21.5. HTMLリクエストのトップCDN。

Cloudflare、Fastly、Akamai、LimelightなどのCDNプロバイダーがこのGeneric CDN providersのリストに表示されます。また、GoogleやAWSのようなプロバイダーも含まれています。これらのプロバイダーは、クラウドホスティングサービスにバンドルされたCDNサービスを提供しているため、このリストに掲載されています。これらのバンドルは、ホスティングインフラの負荷を軽減し、ウェブパフォーマンスを向上させるのに役立つ。

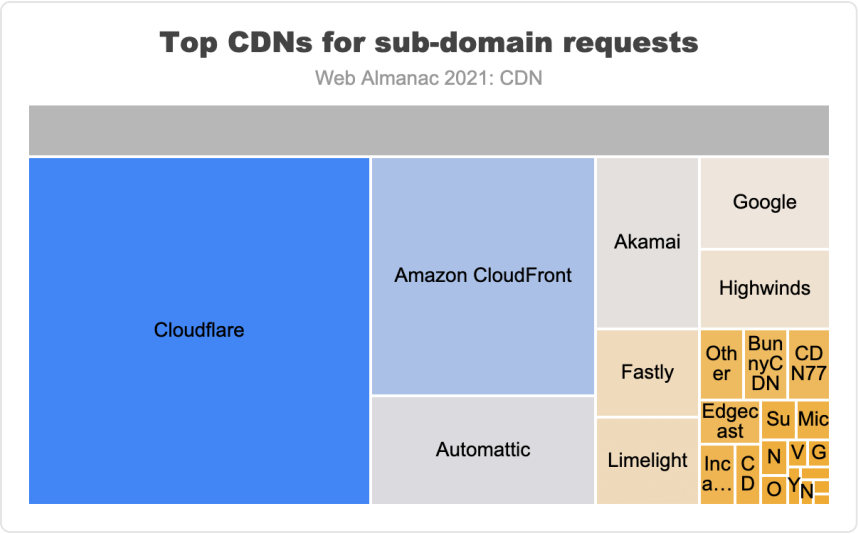


図21.6. サブドメインリクエストの上位CDN。

以下のサードパーティードメインを見ると、トップCDNプロバイダーの傾向が異なっていることがわかります。一般的なCDNプロバイダーの前にGoogleがトップになっているのがわかります。また、このリストでは、Facebookも目立っています。これは、多くのサードパーティードメインの所有者が、他の業界よりもCDNを必要としているという事実に裏打ちされています。そのため、彼らは専用のCDNを構築するために投資する必要があるのです。目的別CDNとは、特定のコンテンツ配信ワークフローに最適化されたCDNのことです。

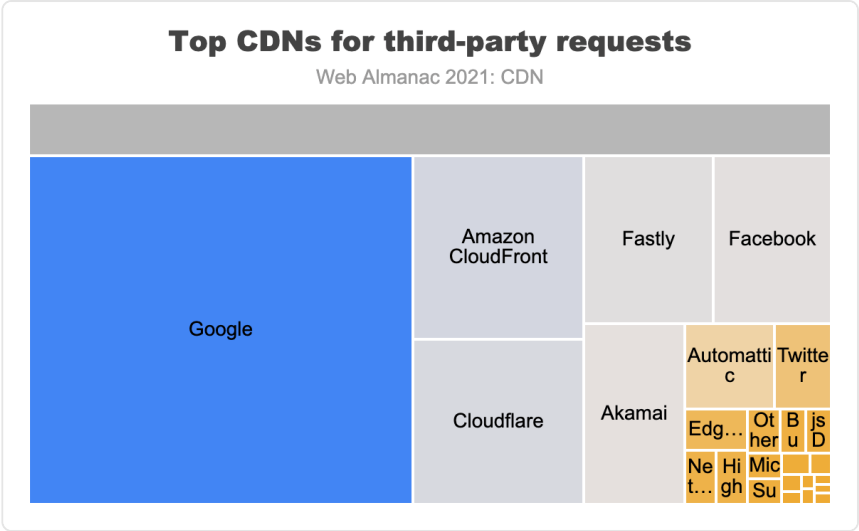


図21.7. サードパーティからのリクエストに対応するトップCDN。

たとえば、広告配信に特化して構築されたCDNは、最適化されます。

- 高い入出力（I/O）動作
- ロングテール[931]コンテンツの効果的な管理方法
- サービスを必要とする企業との地理的な近さ

つまり、汎用CDNソリューションとは対照的に、専用CDNは特定の市場セグメントの要件を正確に満たしているのです。汎用的なソリューションは、より広範な要件を満たすことができますが、特定の業界や市場向けに最適化されているわけではありません。

TLS採用の影響

リクエスト・レスポンス型のワークフローでCDNをセットアップすると、エンドユーザーのTLS接続はCDNで終了します。一方、CDNは2つ目の独立したTLS接続をセットアップし、この接続はCDNからオリジンホストに送られます。CDNワークフローにおけるこのブレークにより、CDNはエンドユーザーのTLSパラメーターを定義できます。CDNはまた、インターネットプロトコルの自動更新を提供する傾向があります。これにより、ウェブオーナーはオリジンに変更を加えることなく、これらの利点を享受できます。

931. https://ja.wikipedia.org/wiki/%E3%83%AD%E3%83%B3%E3%82%B0%E3%83%86%E3%83%BC%E3%83%AB

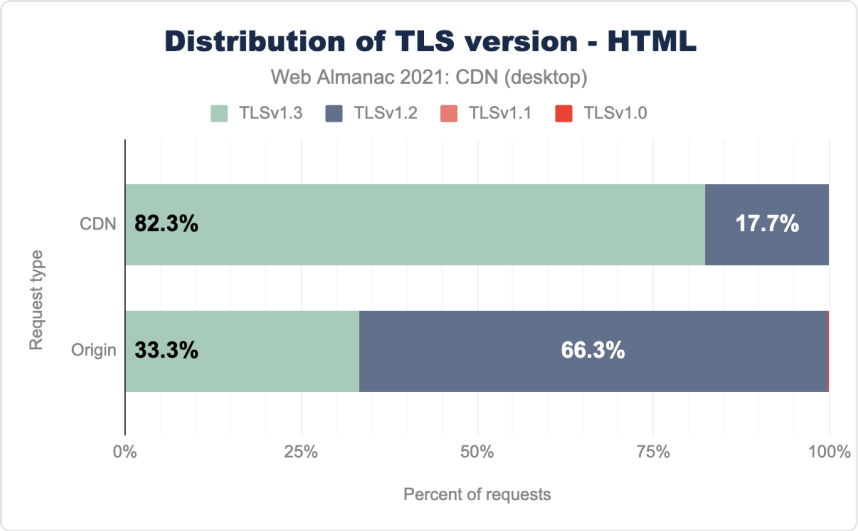


図21.8. HTML（デスクトップ）用TLSバージョンの配布。

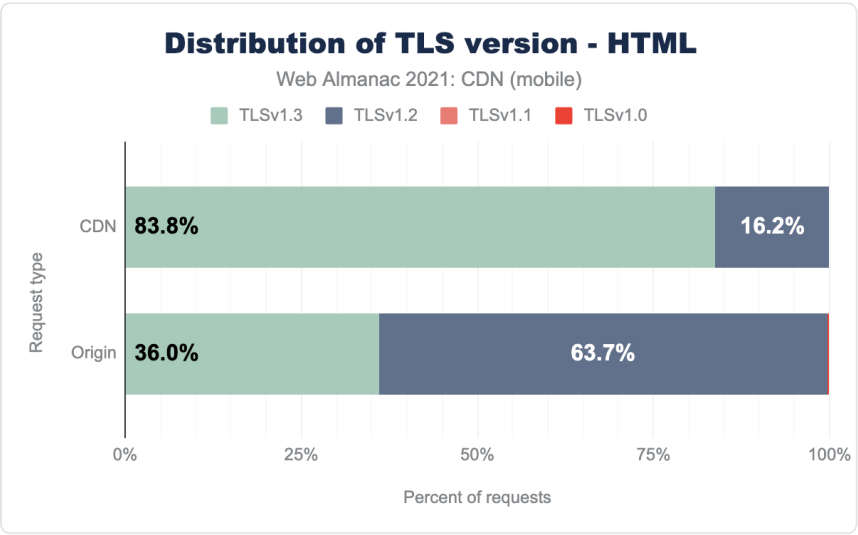


図21.9. HTML（モバイル）用TLSバージョンの配布。

上のデータを見ると、オリジンでは33-36%であるのに対し、CDNでは83%のウェブサイトがTLS 1.3を使用していることがわかります。これは、CDNを利用する大きなメリットです。また、これらのプロトコルのアップグレードは、ウェブオーナーにとって最小限の労力で行うことができます。この傾向は、モバイルおよびデスクトップのWebサイトでも同じです。

同様の傾向は、以下のサードパーティードメインでも観察されます。CDNを持つこれらのウェブサービスは、同じ理由で、持たないものに比べてTLS 1.3の採用率が高いです。

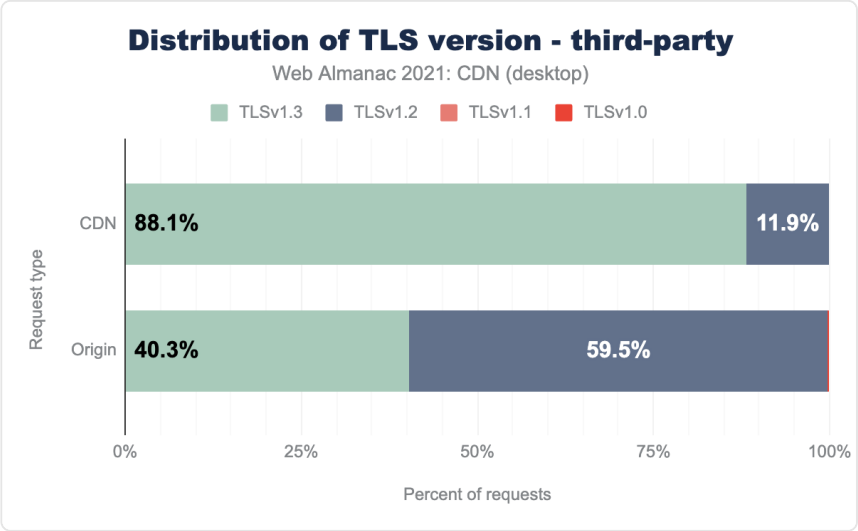


図21.10. サードパーティからの要求に対するTLSバージョンの配布（デスクトップ）。

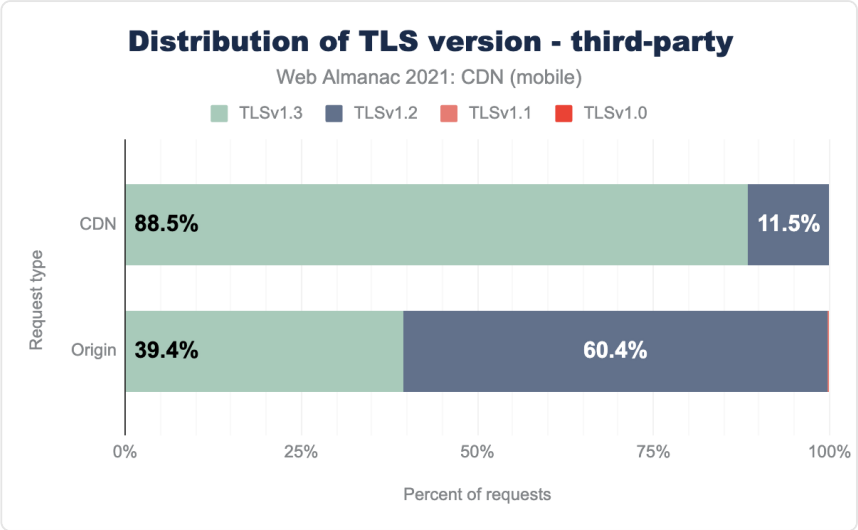


図21.11. サードパーティリクエスト用TLSバージョンの配布（モバイル）。

サードパーティのドメインは、セキュリティ上の理由から、最新のTLSバージョンであることが重要です。ウェブ攻撃の増加に伴い、ウェブ所有者はサードパーティドメインへの安全

でない接続で悪用される可能性のある抜け穴に気づいています。そのため、Webサイトのセキュリティとパフォーマンスの要件を満たす、安全なTLS接続を期待するようになります。このような期待から、CDNがもたらすメリットはますます大きくなっています。

TLSのパフォーマンスへの影響

一般的な論理で、HTTPSのリクエストとレスポンスは通過するホップ数が少なければ少ないほど、ラウンドトリップが速くなると考えられています。では、TLS接続の終端がエンドユーザーの近くにある場合、正確にはどのくらい速くなるのだろうか？答えは、「3倍」です。なんと3倍も速くなるのです

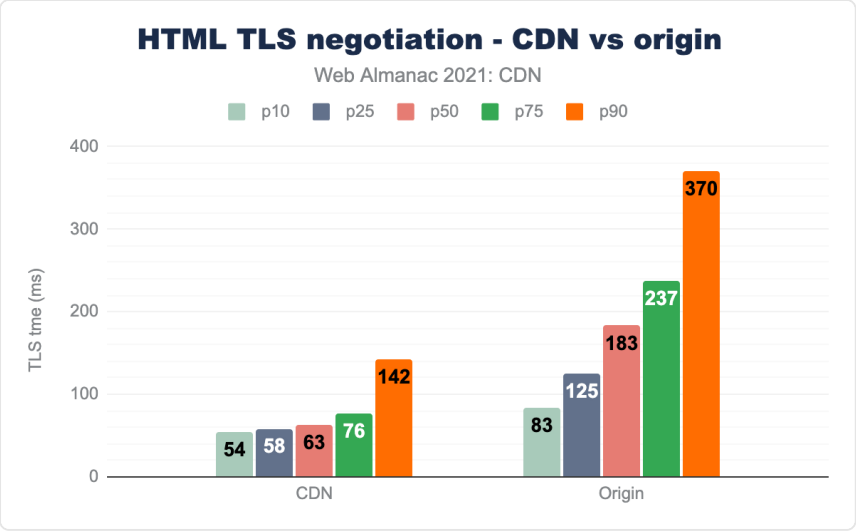


図21.12. HTML TLSネゴシエーション - CDN vs. オリジン。

CDNはTLSの接続時間を短縮するのに役立っています。これは、エンドユーザーとの距離が近いことと、TLSネゴシエーションを最適化する新しいTLSプロトコルを採用していることが要因です。CDNは、すべてのパーセンタイルでオリジンより優位に立っています。P10とP25では、CDNはTLSセットアップ時間においてオリジンより1.5倍から2倍近くも高速です。中央値以上になるとさらに差が広がり、CDNは3倍近く速くなります。CDNを利用する90パーセンタイルのユーザーは、オリジンとの直接接続を利用する50パーセンタイルのユーザーよりも優れたパフォーマンスを発揮できます。

最近、すべてのサイトがTLSを使用しなければならないことを考えると、これは非常に重要です。このレイヤーでの最適なパフォーマンスは、TLS接続に続く他のステップに不可欠です。この点で、CDNは直接オリジン接続と比較して、より多くのユーザーを低いパーセンタ

イルブラケットに移動させることが可能です。

HTTP/2+（HTTP/2以上）の採用

HTTP/2は、多くの誇大広告と期待とともに導入されました。アプリケーション層のプロトコルは、1997年のHTTP 1.1以来、更新されていなかったからです。それ以来、ウェブのトラフィック傾向、コンテンツタイプ、コンテンツサイズ、ウェブサイトデザイン、プラットフォーム、モバイルアプリなどが大きく進化してきました。そこで、現代のWebトラフィックの要求に応えられるプロトコルが必要となり、HTTP/2で実現し、さらに最新のHTTP/3で改良されました。

しかし、HTTP/2の実装上の課題により、採用が見送られました。さらに、これらの変更で期待できる正味の性能向上も明確ではありませんでした。この課題は、HTTP/3の導入でも繰り返されました。

そこで、CDNが仲介することで、ウェブオーナーのHTTP/2実装の課題を解決することができるのです。HTTP/2接続はCDNレベルで終了するため、ウェブ所有者は、それをサポートするためにインフラをアップグレードする必要なく、HTTP/2でウェブサイトとサブドメインを配信する能力を提供します - 新しいTLSバージョンで見たのとまったく同じ理由と利点があります。

CDNはホスト名を統合するレイヤーを提供し、ホスティングインフラへの変更を最小限に抑えながら、トラフィックを関連するエンドポイントにルーティングすることで、ギャップを埋めるプロキシとして機能する。キュー内のコンテンツの優先順位付けやサーバープッシュなどの機能はCDN側で管理することができ、いくつかのCDNはウェブサイトの所有者からの入力なしにこれらの機能を実行する、手動の自動ソリューションさえ提供しており、したがってHTTP/2の採用を後押ししています。

この傾向は、以下のグラフが示すもの以上に明確なものではありません。CDNを利用しているドメインでは、利用していないドメインと比較して、高いHTTP/2+の採用率が見られます。

HTTP/3の動作方法(より多くの情報のためのHTTPの章を参照)のために、HTTP/3はしばしば最初の接続に使われないので、それらのHTTP/2接続の多くが実際にリピート訪問者のためのHTTP/3であるかもしれないので、我々は代わりに「HTTP/2＋」を測定していることに注意します(我々は、HTTP/3を実装しないサーバーがないと仮定しました。).

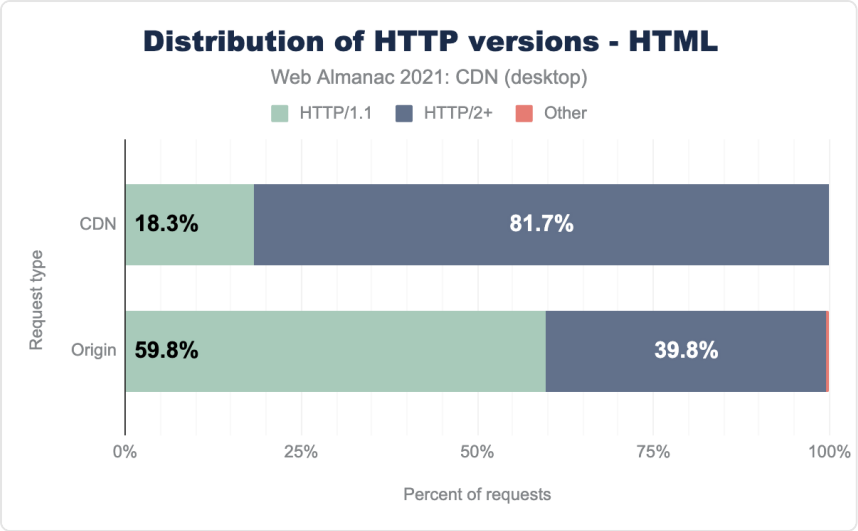


図21.13. HTML（デスクトップ）用HTTPバージョンの配布。

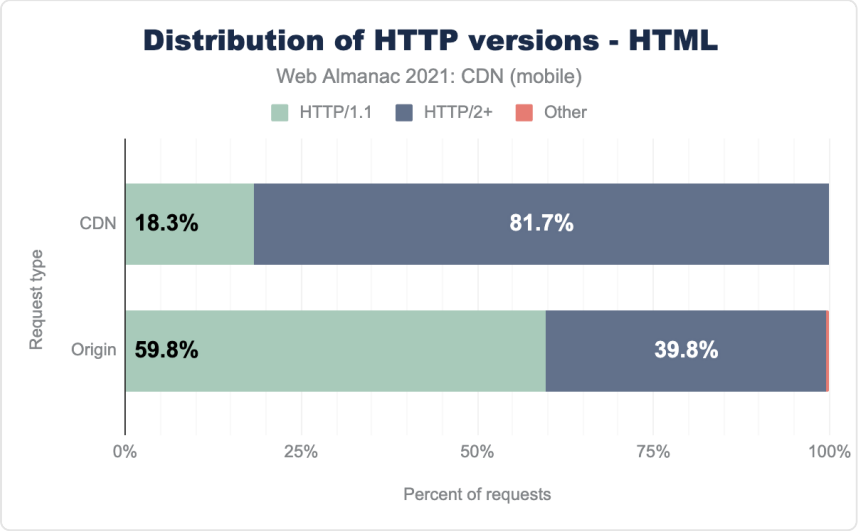


図21.14. HTML（モバイル）用HTTPバージョンの配布。

2019年当時、HTTP/2の採用率はオリジンドメインが27%だったのに対し、CDN上のドメインは71%でした。デスクトップサイトでは、2021年にHTTP/2+をサポートするオリジンが約14%増加していることがわかりますが、CDN上のドメインは15%増加し、そのリードを維持しています。この差はモバイルサイトを見ると、CDNを利用しているドメインは、デスクトップサイトと比較して、HTTP/2+の採用率が若干低くなっていることがわかります。

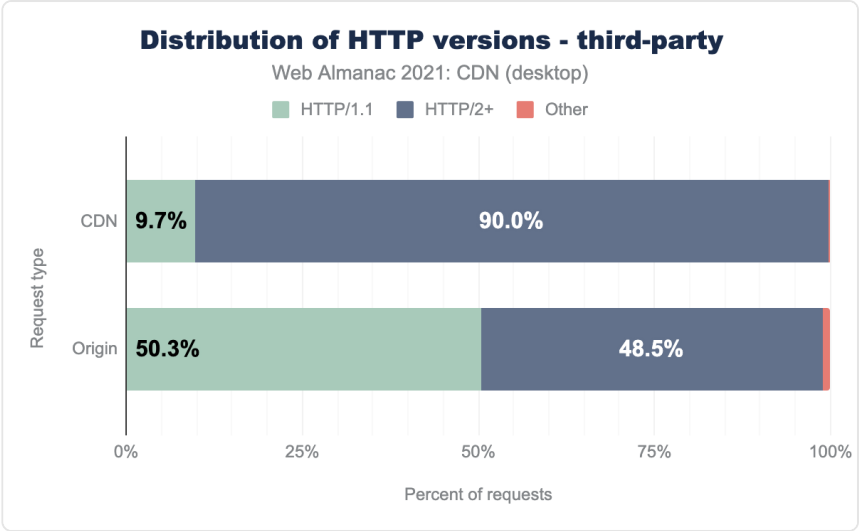


図21.15. サードパーティリクエスト用HTTPバージョンの配布（デスクトップ）。

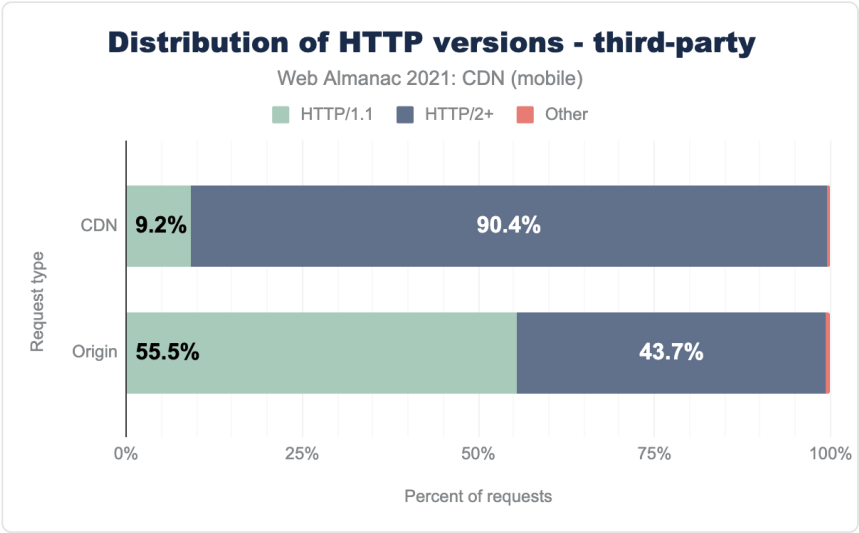


図21.16. サードパーティからのリクエストに対応したHTTPバージョンの配布（モバイル）。

新しいプロトコルをサポートするサードパーティのドメインを見ると、ファーストパーティのドメインと比較して、HTTP/2+プロトコルの採用率が高いという興味深い傾向が見受けられます。これは、上位のサードパーティードメインのほとんどが専用のCDNを使用しているため、コンテンツの開発とコンテンツ配信をよりコントロールできるという事実を考慮する

と、理にかなっていると言えます。さらにサードパーティードメインは、あらゆるネットワーク状況下で一貫したパフォーマンスを発揮する必要があり、そこでHTTP/2+は、従来のTCP接続に加えてUDP（HTTP/3で使用）などの他のプロトコルを混ぜることで価値を高めているのです。

2019年のことですが、UberはTCP（HTTP/3のトランスポート層であるQUIC）とともにUDPがどのように安定したパフォーマンスでコンテンツを配信し、混雑度の高いモバイルネットワークでのパケットロスを克服できるかを理解するための実験を行いました。このブログ[932]で文書化されているこの実験の結果は、HTTP/3が役立つ層について貴重な洞察を投げかけています。このトレンドは時間とともに浸透し、とくにモバイルネットワークのトラフィックがインターネットトラフィック全体に占める割合が高くなっていることから、ウェブオーナーがHTTP/3を採用するようになるはずです。

Brotli採用

インターネット上で配信されるコンテンツは、ペイロードサイズを小さくするために圧縮を採用しています。ペイロードが小さくなれば、サーバーからエンドユーザーへのコンテンツ配信がより速く行えるようになります。これにより、Webサイトの読み込みが速くなり、より良いエンドユーザー体験を提供できます。画像の場合、この圧縮はJPEG、WEBP、AVIFなどの画像ファイル形式によって処理されます（これについては、メディアの章を参照してください）。テキスト形式のWeb資産（HTML、JavaScript、スタイルシートなど）の圧縮は、従来は*Gzip*というファイル形式で処理されていた。Gzipは1992年から存在しています。それはテキスト資産のペイロードを小さくするのに良い仕事をしましたが、新しいテキスト資産圧縮はGzipより良いことができます: *Brotli*（詳細は圧縮の章を参照してください）。

TLSやHTTP/2の採用と同様に、Brotliもウェブプラットフォーム全体で徐々に採用される段階を経ました。この記事の執筆時点で、Brotliは世界中の96%[933]のウェブブラウザでサポートされています。しかし、すべてのWebサイトがBrotli形式でテキスト資産を圧縮しているわけではありません。これは、サポートが十分でないことと、Gzip圧縮に比べてBrotli形式でテキストアセットを圧縮するのに必要な時間が長いことの両方が理由です。また、ホスティングインフラストラクチャーは、Brotli形式をサポートしていない古いプラットフォーム向けにGzip圧縮されたアセットを提供するために下位互換性を持つ必要があり、複雑さを増す可能性があります。

この影響は、CDNを利用しているWebサイトと利用していないWebサイトを比較した場合に確認できます。

932. https://eng.uber.com/employing-quic-protocol/
933. https://caniuse.com/brotli

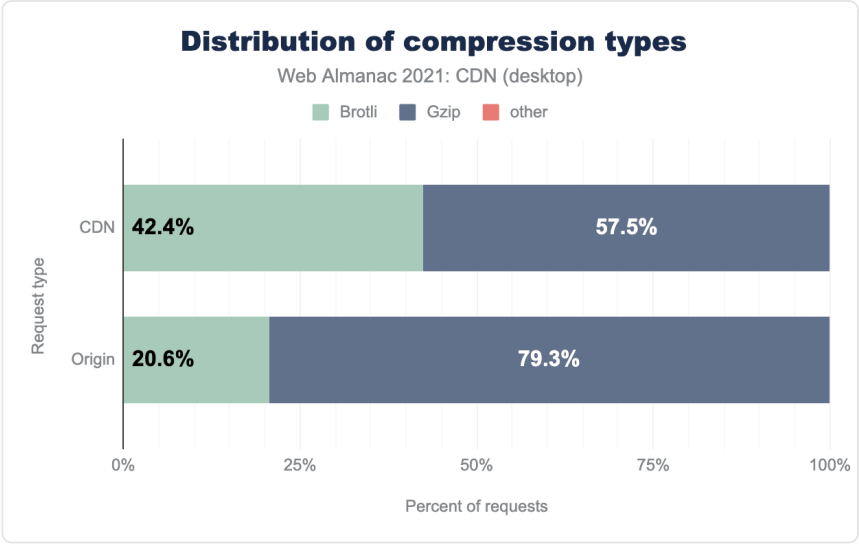


図21.17. 圧縮タイプの分布（デスクトップ）。

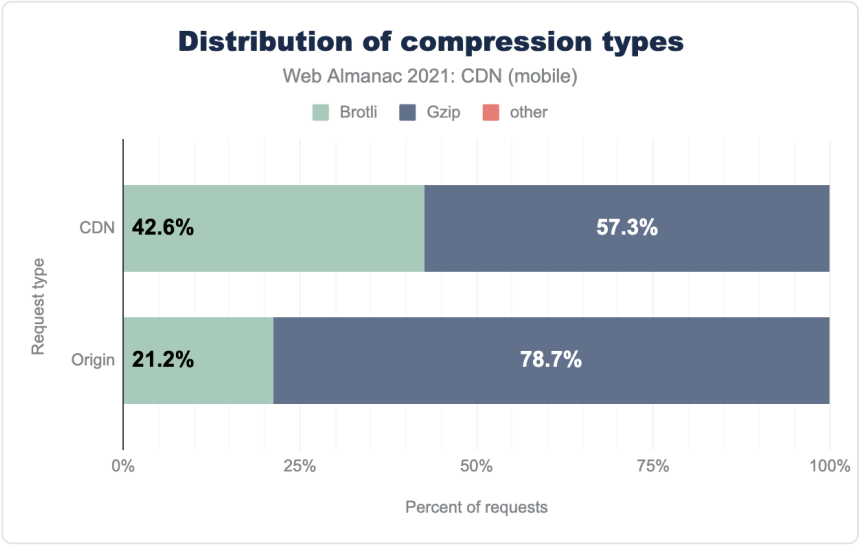


図21.18. 圧縮タイプの分布（モバイル）。

デスクトップとモバイルの両プラットフォームにおいて、オリジンから配信されるドメインと比較して、CDNはBrotliで2倍のテキストアセットを配信していることがわかります。先ほど取り上げたCDN採用のセクションから、サイトを配信するドメインの73%はCDNであり、これらはすべてBrotli圧縮の恩恵を受けることができます。Brotli形式のテキスト資産を

圧縮するための計算負荷をCDNにオフロードすることで、ウェブサイトの所有者はホスティングインフラのためのリソースを投資する必要がありません。

しかし、CDNでBrotli圧縮を使用するかどうかは、ウェブプロパティの所有者の裁量に任されています。全世界のウェブブラウザの95%がBrotli圧縮をサポートしているのに比べ、CDNを導入していてもBrotli形式で配信されているテキスト資産は全体の半分以下であり、この採用率は明らかに改善する余地があります。

結論

CDNを裏で動かしている秘密のソースを知ることは難しいため、外部からCDNについて推論できる洞察には限界があります。しかし、私たちはドメインをクロールし、CDNを利用しているものとそうでないものを比較しました。CDNは、ネットワーク層からアプリケーション層まで、Webサイトが新しいWebプロトコルを採用することを可能にするものであったことがわかります。

この影響は普遍的で、モバイルとデスクトップで同様の採用率となっています。最新のTLSバージョンの使用から、最新のHTTPバージョン（HTTP/2やHTTP/3など）へのアップグレード、Brotli圧縮の使用まで、多岐にわたります。注目すべきは、このインパクトの大きさと、CDNドメインが非CDNドメインに対して築いてきた大きなリードです。

CDNのこのような役割は非常に価値があり、今後もこの傾向は続くと思われます。CDNプロバイダーは、インターネット技術委員会[934]の重要な一員でもあり、インターネットの未来を形作る手助けをしています。今後も、インターネットを活用した産業の円滑な運営を、確実かつ迅速に支援する重要な役割を担っていくことでしょう。

著者



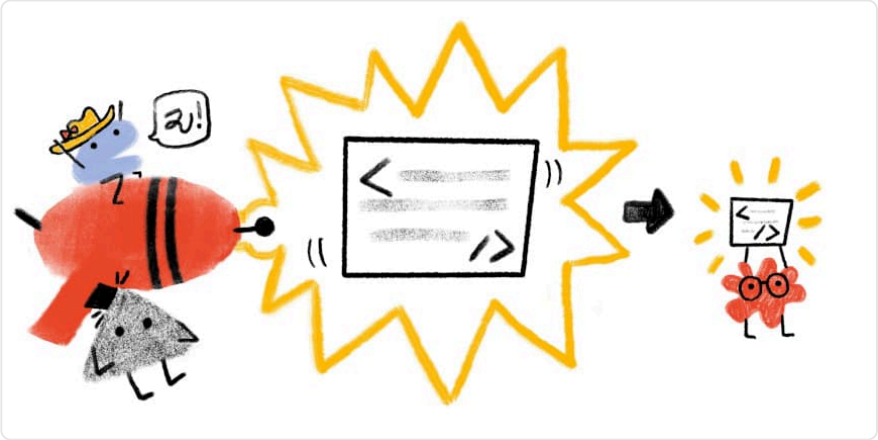
Navaneeth Krishna

@Navanee55755217 Navaneeth-akam

Navaneeth Krishna は、大手 CDN プロバイダーである Akamai[935] でウェブパフォーマンス・アーキテクトを務めています。CDN 業界で 10 年以上の経験を持つ同氏は、CDN が今後数年間のインターネットの成長に不可欠な要素になり、注目すべき空間になると確信しています。彼のツイートは@Navanee55755217で見ることができます。

934. https://www.ietf.org/
935. https://www.akamai.com/

部 IV 章 22
圧縮



Lode Vandevenne、 Moritz Firsching と Jyrki Alakuijala によって書かれた。
Thomas Fischbacher、 Eugene Kliuchnikov と Iulia Comșa によってレビュ。
Paul Calvano による分析。
Shaina Hantsis 編集。
Sakae Kotaro によって翻訳された。

序章

ユーザーの時間は貴重なので、ウェブページの読み込みに長時間待たされることはないはずです。HTTPプロトコルはレスポンスを圧縮することができ、コンテンツの転送に必要な時間を短縮できます。圧縮は、多くの場合、ユーザー体験の大幅な向上につながります。ページの重さを減らし、Webパフォーマンスを向上させ、検索順位を上げることができるのです。そのため、検索エンジン最適化の重要な一部となっています。

この章では、HTTPレスポンスに適用されるロスレス圧縮について説明します。画像、音声、動画などの media[936] フォーマットで使用される非可逆圧縮と可逆圧縮は、ページの読み込み速度を上げるために（それ以上に）同様に重要です。しかし、これらは通常、ファイル形式自体の一部であるため、この章の範囲ではありません。

936. *https://almanac.httparchive.org/ja/2020/media*

HTTP圧縮を使用するコンテンツタイプ

HTTP圧縮は、HTML、CSS、JavaScript、JSON、SVGなどのテキストベースのコンテンツや、`woff`, `ttf`, `ico` ファイルに対して推奨されています。画像のようなすでに圧縮されているメディアファイルは、前述のようにその表現にすでに内部圧縮が含まれているため、HTTP圧縮の恩恵を受けることはありません。

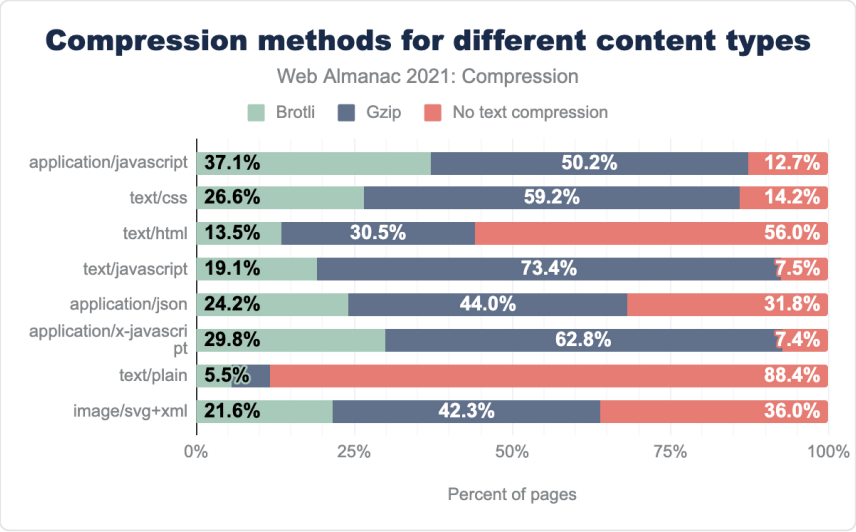


図22.1. コンテンツの種類に応じた圧縮方法

他のタイプのコンテンツと比較すると、`text/plain` と `text/html` はもっとも圧縮率が低く、12%と14%しか圧縮を使用していない。これは、動的に生成されたコンテンツも圧縮することが良い影響を与えるにもかかわらず、`text/html` がJavaScriptやCSSなどの静的コンテンツよりも動的に生成されることが多いためかもしれません。JavaScriptコンテンツの圧縮に関する詳しい分析は、JavaScriptの章にあります。

HTTP圧縮のためのサーバー設定

HTTPのコンテンツエンコーディングについては、HTTP規格で Accept-Encoding[937] リクエストヘッダーが定義されており、HTTPクライアントは、どのコンテンツエンコーディングを扱えるかをサーバーに通知できます。サーバーの応答には、応答本文のデータを変換するためにどのエンコーディングが選ばれたかを指定する Content-Encoding[938] ヘッダーフィールド

937. https://developer.mozilla.org/docs/Web/HTTP/Headers/Accept-Encoding
938. https://developer.mozilla.org/docs/Web/HTTP/Headers/Content-Encoding

を含めることができます。

実質的にすべてのテキスト圧縮は、2つのHTTPコンテンツエンコーディングのうちの1つによって行われます。Gzip[939]とBrotli[940]です。BrotliとGzipの両方は、事実上すべてのブラウザでサポートされています。サーバー側では、nginxやApacheなどもっとも普及しているサーバー[941]でBrotliやGzipを使用するように設定できます。この設定は、コンテンツが生成されるタイミングによって異なります。

- 静的コンテンツ：このコンテンツはあらかじめ圧縮しておくことができます。ウェブサーバーは、ファイル名の拡張子などに基づいて、URLを適切な圧縮ファイルにマップするように設定できます。たとえば、CSSやJavaScriptは静的コンテンツであることが多いので、あらかじめ圧縮しておくと、Webサーバーがリクエストごとに圧縮する手間を省くことができます。
- 動的に生成されるコンテンツ：これは、ウェブサーバー（またはプラグイン）自身がリクエストごとにその場で圧縮する必要があります。たとえば、HTMLやJSONは、場合によっては動的コンテンツになり得ます。

BrotliまたはGzipでテキストを圧縮する場合、異なる圧縮レベルを選択することが可能です。圧縮レベルを高くすると、圧縮ファイルは小さくなりますが、圧縮に時間がかかります。解凍中、CPU使用率は重く圧縮されたファイルほど高くならない傾向があります。むしろ、高い圧縮レベルで圧縮されたファイルは、デコードが若干速くなります。

使用するWebサーバーソフトによっては、圧縮を有効にする必要があり、あらかじめ圧縮されたコンテンツと動的に圧縮されたコンテンツとで設定が、分かれる場合があります。Apache[942]では、Brotliは mod_brotli[943] で、Gzipは mod_deflate[944] で有効にすることができるようになっています。nginxでは、Brotliを有効にする[945]方法、Gzipを有効にする[946] 方法も公開されています。

HTTP圧縮の動向

下のグラフは、過去3年間のHTTP Archiveメトリクスによるロスレス圧縮の使用率シェアの推移を示したものです。2019年からBrotliの利用率が2倍に、Gzipの利用率が若干減少しており、全体的にデスクトップ、モバイルでHTTP圧縮の利用率が伸びていることがわかります。

939. *https://tools.ietf.org/html/rfc1952*
940. *https://github.com/google/brotli*
941. *https://en.wikipedia.org/wiki/HTTP_compression#Servers_that_support_HTTP_compression*
942. *https://httpd.apache.org/*
943. *https://httpd.apache.org/docs/2.4/mod/mod_brotli.html*
944. *https://httpd.apache.org/docs/2.4/mod/mod_deflate.html*
945. *https://github.com/google/ngx_brotli*
946. *https://nginx.org/en/docs/http/ngx_http_gzip_module.html*

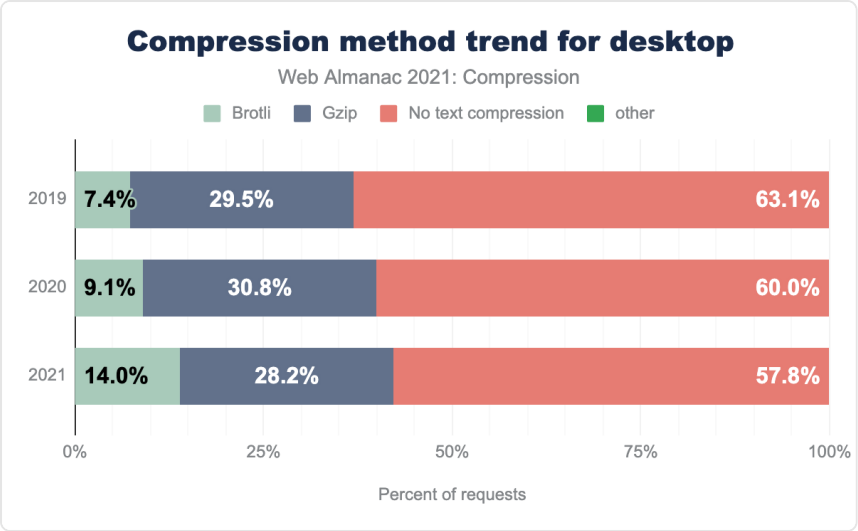


図22.2. デスクトップでの圧縮方法の傾向。

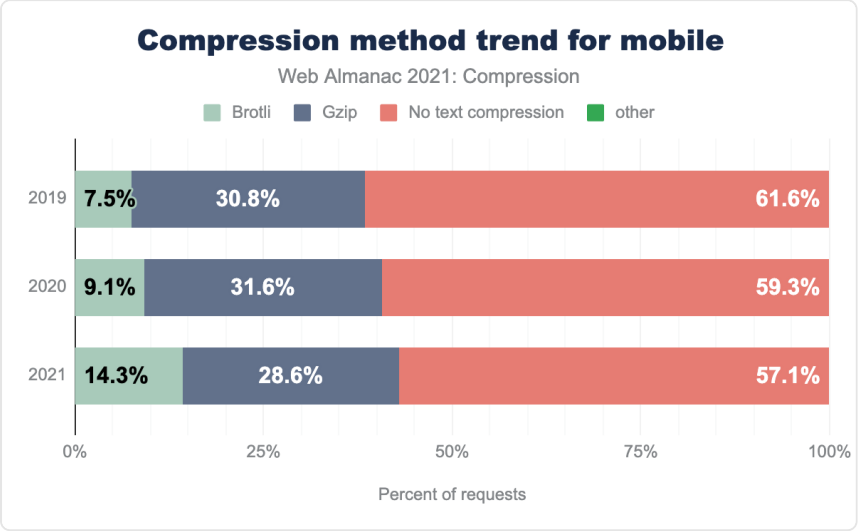


図22.3. モバイル向け圧縮方式の動向。

圧縮されて提供されているリソースのうち、大多数はGzip (66%) またはBrotli (33%) を使用しています。その他の圧縮アルゴリズムが使用されることはほとんどありません。この割合は、デスクトップとモバイルでほぼ同じです。

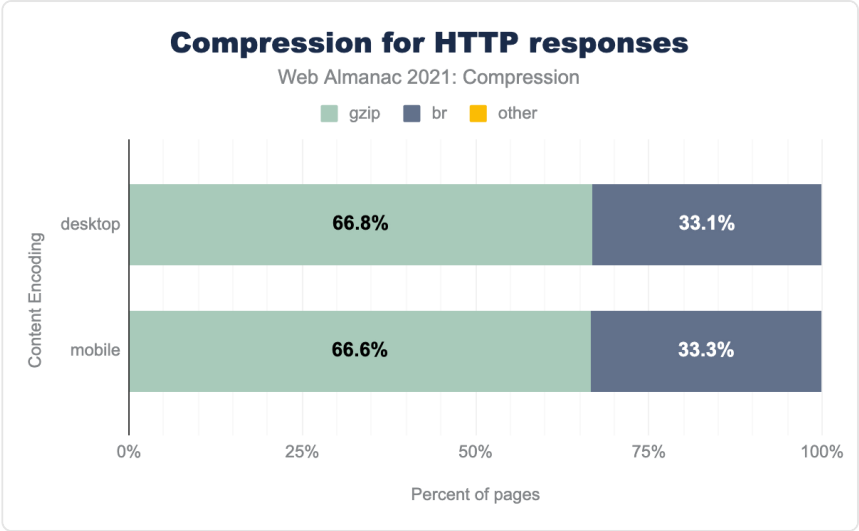


図22.4. HTTPレスポンスの圧縮アルゴリズム。

ファーストパーティとサードパーティの圧縮

サードパーティーは、ウェブサイトのユーザー体験に影響を与えます。歴史的に、サードパーティーと比較してファーストパーティーが使用する圧縮の量は大きく異なっていました。

	デスクトップ		モバイル	
コンテンツエンコーディング	ファーストパーティ	サードパーティ	ファーストパーティ	サードパーティ
テキスト圧縮なし	58.0%	57.5%	56.1%	58.3%
Gzip	28.1%	28.4%	29.1%	28.1%
Brotli	13.9%	14.1%	14.9%	13.7%
Deflate	0.0%	0.0%	0.0%	0.0%
その他/無効	0.0%	0.0%	0.0%	0.0%

図22.5. デバイスタイプ別のファーストパーティ製とサードパーティ製の圧縮率。

この結果から、2020年との比較では、ファーストパーティコンテンツがサードパーティコ

ンテンツに圧縮の利用で追いつき、同等の方法で圧縮を使っていることが分かります。圧縮、とくにBrotliの利用が両カテゴリーで伸びている。Brotliの圧縮率は、ファーストパーティコンテンツにおいて、1年前と比較して2倍になっています。

圧縮レベル

圧縮レベルとは、エンコーダーに与えられるパラメーターで、入力の冗長性を見つけるための努力の量を調整し、結果として高い圧縮密度を達成するために使われます。圧縮レベルが高くなると圧縮速度が遅くなりますが、伸長速度にはあまり影響がなく、むしろ若干速くなります。圧縮前のコンテンツの場合、圧縮に要する時間は事前に設定できるため、ユーザー体験に影響を与えることはない。動的コンテンツの場合、CPUがリソースを圧縮するのに必要な時間は、圧縮されたデータをネットワーク経由で送信する速度向上とトレードオフできる。

Brotliエンコーディングは0から11の圧縮レベルを許容し、Gzipは1から9のレベルを使用します。Zopfliのようなツールを使えば、Gzipでもより高いレベルを達成することができる。これは下のグラフで `opt` として示されている。

HTTPアーカイブの `summary_response_bodies` データテーブルを使用して、現在ウェブで使用されている圧縮レベルを分析しました。これは異なる圧縮レベル設定でレスポンスを再圧縮し、もっとも近い実際のサイズを取ることで、Brotliを使用した約14,000の圧縮レスポンスと、Gzipを使用した約11,000の圧縮レスポンスを基に推定しています。

各レベルのインスタンス数をプロットすると、もっともよく使われるBrotli圧縮レベルについて、圧縮レベル5付近と最大圧縮レベルの2つのピークが、あることがわかる。4以下の圧縮レベルの使用はまれです。

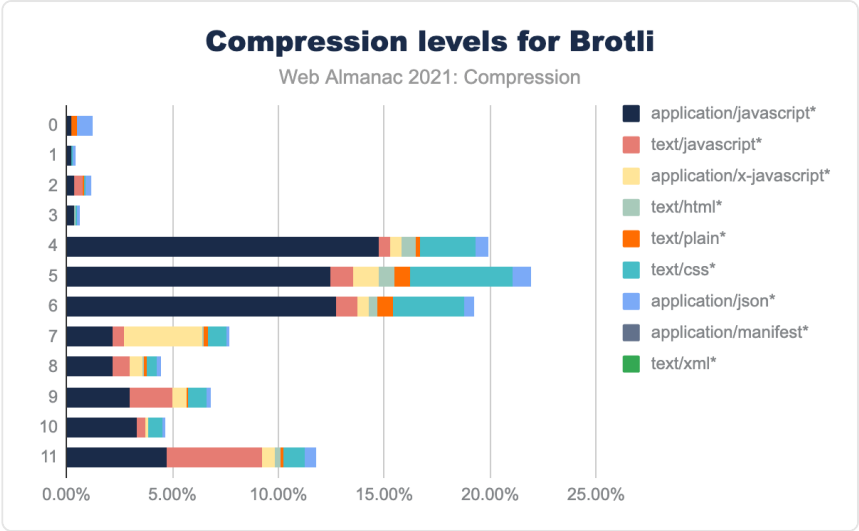


図22.6. Brotliの圧縮レベル。

Gzip圧縮は、圧縮レベル6を中心に、レベル9まで適用されます。レベル1にピークがあるのは、人気のあるWebサーバーnginx[947]のデフォルト圧縮レベルであるためと思われます。比較のため、Gzipレベル9は何千もの冗長性マッチングを試み、レベル6はそれを約100に制限し、レベル1は冗長性マッチングを4つの候補のみに制限し、圧縮率が15%悪いことを意味します。

947. https://nginx.org/

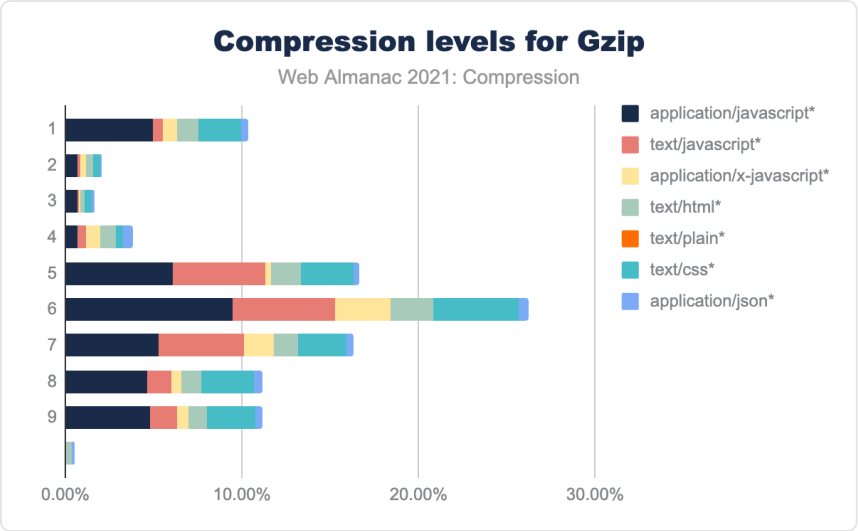


図22.7. Gzipの圧縮レベル。

図は、各圧縮レベルをコンテンツタイプ別に分けたものです。ほぼすべてのケースで、JavaScriptがもっとも一般的なコンテンツタイプです。Brotliでは、もっとも高い圧縮レベルではJavaScriptの割合が低い圧縮レベルよりも高く、低い圧縮レベルではJSONが多くなっている。Gzipでは、JavaScriptのコンテンツタイプの分布はどのレベルでもほぼ同じです。

サイトの圧縮率を分析する方法

WebサイトのどのコンテンツがHTTP圧縮を使用しているかを確認するには、Firefox Developer Tools[948] または Chrome DevTools[949] を使用することが可能です。開発者ツールで、「ネットワーク」タブを開き、サイトを再読み込みしてください。HTML、CSS、JavaScript、フォント、画像などのレスポンスのリストが表示されるはずです。どれが圧縮されているかを確認するには、その応答ヘッダーのコンテンツエンコーディングをチェックします。列を有効にすることで、すべてのレスポンスについて一度に簡単に確認できます。これを行うには、列のタイトルを右クリックし、メニューの［レスポンスヘッダー］に移動して［コンテンツエンコード］を有効にします。

Gzipで圧縮された応答は“gzip“と表示され、Brotliで圧縮された応答は“br“と表示されます。この値が空白の場合、HTTP圧縮は行われません。画像の場合、これらのリソースはすでにそれ自体で圧縮されているため、これは正常です。

948. *https://developer.mozilla.org/docs/Tools*
949. *https://developers.google.com/web/tools/chrome-devtools*

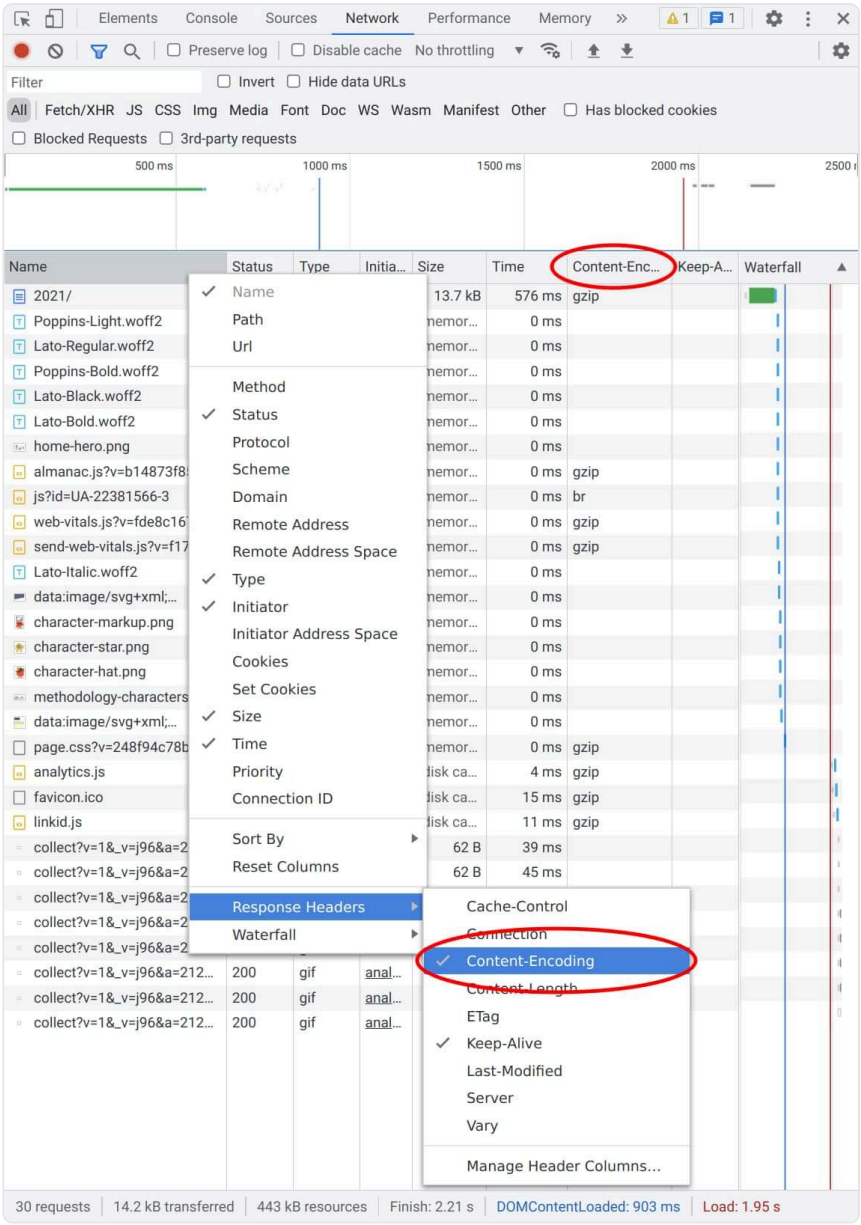


図22.8. Chrome DevToolsでレスポンスのcontent-encodingをチェックする。

サイトの圧縮を分析できる別のツールとして、GoogleのLighthouse[950]というツールがあります。「テキストの圧縮を有効にする」監査[951]を含む一連の監査を実行します。この監査では、リソースの圧縮を試み、少なくとも10%および1,400バイトが削減されたかどうかをチェックします。スコアに応じて、圧縮の推奨事項を結果に表示し、圧縮することでWebサイトに利益をもたらすリソースのリストを表示できます。

HTTP Archiveでは、すべてのモバイルページに対してLighthouse監査を実施しており、このデータから72%のWebサイトがこの監査に合格していることが確認されました。これは、昨年の[952] 74%より2%少なく、これは昨年と比較して全体的にテキスト圧縮の使用率が高いにもかかわらず、わずかに低下しています。

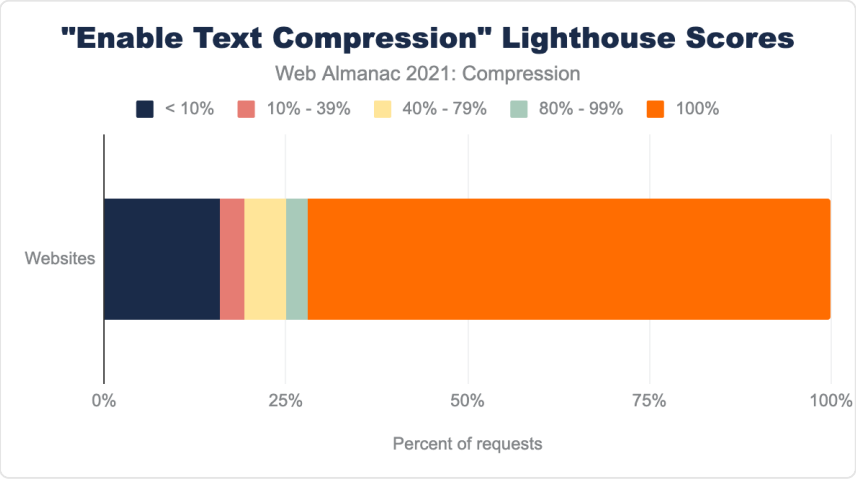


図22.9. テキスト圧縮Lighthouseのスコア。

圧縮率を向上させる方法

コンテンツの圧縮方法を考える前に、そもそも送信するコンテンツを減らすことが、賢明である場合が多い。これを実現する1つの方法は、HTMLMinifier[953]、 CSSNano[954]、UglifyJS[955] などのいわゆる「ミニマイズ」を使うことです。

送信するコンテンツの最小限の形式が決まったら、次は圧縮が有効であることを確認します。前のセクションで強調したように、有効になっていることを確認し、必要に応じてWebサーバーを設定します。

950. *https://developers.google.com/web/tools/lighthouse*
951. *https://web.dev/i18n/ja/uses-text-compression/*
952. *https://almanac.httparchive.org/ja/2020/compression#fig-9*
953. *https://github.com/kangax/html-minifier*
954. *https://github.com/ben-eb/cssnano*
955. *https://github.com/mishoo/UglifyJS2*

Gzip圧縮（DeflateまたはZlibとしても知られている）だけを使用している場合、Brotliのサポートを追加することは有益です。Gzipと比較して、Brotliは同じ速度でより小さなファイルに圧縮し[956]、同じ速度で解凍する。

よく調整された圧縮レベルを選択できます。どの圧縮レベルが正しいかは、複数の要因に依存するかもしれませんが、より重く圧縮されたテキストファイルは、デコード時に多くのCPUを必要としないことに留意してください。したがって、圧縮前の資産の場合、圧縮レベルをできるだけ高く設定しても、ユーザーの観点からは欠点がありません。動的圧縮の場合は、圧縮にかかる時間と転送時間が短くなる可能性の両方を考慮して、より重く圧縮されたファイルをユーザーが、より長く待つ必要がないようにする必要があります。この違いは、両者の圧縮レベルの推奨値を見れば明らかです。

圧縮前のリソースにGzip圧縮を使用する場合、より小さなGzip互換ファイルを生成するZopfli[957]の使用を検討してください。Zopfliは反復的なアプローチで非常にコンパクトな構文解析を見つけ、3-8%の高密度出力をもたらしますが、計算にはかなり時間がかかります。一方、Gzipはより単純ですが、あまり効果的ではないアプローチを採用しています。複数のコンプレッサーの比較[958]と、Gzipの異なる圧縮レベルを考慮したGzipとZopfliの比較[959]を参照してください。

	Brotli	*Gzip*
圧縮前	*11*	*9 か Zopfli*
動的に圧縮	*5*	*6*

図22.10. 使用する推奨圧縮レベル。

ウェブサーバソフトウェアのデフォルト設定を改善することは、ウェブパフォーマンスに時間を投資できない人々に大きな改善をもたらすでしょう。とくにGzip品質レベル1は異常値のようで、HTTPアーカイブ `summary_response_bodies` データで15%圧縮されるデフォルト6が有益でしょう。また、Brotliをサポートしているユーザーエージェントでは、Gzipの代わりにBrotliをデフォルトで有効にすることも大きなメリットになります。

結論

28,000件のHTTPレスポンスに使用された圧縮レベルを分析した結果、Gzip圧縮されたコンテンツの約0.5%にZopfliなどのより高度な圧縮器が使用されており、同様の「最適解析」アプローチはBrotli圧縮されたコンテンツの17%に使用されていることが明らかになりまし

956. *https://quixdb.github.io/squash-benchmark/*
957. *https://ja.wikipedia.org/wiki/Zopfli*
958. *https://cran.r-project.org/web/packages/brotli/vignettes/brotli-2015-09-22.pdf*
959. *https://blog.codinghorror.com/zopfli-optimization-literally-free-bandwidth/*

た。このことは、より効率的な方法が利用できれば、たとえ速度が遅くても、相当数のユーザーが静的コンテンツにこれらの方法を導入することを示しています。

HTTP圧縮の利用が増え続けており、とくにBrotliは前年の章[960]と比較して大きく増加しています。いずれかのテキスト圧縮を使用したHTTPレスポンスの数は2%増加しましたが、Brotliは4%以上増加しました。増えたとはいえ、サーバーの圧縮設定をいじることで、より多くのHTTP圧縮を利用できる機会があることは確かです。ご自身のウェブサイトのレスポンスとサーバーの設定をよく見てみると、その恩恵にあずかれるでしょう。圧縮を使用していない場合は圧縮を有効にすることを検討し、圧縮を使用している場合は、その場で生成されるHTMLなどの動的コンテンツと静的コンテンツの両方で、圧縮レベルを高くするように圧縮方法を調整することを検討できます。一般的なHTTPサーバーのデフォルトの圧縮設定を変更すると、ユーザーに大きな影響を与える可能性があります。

著者



Lode Vandevenne

lvandeve

Lode Vandevenneは、ソフトウェアエンジニアとしてGoogleスイスに勤務し、Zopfli、Brotli、JPEG XL画像フォーマットなどの圧縮プロジェクトに貢献しています。



Moritz Firsching

mo271 https://mo271.github.io/

Moritz Firschingは、Googleスイスのソフトウェアエンジニアで、プログレッシブ画像フォーマットとフォント圧縮を研究しています。それ以前は、数学者として多面体の研究をしていた。

960. *https://almanac.httparchive.org/ja/2020/compression*

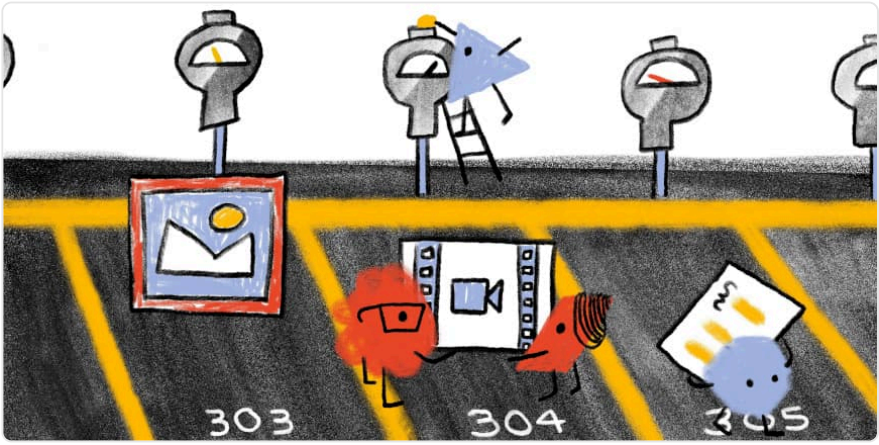


Jyrki Alakuijala

@jyzg jyrkialakuijala

Jyrki Alakuijalaは、オープンソースソフトウェアコミュニティの活発なメンバーであり、データ圧縮の研究者でもあります。最近の研究テーマは、Zopfli、Butteraugli、Guetzli、Gipfeli、WebP lossless、Brotli、JPEG XL圧縮形式とアルゴリズム、および2つのハッシュアルゴリズム、CityHashとHighwayHashです。Google入社以前は、脳神経外科や放射線治療の治療計画のためのソフトウェアを開発していました。

部 IV 章 23
キャッシング



Leonardo Zizzamia と Jessica Nicolet によって書かれた。
Wilhelm Willie と Rory Hewitt によってレビュ。
Rick Viscomi による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

この20年間で、私たちがウェブアプリケーションを体験する方法は変化し、よりリッチでインタラクティブなコンテンツが提供されるようになりました。しかし、残念ながら、このようなコンテンツには、データストレージと帯域幅の両方においてコストがかかっています。ほとんどの場合、使用するネットワークが劣化していたり、デバイスに十分な容量がなかったりすると、多くの人がウェブ製品を完全に体験することが難しくなります。キャッシュは、このような問題の解決策であると同時に原因でもあります。数多くの選択肢を使いこなすことで、ハイエンドデバイスだけでなく、ローエンドデバイスから製品にアクセスする次の10億人のユーザーにも対応した構築が可能になります。

キャッシュは、JavaScript、CSSファイル、基本的な文字列値などの単純な静的アセットから、より複雑なJSON APIレスポンスまで、以前にダウンロードしたコンテンツの再利用を可能にする技術です。

その核心は、キャッシュによって特定のHTTPリクエストを回避し、アプリケーションがユーザーに対してより応答的で信頼性が高いと感じられるようにすることです。各リクエストは通常、主に2つの場所にキャッシュされます。

- **コンテンツデリバリーネットワーク (CDNs)** は通常、サードパーティの会社で、ユーザーがアプリケーションにアクセスしている場所にできるだけ近い場所でデータを複製することを主な目的としています。ほとんどのCDNは、いくつかのデフォルトの動作を持っていますが、主にヘッダーを使用することによって、キャッシュの方法を指示できます。
- **ブラウザ**は、体験を最適化するためにあなたが定義したHTTPヘッダーを尊重するか、またはいくつかの内部デフォルトを適用します。その上、ブラウザは、単純な文字列を*cookies*に、複雑なAPI応答を*IndexedDB*に、あるいはリソース全体を*サービスワーカーで キャッシュストレージ*に保存するなどの手動キャッシュ戦略を利用することも可能です。

この章では、ブラウザとCDNの間で使用されるHTTPヘッダーに主に焦点を当て、サービスワーカーのキャッシュ戦略についても簡単に触れます。

CDNキャッシュの採用

コンテンツデリバリーネットワーク（CDN）は、通常、データのコピーを保存する複数の場所に分散したサーバーのグループです。これにより、サーバーはエンドユーザーへもっとも近いサーバーに基づいてリクエストを処理できます。

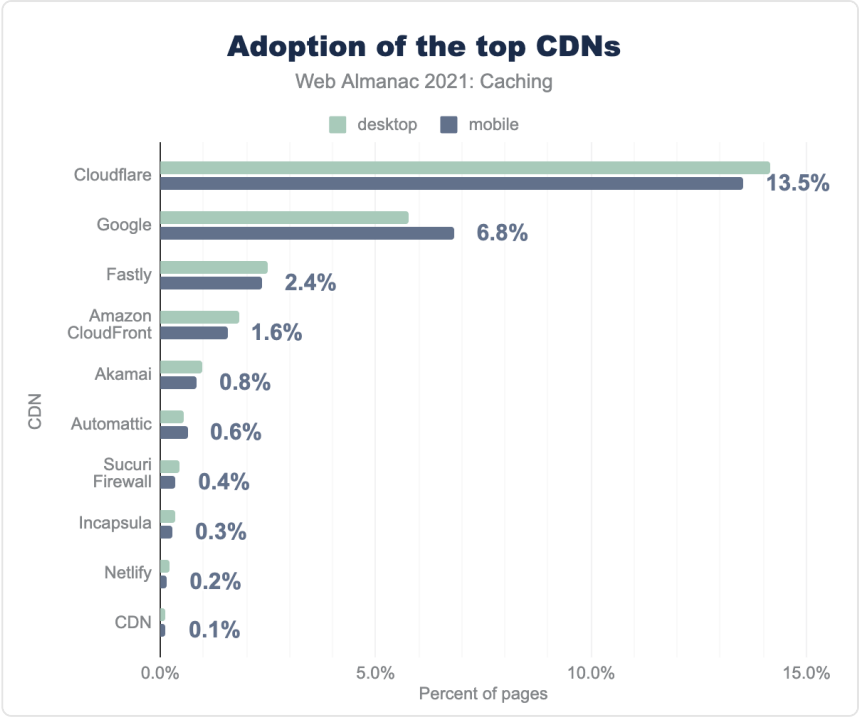


図23.1. トップCDNの採用。

2021年のウェブ全体では、Desktopに使用されているCDNはCloudflareがもっとも普及しており、総ページ数の14%、次いでGoogleが6%となっています。CloudflareとGoogleがもっとも普及していますが、この2つ以外にも、Fastly、Amazon CloudFront、Akamaiなど、多種多様なソリューションが残っています。

サービス・ワーカー採用

サービスワーカーの採用が着実に増え続けています。

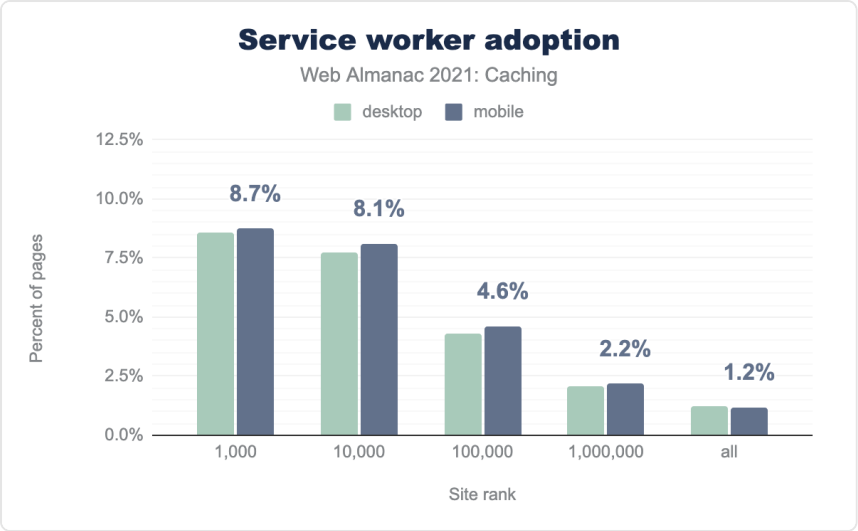


図23.2. サービスワーカーの採用。

サービスワーカーを登録しているページは1%強であるのに対し、アクセス数の多い上位1,000ドメインにランクインしているページの約9%がサービスワーカーを登録しています。

このように、とくに上位1,000ページにおいてサービスワーカーの採用率が高いのは、リモートファースト、そしてそれに関連してモバイルフレンドリーという世界的なトレンドと関係があるのかもしれません。1年を通じて1つの場所で仕事をしたり生活したりすることへの依存度がシフトするにつれて、私たちはデバイスが私たちについていくために、さらにハードでスマートな働きをすることを必要としているのです。サービスワーカーは、ユーザーが信頼性の低いネットワークやローエンドデバイスを使用している場合、パフォーマンスを向上させることができるツールです。

サービスワーカー内のリソースをキャッシュする主な方法は、*キャッシュストレージAPI*を使用することです。これにより、開発者はワーカーを通過するすべてのリクエストに対して、カスタムキャッシュ戦略を作成できます。よく知られているものに、*stale-while-revalidate*、*ネットワークにフォールバックするキャッシュ*、*キャッシュにフォールバックするネットワーク*、キャッシュのみがあります。近年では、プラグアンドプレイでキャッシュを決定できるワークボックス[961]の人気が高まっているおかげで、それらの戦略を採用するのがさらに容易になっています。

サービスワーカー、およびWorkboxについては、PWAの章で詳しく説明します。

961. *https://developers.google.com/web/tools/workbox/modules/workbox-strategies*

キャッシュヘッダーの採用

CDNとブラウザの両方において、HTTPヘッダーは開発者がリソースを適切にキャッシュするために習得しなければならない主要なツールです。ヘッダーとは、HTTPリクエストやレスポンスから読み取られる単純な命令であり、それらのいくつかは、使用するキャッシュ戦略を制御するのに役立ちます。

キャッシュ関連ヘッダーは、それがあるかないかで、ブラウザやCDNに3つの重要な情報を伝えます。

- **Cacheability**: このコンテンツはキャッシュ可能か？
- **Freshness**: キャッシュ可能な場合、どのくらいの期間キャッシュできるのか？
- **Validation**: キャッシュ可能な場合、キャッシュされたバージョンがまだ新鮮であることを確認するにはどうすればよいか？

ヘッダーは単独、あるいは一緒に使うものです。コンテンツが**キャッシュ可能**で**新鮮**であるかどうかを判断するため。

- `Expires` は、明示的に有効期限を指定します（つまり、コンテンツの正確な有効期限を指定します）。
- `Cache-Control` はキャッシュ期間 (すなわち、コンテンツが生成されたときから相対的に、ブラウザにキャッシュできる期間）を指定します。

両方が指定された場合、 `Cache-Control` が優先されます。

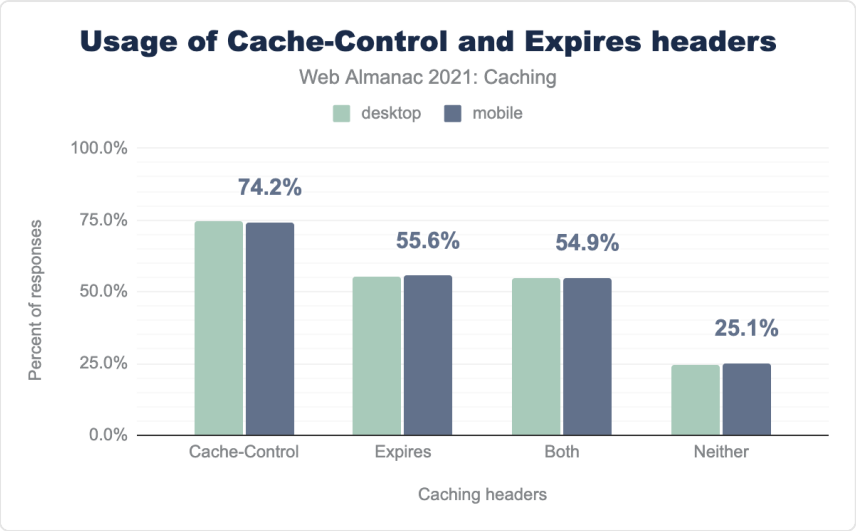


図23.3. `Cache-Control` ヘッダーと `Expires` ヘッダーを設定したレスポンスの割合。

2019年以降、`Cache-Control` ヘッダーの使用率は着実に増加しています。モバイルリクエストのレスポンスの74.2%が `Cache-Control` ヘッダーを含み、デスクトップリクエストのレスポンスの74.8%がこのヘッダーを利用していました。

2020年以降、この特定のヘッダーの使用率は、モバイルで0.71%、デスクトップでは1.13%増加しました。しかし、モバイルではまだ25.1%のリクエストが `Cache-Control` と `Expires` ヘッダーのどちらも使用していません。このことから、コミュニティでは `Cache-Control` の適切な使用に関する認識が高まっていると思われますが、これらのヘッダーを完全に採用するにはまだ長い道のりがあります。

内容を**検証**するために、私たちは

- `Last-Modified` は、オブジェクトが最後に変更された日時を示します。この値は日付のタイムスタンプです。
- `ETag` (Entity Tag) は引用符で囲まれた文字列として、コンテンツに一意の識別子を与えます。これは、サーバーが選択した任意の形式を取ることができます。通常はファイルの内容のハッシュ値ですが、タイムスタンプや単純な文字列でもかまいません。

両方が指定された場合、`ETag` が優先される。

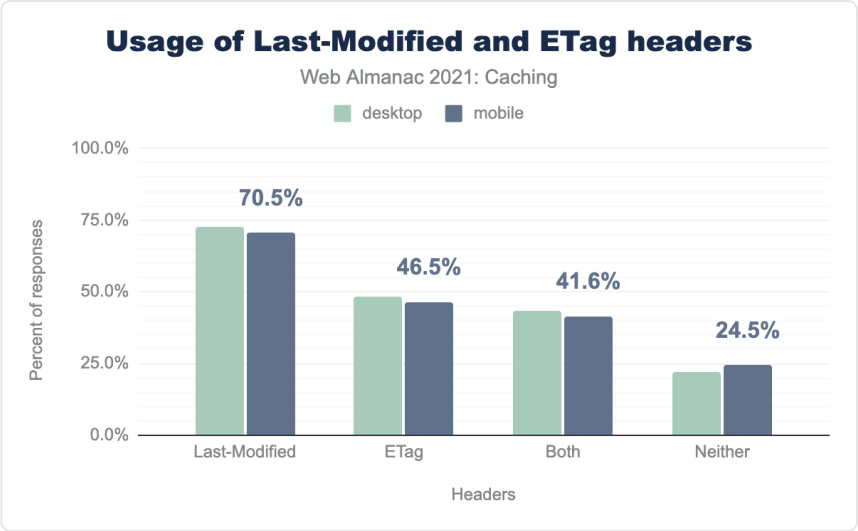


図23.4. `Last-Modified` と `ETag` ヘッダーを設定したレスポンスのパーセンテージ。

2020年と2021年を比較すると、`ETag` の利用が年々わずかに増え、`Last-Modified` の利用が1.5%減るという、過去繰り返された傾向にあることがわかります。来年注目すべきは、`ETag` や `Last-Modified` ヘッダーを使用しない回答が1.4%増加するという新しい傾向で、これはコミュニティがこれらのヘッダーの価値を理解していないことを示唆している可能性があるのです。

`Cache-Control` ディレクティブ

`Cache-Control` ヘッダーを使用する場合、特定のキャッシュ機能を示す1つ以上のディレクティブ定義された値を指定します。複数のディレクティブはカンマで区切られ、どのような順番でも設定できますが、中には互いに衝突するものもあります（たとえば、`public` と `private` など）。さらに、いくつかのディレクティブは `max-age` のような値をとります。

以下は、もっとも一般的な `Cache-Control` ディレクティブを示した表です。

ディレクティブ	説明
`max-age`	現在時刻を基準として、リソースのキャッシュが可能な秒数を示す。たとえば、`max-age=86400`などです。
`public`	ブラウザや CDN を含む、任意のキャッシュが応答を保存できることを示します。これはデフォルトで想定されています。
`no-cache`	キャッシュされたリソースは、たとえそれが古いとマークされていなくても、使用前に条件付きリクエストによって再検証されなければならない。
`must-revalidate`	キャッシュされた古いエントリは、使用前に条件付きリクエストによって再検証されなければならない。
`no-store`	レスポンスがキャッシュされてはならないことを示す。
`private`	このレスポンスは特定のユーザーを対象としたものであり、CDNなどの共有キャッシュに保存されるべきものではありません。
`immutable`	キャッシュされたエントリがその TTL の間、決して変更されず、再バリデーションが必要ないことを示す。

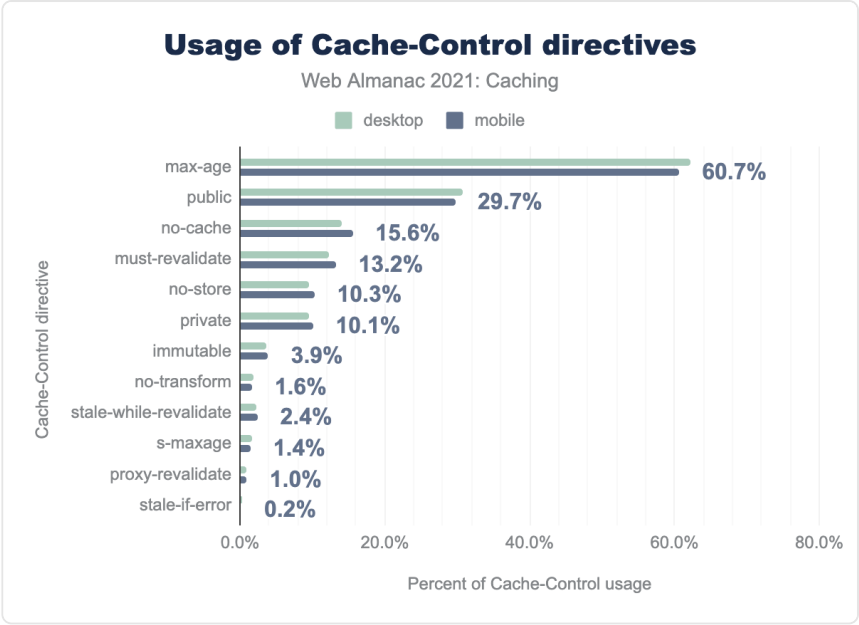


図23.5. `Cache-Control` *ディレクティブの使用法。*

`max-age` ディレクティブはもっとも一般的で、62.2%のデスクトップ・リクエストがこのディレクティブを含む `Cache-Control` レスポンス・ヘッダーを含んでいます。

2020年との比較では、上図の上位7つの指令のほとんどとともに、デスクトップでは `max-age` の採用が2%増加しました。

`immutable` ディレクティブは比較的新しいもので、特定のタイプのリクエストのキャッシュ性を大幅に向上させることができます。しかし、まだ一部のブラウザでしかサポートされておらず、*Wix*が16.4%、*Facebook* 8.6%、*Tawk* 2.8%、*Shopify* 2.4%といったホストネットワークから来るリクエストが、ほとんどであることが分かっています。

もっとも誤用されている `Cache-Control` ディレクティブは、引き続き `set-cookie` で、デスクトップでは *ディレクティブ*全体の0.07% で、モバイルでは0.08% で使用されています。しかし、2020年からは0.16%の使用量削減という意味でも喜ばしいことです。

`no-cache`、`max-age=0`、`no-store` が一緒に使われている場合を見てみると年々増加傾向にあり `no-store` が `no-cache` と `max-age=0` のいずれか/両方と一緒に指定されると、`no-store` ディレクティブが優先され、他のディレクティブが、無視されることが分かってきています。これらのディレクティブの使用について、たとえば大規模なカンファレンスの際にもっと認識を高めることで、誤ってムダなバイトを使用することを避けることができるかもしれません。

51兆年

図23.6. `max-age` で記録されたもっとも大きな値。

おもしろい事実：もっとも一般的な`max-age`の値は30日であり、もっとも大きな値は51兆年です。

`304` Not Modified ステータス

サイズに関して言えば、`304 Not Modified`のレスポンスは`200 OK`のレスポンスよりもずっと小さいので、必要なサイズのデータのみを配信することでページのパフォーマンスを向上させることができるということになります。そこで、条件付きリクエストを正しく使用することが重要になります。再バリデーション、つまりデータの節約は、`ETag`ヘッダーまたは`Last-Modified`ヘッダーのどちらかを使用することで行うことができます。

`Last-Modified`レスポンスヘッダーは`If-Modified-Since`リクエストヘッダーと一緒に動作し、リクエストされたファイルに何らかの変更がなされたかどうかをブラウザに知らせます。

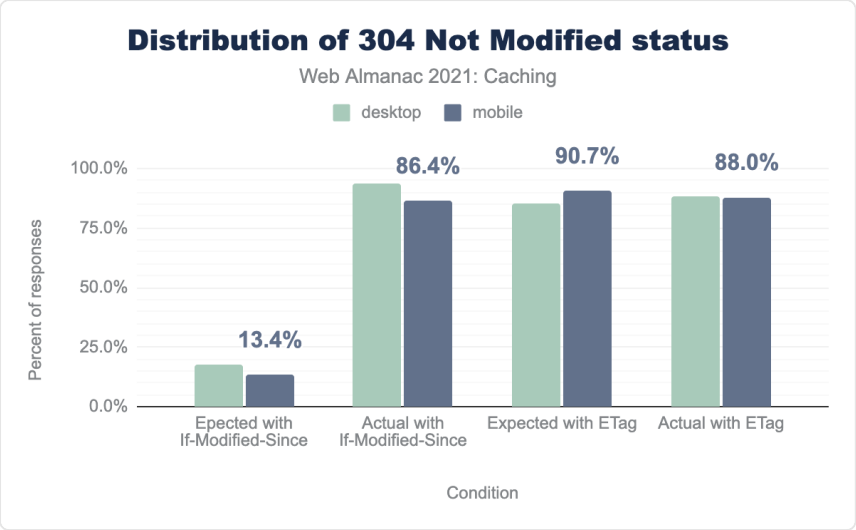


図23.7. キャッシュ戦略別のHTTP 304応答率。

2020年から2021年にかけて、`If-Modified-Since`の304レスポンスの分布が、7.7%増加したことがわかりました。これは、コミュニティがこれらのデータ節約を活用していることを示しています。

日付文字列の有効性

タイムスタンプを表すために使用される3つの主要なHTTPヘッダー、`Date`、`Last-Modified`、`Expires`はすべて日付の書式を持つ文字列を使用します。HTTPレスポンスヘッダーの`Date`はほとんどの場合、ウェブサーバによって自動的に生成されるため、無効な値は非常に稀です。しかし、日付が正しく設定されていない場合、それが提供されるレスポンスのキャッシュに影響を与える可能性があります。

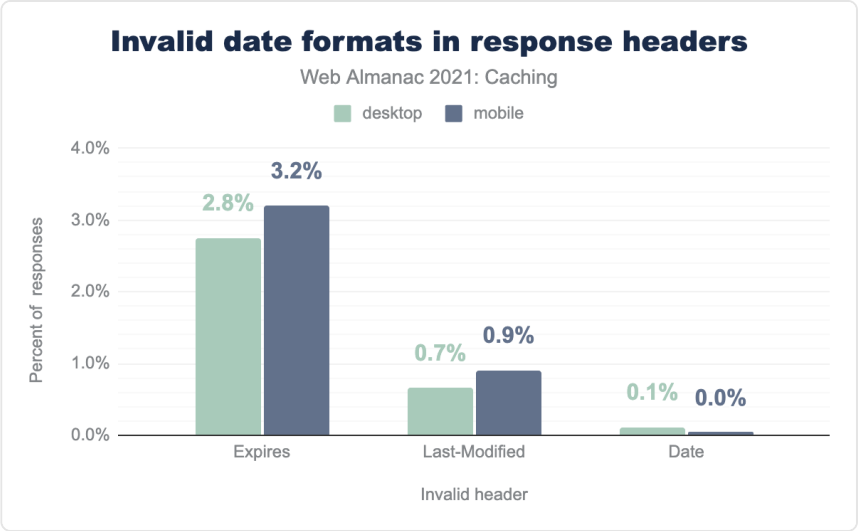


図23.8. 日付の形式が無効な回答の割合。

2020年から2021年にかけて無効な`Date`を使用する割合は0.5%改善しましたが、`Last-Modified`と`Expires`については悪化しており、キャッシュ時の日付設定に関係が、あることがわかります。

このことから、日付ベースのヘッダーの自動化には、さらなる注意が、必要であることがわかる。

`Vary`

リソースのキャッシュに不可欠なステップは、そのリソースが以前にキャッシュされていたかどうかを理解することです。ブラウザは通常、キャッシュキーとしてURLを使用します。同時に、同じURLに対するリクエストでも`Accept-Encoding`が異なるとレスポンスが異なるため、不正にキャッシュされる可能性があります。そのため、`Vary`ヘッダーを使用して、1つ以上のヘッダーの値をキャッシュキーに追加するようにブラウザに指示します。

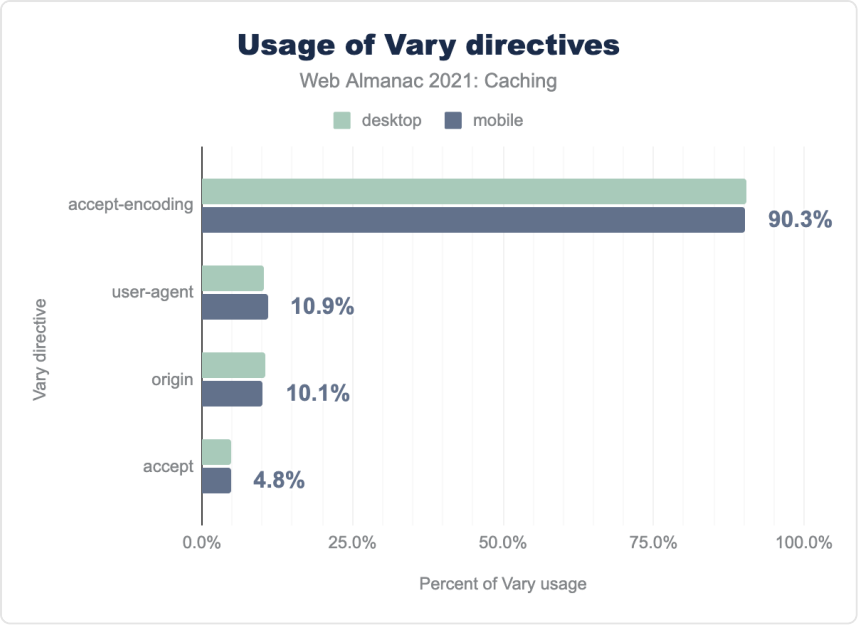


図23.9. `Vary` *ディレクティブの使用法。*

もっとも普及している `Vary` ヘッダーは `Accept-Encoding` で90.3%の使用率、次いで `User-Agent` で10.9%、 `Origin` で10.1%、そして `Accept` で4.8%の使用率です。

2020年から `Accept-Encoding` の使用率が1.5%減少していることがわかりました。

46.3%

図23.10. `Vary` *ヘッダーを設定したモバイル用レスポンスのパーセンテージ。*

監査した総リクエストのうち、 `Vary` ヘッダーを使用しているのは46.25%に過ぎませんが、2020年と比較すると、全体で2.85%増加していることを指摘することが重要です。

83.4%

図23.11. `Vary` *ヘッダーを持つモバイルレスポンスのうち、* `Cache-Control` *も設定されているものの割合です。*

`Vary` ヘッダーを使用するリクエストのうち、83.4%は `Cache-control` も使用しています。これは、2020年から2.1%改善されたことを示しています。

キャッシュ可能なレスポンスにCookieを設定する

2020年のキャッシュの章では、キャッシュ可能なレスポンスで `set-cookie` を使用することに注意するよう念を押しました。なぜなら、レスポンスのわずか4.9%が `private` ディレクティブを使用しており、ユーザーの個人データがCDN経由で誤って別のユーザーに提供される危険性があるためです。

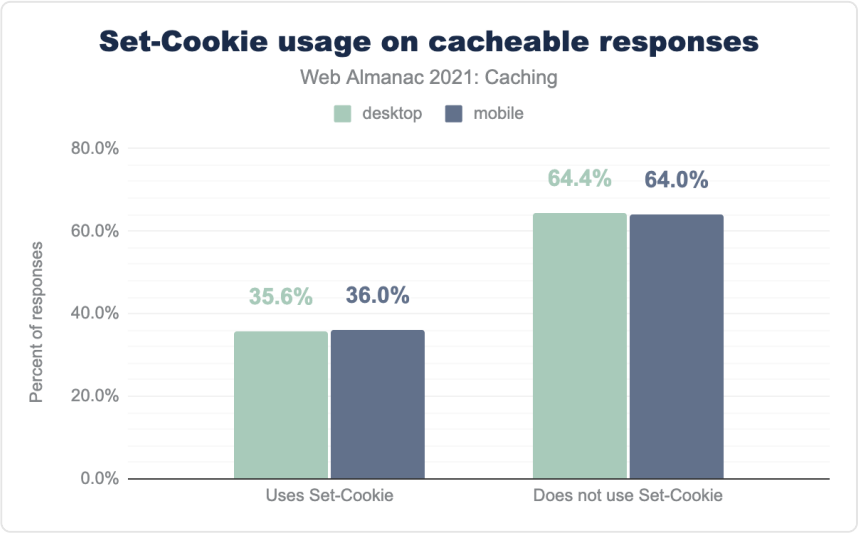


図23.12. `Set-Cookie` を使用するキャッシュ可能なレスポンスのパーセンテージ。

2021年には、 `set-cookie` とキャッシュの共存に関する認識が高まっていることがわかります。 `set-cookie` でprivateディレクティブを使用しているウェブページはまだ5%しかありませんが、キャッシュ可能な `set-cookie` レスポンスの総数は4.41%減少しています。

どのようなコンテンツをキャッシュしているのか？

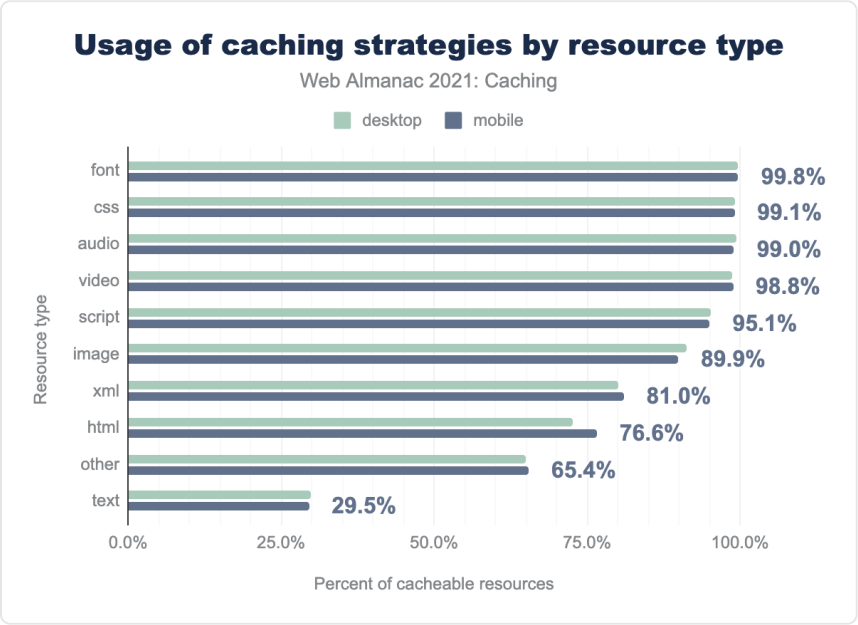


図23.13. リソースタイプ別にキャッシュ戦略を使用したリクエストのパーセンテージ。

フォント、CSS、オーディオファイルは99%以上キャッシュ可能で、現在ほぼ100%のページでフォントがキャッシュされています。これは、静的なファイルであるため、キャッシュに適しているためと思われます。

しかし、もっともよく使われるリソースの中には、動的な性質のためか、キャッシュ不可能なものがあります。とくに、HTMLは23.4% ともっとも高い割合でキャッシュ不可能なリソースがあり、画像は10.1% でそれに続いています。

2020年と2021年のモバイルデータを比較すると、キャッシュ可能なHTMLが5.1%増加していることがわかります。これは、サーバーサイドレンダリングアプリケーションによって生成されたようなHTMLページをキャッシュするために、CDNをより良く利用する方向に進んでいる可能性を物語っています。ページは通常、特定のウェブページのコンテンツが頻繁に変更されない場合、SSRによって生成されます。このURLは、数週間あるいは数ヶ月間、同じHTMLを提供する可能性があり、そのコンテンツは高度にキャッシュ可能です。

タイプ	デスクトップ	モバイル
テキスト	0.2	0.2
XML	1	1
その他	1	1
動画	4	8
HTML	3	14
オーディオ	0.2	30
CSS	30	30
画像	30	30
スクリプト	30	30
フォント	365	365

図23.14. TTLの中央値（単位：日）

すべてのリソースタイプで*Time To Live*（TTL）の中央値を見てみると全体で同じような割合のキャッシュをしていても、モバイルでは、とくにHTML、オーディオ、ビデオでかなり長いキャッシュが、存在することがわかります。

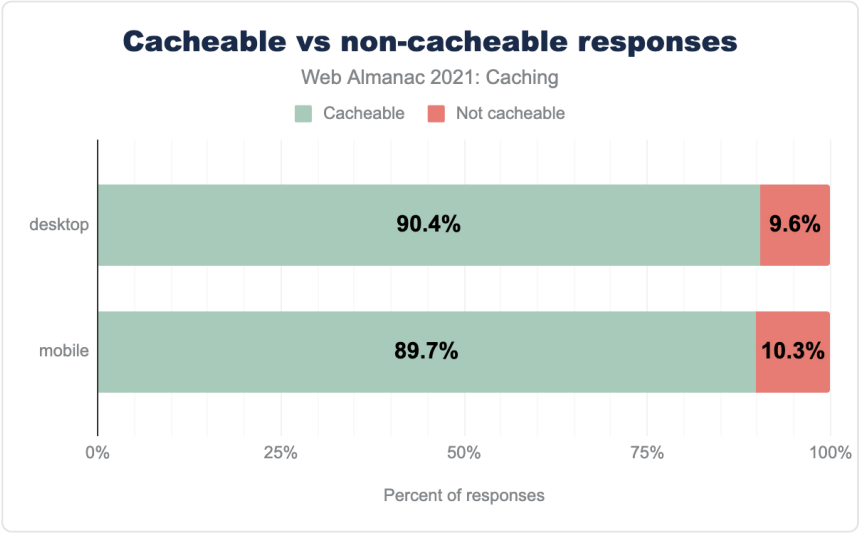


図23.15. キャッシュ可能なレスポンスとキャッシュ不可能なレスポンスの割合。

とはいえ、モバイル体験のための最適化を続けていても、キャッシュ可能なデスクトップリソースの潜在的な量は、モバイル用のリソースよりわずかに多いままであることは興味深いことです。

キャッシュTTLとリソースエイジの比較は？

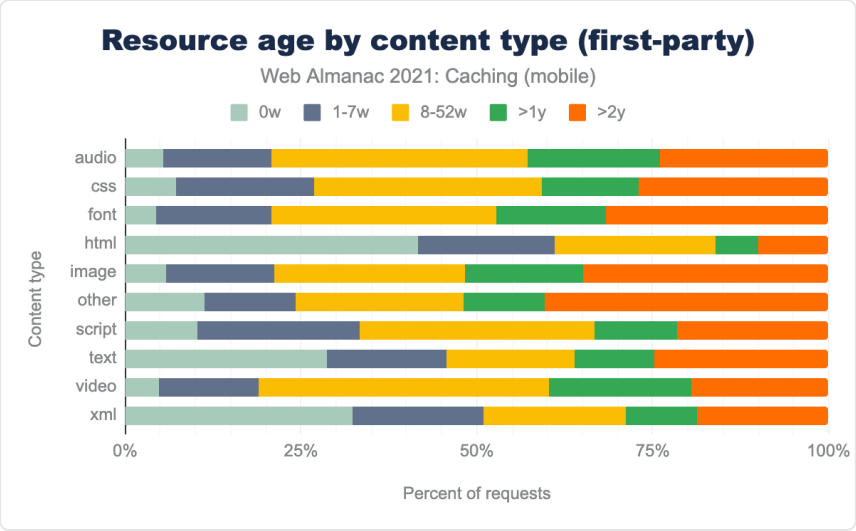


図23.16. コンテンツタイプ別のファーストパーティリソースエイジの分布（モバイルのみ）。

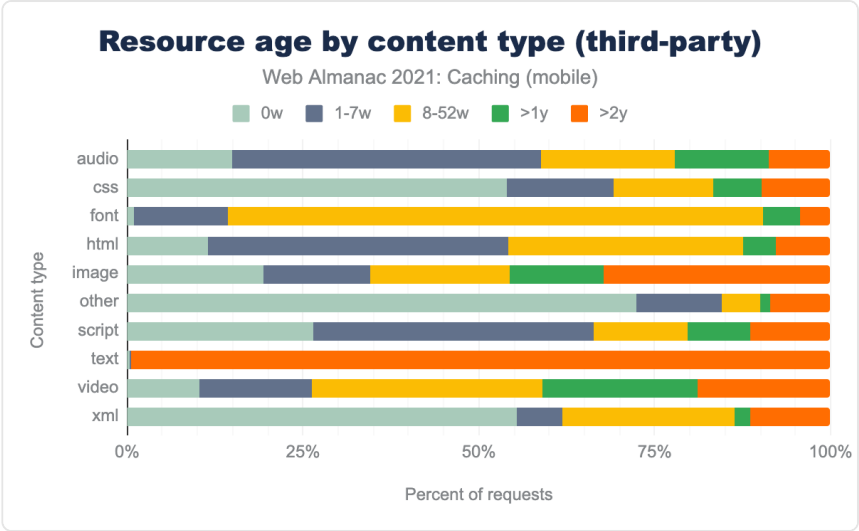


図23.17. コンテンツタイプ別のサードパーティリソースエイジの分布（モバイルのみ）。

画像と動画は、ファースト・ソースでも、サード・パーティ・ソースでも、同じ平均年齢を維持していることがわかる。画像は一貫してリソース年齢が2年であるのに対し、動画のリソースはほとんどが8〜52週であった。

他のコンテンツタイプに分けると、サードパーティーのフォントは、8〜52週間の間にもっとも多くキャッシュされ、72.4%であることがわかりました。しかし、ファーストパーティーの場合、リソースの年齢層は8〜52週と2年以上に均等に分かれており、かなり大きなばらつきがあります。オーディオとスクリプトについても同様の結果が出ており、ファーストパーティでは8〜52週、サードパーティでは1〜7週が大半を占めています。

オーディオは、ファーストパーティーとサードパーティーの両方で、もっとも高度にキャッシュされたリソースでした。しかし、リソース年齢は、ファーストパーティ（平均8〜52週間）とサードパーティ（わずか1〜7週間）で大きく異なっていました。ファーストパーティーのオーディオリソースは更新頻度が、低い傾向があるため（なぜ？）、サードパーティーは、より新鮮なリソースを提供することでキャッシュの機会を利用している可能性があります。

キャッシュされたファーストパーティーのCSSの最大グループ（32.2%）は8〜52週間の傾向があり、サードパーティーの最大グループは1週間未満で、その期間にキャッシュされたリソースは51.8%でした。

もっとも、HTMLは1週間未満で42.7%、サードパーティは1〜7週間で43.1%と、ファーストパーティのグループがもっとも多くサービスを提供しています。

このデータを検討した上での考察。

- ファーストパーティではHTML、サードパーティではCSSがもっとも新鮮なコンテンツです。
- ファースト、サードパーティともに、もっとも陳腐化したコンテンツは画像です。

このデータからファーストパーティは、JSやCSSファイルへのリンクを含むHTMLコンテンツの更新を優先し、ブラウザ拡張機能のようにCSSやスクリプトを主体とするサードパーティは、CSSを最新の状態に保つことを優先していることがわかります。ファーストパーティとサードパーティの違いを考えると、サードパーティにとってコンテンツの配信方法は実際のコンテンツよりも重要であり、そのためコンテンツの表示と最適化が、より重要であることが分かります。

コンテンツ年齢と比較してキャッシュTTLが短すぎるとされたモバイルリソースは、2020年以降に改善が見られました。このデータは、コミュニティが適切な相対キャッシュについて理解を深めていることを示唆するものであり、エキサイティングなものです。

54%

図23.18. 54%のモバイルリソースはTTLより古い

キャッシュTTLが長すぎると、古くなったコンテンツが、提供される可能性がありますが、短すぎるとエンドユーザーにとって何のメリットもありません。キャッシュTTLとコンテンツエイジの関係は、2020年の60.2%から2021年には54.3%となり、この差は徐々に縮まってきています。コンテンツエイジ（ページのHTMLやCSSなどの改修頻度）に対する気配りができればできるほど、より正確にキャッシュの上限を設定することができるようになるのです。

開発者はキャッシュ期間をコンテンツ年齢により正確に設定できるようになってきており、その結果、より責任ある、つまりより効果的なキャッシュを実現できるようになっています。

クライアント	ファーストパーティ	サードパーティ	全体
デスクトップ	59.5%	46.2%	54.3%
モバイル	60.1%	44.7%	54.3%

図23.19. 短いTTLを持つリクエストの割合。

ファーストパーティプロバイダーとサードパーティプロバイダーに分けると、もっとも改善されたのはサードパーティで、13.2%の改善が見られました。世界中の企業が、開発者向けにキャッシュを最適化する製品を開発していることは、非常に心強いことです。開発者コミュニティのパフォーマンス向上に対する関心の高まりが、サードパーティによるキャッシュ戦略の最適化を促し、さらにはその動機付けになった可能性があります。

しかし、今後数年間、ファーストパーティがどのように効果的な改善をしていくかという課題は残されています。

キャッシュの機会を特定する

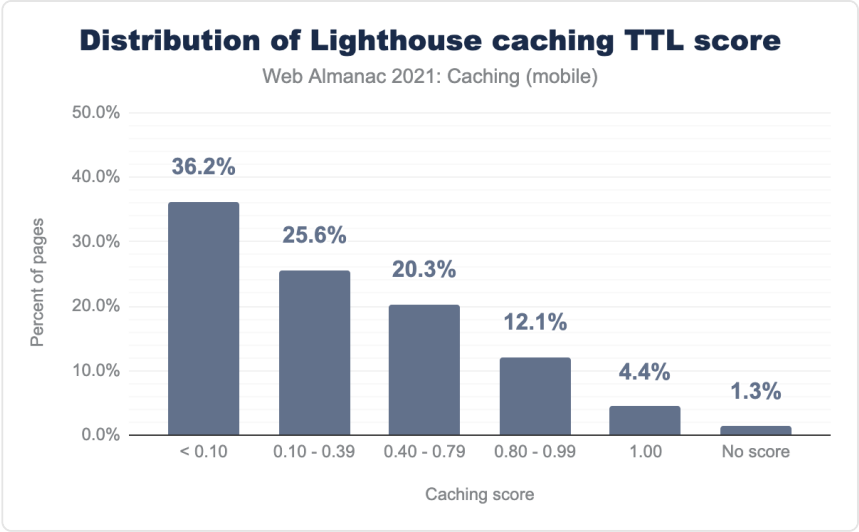


*図23.20. Lighthouseキャッシュの*TTL*スコアの分布。*

Lighthouseのキャッシング*TTL*スコアに基づき、100点満点でランクインしたページが2020年の3.3%から2021年には4.4%に増加するという改善が見られました。

このスコアは、ページがキャッシュポリシーの追加的な改善によって恩恵を受けることができるかどうかを反映しています。31% のページが50パーセンタイルのスコアを上回ったことは喜ばしいことですが、25パーセンタイル以下の52% のページには大きな改善の可能性があります。

このことから、Webページにはある程度のキャッシュ機能が備わっていても、そのポリシーの使い方が古く、自社製品の最新状態に最適化されていないと考えざるを得ません。

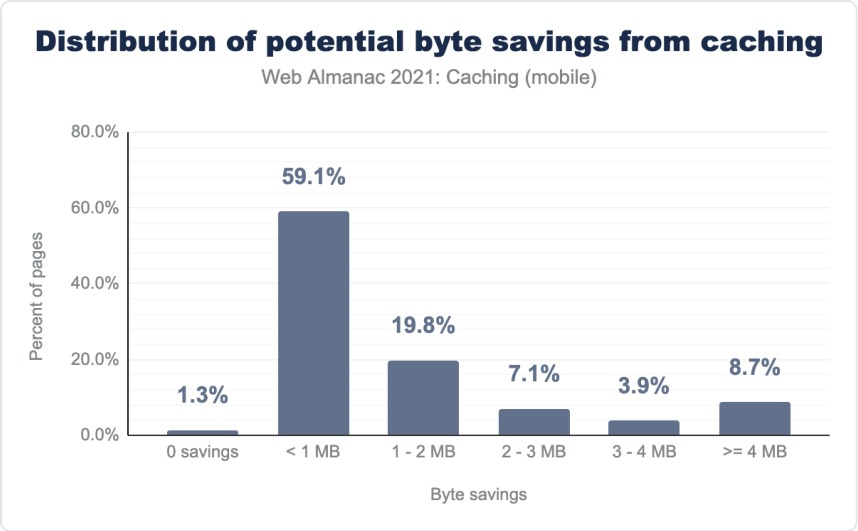


図23.21. キャッシュによる潜在的なバイト数削減の分布。

2020年から2021年にかけてのLighthouseの*wasted bytes*監査に基づくと、繰り返し閲覧される監査対象の全ページにおいて、ムダなバイトが3.28%改善されました。これは、1MBをムダにするページの割合を42.8%から39.5%に下げ、有料のインターネットデータプランを持つ海外のユーザーにとって、よりコストの低い製品を作るというコミュニティのかなりの傾向を示しています。

現在、監査されたページのうち、ムダなバイトが0である割合は1.34%とまだ比較的低いです。今後数年間は、コミュニティがウェブパフォーマンスの最適化に注力し続けるため、この割合が増加することを期待しています。

結論

故・偉大なPhil Karlton[962]は、「コンピューター サイエンスには難しいことが2つしかない」という有名な言葉を残しました。正直なところ、私はキャッシュがなぜそんなに難しいのか、いつも不思議に思っています。私の考えでは、キャッシュをうまく行うには、2つの重要な要素が必要です：シンプルに保つこと、そして潜在的なエッジケースをすべて理解することです。

残念ながらキャッシュを賢くしようとしすぎると、間違ったものをキャッシュしてしまったり、もっと悪いことに、過剰にキャッシュしてしまったりすることがあるのです。同じよう

962. https://www.karlton.org/karlton/

なことですが、すべてのエッジケースを理解するには、広範な調査とテスト、そしてゆっくりとした漸進的な改良が必要です。それでも、古いブラウザがあなたをバスの下に投げ出さないことを願わなければなりません。しかし私たちがいまだに優れたキャッシュ戦略を追い求める理由は、ラウンドトリップリクエストの大幅な削減、サーバーの高い節約、ユーザーから求められるデータの削減、そして最終的にはより良いユーザー体験という、究極の報酬が非常に大きいからです。

どのような場合でも、キャッシュの最適な使用方法のプレイブックを用意するようにしてください。

- 開発サイクルの初期段階、および製品出荷後のキャッシュ作業を優先させる。
- 主要なエッジケースを再現するエンドツーエンドのテストを書く
- 定期的なサイト監査と、古くなったり欠落している可能性のあるキャッシュルールの更新

最終的には私たちが仲間を指導し、理解しやすい良いドキュメントを書くことによって知識を広めることができればキャッシングは、それほど複雑なものではなくなります。キャッシングは、一部の人だけがマスターすればいいというものではありません。私たちの目標は、会社全体の共通認識として定着させることです。なぜなら、最終的に私たちが本当に注力したいのは、ユーザーにとって簡単で摩擦のない体験を構築することだからです。

著者



Leonardo Zizzamia

@Zizzamia Zizzamia

Leonardo は、Coinbase[963] のソフトウェアエンジニアスタッフで、製品エンジニアが世界でもっとも高い品質のアプリケーションを大規模に出荷できるようにするためのイニシアチブをリードしています。彼はNGRome Conference[964]をキュレーションしている。また、レオはPerfume.js[965] ライブラリのメンテナンスも行っており、企業がパフォーマンス分析を通じてロードマップの優先順位付けやより良いビジネス決定を行えるよう支援しています。

963. *https://www.coinbase.com/*
964. *https://ngrome.io*
965. *https://github.com/Zizzamia/perfume.js*



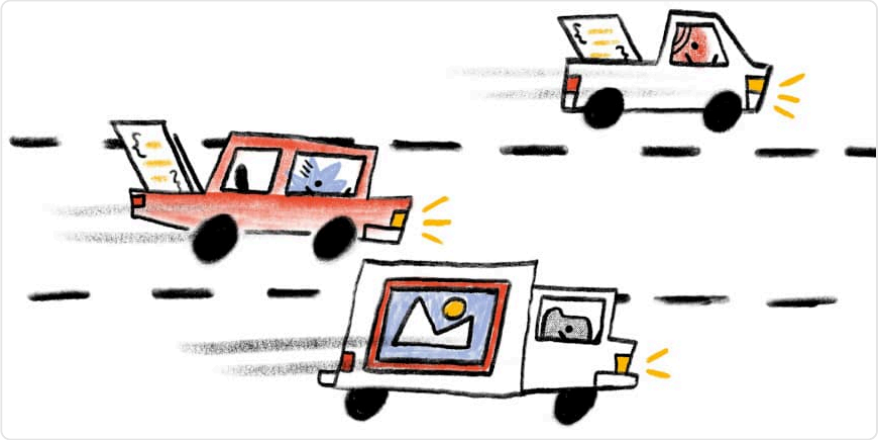
Jessica Nicolet

@jessica_nicolet jessnicolet https://www.jessicanicolet.com/

Jessicaはオペラ歌手としてキャリアをスタートし、過去10年間はクラシック音楽業界に身を置いてきました。2020年初頭、パンデミックにより、彼女は技術、特にウェブ開発の分野で新しいキャリアをスタートさせることを決意しました。彼女はもともと、ステージ上でもオフでも、文章を書いたり物語を語ったりするのが好きで、この新しい分野に移行した体験を記録した3つの記事のシリーズ[966]をMediumで発表しています。現在、テクニカルライティングの正社員を募集しています。

966. *https://jessicanicolet.medium.com/*

部 IV 章 24
HTTP



***Dominic Lovell** によって書かれた。*
Barry Pollard と Robin Marx によってレビュ。
Barry Pollard による分析。
Shaina Hantsis 編集。
Sakae Kotaro によって翻訳された。

序章

HTTPプロトコルは、Webを構成する重要な要素の1つです。HTTP自体は、1997年にHTTP/1.1が導入されて以来、20年近く変更されることはありませんでした。HTTPの実装方法に大きな設計変更があったのは、HTTP/2の導入が行われた2015年までででした。HTTP/2は、主にプロトコルのトランスポートレベルで変更を導入するよう設計されていました。これらのプロトコルの変更は、その動作が重要である一方で、バージョン間の後方互換性をまだ可能にしていました。

今年もまた、HTTP/2を詳しく見て、その主要な機能のいくつかを議論します。そして、HTTP/2の利点のいくつかと、なぜそれがウェブパフォーマンスコミュニティ全体で大きく採用されているのかを見ていきます。HTTP/2は接続制限、より優れたヘッダー圧縮、より優れたペイロードのカプセル化を可能にするバイナリサポートなどHTTPの多くの問題の解決を目指したが、提示されたすべての機能がその設計で成功したわけではなかった。

HTTP/2が野放しになって数年、HTTP/2の意図のいくつかはまだ実現されていない。たとえば、昨年、私たちは、HTTP/2プッシュに別れを告げるかどうかという問いを提起しました。今年は、2021年のデータを見ることで、より確信を持ってこの問いに答えることを目指しています。これらの欠点が明らかになるにつれて、HTTPの次のイテレーションであるHTTP/3で対処されたり省略されたりしています。

過去1年間のHTTP/3のサポートの増加は、ウェブでのHTTP/3の採用について内省することを可能にしました。この章では、HTTP/3のコア機能のいくつかと、それぞれの利点を詳しく見ていきます。また、HTTP/3の進化をサポートしている主要なベンダーと、HTTP/3の継続的な批評のいくつかを検証します。

Web AlmanacがHTTPの章を通して回答しようとするデータポイントには、HTTPバージョン間の採用状況、主要ソフトウェアベンダーやCDN企業のサポート、ファーストパーティとサードパーティのこの分布が採用にどのように影響するかが含まれます。また、接続、サーバープッシュ、レスポンスデータサイズなどのHTTP属性に関する指標を含め、ウェブ上のトップランカーサイトにおける使用状況も見ています。

これらのデータは、2021年のウェブ上でのHTTPの使用状況や、主要なバージョン間でのプロトコルの進化についてのスナップショットを提供します。そして、今後数年間の主要な機能の採用について洞察を提供します。

HTTPの進化

インターネット技術特別調査委員会 (IETF)[967] が HTTP/2[968]を紹介して以来6年が経ちますが、そもそもHTTP/2に至った経緯について理解する価値があるでしょう。30年前（1991年）、私たちははじめてHTTP 0.9を紹介されました。HTTPは、機能が制限されていた0.9からは大きく進歩しました。0.9は、GETメソッドのみをサポートし、ヘッダーやステータスコードをサポートしない、1行のプロトコル転送に使用されていました。回答はハイパーテキストで提供されるだけだった。5年後、これはHTTP/1.0で強化されました。1.0バージョンには、レスポンスヘッダー、ステータスコード、 `GET` 、 `HEAD` 、 `POST` メソッドなど、現在私たちが知っているほとんどのプロトコルが含まれています。

1.0で対処できなかった問題は、応答を受け取った後すぐに接続が終了してしまうことでした。つまり、リクエストごとに新しいコネクションを開き、TCPハンドシェイクを行い、データを受け取った後にコネクションを閉じる必要があったのです。この大きな非効率性から、わずか1年後の1997年にHTTP/1.1が導入され、一度開いたコネクションを再利用できる持続的なコネクションが可能になりました。このバージョンは、2015年まで何の変更も加えられることなく、18年間その目的を果たしました。この間、Googleは SPDY[969] という、HTTPメッセージの送信方法を完全に作り直す実験を行いました。これは最終的にHTTP/2で

967. https://www.ietf.org/
968. https://datatracker.ietf.org/doc/html/rfc7540
969. https://ja.wikipedia.org/wiki/SPDY

正式化されました。

HTTP/2は、Web開発者がパフォーマンスの向上を図る際に直面していた問題の多くに対処することを目的としています。HTTP/1.1の非効率性を回避するためには、ドメインシャード、アセットスプリング、ファイルの連結といった複雑な処理が必要でした。リソースの多重化、優先順位付け、ヘッダー圧縮を導入することで、HTTP/2はプロトコルレベルでネットワークの最適化を提供するように設計されています。既知のパフォーマンス問題に対処するだけでなく、HTTP/2は *HTTP/2 プッシュ* のような機能によって、新しいパフォーマンスの最適化の可能性を導入しました。

HTTP/2の採用

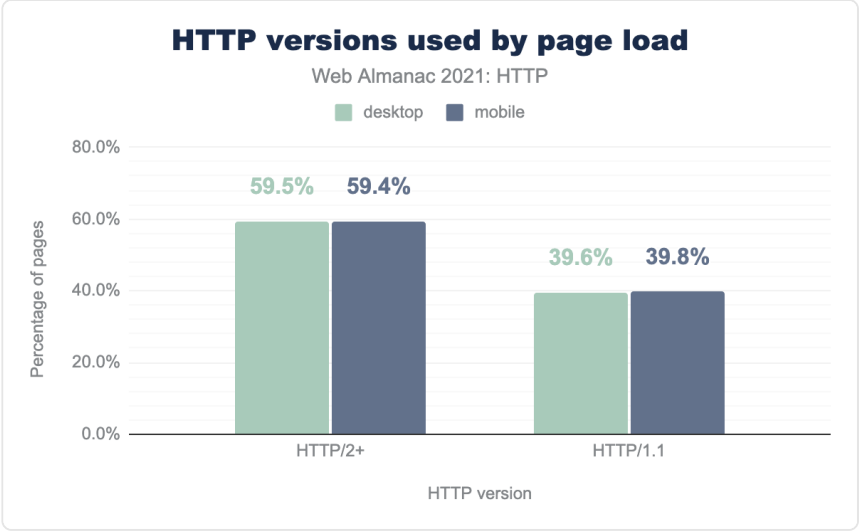


図24.1. ページロードで使用されるHTTPのバージョン。

HTTPバージョン0.9から30年の間に、プロトコルの採用状況に変化がありました。600万以上のウェブページを分析した結果、HTTP Archiveは最初のページリクエストにHTTP 0.9を使用した例は1つだけで、1.0を使用しているページはわずか数千件であることを発見しました。しかし、ほぼ40%のページがバージョン1.1を使用しており、残りの60%はHTTP/2以上を使用しています。このように、HTTP/2の採用率は、2020年に同じ分析を行ったときよりも10%増加しています。

注: 後述するように、HTTP/3の動作方法と、毎回新鮮なインスタンスでクロールが動作する方法のため、HTTP/3は最初のページリクエスト、あるいはその後のリクエストに使用される可能性は低いです。そのため、本章ではいくつかの統計情報を「HTTP/2+」として報告し、

現実世界でHTTP/2またはHTTP/3が使用される可能性があることを示す。私たちは、この章の後半で、（私たちのクロールで使用されていないとしても）実際にどの程度HTTP/3がサポートされているのかを調査します。

リクエストによる採用

最初のページへのリクエストは、他の多くのリクエストによって補完されます。多くの場合、サードパーティによって提供され、異なる、あるいはより優れたプロトコルサポートを持っているかもしれません。このため、最初のページだけでなく、リクエストのレベルで見ると、使用率が非常に高くなることが過去数年間わかっており、今年もその傾向が見られます。

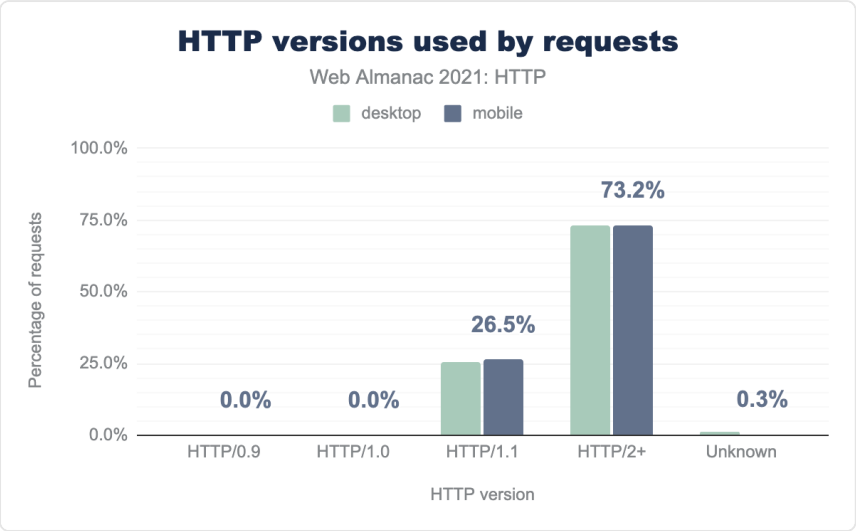


図24.2. リクエストで使用されたHTTPのバージョン。

2021年、HTTP Archiveのデータは、HTTP/0.9とHTTP/1.0が事実上すべて消滅したことを示唆しています。0.9には数百のリクエストが存在しましたが、データセット全体で集計するとゼロに切り捨てられます。HTTP/1.0は数千のリクエストがありますが、これも全体の0.02%に過ぎません。

25%

図24.3. 昨年度のHTTP/1.1リクエストの減少。

興味深いことに、4分の1以上のリクエストは依然としてHTTP/1.1経由で提供されています。2020年と比較すると、これは25%減少していることになります。70%以上のリクエストがHTTP/2以上で処理されており、HTTP/2とHTTP/3がWebの支配的なプロトコルバージョンであることを示唆しています。

ページごとに使用されているプロトコルを見ると、やはりHTTP/2以上の優位性がプロットされます。

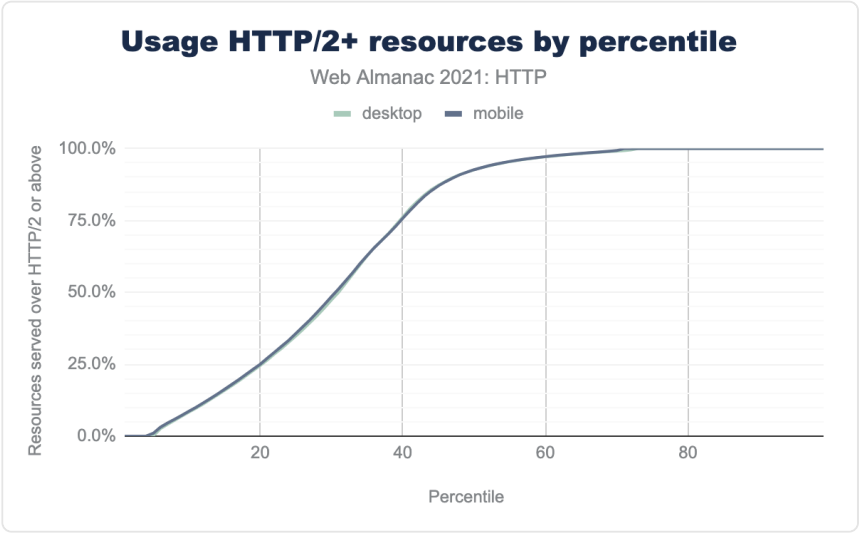


図24.4. HTTP/2+リソースのパーセンタイル別利用状況。

50パーセンタイル以上のページでは、リソースの92%以上がHTTP/2+ で提供されています。そして70パーセンタイル以上では、100% のサイトのリソースがHTTP/2以上で読み込まれています。別の言い方をすれば、30%のサイトがHTTP/1.1のリソースをまったく使用していないことになります。

第三者による採用

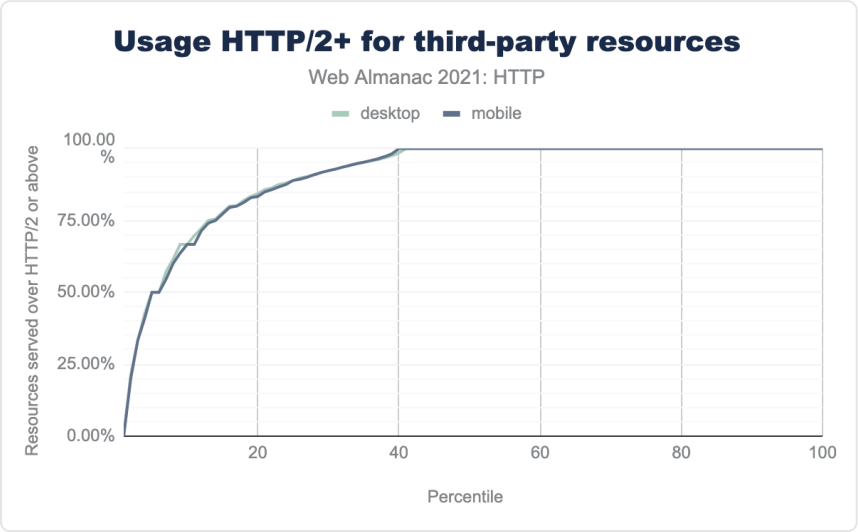


図24.5. サードパーティのリソースにHTTP/2+を使用する。

サードパーティコンテンツによるHTTP/2の採用は非常に偏っており、サードパーティのリクエストの40パーセンタイルを超えると、100%のトラフィックがHTTP/2で提供されるようになります。実際、10パーセンタイルでも、66%以上のリクエストがHTTP/2を活用しています。このことから、導入の大半はまだサードパーティコンテンツとCDNを活用したドメインが提供するコンテンツに影響されていることがわかります。

サーバーでの採用

caniuse.com[970] によると、全世界で97%のブラウザがHTTP/2をサポートしているとのことです。HTTPSは、HTTP/2対応のためにブラウザが要求しているもので、従来はブロックされていた可能性があります。しかし、デスクトップで93%、モバイルでは91%のサイトがHTTPSに対応しています[971]。これは、2020年に昨年より5%増加し、2019年から2020年にかけては前年比6%増加しました。HTTPSの導入はもはやブロッカーではない。

ブラウザ間での高い普及率とHTTPSの高い普及率から、HTTP/2のさらなる普及を制限する要因は、まだサーバーの実装に大きく左右されることを理解することが重要です。HTTP/2の利用が急速に増加しているにもかかわらず、ウェブサーバ別に分けた場合、採用の数値はより断片的なストーリーを示しています。

970. *https://caniuse.com/http2*
971. *https://httparchive.org/reports/state-of-the-web#pctHttps*

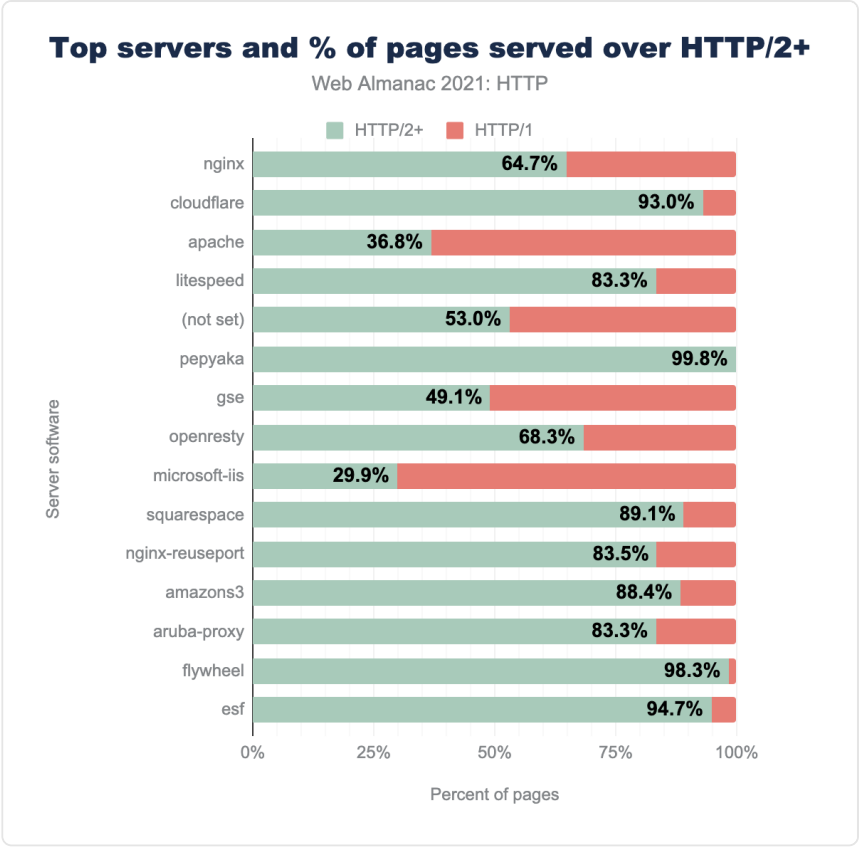


図24.6. 上位サーバーとHTTP/2+で提供されたページの割合。

Apache HTTPサーバーを使用しているサイトでは、HTTP/2にアップグレードしている可能性は低く、新しいプロトコルを活用しているApacheサーバーはわずか3分の1に過ぎません。Nginxは、全サーバーの3分の2がHTTP/2にアップグレードしており、より有望な数字を示しています。CDNとクラウドサーバーは、Cloudfront、Cloudflare、Netlify、S3、Flywheel、Vercelなどのサービスから、いずれも高い採用率を誇っています。CaddyやIstio-Envoyのようなニッチなサーバ実装も、高い採用率を示しています。一方、IIS、Gunicorn、Passenger、Lighthttpd、Apache Traffic Server（ATS）などの実装はいずれも採用率が低く、Scuriも採用率はほぼゼロと報告されています。

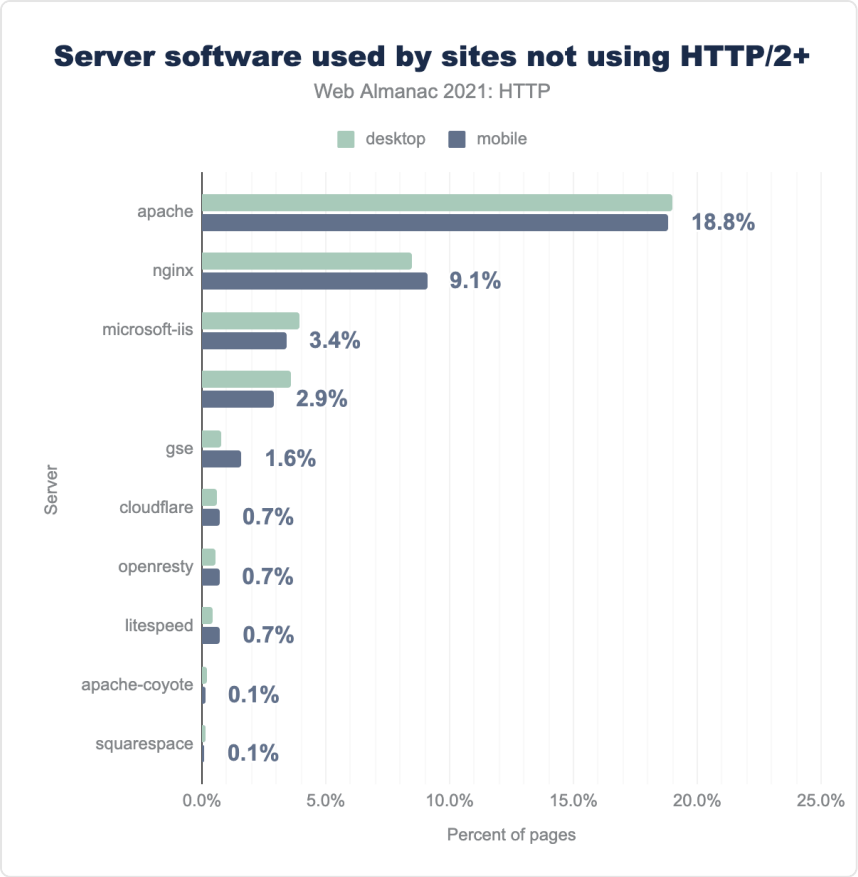


図24.7. HTTP/2+を使用していないサイトが使用するサーバーソフト。

実際、HTTP/1.1レスポンスを報告している全サーバーのうち、もっとも多いのはApacheサーバーで20%でした。Apacheはウェブ上でもっとも普及しているウェブサーバーの1つであるため、古いApacheによってウェブが新しいプロトコルを全面的に採用するのを妨げている可能性があることを示唆しています。

CDNによる採用

CDNは、HTTP/2のような新しいプロトコルの採用を推進する上で極めて重要な役割を果たすことが多く、統計データを見れば、このことが証明されます。

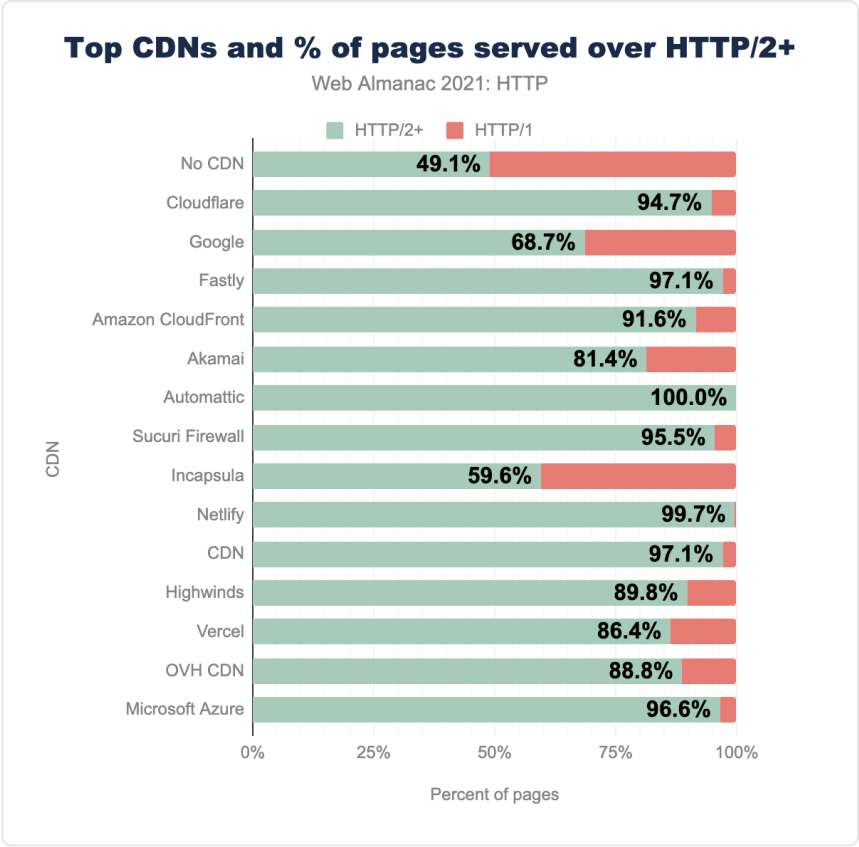


図24.8. 上位CDNとHTTP/2+で提供されるページの割合。

大多数のCDNは、HTTP/2搭載サイトの採用率が70%以上であり、非CDNトラフィックの49.1%よりはるかに高い。Yottaa、WP Compress、jsDeliverなどの一部のCDNは、すべてHTTP/2の採用率が100%となっています!

採用率の高いサービスは、広告ネットワーク、分析、コンテンツプロバイダー、タグマネージャー、ソーシャルメディアサービスなどの周辺サービスが典型的です。これらのサービスにおけるHTTP/2の採用率の高さは、少なくとも50%がHTTP/2を有効にしている5%台以上でも明らかです。中央値では、これらのサービスの95%がHTTP/2を使用することになります。

ランク別採用状況

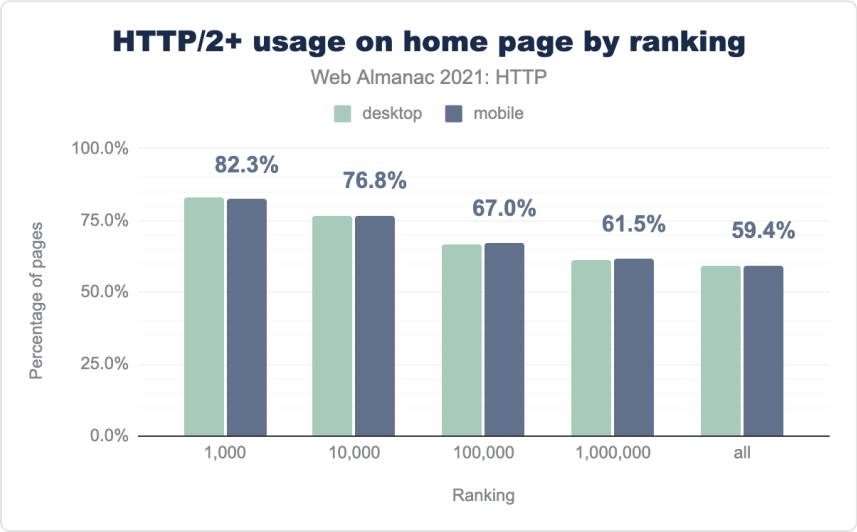


図24.9. ランキング別のトップページでのHTTP/2+の利用状況。

また、HTTP Archiveにおけるサイトのページランクと HTTP/2への対応には直接的な相関があります。トップ1,000に掲載されているサイトの82%がHTTP/2を有効にしています。上位1万サイトでは76%以上、次いで上位10万サイトで66%、そして上位100万サイトは少なくとも60%がHTTP/2を有効にしていることになります。このことは、ランキング上位のサイトが、提供されるセキュリティとパフォーマンスの利点を求めてHTTP/2を有効にしていることを示唆しています。ランキング上位のサイトほど、HTTP/2を有効にしている可能性が高くなります。

HTTP/2をもう少し深く掘り下げる

HTTP/2の主な利点の1つは、テキストベースのプロトコルの代わりにバイナリであることです。ストリームで送信されるリクエストは、1つまたは複数の *フレーム* で構成されることがあります。これは、クライアントとサーバーの間の仕組みを変えます。

メッセージをフレームにチャンキングし、それらのフレームをワイヤ上でインターリーブすることで、1つのTCPコネクションで複数のメッセージを送受信できます。これにより、ドメインハックやその他のHTTP/1.1パフォーマンスの回避策が不要になります。

しかし、このまったく新しいHTTPトラフィックの送信方法は、HTTP/2が以前のバージョンと互換性がないことを意味するため、クライアントとサーバーはそれぞれHTTP/2を話して

いることを知る必要があります。HTTPSは、HTTP/2のデファクトスタンダードとして採用されています。HTTP/2はHTTPSなしでも実装できますが、すべての主要なブラウザベンダーは、HTTP/2がHTTPS上で使用されることを保証しています。また、HTTP/2はALPN[972]を使用しており、最初の接続時にプロトコルを特定できるため、より高速な暗号化接続が可能になります。

プロトコル間の切り替え

HTTPSの使用は、HTTP/1.1とより新しいHTTP/2のどちらを「話す」かを決めるのに役立ちますが、より新しいプロトコルに切り替える方法は他にもあります。HTTP/2のサポートはHTTP/1.1接続で `upgrade` HTTPヘッダーによって告知され、クライアントは101 (Switching Protocols) 応答ステータスコードを使って切り替えを行うことが可能です。HTTP/2からHTTP/3へは、同様の `alt-svc` (Alternative Service) ヘッダーが使用されますが、これについては本章で後ほど説明します。

HTTP Archiveのデータによると、`Upgrade` ヘッダーの使用は、しばしば誤用されたり、正しく設定されなかったりすることがあるようです。この機能は、実際、HTTP/2の次のバージョンから削除[973]される予定です。`Upgrade` ヘッダーを提供しているサイトはごく一部です。もっとも多く報告されているヘッダーは、HTTP/2オプション、またはHTTP/2 over cleartextの詳細を示す `h2,h2c` で、デスクトップサイトの0.09% とモバイルサイトの0.16% がこのヘッダーを報告しています。

同様の割合で `websockets` も `Upgrade` オプションとして提供しているサイトがあり、0.08%となっています。`Upgrade` は、リクエストが行われた既存のHTTP/1.1接続以外の、互換性のない、またはより適切なプロトコルを通知するために使用されるべきものだからです。0.04% のサイトが、この接続がすでにHTTP/2であるにもかかわらず、H2を `Upgrade` オプションとして不正確に報告しています。

972. https://ja.wikipedia.org/wiki/Application-Layer_Protocol_Negotiation
973. https://github.com/httpwg/http2-spec/issues/772

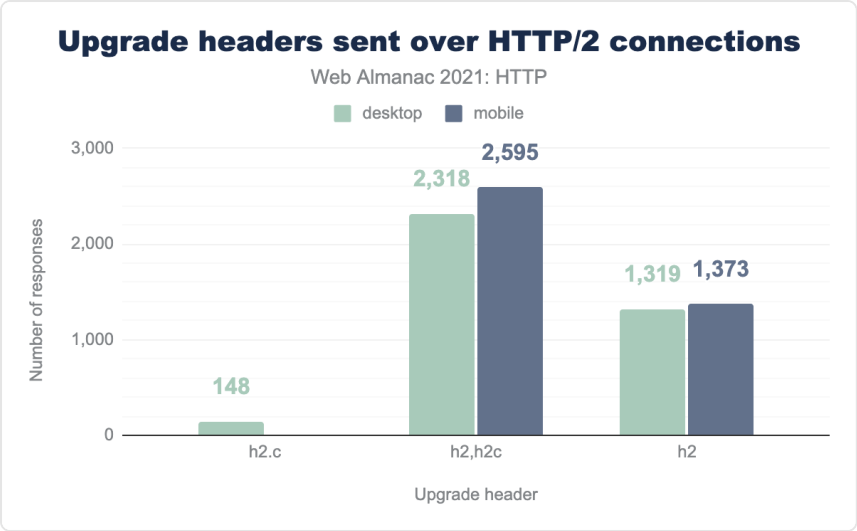


図24.10. HTTP/2コネクションで送信されるヘッダーをアップグレードする。

さらに心配なのは、HTTP/2接続をHTTP/2に「アップグレード」することを提案するサイトの数です。これは明らかなエラーであり、HTTP/2の初期にはブラウザを混乱させるために使用されました。

また、約12万件のモバイルサイトがHTTPで発見されましたが、HTTP/2への`Upgrade`ヘッダーが報告されています。より良い方法は、HTTPからHTTPSへのリダイレクトを発行し、安全な接続で直接HTTP/2を活用することです。

26,000

図24.11. モバイルサイトがHTTP/2をサポートしていないにもかかわらず、サポートしていると主張すること。

また、デスクトップとモバイルでそれぞれ22,000と26,000のWebページがHTTPSでありながらHTTP/2をサポートしていないことが判明しました。同様に、数百のウェブページで、接続自体がすでにHTTP/2であるにもかかわらず、HTTP/2にアップグレードするよう誤ったシグナルが表示されていました。

接続台数

HTTP/2の導入以降、1ページあたりのTCP接続数の中央値は着実に減少しています。

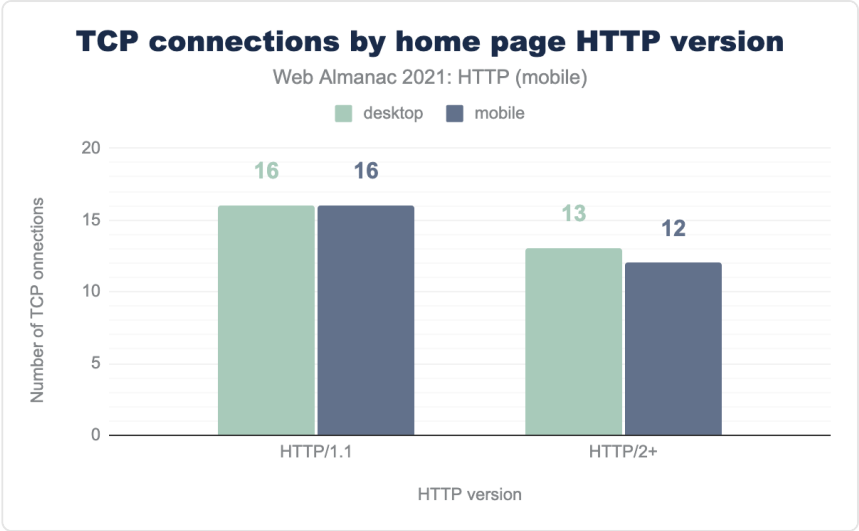


図24.12. ホームページのHTTPバージョン別のTCPコネクション。

本稿執筆時点では、デスクトップ接続数は12か月間で44%減の16接続（中央値）。モバイルは7%減で、接続数の中央値は12です。これは、2020年以降にHTTP/2の採用が急増したため、時間の経過とともに接続数が減少していることを表しています。

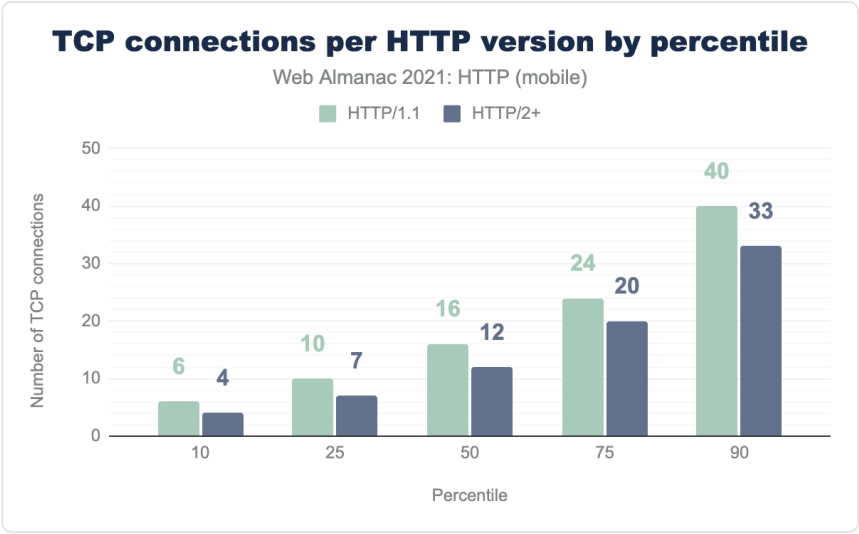


図24.13. HTTPバージョンごとのTCPコネクション数（パーセンタイル別）。

収集したHTTP Archiveのデータによると、HTTP/1.1サイトの中央値は、1ページあたり16コネクションになります。75パーセンタイルでは24コネクションになります。モバイルとデスクトップでは、90パーセンタイルで40と2倍以上になります。一方、HTTP/2サイトでは、中央値で12、75パーセンタイルで21、90パーセンタイルで33の接続が発生します。トップエンドでも、ウェブサイト全体で使用される接続数が21%減少していることになります。

TLSはパフォーマンスに若干のオーバーヘッドを追加し、HTTPS上のHTTP/2の事実上の実装では、使用するTLSのバージョンによってパフォーマンスの考慮が必要であることを意味します。TLS1.3[974] が導入されて以来、TLS false starts[975] など、パフォーマンスへの配慮が追加され、クライアントが最初のTLSラウンドトリップ後に直ちに暗号化データの送信を開始できるような仕組みが導入されています。同様に、TLSハンドシェイクを改善するためにゼロラウンドトリップタイム（0-RTT[976]）を実現しました。TLS 1.2ではTLSハンドシェイクを完了するために2回のラウンドトリップが必要ですが、1.3では1回で済むため、暗号化の待ち時間が半分に短縮されます。

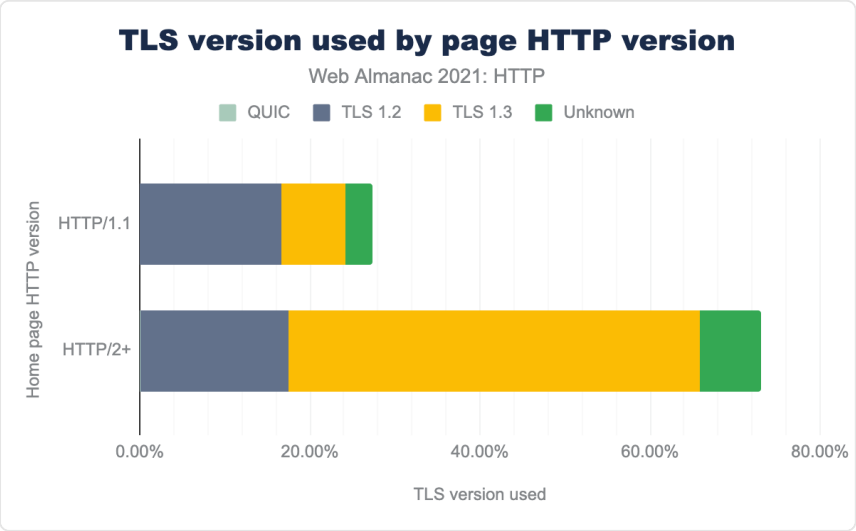


図24.14. ページのHTTPバージョンで使用されるTLSバージョン。

HTTP Archiveのデータによると、デスクトップページの34%がTLS 1.2を、56%がTLS 1.3を使用しており、残りの10%は不明（接続に失敗したHTTPSサイトなど）です。モバイルではやや低く、36%がTLS 1.2、55%がTLS 1.3、9%が不明となっています。大半のサイトがTLS1.3を使用していますが、Webサイトの3分の1はアップグレードすることで、これらのパ

974. https://blogs.windows.com/msedgedev/2016/06/15/building-a-faster-and-more-secure-web-with-tcp-fast-open-tls-false-start-and-tls-1-3/
975. https://blogs.windows.com/msedgedev/2016/06/15/building-a-faster-and-more-secure-web-with-tcp-fast-open-tls-false-start-and-tls-1-3/
976. https://blog.cloudflare.com/introducing-0-rtt/

フォーマンスを向上させることができます。

ヘッダーの削減

HTTP/2で提唱されたもう1つの機能は、ヘッダーの圧縮です。HTTP/1.1は、多くの重複した、または繰り返されるHTTPヘッダーが電線で送信されていることを証明しました。これらのヘッダーは、クッキーを扱うときにとくに大きくなることがあります。このオーバーヘッドを減らすために、HTTP/2はHPACK圧縮形式[977]を利用して、送受信するヘッダーのサイズを小さくしています。クライアントとサーバーの両方が、よく使われるヘッダーや以前に転送したヘッダーのインデックスをルックアップテーブルに保持し、個々の値を前後に送信するのではなく、テーブル内のそれらの値のインデックスを参照できます。これにより、電線で送信されるバイト数を節約できます。

977. *https://datatracker.ietf.org/doc/html/rfc7541*

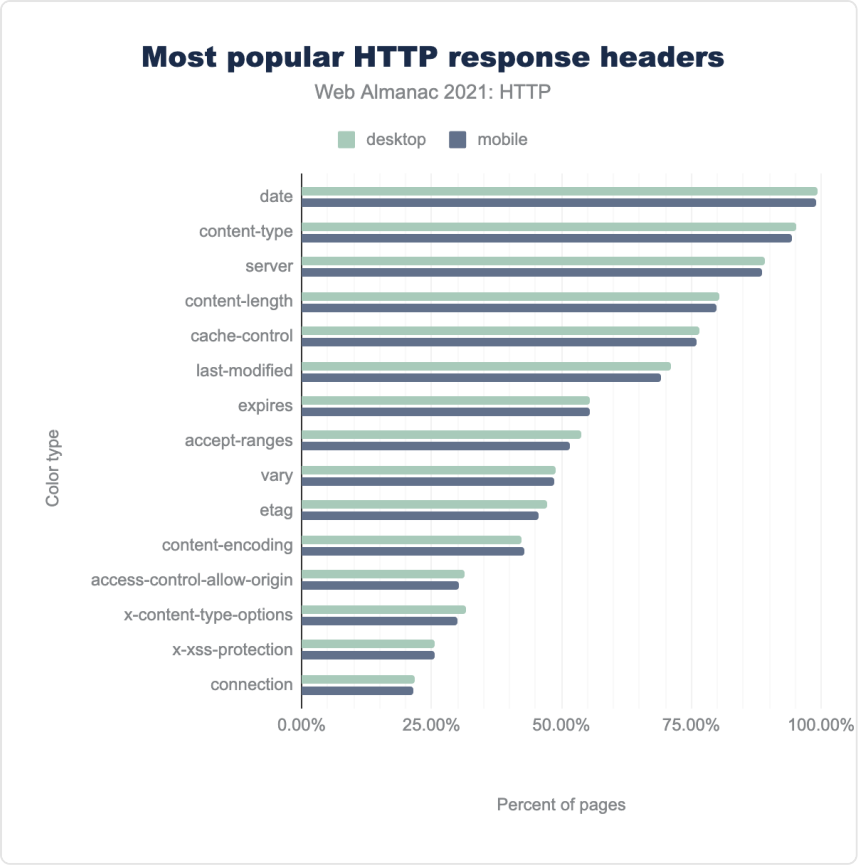


図24.15. もっとも普及しているHTTPレスポンスヘッダー。

受信したレスポンスヘッダーの中で、もっとも多いヘッダーは上位5つです。それぞれ、`date`, `content-type`, `server`, `cache-control`, `content-length` です。もっとも一般的な非標準ヘッダーはCloudflareの `cf-ray` で、Amazonの `x-amz-cf-pop` と `X-amz-cf-id` がそれに続いています。コンテンツ情報 (`length`, `type`, `encoding`) 以外では、キャッシュポリシー (`expires`, `etag`, `last-modified`) やオリジンポリシー (STS, CORS[978]), `expect-ct` レポート証明書の透明性やCSP `report-to` ヘッダーはもっともよく使われるヘッダーとなります。

これらのヘッダーの一部 (たとえば `date` や `content-length`) はリクエストごとに変わるかもしれませんが、大部分はリクエストごとに同じか、限られた数のバリエーションを送信します。同様にリクエストヘッダーも、リクエストごとに同じデータ（長い `user-agent`

978. *https://developer.mozilla.org/docs/Web/HTTP/Headers/Access-Control-Allow-Origin*

ヘッダーなど）を何度も送信することがよくあります。したがって、影響を考慮するためには、ページが行っているリクエストの数に注目する必要があります。

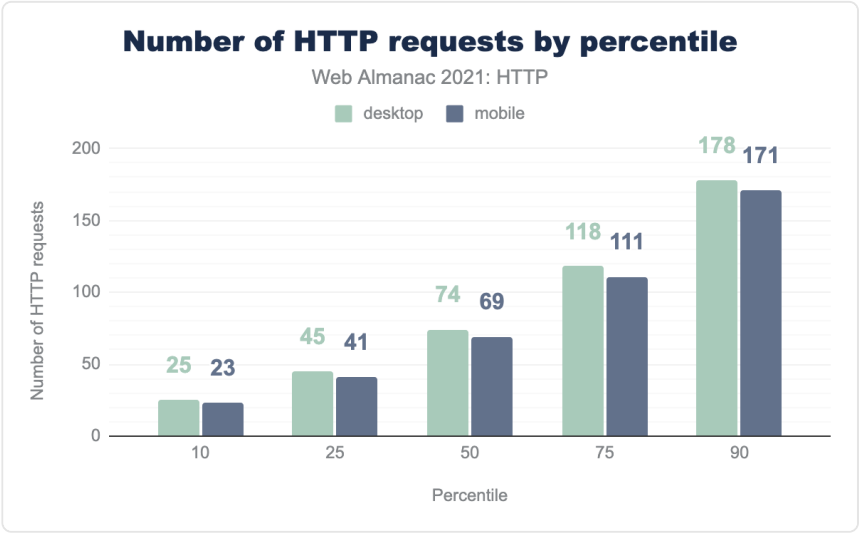


図24.16. パーセンタイル毎のHTTPリクエスト数。

デスクトップ用サイトの中央値は74リクエスト、モバイル用サイトの中央値は69リクエストです。1ページあたりのリクエスト数が数千を超えるサイトも何百とありました。もっとも高かったのは、合計17,923リクエストで、次いで10,224リクエストとなっています。HTTP/2は、以前のリクエストで送信されたヘッダーを圧縮して再利用することで、繰り返されるリクエストの影響を軽減しています。

この分析では、ブラウザのネットワークスタックの奥深くに埋もれているため、ヘッダー圧縮の正確な影響を測定することはできませんが、圧縮されていないヘッダーサイズを見ることで、潜在的なメリットの指標を示すことはできます。

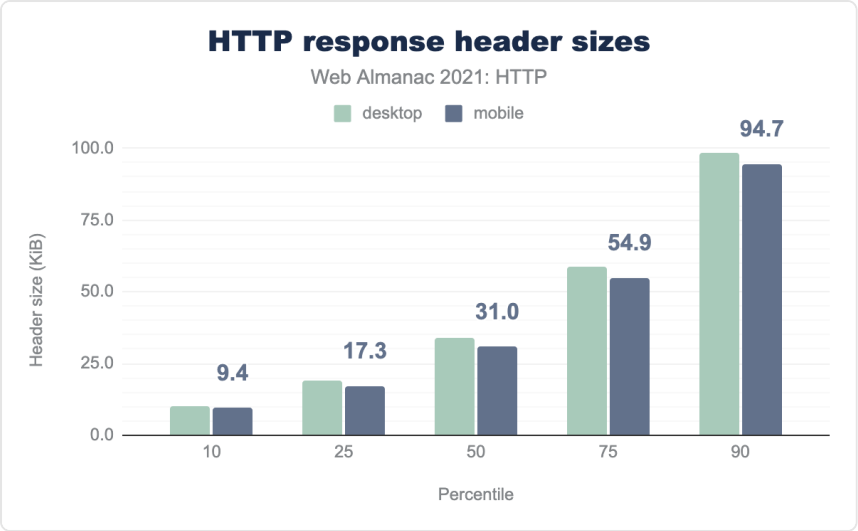


図24.17. HTTPレスポンスヘッダーサイズ。

ウェブページの中央値では、デスクトップで34KB、モバイルで31KB相当のヘッダーが返されます。90パーセンタイルでは、デスクトップで98KB、モバイルで94KBに増加します。しかし、レスポンスヘッダーの最大のインスタンスは5.38MB以上でした。多くのサイトで、1MBを超えるレスポンスヘッダーが発見されました。一般的に、これらの大きなレスポンスヘッダーは、オーバーウェイトの `CSP` または `P3P` ヘッダーが原因であり、ウェブサイト間でこれらのヘッダーの複雑さや誤った管理を示唆しています。他の極端な例では、複数の `Set-Cookies` や `Cache-Control` の設定を重複させるアプリケーションの設定ミスやエラーが原因で、ヘッダーが過大になっています。

優先順位付け

また、あるストリームを別のストリームに依存させることでストリームをリンクさせたり、1〜256の整数を割り当てて重み付けをしたりすることもできます。これらの依存関係や重み付けによって、サーバーは特定のキーストリームを優先し、他のストリームよりも先に応答データを送信できます。

HTTP/2が導入されて以来、優先順位付けはウェブのさまざまな部分で矛盾なく実装されています。Andy Davis[979]は、この矛盾が、ウェブ上のユーザーに最適とはいえない体験をもたらす可能性があることを発見しました。これは、サーバーが優先順位付けを無視して、先着順でサービスを提供することが多いからです。実際、Andyの研究[980]は、主要なCDNの多くが

979. https://x.com/AndyDavies
980. https://github.com/andydavies/http2-prioritization-issues

HTTP/2の優先順位付けを正しく実装していないことを強調しています。これには、人気のあるクラウドロードバランサーも多数含まれています。2021年のデータでは、優先順位付けを正しく実装しているCDNはわずか6社で、例年と同様の調査結果となっています。これには、Akamai、Fastly、Cloudflare、Automattic、section.io、Facebook独自のCDNが含まれています。

Patrick Meehan[981]は、優先順位付けを正しく実装しているCDNを使っていない場合、BBRや`tcp_notsent_lowat`を含む多くのTCP最適化[982]があり、サーバー側での優先順位付けを改善できるだろうと提案しています。

この不整合はクライアントレベルでも存在し、ブラウザベンダーによってこの動作の実装が異なります。Safariは、アセットタイプに応じて優先順位付けを行う静的なアプローチを実装しており、依存関係をマッピングしません。Chrome、Edge、Firefoxは、ストリーム全体の論理的な依存関係を構築する、より高度なアプローチを持っており、発見された優先順位に基づいてストリーム上の要求されたアセットの優先順位を変更できます。

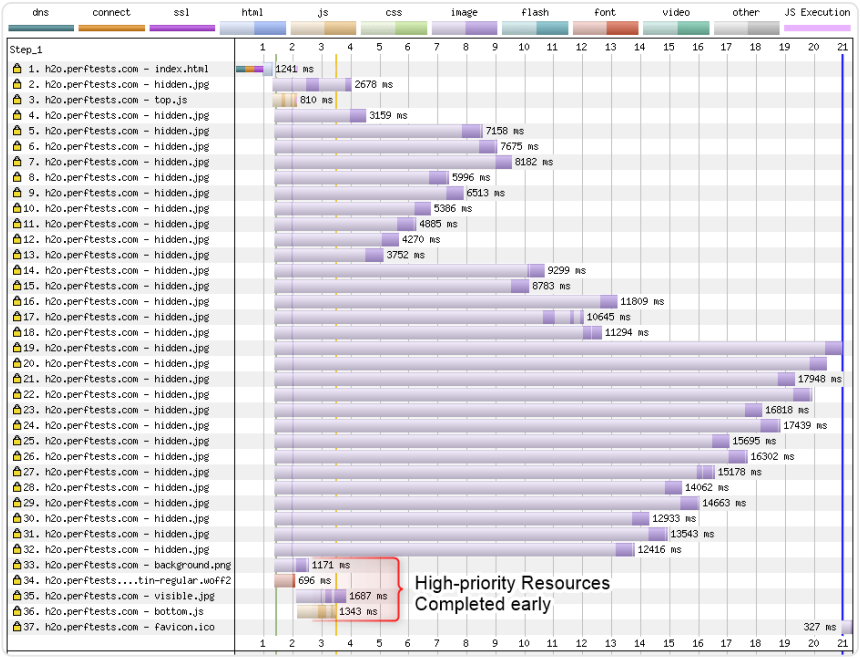


図24.18. WebPageTestのウォーターフォール例。

HTTP/2からは、HTTPの拡張可能な優先順位付けスキーム[983]という提案で、優先順位付けに

981. *https://x.com/patmeenan*
982. *https://blog.cloudflare.com/http-2-prioritization-with-nginx/*
983. *https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-priority*

対する最新の提案がなされています。これには、レスポンスに `priority` ヘッダーを追加することと、HTTP/2用の新しい `PRIORITY_UPDATE` フレームを追加することが含まれます。この `PRIORITY_UPDATE` フレームは、HTTP/3においても提案されています。これはまだウェブ全体に完全に採用されていませんが、Cloudflare[984]が優先順位付け[985]の基本動作を改善する取り組みとして注目されています。

HTTP/2プッシュの死？

もう1つの大きな特徴は、サーバープッシュの仕組みが導入されたことです。HTTP/2のサーバープッシュでは、サーバーがクライアントのリクエストに応答して複数のリソースを送信できます。このため、クライアントがその存在を気づく前に、サーバーはクライアントが必要とする可能性のあるアセットを通知する。一般的な使用例としては、ブラウザがベースHTMLを解析してそれらの重要な資産を特定し、その後自ら要求する前に、JavaScriptやCSSなどの重要な資産をクライアントにプッシュすることが挙げられます。クライアントには、プッシュメッセージを拒否するオプションもあります。

ゼロ・ラウンドトリップ、先制的な重要資産、パフォーマンス向上の可能性などの約束にもかかわらず、HTTP/2プッシュは誇大広告に見合うものではありません。



図24.19. HTTP/2プッシュを使用しているサイト。

2019年に分析したところ、HTTP/2はほとんど採用されておらず、平均0.5%程度でした。翌2020年には、デスクトップで0.85%、モバイルでは1.06%の採用率に増加しました。今年2021年はデスクトップで1.03%、モバイルで1.25%とわずかに増加しています。相対的にモバイルは前年比で大幅に増加していますが、HTTP/2の採用率は全体で1.25%と、まだ無視できるレベルです。ページレベルでは、デスクトップが64kリクエスト、モバイルが93kリクエストとなっています。

984. *https://blog.cloudflare.com/better-http-2-prioritization-for-a-faster-web/*
985. *https://blog.cloudflare.com/adopting-a-new-approach-to-http-prioritization/*

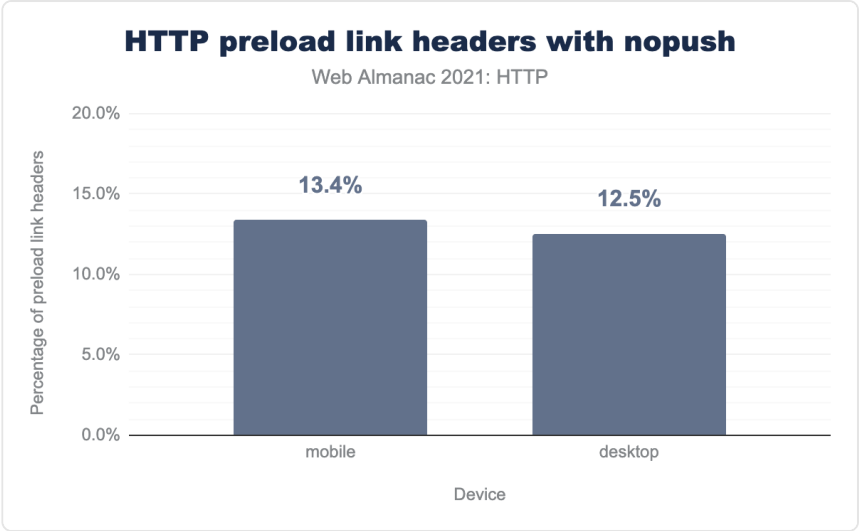


図24.20. HTTPのプリロードリンクヘッダーは `nopush` を使用します。

多くのHTTP/2の実装では、`preload` resource hint をプッシュするためのシグナルとして再利用していました。しかし、場合によっては、開発者がアセットをプリロードしたいが、HTTP/2プッシュメカニズムで配信されるのは嫌だと判断することがあります。CDNや他の下流のサーバーに、`nopush` ディレクティブを使って、プッシュを試みないように合図したいかもしれません。今年のデータでは、20万件以上のプリロードヘッダーが使用され、そのうちの平均12%が`nopush`属性で発行されたことが分かっています。

課題の1つはNginx[986] や Apache[987] 設定でグローバルに定義されているような、サイト全体に一様に適用する一連のプッシュに対して、現在のページとそのページの重要な資産に基づいて形成される、ページ レベルでの動的プッシュ ディレクティブを導入することです。Akamai[988] や Google[989] が実際のユーザー データと分析を使ってこの動的プッシュの設定を決定する実装例を示しているものの、ウェブ全体での実装が限定されていることがデータからうかがえます。Akamai[990] の研究によると、正しく適用すればHTTP/2プッシュはウェブパフォーマンスに明確な利益をもたらすことが示唆されています。

しかし、他のCDNプロバイダーやサーバー実装からの投資は、HTTP/2プッシュのための設計が困難であることを証明しています。実際にJake Archibald[991]は、2017年の時点でこれらの課題[992]のいくつかを説明しています。これらは、プッシュキャッシュの問題、ブラウザの不整合、クライアントがプッシュを必要としないと判断した場合にサーバーから送信される

986. *https://www.nginx.com/blog/nginx-1-13-9-http2-server-push/*
987. *https://httpd.apache.org/docs/2.4/howto/http2.html#push*
988. *https://medium.com/@ananner/http-2-server-push-performance-a-further-akamai-case-study-7a17573a3317*
989. *https://github.com/guess-js/guess/*
990. *https://medium.com/@ananner/http-2-server-push-performance-a-further-akamai-case-study-7a17573a3317*
991. *https://x.com/jaffathecake*
992. *https://jakearchibald.com/2017/h2-push-tougher-than-i-thought/*

余分なバイトに焦点を当てたものです。いくつか[993]のこれらの[994] issuesを解決しようとする試みは、キャッシュ ダイジェストがユーザーの特定に使われるかもしれないプライバシーとセキュリティの懸念に関する問題から、主に放棄されました。

Patrick Meehan氏は、代替案に関するこの投稿103 Early Hintsで、問題点のいくつかを取り上げています[995]。その投稿では、Pushは通常、HTMLやその他のレンダーブロッキングアセットを遅延させる結果になると詳しく説明しています。

プッシュされたアセット

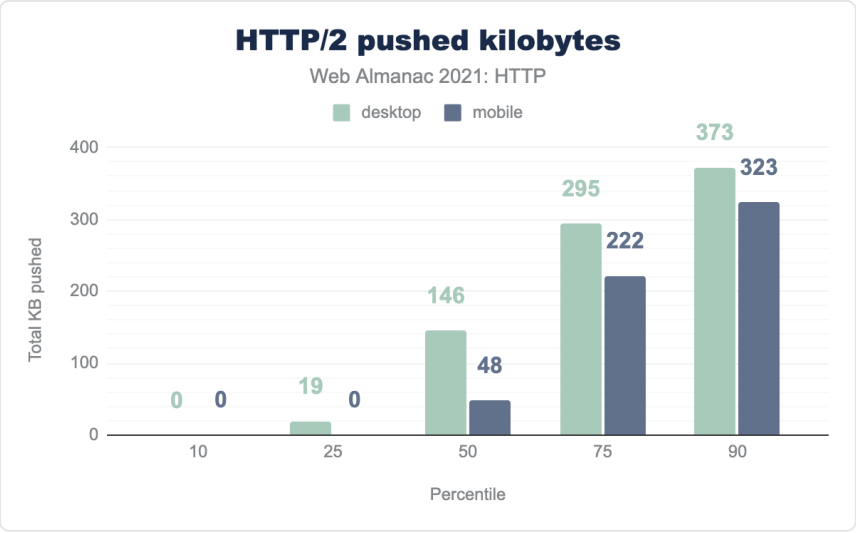


図24.21. HTTP/2 pushed kilobytes.

アイテムがプッシュされた場合のバイトサイズの中央値は、デスクトップで145KB、モバイルで48KBでした。これが75パーセンタイルになると、デスクトップでは294KBとほぼ倍増し、モバイルでは221KBと4倍以上になっています。トップエンドでは、90パーセンタイルで372KB、モバイルでは323KBがプッシュされていることがわかります。

この90%台の数字は問題ないように見えますが、プッシュ回数を見直すようになると、プッシュ機能の誤用が浮き彫りになってくるのです。

993. https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cache-digest#appendix-A
994. https://datatracker.ietf.org/doc/html/draft-vkrasnov-h2-compression-dictionaries-03
995. https://blog.cloudflare.com/early-hints/#:~:text=summarized%20server%20push%E2%80%99s%20gotchas

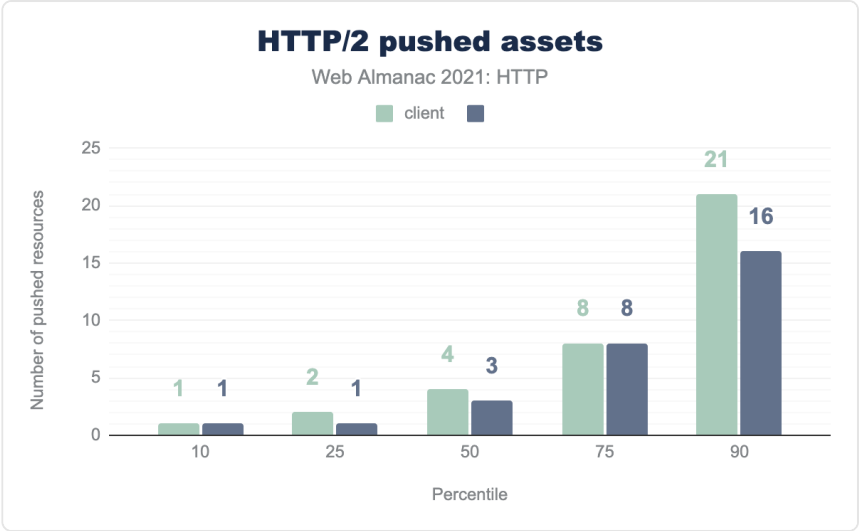


図24.22. HTTP/2はキロバイトをプッシュしています。

プッシュ回数の中央値は、デスクトップとモバイルでそれぞれ4回と3回です。75パーセンタイルでは8回、90パーセンタイルでは21回と16回に跳ね上がります。100パーセンタイルでは、517回と630回という驚くべきプッシュが一部のサイトで行われており、とくにプッシュが本来リクエストの初期に少数の重要なアセットを提供するために設計されたことを考えると、この機能の危険性が浮き彫りになっています。

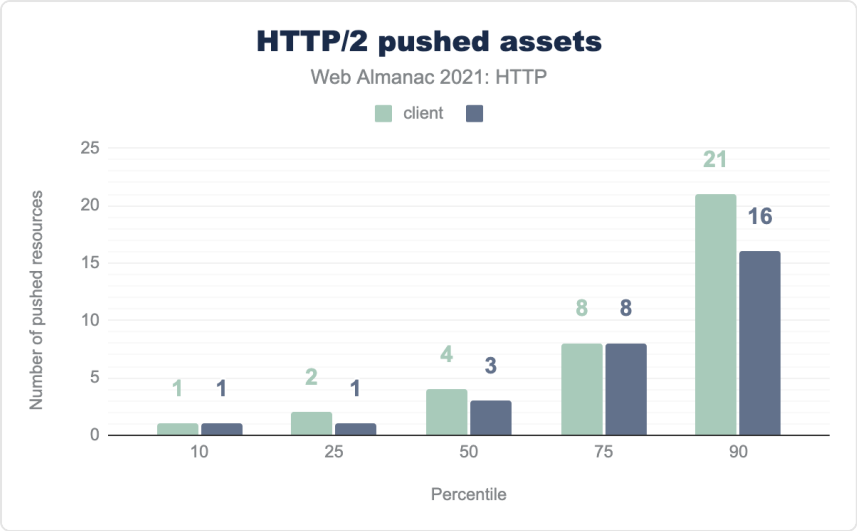


図24.23. HTTP/2のプッシュ回数。

コンテンツの種類別に分析すると、フォントがもっとも多く、次いで画像、CSS、スクリプト、動画の順になっています。この数字は、アセットタイプのサイズを見ると、異なるストーリーを描いています。フォントは依然としてもっとも多く利用されているアセットですが、スクリプトも遠く及ばない程度に利用されています。次いで、画像、動画、CSSの順です。したがって、より多くのCSSファイルがプッシュされているにもかかわらず、そのサイズは小さいということになります。スクリプトは、フォント、画像、CSSほど頻繁にプッシュされていませんが、プッシュされたデータの中ではより大きなボリュームを占めています。

上記の数字が示すように、また以前の記事でも紹介したように、HTTPプッシュは十分に活用されていません。利用されていても、誤用されたり、意図したとおりに使われなかったりすることが多く、エンドユーザーにとってパフォーマンスの障害となる可能性が高いのです。

Googleは、Chromeからプッシュを削除する意向を表明しています。しかし、2021年を通じて、HTTP/2 Pushの有効性については、まだ継続的な議論[996]が続いていました。この削除はまだ実現しておらず、Pushを正しく実装しているCDNを通じて活用できることが大きく示唆されています。Googleは、Pushの代替手段として `<link rel="preload">` ディレクティブの活用を推奨していますが、それでも1RTTが発生するため、Pushが解決しようとすることは同じです。GoogleもHTTP/3にPushを実装していないと報告[997]しており、Cloudflareのような他の企業も同様です。

996. https://groups.google.com/a/chromium.org/g/blink-dev/c/K3rYLvmQUBY/m/vOWBKZGoAQAJ
997. https://groups.google.com/a/chromium.org/g/blink-dev/c/K3rYLvmQUBY/m/vOWBKZGoAQAJ

プッシュの代替

Pushの代わりによく提案されるのが、アーリーヒントを使うことです。これは、サーバーに `103` ステータスコード応答メッセージを報告させ、Linkヘッダーに `preload` ヒントを含めることで動作します。アーリーヒントにより、サーバはページのHTMLを取得する前にクライアントが `preload` すべきアセットを報告できます。

```
HTTP/1.1 103 Early Hints
Link: <style.css>; rel="preload"; as="style"
```

Fastly[998]やCloudflare[999]などのCDNは、アーリーヒントヒントの実験を行っていますが、アーリーヒントヒントはまだ日が浅いです。この記事を書いている時点ではChrome内のHTTP/2におけるアーリーヒントのサポートはまだ 作業中[1000] であり、他のブラウザベンダーはアーリーヒントのサポートを発表し、Cloudflareは野放し状態でサポートを導入していますが、他の多くのベンダーはまだ具体的に実装していない状態です。

HTTP/2プッシュの採用が年々増えているにもかかわらず、Googleや他のブラウザベンダーはプッシュのサポートを放棄し、アーリーヒントのような代替手段を選ぶと思われる。CDNからのサポートと相まって、アーリーヒントがその代替となる可能性が高い。昨年、私たちはHTTP/2プッシュに別れを告げるかどうかという問題を提案しました。今年は、少なくともウェブブラウジングのユースケースでは、HTTP/2のメインストリーム利用は死んだと提案します。

HTTP/3

HTTP/3は、HTTP/2の次の進化であり、その基礎の上に、プロトコル全体にさらなる変更を加えて構築されています。最大の変化は、TCPからQUICと呼ばれるUDPベースのトランスポートプロトコルに移行したことです。これにより、インターネット全体に浸透しているTCPの実装がサポートされるのを待つことなく、HTTPの迅速な進歩が可能になりました。たとえば、HTTP/2は独立したストリームの概念を導入しましたが、TCPレベルではこれらはまだ1つのTCPストリームの一部であり、本当の意味で独立したものではありません。これをサポートするためにTCPを変更することは、安全に使用できるほど広くサポートされるようになるまで、かなりの時間を要するでしょう。そのため、HTTP/3は代替のトランスポートプロトコルに切り替えています。QUICは多くの点でTCPに似ており、基本的にTCPの多くの便利な機能をすべて再構築し、新しい機能を追加しています。QUICは暗号化され、よくサポートされている軽量のUDPトランスポートプロトコルで配信されます。

998. *https://www.fastly.com/blog/beyond-server-push-experimenting-with-the-103-early-hints-status-code*
999. *https://blog.cloudflare.com/early-hints/*
1000. *https://bugs.chromium.org/p/chromium/issues/detail?id=671310*

HTTP/3の採用

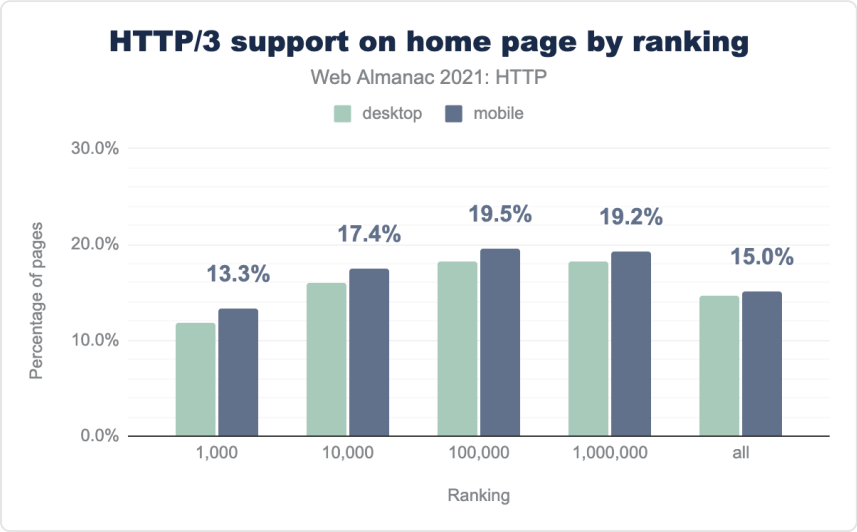


図24.24. ランキングによるトップページのHTTP/3対応。

この章の前半で、上位にランクされたサイトほどHTTP/2の採用率が、高いことがわかりました。意外なことに、HTTP/3ではその逆が当てはまります。上位1000サイトのサポートは上位100万サイトよりも少なく、モバイルサイトでのサポートがわずかに多く実装されていることがわかります。

上位10万サイトと上位100万サイトでの分布は、デスクトップが18%、モバイルは19%。上位1万サイトでは、16%と17%に低下しています。上位1,000サイトでは、デスクトップとモバイルでそれぞれ11%と13%となっています。上位100万サイトを超えるサイトでは、ホームページへの導入率は15%程度です。全体として、これは非常に強力な採用であり、おそらくいくつかの主要なCDNからのサポートが先導しているのでしょう。このことは、上位のウェブサイトがHTTP/2を主流として採用している一方で、多くのウェブサイトがHTTP/3をまだ探求していないことを示唆しています。

HTTP/3対応

HTTP/3へのWebサーバーの対応は、まだ市場でも限られています。Nginxはウェブ上でもっとも一般的なHTTPサーバーであり、HTTP/2サイトの約3分の2がNginxのバージョンを使用しています。NginxはHTTP/3のサポートを公に表明しており、ロードマップの議論[1001]で完全サポートを展開し、2021年末までに完全サポートを実現することを目指しています。それ

1001. https://www.nginx.com/blog/our-roadmap-quic-http-3-support-nginx/

に比べ、Apacheサーバーは、HTTP/3がいつサポートされるのか、まだ何のガイダンスも出していない。Microsoftは、新しい Windows Server 2022[1002] でHTTP/3をサポートすることを発表しました。LiteSpeed Webサーバーのような他の代替品は、HTTP/3のサポートに傾いています[1003]が、CaddyはHTTP/3のサポートを 実験機能[1004] として利用できるようにしました。Node.jsのサポートは、OpenSSLのサポートがないため、保留[1005]されています。

また、多くのCDNがHTTP/3のサポートを表明しています。Cloudflareは、2019年から[1006] HTTP/3の実験を行っており、その中で多くの事例でパフォーマンスが向上していることを報告しています。Cloudflareは、エッジネットワークでのHTTP/3展開を強化する quiche[1007]ライブラリも公開しています。Fastlyは、HTTP/3のサポートについても議論しており[1008]、BETAサービス[1009]として利用できるようにしています。Fastlyは、quicly[1010]という独自の実装もオープンソース化しており、FastlyがエッジネットワークでH2O HTTP[1011]サーバ用に設計されています。Akamaiはまた、HTTP/3とQUICの継続的なサポート[1012]を表明し、マイクロソフト社と協力して QUICによるOpenSSL[1013] のバージョンをフォークし、サポートの前進[1014] を支援しました。

HTTP/3のブラウザサポートは、現在も進化を続けています。2021年10月現在、Microsoft Edge、Firefox、Google Chrome、Operaの最新バージョンでサポートされており、一部のAndroidの亜種とOpera mobileについてはモバイル全体で部分的に利用可能です。Safariからのサポートは macOS 11 Big Surに限定されており、「実験的機能」を介して有効にする必要があります、iOSのサポートもフラグの後ろにある実験的機能としてのみ利用可能です。

HTTP/3のネゴシエーション

HTTP/3は従来のTCPベースのHTTPと完全に、異なるトランスポート層にあるので、HTTPSネゴシエーションを通じてHTTP/2で起こるような、接続設定の一部としてHTTP/3をネゴシエートすることはできません。この段階で、あなたはすでにトランスポートプロトコルを選んでいます!

HTTP/3は代わりに `alt-svc` ヘッダーを必要とします。TCPベースのHTTP接続 (クライアントがHTTP/3をサポートするほど高度であれば、おそらくHTTP/2) で開始し、サーバーはリクエストに対するレスポンスで `alt-svc` ヘッダーを通して、このサーバーはUDPとQUIC上のHTTP/3もサポートしていると通知できます。ブラウザはそれを使って接続を試みることを決定できます。HTTP/3のいくつかの繰り返しのために、このヘッダーは、クライアン

1002. https://blog.workinghardinit.work/2021/10/11/iis-and-http-3-quic-tls-1-3-in-windows-server-2022/
1003. https://docs.litespeedtech.com/cp/cpanel/quic-http3/
1004. https://caddyserver.com/docs/caddyfile/options
1005. https://github.com/nodejs/node/pull/37067
1006. https://blog.cloudflare.com/http3-the-past-present-and-future/
1007. https://github.com/cloudflare/quiche
1008. https://www.fastly.com/blog/why-fastly-loves-quic-http3
1009. https://www.fastly.com/blog/modernizing-the-internet-with-http3-and-quic
1010. https://github.com/h2o/quicly
1011. https://h2o.examp1e.net/
1012. https://www.akamai.com/blog/performance/http3-and-quic-past-present-and-future
1013. https://github.com/quictls/openssl
1014. https://daniel.haxx.se/blog/2021/10/25/the-quic-api-openssl-will-not-provide/

トとサーバーがどのバージョンのHTTP/3に決定することができるかということでもあります。

そのため、最初のケースでは、最初のリクエストでHTTP/2が使用されます。いったんブラウザが `alt-svc` ヘッダーを検出すると、プロトコルを切り替えてHTTP/3を使用し始めることができます。将来のケースでは、ブラウザは `alt-svc` ヘッダーをキャッシュし、次回はHTTP/3を試すために直接ジャンプできます。

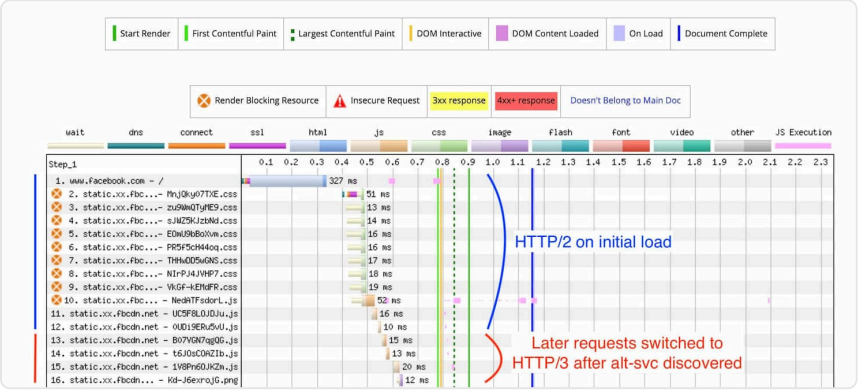


図24.25. ページロード中にHTTP2からHTTP3に切り替わる様子を示すWebPageTestの例。

また、コネクション合体（コネクションの再利用）により、2つのホスト名がDNS上で同じIPに解決し、同じTLS証明書とバージョンを使用する場合、クライアントは両方のホスト名で同じコネクションを再利用できる場合があります。したがって、使用するホストとTLS証明書の数に応じて、HTTP/2とHTTP/3の両方が混在するウォーターフォール リクエストを目にすることは珍しくありません。

ページレベルでは、リクエストの約15%が `alt-svc` ヘッダーを提供します。これらはQUICを提供する構文、さまざまなH3プレリリースバージョンの1つ（公式にはHTTP/3は執筆時点で標準化されていませんが、非常に最終段階にあります）間で異なります。いくつかのサイトは `quic=":443"; ma=2592000; v="39,43,46,50"` のように複数のバージョンのQUICのサポートを宣伝していますが、中には1つのバージョンしか提供しないサイトもあります。もっとも一般的な `alt-svc` のアドバタイズは `"h3-27=":443"; ma=86400, h3-28=":443"; ma=86400, h3-29=":443"; ma=86400, h3=":443"; ma=86400"` で、すべての `alt-svc` 応答に対して11%に及んでいます。このヘッダーは、HTTP/33バージョン27、28、29をサポートし、 `max-age` が24時間であることをクライアントに指示します。

`alt-svc` がある場合、ほとんどのサイトは新しいプロトコルのバージョンをサポートするようにバージョン番号を追加していますが、 `clear` ディレクティブを使って以前に宣伝したサポートを無効にしているケースも多く見られました。

この記事の執筆時点では、HTTP/3仕様[1015]の最新バージョンは34です。しかし、この最新バージョンを報告している回答は、わずか0.01%です。リクエストレベルで `alt-svc` の詳細を見る場合、レスポンスヘッダーでもっともよく要求されるバージョンは27であす。サーバーは左から右の順に優先されるバージョンを表示します。6%のリクエストでは、最初に `h3-27` が優先され、同じレスポンスで28と29が代替バージョンとして提供されると報告されます。2%のレスポンスは `h3-29` をアップグレードのための唯一の優先バージョンとして提供します。QUICは優先的に更新されるプロトコルですが、わずか0.11% しか受信していません。実際には `h3-29` 以降は技術的にほとんど違いがなく、ほとんどの実装は `h3` の正式リリースを待ってバージョンを凍結していました。

ほとんどの `alt-svc` は `max-age` が24時間しかないと報告しています。これは、指定しない場合のデフォルトです。もっとも長い `alt-svc` の `max-age` は30日または2592000秒でした。

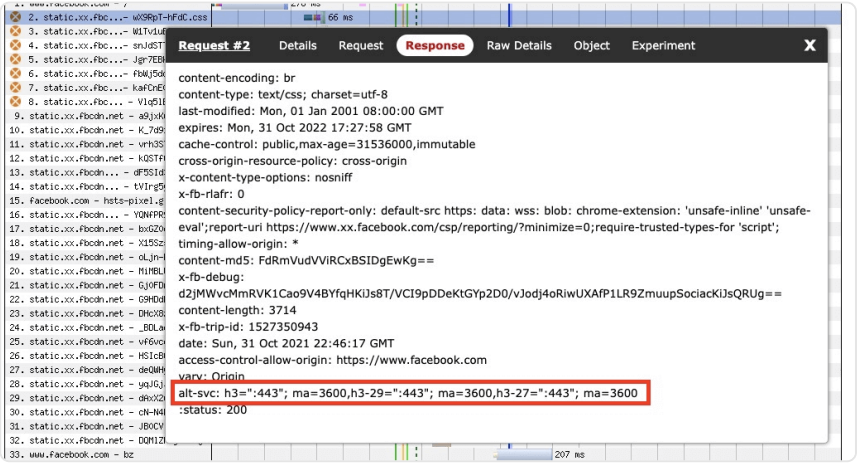


図24.26. WebPageTestの `alt-svc` の例。

HTTP/3に関する考察と懸念

HTTP/3の長所が多く語られる一方で、懸念や批判も指摘されている。多くの開発者は、HTTP/1.1からの制限を克服するために多くのウェブパフォーマンスの回避策をロールバックしなければならず、それらの回避策が後にHTTP/2で アンチパターン[1016] となったため、今ようやくHTTP/2から導入した変更に慣れてきたというところです。

場合によっては、開発者とサイトオーナーは、HTTP/3による増分の利益は、ウェブサーバ

1015. https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34
1016. https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19

一の大規模なアップグレードに見合わないかもしれないと主張するかもしれません。とくに、HTTP/3が、優先順位付けやサーバープッシュの効果的な使用など、HTTP/2で特定された問題をすべて解決していない場合はなおさらです。そのため、採用はウェブアプリケーション内ではなく、CDNレベルで推進されるかもしれません。これは、いくつかのサーバーがHTTP/3をサポートしないか、OpenSSLのサポート不足によってブロックされる場合、とくにそうなる可能性があります。

この章を通して説明したように、QUICはUDPプロトコルに依存しています。HTTP/3の導入に伴い、UDPのトラフィックはウェブ全体で増加する予定です。しかし、現状ではUDPはリフレクション攻撃[1017]などの攻撃ベクトルとしてよく利用されています。QUICはいくつかの保護メカニズム[1018]を備えていますが、これは、ウェブ上でのUDPの扱いの変更と、あるネットワークとファイアウォールで許可されるUDPトラフィックの量の変更を意味するかもしれません。同じ例で、TCPヘッダーとパケットの非暗号化部分がファイアウォールやウェブ上の他のミドルボックス[1019]で使用されている場合、採用の後押しがあるかもしれません。QUICはパケットの多くの部分を暗号化するため、パケットに関する検査の可視性が低くなり、追加のセキュリティチェックを行う能力など、これらのミドルボックスの動作が、制限される可能性があります。

また、QUICはサーバー側でパフォーマンスの問題が、発生することが懸念されています。これは、UDPを扱う際に必要となるCPU要件が高くなるためです。HTTP/2と比較した場合、2倍のCPUが必要になるとの試算もあります。とはいえ、QUIC CPUパフォーマンス[1020]を最適化する試みは現在進行形で数多く行われています。

このような懸念はあるものの、本当のメリットはウェブのエンドユーザーから受けることになります。QUICは、ネットワーク接続を切り替えても接続を維持できるため、モバイルファーストの世界でもモバイルファーストの体験を可能にします。ヘッドオブラインブロッキングの改善により、ページロードがより向上し、1ミリ秒[1021]が重要であることを私たちは知っています。また、QUICが導入する強化された暗号化により、より安全でセキュアなWebを実現できます。また、HTTP/3では0-RTTが可能であるため、パフォーマンスの向上も期待できます。

結論

本章を通じて、私たちはHTTP/2の採用が増加していることと、より新しいバージョンのプロトコルが提供する利点に主な焦点を当てながら、HTTPの進化を見てきました。続いて、HTTP/3について詳しく見ていき、バージョン3がウェブ上で数年間HTTP/2が使用された後に特定された懸念の多くをどのように解決することを目指しているかを見てきました。

1017. https://blog.cloudflare.com/reflections-on-reflections/
1018. https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-27#section-8.1
1019. https://ja.wikipedia.org/wiki/Middlebox
1020. https://conferences.sigcomm.org/sigcomm/2020/files/slides/epiq/0%20QUIC%20and%20HTTP_3%20CPU%20Performance.pdf
1021. https://ai.googleblog.com/2009/06/speed-matters.html

HTTP Archiveのデータによると、今年はHTTP/2の採用が大きく進み、リクエストの72%がHTTP/2を使用し、ベースとなるHTMLページの59%がHTTP/2を使用していることがわかります。この採用は、CDNプロバイダーによる採用の増加が主な要因となっています。HTTP/1.1はウェブ上で少数派となっています。

HTTP/2の普及にもかかわらずHTTP/2のプッシュ機能は、実装の複雑さのために十分に活用されていないままであり、私たちは、プッシュは実際に到着時に死んでいる可能性があることを示唆しています。同時に、リソースの優先順位付けに関する継続的な懸念や、主要なCDNベンダー以外の不正な実装が見られます。優先順位付けの複雑さは、HTTP/3仕様から削除されるほど、依然として広まっています。

2021年は、HTTP/3の採用に関しても詳しく検証することができました。GoogleやFacebookなどの主要プレイヤーは、数年前からHTTP/3のサポートを独自に展開しています。HTTP/3のより広い採用は、ウェブの他の部分に対するHTTP/3のサポートへ公に取り組んできたAkamai、Cloudflare、Fastlyによって影響されています。

HTTP/3はTCPに課せられたヘッドオブラインブロッキングなど、HTTP/2の改善点をベースにしつつ、QUICのTLS 1.3のより厳密なカプセル化により、プロトコルスタックのより多くの部分の安全性を確保することを目的としています。しかし、HTTP/3はまだ初期段階です。私たちは2022年にHTTP/3の採用を測定することを楽しみにしており、HTTP/2のサポートが主流になり、人々が現在の展開よりもさらなる改善を得たいと考えるようになるにつれて、さらに牽引される可能性が高いと信じています。

HTTP/3にはいくつかの懸念が示されていますが、これらの懸念のいずれもが、エンドユーザーへ得られるパフォーマンスによって打ち消されるはずです。HTTP/3の採用は、HTTP/2で見られたように、CDNが独自の実装に向けて取り組むことで、CDNのロールアウトによって促進される可能性もあります。とくに、主要なウェブフレームワークでの実装はまだ確認されていません。また、今後数年間は、HTTP/2とHTTP/3が混在することになると思われます。

著者



Dominic Lovell

@dominiclovell dominiclovell

Dominic Lovell は現在 Akamai Technologies のソリューション・エンジニアリング・マネージャーで、ウェブ上でのサイトのパフォーマンスと安全性を高めることに長年取り組んでいます。ツイッターでは@dominiclovell、またはLinkedIn[1022]で彼とつながることができます。

1022. https://www.linkedin.com/in/dominiclovell/

付属資料 A
方法論



概要

Web Almanacは、HTTP Archive[1022]が主催するプロジェクトです。HTTP Archiveは、ウェブがどのように構築されているかを追跡することを使命として、2010年にSteve Soudersによって開始されました。何百万ものウェブページの構成を毎月評価し、その何テラバイトものメタデータをBigQuery[1024]で分析できるようにしています。

Web Almanacの使命は、ウェブの状態に関する一般的な知識の年次リポジトリとなることです。私たちの目標は、専門家が文脈に応じた洞察を提供することで、HTTP Archiveのデータ

1023. https://httparchive.org
1024. https://httparchive.org/faq#how-do-i-use-bigquery-to-write-custom-queries-over-the-data

ウェアハウスをウェブコミュニティにとってさらに利用しやすいものにすることです。

Web Almanacの2021年版は、コンテンツ、体験、出版、流通の4つのパートに分かれています。各パートでは、いくつかの章がその包括的なテーマをさまざまな角度から探求しています。たとえば、パートIIでは、パフォーマンス、セキュリティ、アクセシビリティなどの各章で、ユーザー体験をさまざまな角度から探求しています。

データセットについて

HTTP Archiveのデータセットは、毎月新しいデータが追加され、継続的に更新されています。Web Almanacの2021年版では、その章でとくに断りがない限り、すべてのメトリクスは2021年7月のクロールから取得されました。これらの結果は、BigQuery上で公に照会可能[1025]で`2021_07_01`というプレフィックスが付いたテーブルで提供されています。

Web Almanacに掲載されているすべてのメトリクスは、BigQuery上のデータセットを使用して一般に再現可能です。GitHub リポジトリ[1026]で、すべての章で使われているクエリを閲覧できます。

これらのクエリの中には非常に大きなものがあり、自分で実行すると高くつく[1027]可能性があることに注意してください。出費を抑えるには、Tim Kadlecのポスト BigQueryを低コストで利用する[1028] を参照してください。

たとえば、デスクトップとモバイルのページあたりのJavaScriptのバイト数の中央値を理解するには、bytes_2021.sql[1029] をご覧ください。

```
#standardSQL
# 1ページあたりのJSリクエストバイトの合計(2021)
SELECT
  percentile,
  _TABLE_SUFFIX AS client,
  APPROX_QUANTILES(bytesJs / 1024, 1000)[OFFSET(percentile *
10)] AS js_kilobytes
FROM
```

1025. https://github.com/HTTPArchive/httparchive.org/blob/main/docs/gettingstarted_bigquery.md
1026. https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2021
1027. https://cloud.google.com/bigquery/pricing
1028. https://timkadlec.com/remembers/2019-12-10-using-bigquery-without-breaking-the-bank/
1029. https://github.com/HTTPArchive/almanac.httparchive.org/blob/main/sql/2021/javascript/bytes_2021.sql

```
    `httparchive.summary_pages.2021_07_01_*`,
    UNNEST([10, 25, 50, 75, 90, 100]) AS percentile
  GROUP BY
    percentile,
    client
  ORDER BY
    percentile,
    client
```

各指標の結果は、たとえばJavaScript results[1030]のように、章ごとのスプレッドシートで一般に閲覧できます。生の結果やクエリへのリンクは各章の最下部にあります。また、指標別の結果とクエリも各図から直接リンクされています。

ウェブサイト

データセットには8,198,531のWebサイトが含まれています。これは、Web Almanacの2020年版と比較して9%の増加です。そのうち7,499,763件がモバイルサイト、6,294,605件がデスクトップサイトです。ほとんどのウェブサイトは、モバイルとデスクトップの両方のサブセットに含まれています。

HTTP Archiveは、Chrome UX Report からウェブサイトのURLを取得しています。Chrome UXレポートは、Googleが公開しているデータセットで、Chromeユーザーが積極的に訪問した数百万のウェブサイトのユーザー体験を集約しています。これにより、最新のWebサイトのリストが得られ、実際のWebの使用状況を反映できます。Chrome UXレポートのデータセットには、フォームファクターのディメンションが含まれており、これを使用して、デスクトップまたはモバイルユーザーがアクセスしたすべてのWebサイトを取得します。

7月の2021はWeb Almanacが使用したHTTP Archiveのクロールでは、ウェブサイトのリストとして、最新のChrome UXレポートのリリースが使用されました。202105データセットは、2021年6月8日に公開され、5月の間にChromeユーザーが訪問したウェブサイトをキャプチャしています。

リソースの制限により、HTTP Archiveでは、Chrome UXレポートの各ウェブサイトから1ページのみテストできます。このため、ホームページのみを対象としています。ホームページは必ずしもWebサイト全体を代表するものではないため、結果に偏りが生じることをご了承ください。

1030. https://docs.google.com/spreadsheets/d/1zU9rHpI3nC6jTz3xgN6w13afW7x34xAKBh2IPH-IVxk/edit#gid=18398250

HTTP Archiveはラボテスト用のツールであり、データセンターからWebサイトをテストし、実際のユーザー体験からデータを収集するものではありません。すべてのページは、ログアウトした状態で空のキャッシュを使用してテストされており、実際のユーザーがどのようにアクセスするかを反映していない可能性があります。

Metrics

HTTP Archiveは、ウェブがどのように構築されているかについての数千もの指標を収集しています。ページあたりのバイト数、ページがHTTPSで読み込まれたかどうか、個々のリクエストおよびレスポンスヘッダーなどの基本的な測定基準が含まれています。これらのメトリクスの大部分は、各ウェブサイトのテスト ランナーとして機能する WebPageTest によって提供されます。

他のテストツールは、ページに関するより高度な指標を提供するために使用されます。たとえば、Lighthouseは、ページに対して監査を実行し、アクセシビリティやSEOなどの分野でその品質を分析するために使用されます。以下のツールの項では、それぞれのツールについて詳しく説明しています。

ラボのデータセット特有の制限を回避するために、Web Almanacは、とくにWebパフォーマンスの分野におけるユーザー体験の測定基準として Chrome UX レポート も利用しています。

測定基準によっては、まったく手の届かないものもあります。たとえば、ウェブサイトを構築するために使用されるツールを検出する能力は、必ずしもありません。ウェブサイトがcreate-react-appで構築されている場合、Reactフレームワークを使用していることはわかりますが、特定の構築ツールが使用されていることは必ずしもわかりません。これらのツールが、ウェブサイトのコードに検出可能なフィンガープリントを残さない限り、その使用状況を測定することはできないのです。

その他の指標は、必ずしも測定できないわけではないが、困難であったり、信頼性に欠けたりする場合がある。たとえば、ウェブデザインは本質的に視覚的なものであり、ページに邪魔なモーダルダイアログはあるかどうかなど定量化が、困難な場合があります。

ツール

Web Almanacは、以下のオープンソースツールの協力により実現されています。

ウェブページテスト

ウェブページテスト[1031]は著名なWebパフォーマンステストツールで、HTTP Archiveのバックボーンです。ウェブページテストのプライベートインスタンス[1032]を使用し、各ウェブページをテストする実際のブラウザであるプライベートテストエージェントを使用しています。デスクトップとモバイルのウェブサイトは、異なる設定の下でテストされます。

コンフィグ	デスクトップ	モバイル
デバイス	Linux VM	エミュレートされたMoto G4
ユーザーエージェント	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36 PTST/210702.163639	Mozilla/5.0 (Linux; Android 6.0.1; Moto G (4)) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Mobile Safari/537.36 PTST/210702.163639
ロケーション	Google Cloud Locations, USA	Google Cloud Locations, USA
接続	Cable (5/1 Mbps 28ms RTT)	3G (1.600/0.768 Mbps 300ms RTT)
ビューポート	1376 x 768px	512 x 360px

デスクトップ用ウェブサイトは、Linux VM上のデスクトップ用Chrome環境内から実行されます。ネットワーク速度は、ケーブル接続と同等です。

モバイルサイトは、3G回線相当のネットワーク速度を持つMoto G4のエミュレーション端末で、モバイルChrome環境内から実行されます。

テストエージェントは、米国にあるさまざまなGoogle Cloud Platformの拠点[1033]から実行されます。

1031. https://www.webpagetest.org/
1032. https://github.com/WPO-Foundation/webpagetest-docs/blob/master/user/Private%20Instances/README.md
1033. https://cloud.google.com/compute/docs/regions-zones/#locations

HTTP Archiveのウェブページテストのプライベートインスタンスは最新のパブリックバージョンと同期しており、カスタムメトリクス[1034]（テスト終了時に各ウェブサイト上で評価されるJavaScriptのスニペット）を追加しています。

各テストの結果は、HARファイル[1035]という、Webページに関するメタデータを含むJSON形式のアーカイブファイルとして提供されます。

Lighthouse

Lighthouse[1036]は、Googleが開発した自動ウェブサイト品質保証ツールです。ウェブページを監査し、最適化されていない画像やアクセスしにくいコンテンツなど、ユーザー体験を阻害するパターンが含まれていないことを確認します。

HTTP Archiveは、すべてのモバイル向けウェブページで最新バージョンのLighthouseを実行しています。デスクトップ向けページは、リソースが限られているため含まれていません。2021年7月のクロールの時点で、HTTP Archiveは8.0.0[1037]と 8.1.0[1038]のバージョンのLighthouseを組み合わせて使用していました。

Lighthouseは、ウェブページテスト 内から独立したテストとして実行されますが、独自の構成プロファイルを持っています。

コンフィグ	値
CPUの速度低下	1x/4x
ダウンロードのスループット	1.6 Mbps
アップロードのスループット	0.768 Mbps
RTT	150 ms

LighthouseとHTTP Archiveで利用できる監査の詳細については、Lighthouse開発者向けドキュメント[1039]を参照してください。

Wappalyzer

Wappalyzer[1040] は、Webページで使用されている技術を検出するためのツールです。テスト

1034. https://github.com/HTTPArchive/custom-metrics
1035. https://en.wikipedia.org/wiki/HAR_(file_format)
1036. https://developers.google.com/web/tools/lighthouse/
1037. https://github.com/GoogleChrome/lighthouse/releases/tag/v8.0.0
1038. https://github.com/GoogleChrome/lighthouse/releases/tag/v8.1.0
1039. https://developers.google.com/web/tools/lighthouse/
1040. https://www.wappalyzer.com/

される技術は、JavaScriptフレームワークからCMSプラットフォーム、さらには暗号通貨マイナーまで、90 のカテゴリ[1041]に分類されています。サポートされている技術は2,600以上（昨年の1,400から増加）です。

HTTP Archiveは、すべてのウェブページで最新版のWappalyzerを実行しています。2021年7月の時点で、Web AlmanacはWappalyzerのバージョン6.7.7[1042]を使用しています。

Wappalyzerは、WordPress、Bootstrap、jQueryなどの開発者ツールの人気を分析する多くの章をパワーアップしています。たとえば、EコマースとCMSの章はそれぞれのEコマース[1043]とCMS[1044]は、Wappalyzerが検出した技術のカテゴリーです。

Wappalyzerを含むすべての検出ツールには、限界があります。その結果の妥当性は、常にその検出メカニズムがいかに正確であるかに依存します。Web Almanacは、Wappalyzerを使用されているすべての章に注釈を加えますが、特定の理由によりその分析が、正確でない可能性があります。

Chrome UXレポート

Chrome UX レポート[1045]は、実際のChromeユーザーの体験をまとめた公開データセットです。体験は、たとえば`https://www.example.com`のように、Webサイトのオリジンでグループ化されています。このデータセットには、ペイント、負荷、インタラクション、レイアウトの安定性などのUX指標の分布が含まれています。月別のグループ化に加え、国レベルの地域、フォームファクター（デスクトップ、携帯、タブレット）、効果的な接続タイプ（4G、3Gなど）などの切り口で体験を分類することも可能です。

今年から、Chrome UXレポートのデータセットに、相対的な Web サイト ランキング データ[1046]が含まれるようになりました。これらは、ランク マグニチュードと呼ばれます。なぜなら、もっとも普及しているウェブサイトの1位や116位といった細かいランクとは対照的に、ウェブサイトは上位1K、上位10K、上位10Mまでのランクバケットにグループ化されるからです。各ウェブサイトは、そのすべてのページを合わせた対象[1047]ページビューの数によってランク付けされています。今年のWeb Almanacは、サイトの人気度によるウェブの構築方法のバリエーションを探る方法として、この新しいデータを広範に利用しています。

Chrome UX Reportの実際のユーザー体験データを参照するWeb Almanacの指標には、2021年7のデータセット（202107）が使用されています。

データセットについては、web.dev[1048]のBigQueryのChrome UX レポートの活用[1049]のガイドで

1041. https://www.wappalyzer.com/technologies
1042. https://github.com/AliasIO/Wappalyzer/releases/tag/v6.7.7
1043. https://www.wappalyzer.com/categories/ecommerce
1044. https://www.wappalyzer.com/categories/cms
1045. https://developers.google.com/web/tools/chrome-user-experience-report
1046. https://developers.google.com/web/updates/2021/03/crux-rank-magnitude
1047. https://developer.chrome.com/docs/crux/methodology#eligibility
1048. https://web.dev/
1049. https://web.dev/chrome-ux-report-bigquery

詳しく説明しています。

Blink機能

ブリンク機能[1050]は、特定のウェブプラットフォーム機能が使用されていることが検知されるたびに、Chromeがフラグを立てるインジケーターです。

私たちはBlink機能を利用して、機能の採用状況を別の角度から見ています。このデータは、ページに実装された機能と実際に使用された機能を区別するのにとくに有効です。たとえば、CSS の章のグリッド レイアウトに関するセクションでは、実際のページ レイアウトの一部がグリッドで構築されているかを測定するのにBlink機能データが使用されています。それに比べて、より多くのページが、たまたまスタイルシートに未使用のグリッド スタイルを含んでいます。どちらの統計もそれなりに興味深く、ウェブがどのように構築されるかについて何かを教えてくれます。

Blink機能については、ウェブページテスト によって定期テストの一部として報告されます。

サードパーティ ウェブ

サードパーティ ウェブ[1051]はPatrick Hulce による研究プロジェクトであり、その著者は2019 サードパーティ の章で、HTTP ArchiveとLighthouseのデータを使用して、ウェブ上のサードパーティ製リソースの影響を特定し分析しています。

ドメインは、少なくとも50のユニークページに表示されていれば、サードパーティ プロバイダーとみなされます。また、このプロジェクトでは、広告、分析、ソーシャルといったカテゴリーで、プロバイダーをそれぞれのサービスごとにグループ化しています。

Web Almanacのいくつかの章では、このデータセットのドメインとカテゴリを使用して、サードパーティの影響を理解しています。

Rework CSS

Rework CSS[1052]はJavaScriptベースのCSSパーサーです。スタイルシート全体を受け取り、個々のスタイルルール、セレクター、ディレクティブ、および値を区別するJSONエンコードされたオブジェクトを生成します。

この特別な目的のツールは、CSS 章の多くの指標の精度を著しく向上させました。各ページ

1050. https://chromium.googlesource.com/chromium/src/+/HEAD/docs/use_counter_wiki.md
1051. https://www.thirdpartyweb.today/
1052. https://github.com/reworkcss/css

のすべての外部スタイルシートとインライン スタイル ブロックのCSSが解析され、クエリされることで解析が可能になりました。BigQuery上のHTTP Archiveデータセットとの連携については、このスレッド[1053]をご覧ください。

Rework Utils

今年の CSS 章では、 Lea Verou によって導かれた昨年のCSS章で紹介された多くのメトリクスを再確認しています。LeaはRework Utils[1054]を書き、Rework CSSの出力からより簡単に洞察を抽出できるようにしました。CSSの章で見ることができる統計のほとんどは、これらのスクリプトによって提供され続けています。

Parsel

Parsel[1055]はCSSセレクタパーサーと特異性計算機で、元々は2020 CSS章のリーダーLea Verouによって書かれて、別のライブラリとしてオープンソース化されています。セレクターと特異性に関連するすべてのCSSメトリクスで広範囲に使用されています。

分析プロセス

Web Almanacは、ウェブ・コミュニティーの100人以上の貢献者の協力を得て、約1年かけて企画・実行されました。このセクションでは、Web Almanacに掲載されている章を選んだ理由、そのメトリクスを照会した方法、そしてその解釈について説明します。

計画

Web Almanacは、2021年4月に投稿者募集[1056]でスタートしました。私たちは、前年度の全23章でプロジェクトを初期化しましたが、コミュニティから追加のトピックが提案され、今年度は新たに2つの章を設けました。構造化データと WebAssembly です。

初年度の方法論で述べたとおりです。

> *Web Almanacの今後の明確な目標の1つは、著者や査読者として、代表的でない異質な声をより多く取り入れることを奨励することです。*

そのため、今年は著者の選考方法[1057]をより洗練させました。

1053. https://discuss.httparchive.org/t/analyzing-stylesheets-with-a-js-based-parser/1683
1054. https://github.com/LeaVerou/rework-utils
1055. https://projects.verou.me/parsel/
1056. https://github.com/HTTPArchive/almanac.httparchive.org/issues/2167
1057. https://github.com/HTTPArchive/almanac.httparchive.org/discussions/2165

- これまでの著者は、異なる視点を持つために、とくに再執筆を控えるようにした。
- 2021年の著者を推薦する皆さんには、とくに「似たような人ばかりを推薦しない」ということを意識してもらいました。
- プロジェクトリーダーは、推薦されたすべての著者を検討し、新しい視点をもたらし、コミュニティで十分に代表されていないグループの声を増幅するような著者を選ぶように努めました。

今後、このプロセスを繰り返し、Web Almanacがあらゆるバックグラウンドの貢献者による、より多様で包括的なプロジェクトとなることを期待します。

分析

2021年5月と6月に、データアナリストは著者や査読者とともに、各章で照会が必要となる指標のリストを作成した。場合によっては、カスタム指標[1058]が、私たちの分析能力のギャップを埋めるために作成されることもありました。

2021年7月の間、HTTP Archiveデータパイプラインは数百万のウェブサイトをクロールし、Web Almanacに使用されるメタデータを収集しました。これらの結果は、後処理をしてBigQuery[1059]に保存されました。

3年目ということで、以前のアナリストが書いたクエリを更新して再利用することができました。それでも、ゼロから書く必要のある新しいメトリクスがたくさんありました。GitHubのオープン ソース query repository[1060] で、すべてのクエリを年別、章別に閲覧できます。

解釈

著者はアナリストと協力し、結果を正しく解釈し、適切な結論を導き出しました。著者はそれぞれの章を執筆する際、ウェブの現状を説明するために、これらの統計データを参考にしました。査読者は、著者の分析が技術的に正しいことを確認するために、著者と協力しました。

読者に結果をより分かりやすく伝えるため、ウェブ開発者とアナリストがデータビジュアライゼーションを作成し、章に埋め込んでいます。いくつかのビジュアライゼーションは、ポイントをより明確にするため簡略化されています。たとえば、分布をすべて表示するのではなく、ごく一部のパーセンタイルだけを表示する。とくに断りのない限り、すべての分布は平均値ではなく、パーセンタイル、とくに中央値（50パーセンタイル）を使って要約されて

1058. https://github.com/HTTPArchive/custom-metrics
1059. https://console.cloud.google.com/bigquery?p=httparchive&d=almanac&page=dataset
1060. https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2021

います。

最後に、編集者は簡単な文法的な誤りを修正し、読書体験全体の一貫性を確保するために各章を改訂しました。

今後に向けて

Web Almanacの2021年版は、ウェブ・コミュニティにおける内省と前向きな変化へのコミットメントという毎年の伝統として続けていきたいと考えていることの3回目です。ここまで来るには、多くの献身的な貢献者のおかげで大変な努力をしました。この仕事をできるだけ活用して、将来の版をさらに合理的にしたいと思います。

Web Almanacの2022年版への貢献にご興味のある方は、関心フォーム[1061]にご記入をお願いします。一緒にウェブの状況を追跡しましょう!

1061. https://forms.gle/55uatdX9T3JZG2837

付属資料 B
貢献者








Web Almanacは、ウェブ・コミュニティの努力によって実現しています。2021 Web Almanac々は、企画、調査、執筆、制作の段階で122人が数え切れないほどの時間をボランティアで費やしてきました。



Abby Tsai
AbbyTsai
開発者



Adam Argyle
@argyleink
argyleink
https://nerdy.dev
レビュワー



Addy Osmani
@addyosmani
addyosmani
https://www.addyosmani.com
レビュワー



Adriana Jara
@tropicadri
tropicadri
レビュワー



Akshay Ranganath
@rakshay
akshay-ranganath
akshayranganath
https://akshayranganath.github.io/
分析者 と レビュワー



Alan Kent
@akent99
alankent
https://alankent.me
レビュワー



Alba Silvente Fuentes
@dawntraoz
Dawntraoz
https://www.dawntraoz.com/
レビュワー



Alex Lakatos
@avolakatos
AlexLakatos
http://alexlakatos.com/
著作者



Alex Tait
@at_fresh_dev
alextait1
https://atfreshsolutions.com
著作者



Alon Kochba
@alonkochba
alonkochba
alonkochba
著作者



Alon Zakai
@kripken
kripken
レビュワー



Andrea Volpini
@cyberandy
cyberandy
https://wordlift.io/blog/en/entity/andrea-volpini
著作者



Andrey Lipattsev
@AndreyLipattsev
andreylipattsev
レビュワー



André Cipriani Bandarra
@andreban
andreban
レビュワー



Andy Davies
@AndyDavies
andydavies
http://andydavies.me/
レビュワー



Artem Denysov
@denar90_
denar90
分析者 と 著作者



Ashley Berman Hale
ashleyish
著作者



Barry Pollard
@tunetheweb
https://webperf.social/@tunetheweb
tunetheweb.com
tunetheweb
tunetheweb
https://www.tunetheweb.com
分析者、 著作者、 開発者、 編集者、 リーダー と レビュワー



Brian Kardell
@briankardell
https://toot.cafe/@bkardell
bkardell
https://bkardell.com
レビュワー



Caleb Queern
@httpsecheaders
cqueern
レビュワー



Carlie Dixon
cdixon83
レビュワー



Carlo Piovesan
@carlop54002226
carlopi
レビュワー



Cassey Lottman
clottman
https://cassey.dev/
レビュワー



Chris Lilley
@svgeesus
svgeesus
https://svgees.us
レビュワー



Chris Sater
christophersater
レビュワー



Christian Liebel
@christianliebel
https://mastodon.cloud/@christianliebel
christianliebel
christianliebel
https://christianliebel.com
著作者



Dave Smart
https://seocommunity.social/@dwsmart
tamethebots.com
dwsmart
davewsmart
https://tamethebots.com
著作者



David Fox
@theobto
foxdavidj
https://www.lookzook.com
分析者、 リーダー と レビュワー



Demian Renzulli

@drenzulli
demianrenzulli
分析者 と 著作者



Dominic Lovell

@dominiclovell
dominiclovell
著作者



Doug Sillars

@dougsillars
dougsillars
https://dougsillars.com
分析者 と 著作者



Edmond W. W. Chan

edmondwwchan
https://edmondwwchan.github.io/
レビュワー



Eric A. Meyer

@meyerweb
meyerweb
http://meyerweb.com/
著作者



Eric Bailey

@ericwbailey
ericwbailey
https://ericwbailey.design/
レビュワー



Eric Portis

@etportis
ericportis.com
eeeps
https://ericportis.com
分析者 と 著作者



Estelle Weyl

@estellevw
estelle
http://standardista.com
レビュワー



Eugene Kliuchnikov

eustas
レビュワー



Fili Wiese

@filiwiese
fili
filiwiese
https://fili.com/
レビュワー



Gary Wilhelm

gwilhelm
著作者



Gertjan Franken

@GJFR_
gjfr
分析者



Gigi Rajani

GigiRajani
レビュワー



Greg Brimble

@gregbrimble
GregBrimble
https://gregbrimble.com/
分析者



Harry Roberts

@csswizardry
csswizardry
https://csswizardry.com/
レビュワー



Hemanth HM

@gnumanth
hemanth
http://h3manth.com
レビュワー



Ian Lurie

@ianlurie
wrttnwrd
https://www.ianlurie.com
著作者



Ingvar Stepanyan

@RReverser
RReverser
https://rreverser.com/
分析者 と 著作者



Iulia Comșa

iulia-m-comsa
https://sites.google.com/view/iuliacomsa/
レビュワー



JR Oakes

@jroakes
jroakes
分析者



Jamie Indigo

@Jammer_Volts
not-a-robot.com
fellowhuman1101
jamie-indigo
https://not-a-robot.com
著作者 と レビュワー



Jarno van Driel

@JarnoVanDriel
jvandriel
jarno-van-driel-36a47075
編集者



Jarrod Overson

jsoverson
http://jarrodoverson.com/
レビュワー



Jasmine Drudge-Willson

@jasminedwillson
JasmineDWillson
編集者



Jeff Posnick

@jeffposnick
jeffposnick
https://jeffy.info
レビュワー



Jens Oliver Meiert

@j9t
https://mas.to/@j9t
meiert.com
j9t
meiert
https://meiert.com/en/
レビュワー



Jess Peck

@jessthebp
jessthebp
https://jessbpeck.com/
分析者



Jessica Nicolet

@jessica_nicolet
jessnicolet
https://www.jessicanicolet.com/
著作者



John Teague

@jtteag
logicalphase
https://gemservers.com
著作者 と レビュワー



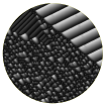
Jono Alderson

@jonoalderson
jonoalderson
https://www.jonoalderson.com
著作者



Julia Yang

@Jules_Yang
jzyang
yangzhe
編集者 と レビュワー



Jyrki Alakuijala

@jyzg
jyrkialakuijala
著作者



Kai Hollberg

@schweinepriestr
Schweinepriester
レビュワー



Katriel Paige

kachiden
https://www.flowerstorm.tech/
著作者



Kevin Farrugia

@imkevdev
https://webperf.social/@kevinfarrugia
kevinfarrugia
imkevdev
https://imkev.dev
分析者、 著作者 と レビュワー



Koen Van den Wijngaert

@vdwijngaert
vdwijngaert
https://www.neok.be/
レビュワー



Lea Verou

@leaverou
LeaVerou
https://lea.verou.me/
レビュワー



Leonardo Zizzamia

@Zizzamia
Zizzamia
著作者



Lode Vandevenne

lvandeve
著作者



Lucas Gonçalves

lucasbona05
開発者



Manuel Garcia

@corrosion_pt
soulcorrosion
manuel-garcia-12b6928
https://farfetchtechblog.com/en/blog/authors/manuel-garcia/
レビュワー



Matteo Große-Kampmann

@pizzahax
lord-r3
https://lord-r3.github.io/
レビュワー



Maud Nalpas

maudnals
レビュワー



Max Ostapenko

max-ostapenko
max-ostapenko
https://maxostapenko.com
分析者



Maxim Salnikov

@webmaxru
webmaxru
https://medium.com/@webmaxru
レビュワー



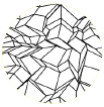
Michelle O'Connor

デザイナー



Minko Gechev

@mgechev
mgechev
https://blog.mgechev.com/
レビュワー



Moritz Firsching

mo271
https://mo271.github.io/
著作者



Mukesh Jat

mukeshjat
翻訳者



Navaneeth Krishna

@Navanee55755217
Navaneeth-akam
著作者 と レビュワー



Nikita Dubko

@dark_mefody
MeFoDy
https://mefody.dev/
翻訳者



Nishu Goel

@TheNishuGoel
NishuGoel
https://unravelweb.dev/
著作者



Nitin Pasumarthy

Nithanaroy
nitinpasumarthy
https://nithanaroy.medium.com/
分析者



Nurullah Demir

@nrllah
nrllh
https://ndemir.com
著作者



Olu Niyi-Awosusi

@oluoluoxenfree
oluoluoxenfree
https://olu.online/
著作者



Pankaj Parkar

@pankajparkar
pankajparkar
https://pankajparkar.dev
分析者、 編集者 と レビュワー



Pascal Schilp

thepassle
レビュワー



Patrick Hulce

@patrickhulce
patrickhulce
http://patrickhulce.com
レビュワー



Patrick Stox

@patrickstox
patrickstox
https://patrickstox.com
著作者



Paul Calvano

@paulcalvano
paulcalvano
https://paulcalvano.com
分析者 と リーダー



Phil Barker

@philbarker
philbarker
https://blogs.pjjk.net/phil/
レビュワー



Rajiv Ramnath

rrajiv
rajivramnath
分析者



Rebecca Holmlund

RMHolmlund
レビュワー



Rick Viscomi

@rick_viscomi
rviscomi
https://rviscomi.dev/
分析者、 編集者、 リーダー と
レビュワー



Rob Teitelman

@teitelmanrob
SeoRobt
https://www.paulteitelman.com/
レビュワー



Robin Marx

@programmingart
programmingart.bsky.social
rmarx
rmarx
http://internetonmars.org
レビュワー



Rockey Nebhwani

@rnebhwani
rockeynebhwani
rockeynebhwani
レビュワー



Rory Hewitt

@roryhewitt3
roryhewitt
roryhewitt
https://romche.com
レビュワー



Ruth Everett

@rvtheverett
rvth
分析者



Sakae Kotaro

@beltway7
ksakae1216
https://ksakae1216.com/
翻訳者



Samar Panda

samarpanda
レビュワー



Saptak Sengupta

@Saptak013
saptaks
https://saptaks.website/
著作者 と 開発者



Scott Davis

@scottdavis99
scottdavis99
http://thirstyhead.com
著作者



Shaina Hantsis

shantsis
デザイナー、 編集者 と レビュワー



Shilpa Raghunathan

boosef
レビュワー



Shuvam Manna

@shuvam360
GeekBoySupreme
https://shuvam.xyz
著作者 と デザイナー



Sia Karamalegos

https://front-end.social/@sia
sia.codes
siakaramalegos
karamalegos
https://sia.codes
分析者、 著作者 と レビュワー



Simon Hearne

@simonhearne
simonhearne
https://simonhearne.com
レビュワー



Tamas Piros

@tpiros
tpiros
https://tamas.io
レビュワー



Thom Krupa

@thomkrupa
thomkrupa
https://www.thomkrupa.com/
レビュワー



Thomas Fischbacher

fischbacher
レビュワー



Thomas Steiner

@tomayac
tomayac
https://blog.tomayac.com/
分析者 と レビュワー



Timur Kartashov

@krutoi_paren
kartashovio
kartashov-io
https://kartashov.io/
翻訳者



Tom Robertshaw

@bobbyshaw
bobbyshaw
tomrobertshaw
https://www.space48.com
著作者



Tom Van Goethem

@tomvangoethem
tomvangoethem
https://tom.vg/
著作者



Tomek Rudzki

@TomekRudzki
Tomek3c
https://tomekseo.com/
著作者



Tosin Arasi

tosinarasi
分析者



Victor Le Pochat

@VictorLePochat
VictorLeP
victor-le-pochat
https://lepoch.at
分析者、 著作者 と 翻訳者



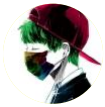
Weston Ruter

@westonruter
westonruter
https://weston.ruter.net/
レビュワー



Wilhelm Willie

WilhelmWillie
レビュワー



William Sandres

@hakacode
HakaCode
https://hakacode.github.io
翻訳者



Yana Dimova

ydimova
著作者



Yusuf Seyhan

@yuseyhan
yuseyhan
https://webpen.de/
デザイナー



Ziemek Bućko

ziemek-bucko
レビュワー