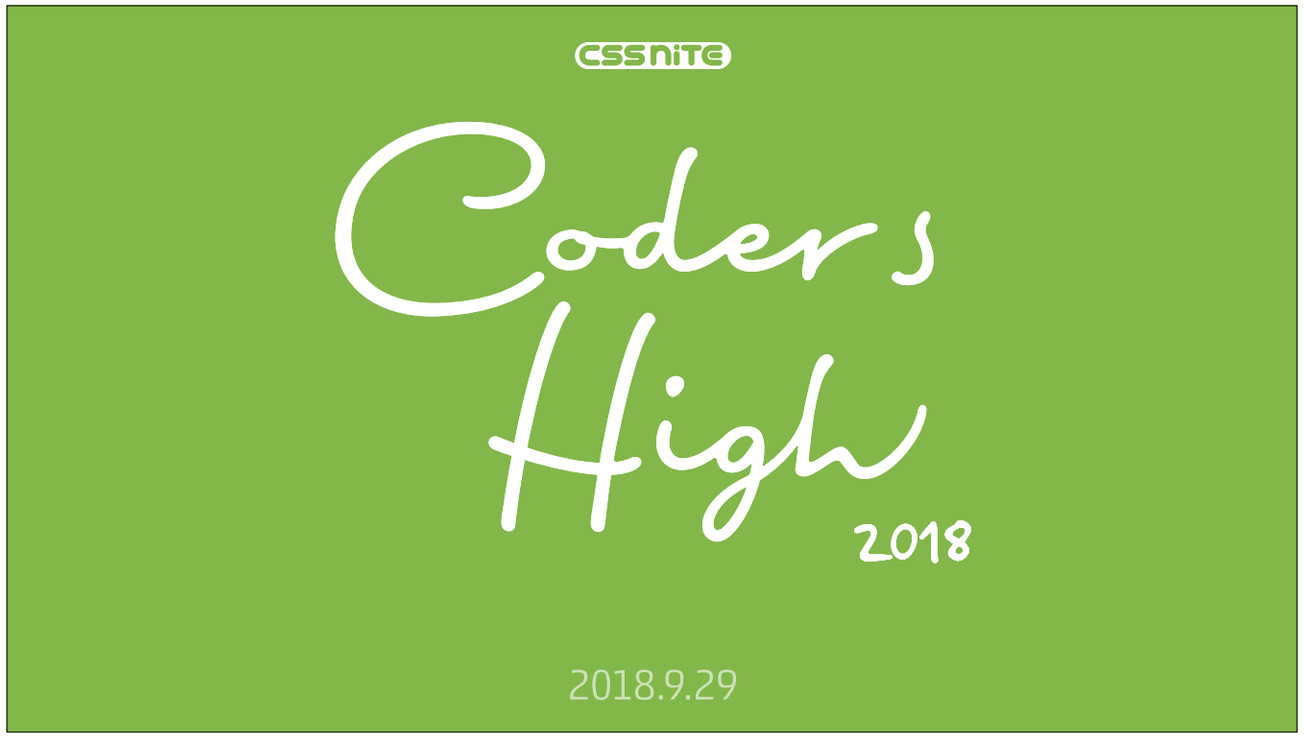

CSSnite
Coders High
2018
2018.9.29

フロントエンドで
サイトスピードUP！

ism / PentaPROgram 佐藤あゆみ

まずは自己紹介をさせてください。
私は、ペンタプログラム　佐藤あゆみと申します。
「さとペン」と呼んでください！

1997年、12歳頃から、趣味でWeb制作を開始しました。
2014年より、フリーランスのコーダーとして活動しています。

ECサイト
ブログ
コーポレートサイト

ECサイトのテンプレート制作、ブログのコーディングや、コーポレートサイトの制作、運用などを行っています。
また、Web制作に加えまして、

ismという会員組織の中で、高速化の担当として、入門セミナーや高速化サービスも提供しています。
https://www.internet-strategy-marketing.org/

皆さんの手元にお配りしている資料の中に、私のセミナーの案内が入っていますので、ぜひご覧ください。
※駒込会の日程は10/9(火)になりました。
https://www.facebook.com/events/1495941003840807/

1. 事例
2. 最も簡単で効果があるのは画像の最適化
3. そもそも、不要なコンテンツは消すべき！
4. その他のテクニック
5. FMP

こちらが、今回のセッションでお伝えする内容です。

前半に、必ず行って頂きたい、「画像の最適化」と「コンテンツの整理」についてお話します。
後半は、やらなくても良いけれど、やれば少しずつ速くなるテクニックをご紹介します。

それでは、

事例

静的HTMLサイト・WordPressサイト

さっそく、高速化の改善事例をご紹介します。

まず、完全静的なコーポレートサイトの改善例です。

こちらはismを運営するゴンウェブコンサルティングのコーポレートサイトです。
改善前のサイトは、Googleの採点ツールPageSpeed Insightsでは68点でした。

静的HTMLサイトの高速化

✦画像の圧縮

✦ブラウザキャッシュの設定

✦圧縮配信の設定

✦CSS & JavaScriptの非同期読込

- 複数のCSSファイルを1つに結合
- CSS & JavaScriptのMinify

このようなテクニックで高速化しました。
各テクニックについては、事例の後にご説明するのでご安心くださいね。

まず、画像を圧縮しました。
ブラウザキャッシュを設定し、テキストファイルを圧縮配信し、CSSとJavaScriptを非同期読込にしました。

また、手間の割に効果があまりに小さいため今回の講演では省略しますが、
・複数のCSSファイルを1つのファイルにまとめることで通信回数を減らして速くする
・CSSとJavaScriptから改行やコメントを取り除いて速くする、Minifyという対策
もついでに行いました。

このような対策を施したところ…

99点まで改善することができました。
もちろん体感的にも早くなっています。

実は、この改善結果をみた方から「これが完全静的じゃないならすごいけどね〜」というご意見を頂きました。
でも、実は動的なサイトでも同じテクニックを使用して高速化できるんです。
次は動的にWordPressで生成しているサイトの高速化をご紹介します。

縄合屋さんという、しめ縄を販売している会社のサイトです。

今回は完全静的だったサイトをWordPressに作り替えるというリニューアルオープンのため、改善前のスコアというのが存在しません。

WordPressサイトの高速化

✦画像の圧縮(EWWW Image Optimizer)

✦ブラウザキャッシュの設定

✦圧縮配信の設定

✦CSS & JavaScriptの非同期読込

- 複数のCSSファイルを1つに結合
- CSS & JavaScriptのMinify

今回の案件では、テーマ内の画像は予め圧縮したものをアップロードしましたが、
それ以外の画像はWordPressの管理画面からアップロードするため、EWWW Image Optimizerというプラグインを入れました。

ブラウザキャッシュ、圧縮配信、CSSとJavaScriptの非同期読込とMinifyを行いました。
このような対策を施したところ…

99点です。
先ほどのゴンウェブコンサルティングのサイトと同じ点数です。
WordPressでも、同じ方法で高速化を行えるということです。

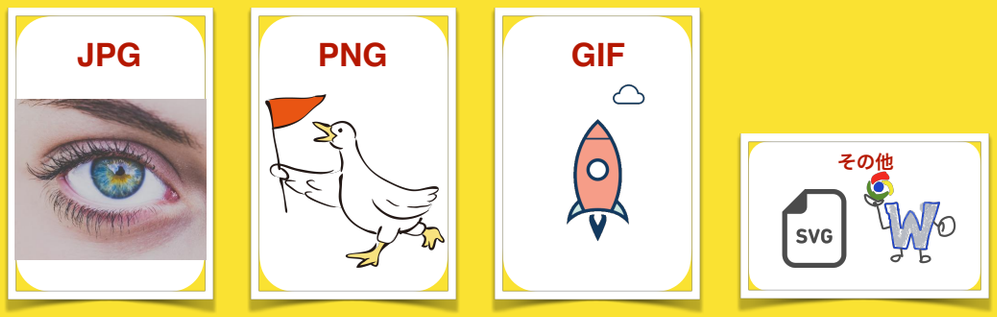
それでは本編に入ります。

これからご紹介する中で、一番簡単に効果を体感できるのが、画像を最適化することです。

画像のテクニック①

最適な形式はどれ？

ウェブで使われる画像の種類



現在、日本のウェブサイトで主に使われている画像の形式はJPEG・PNG・GIFの3つです。
適していない形式を使用すると、大幅にファイルサイズが増えてしまいますので、
まずは基本中の基本から、さっとおさらいしましょう。

	色数	透過	形式	アニメ	軽さ	
JPEG	○	×	ラスター	×	○	**写真を載せるならコレ一択**
PNG-8	△	△	ラスター	×	◎	**色数の少ないイラスト・ロゴならコレ** 透過はできるけど半透明を表現できない
PNG-24	○	×	ラスター	×	△	重いけど劣化しない
PNG-32	○	◎	ラスター	×	△	**イラストを透過させたいならコレ**
GIF	△	△	ラスター	○	○	**アニメーションさせるならコレ** 透過はできるけど半透明を表現できない
SVG	○	◎	ベクター	○	◎	**カンタンシンプルな図形ならコレ** 入り組んだ複雑なイラストは重くなる
WebP	○	◎	ラスター	○	◎	対応ブラウザが少なすぎてまだ×

JPGは、写真を載せるとき
PNGは、イラストやロゴ、図を載せるとき
GIFは、ちょっとしたパラパラ漫画のようなアニメーションを載せたいとき
に使います。

SVGは、ベクター形式という拡大縮小しても荒れない便利な形式で、レスポンシブデザインの広がりを受けて徐々に広まりつつあります。
ただ、JPG/PNG/GIFがラスター形式なのと比べ、ベクター形式のSVGは画像加工ソフトやブラウザ上での扱い方が異なるため、使い方を事前に学習しておく必要があります。
また、WebPに関しては、軽量化に優れた方式ですが、対応ブラウザが少ないため、まだ現場で使うのは難しいです。

画像のテクニック②

画像リサイズ

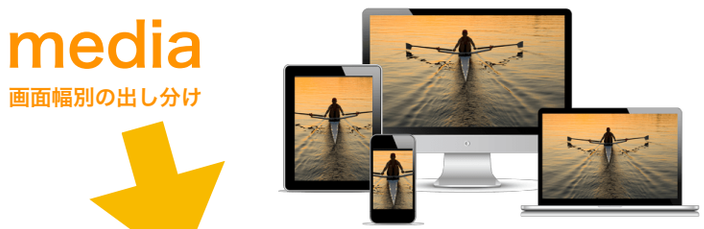
適したファイル形式を選んだら、必ず(Click)リサイズ・縮小して保存しましょう。

こちらは、デジタルカメラで撮影して書き出したままの画像で、
横幅3008px、サイズは6.7MBあります。
たとえば、これを実際のページ上での表示幅である横幅680pxにしてみます。

この作業で368KBまで小さくなりました。

デジカメの画像をそのままアップしないよ？と思われる方もいるかもしれませんが、私が今までお預かりしてきたサイトではそういった例を何回も見かけたことがあります。
また、いつかリサイズするつもりで入れていた仮の画像ファイルを、うっかりそのまま本番に掲載してしまうこともあるようですので、見直しましょう。

media&srcset属性




```
<picture>
  <source media="(min-width: 800px)" srcset="head.jpg, head-2x.jpg 2x">
  <source media="(min-width: 450px)" srcset="head-small.jpg, head-small-2x.jpg 2x">
  <img src="head-fb.jpg" srcset="head-fb-2x.jpg 2x" alt="a head carved out of wood">
</picture>
```

画像引用元）
https://developers.google.com/web/fundamentals/design-and-ux/responsive/images?hl=ja

また、画像を呼び出すHTML側にて、mediaやsrcset属性を使うと、画面解像度や画面幅に応じて画像を出し分けられます。
例えば、画面解像度が2倍の端末には、srcset属性（Click）で2倍の解像度の画像を出せます。
また、media属性（Click）で画面の幅に応じて適した横幅の画像を出し分けられます。

ただし！！
必ずそうしなければいけないと言うことはありません。
この画像は美しく見せたいという勝負どころでは解像度の高い画像を出し、それ以外ではあえて低解像度の画像を使用することで、管理が楽になりますし、表示も速くなります。

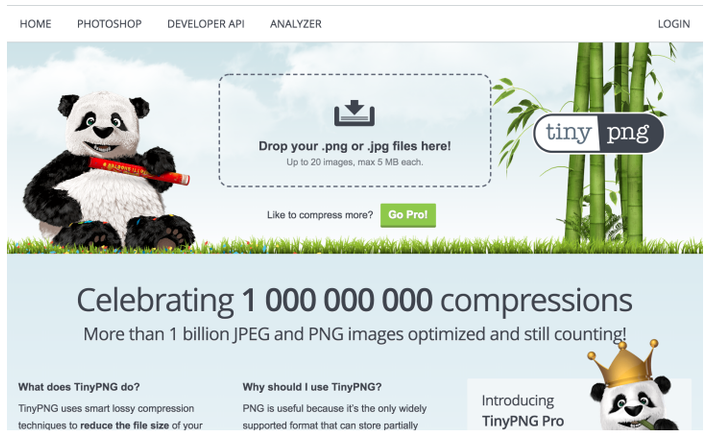
画像のテクニック③

画像圧縮

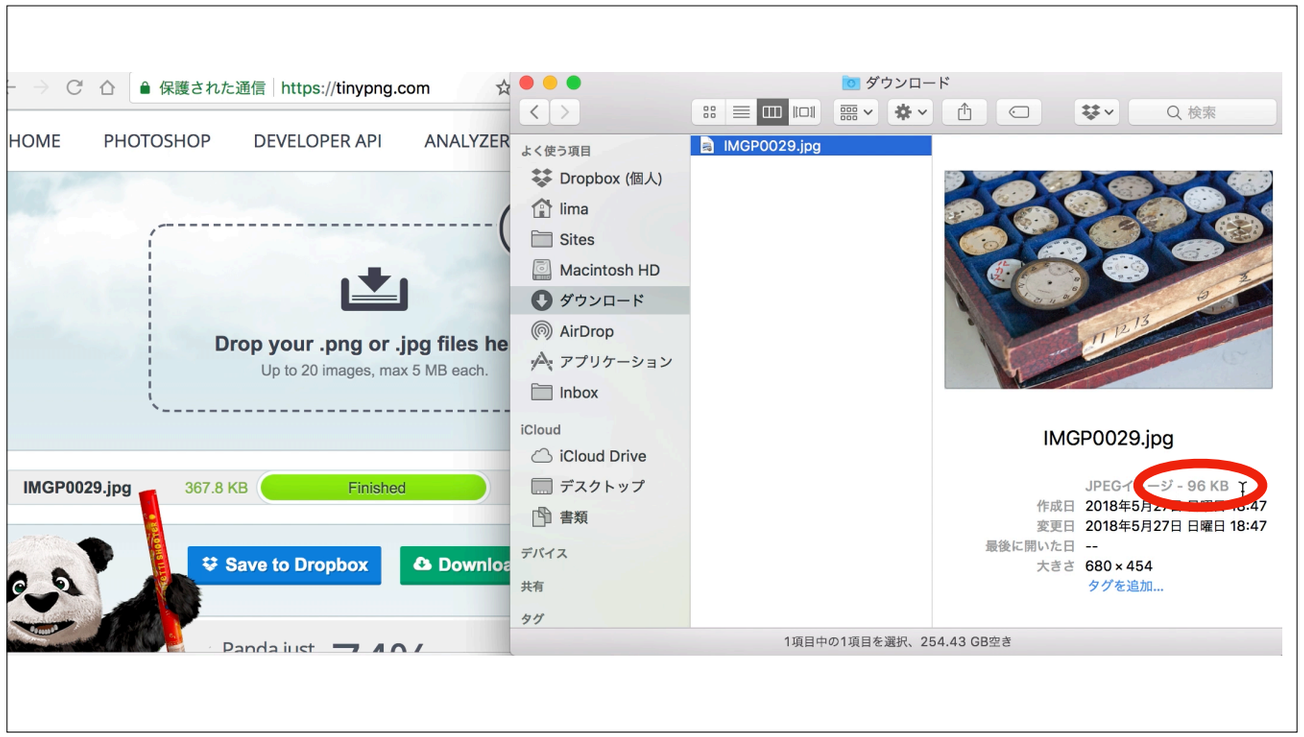
これから、さらに、画像を圧縮(Click)してみます。

圧縮アプリケーションを使う

- TinyPng
  （ブラウザから使用可）
  https://tinypng.com/



このTinyPNGというサイトからブラウザで画像をアップロードすると、ファイルサイズが減らされた画像をダウンロードできます。
TinyPNGという名称ですが、PNGだけでなく、JPEGも対応しています。



さっそく、さきほどリサイズ済みの画像を圧縮してみます。
（Click）
完了したJPGをダウンロードすると、96KBに軽量化されました。
このTinyPNGは、Photoshopで有料プラグインとしても利用できます。

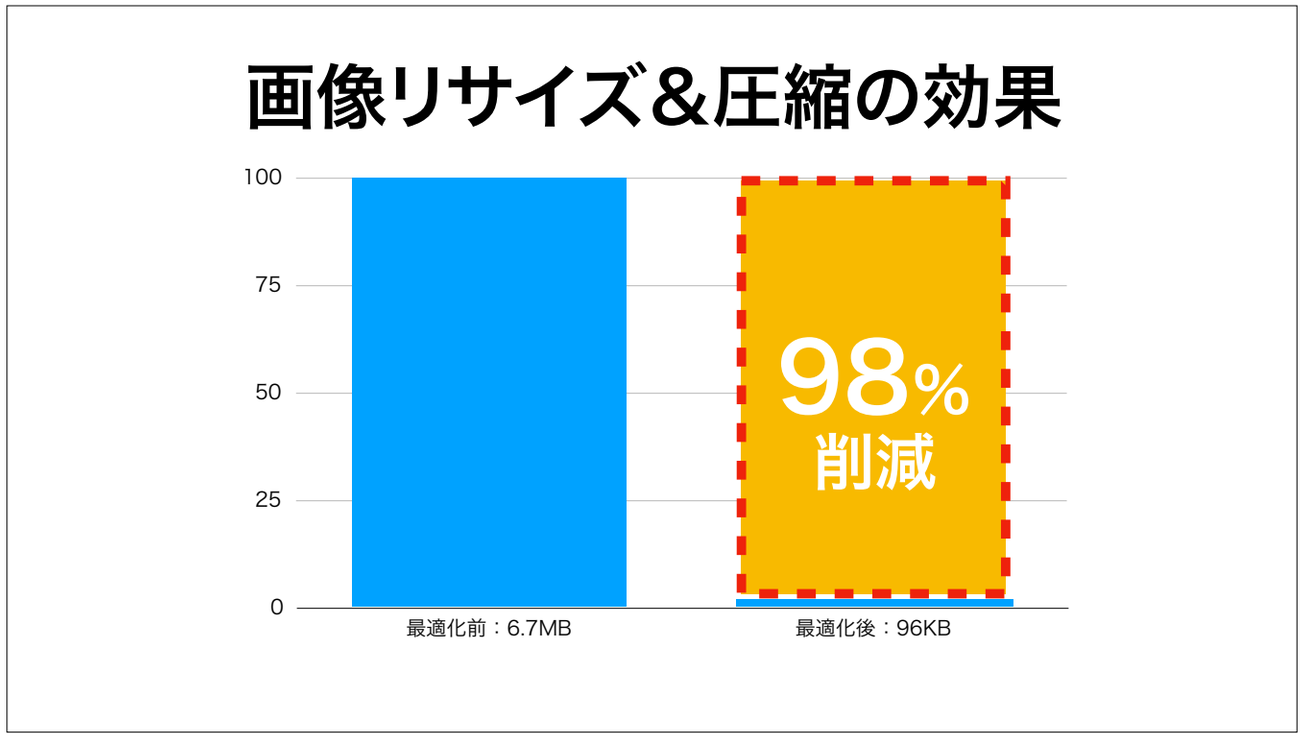
圧縮アプリケーション

- **ImageOptim**（Mac用無料ソフト）
  https://imageoptim.com/mac
- **Antelope**（Windows用無料ソフト）
  http://www.voralent.com/ja/products/antelope/
- **EWWW Image Optimizer**（WordPressプラグイン）
  https://ja.wordpress.org/plugins/ewww-image-optimizer/
- **pngquant、Jpegoptim**など（無料コマンドラインツール）

もちろん単体のネイティブアプリでも同様のことができます。
Macをお使いでしたら「ImageOptim」、Windowsをお使いでしたら「Antelope」というフリーウエアがあります。

また、WordPressも、プラグインを利用すると、画像をアップロードするたびにその画像を自動で圧縮してくれるようになります。

コマンドラインを使える方には、pngquantとJpegoptimがおすすめです。
さて、リサイズや圧縮する前の、元画像の6.7MBと比較すると



（Click）
圧縮大成功！およそ98%もファイルサイズを減らせたことになります。

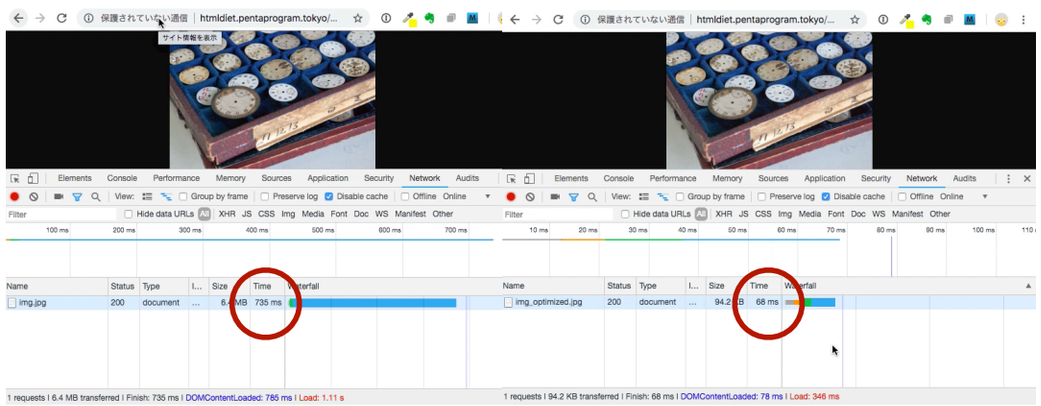
画像リサイズ＆圧縮の効果

圧縮前　　圧縮後

さて、いったい、どれぐらい表示が速くなるのか。
アクセスするタイミング等で通信速度は変わりますのであくまで参考としてご覧頂きたいのですが…
…あっという間なので、よく見ていてくださいね。

画像リサイズ&圧縮の効果

圧縮前　　圧縮後



（Click）（Click）
圧縮前のファイルは735ミリ秒、圧縮後は68ミリ秒ですので、およそ10倍の速さです。
ここまで違うと、目視でも分かりますね。
画像1枚でこの違いですから、ページ内に何枚も画像があれば、その分だけ違いが出て来ます。
画像についてはこれで終わります。

そもそも、不要なコンテンツは消すべき！
ページから不要なコンテンツを削ることで、根本から高速化します。

（Click）
たとえば、動画や特に意味もなく動くだけのJavaScript等はないでしょうか。
これらのコンテンツはページの表示や実行速度に大きな影響を与えます。
たとえば、動かないキービジュアル1枚画像にした方が、表示が速くなる上に気が散らないので、伝えたいメッセージをより強く伝えられる可能性もありますよ。

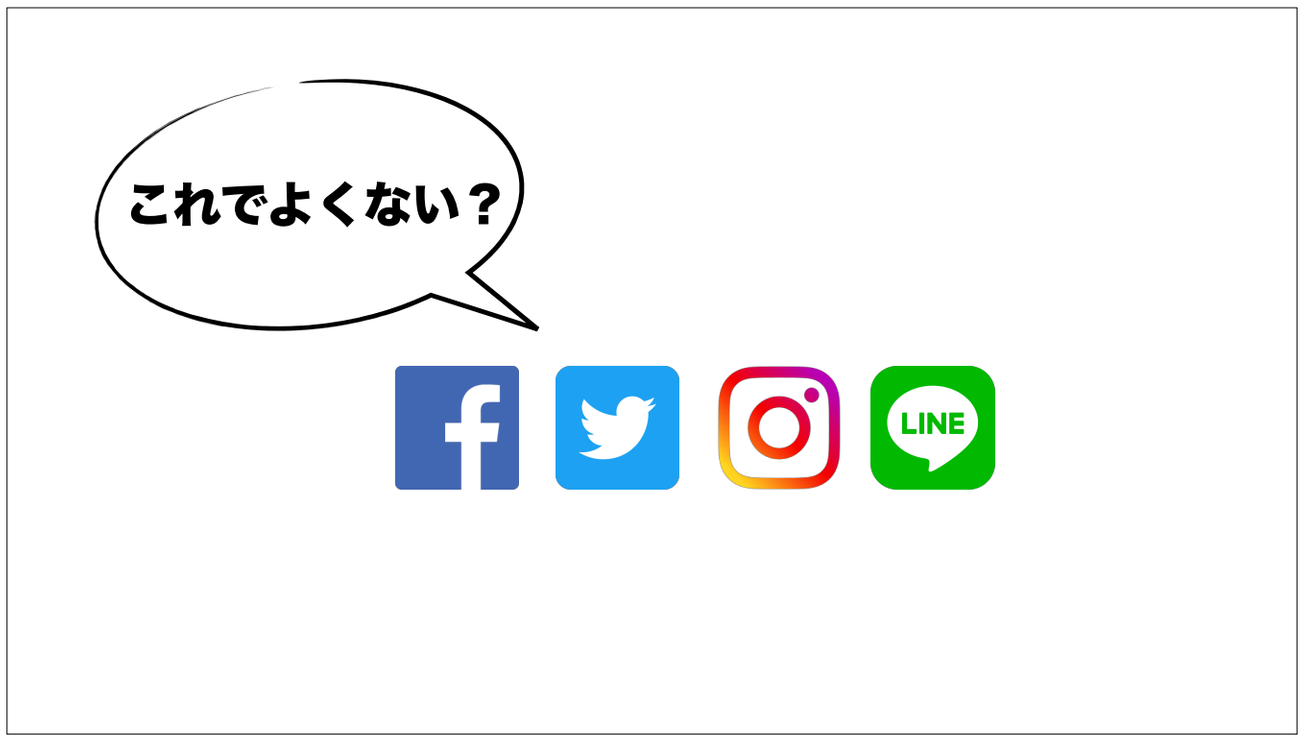
(Click)
例えば、すでに新着ではない新着情報はありませんか？
古い情報はページ上から消しましょう。
思わぬムダを生んでいる可能性があります。
サムネイルやバナーが画像を使っている場合には、特に注意します。

長く運用しているサイトでは、担当者自身もそのコンテンツがある状態に見慣れてしまい、あるのが当たり前として感じられてしまいます。
一歩引いた目でウェブページを眺めましょう。

また、その他の例としては、FacebookやTwitterなどの埋め込みがあります。

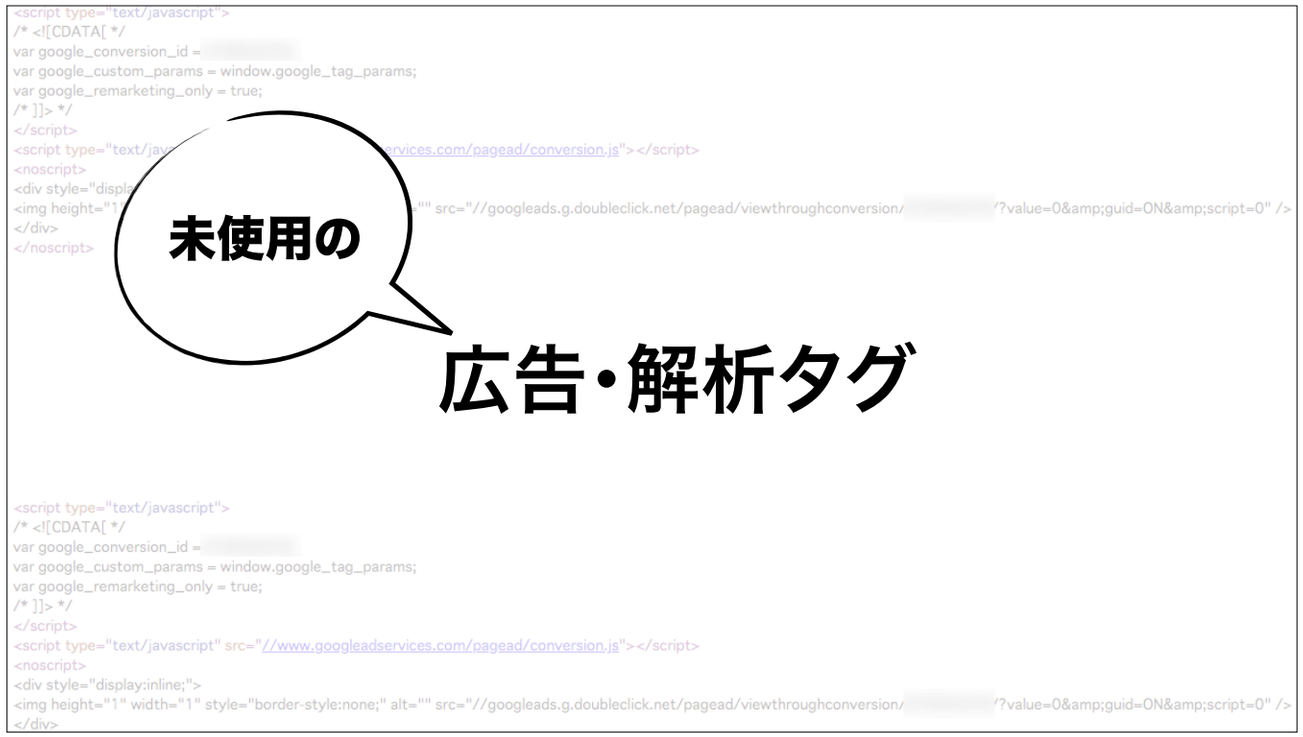
これらをページ内へ埋め込むと、ページの表示は遅くなります。
FacebookやTwitterのサーバに接続して、その都度、最新の投稿や、誰が「いいね！」したか等の情報を一つ一つダウンロードしてくる必要があるからです。

そこで、これらの埋め込みを外して…



単純なアカウントへのリンクに置き換えてしまった方が良いです。

もちろん、なんでもかんでも消せということではなく、
これらのSNSをフル活用しているなら残すという判断もできます。

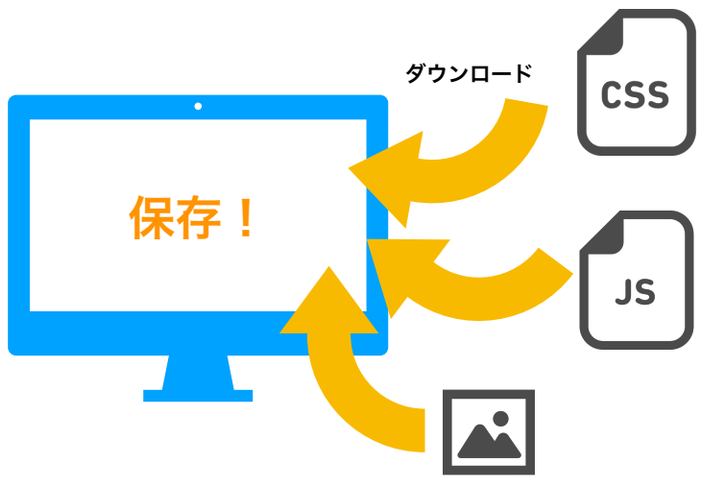


さらに、意外とページスピードを遅くしているのが、いま使っていないにもかかわらず、ページ内でそのままになっている、広告やアクセス解析系のタグです。

例えば、お試しで導入したものの本契約までは至らなかった解析サービスや、もう終了した広告用のタグなどです。
消すべきタグがないか、一度精査しましょう。

これから、やらなくても良いけれど、やれば少しずつ早くなって、採点サイトで高評価をもらえるテクニックをご紹介します。

ブラウザにキャッシュさせる



ブラウザにキャッシュさせる。

ウェブページを表示する際、ブラウザはサーバからファイルをダウンロードして（Click）、画面に表示します。
そこで、これらのファイルを一定期間ブラウザにキャッシュ、保存（Click）させてみましょう。

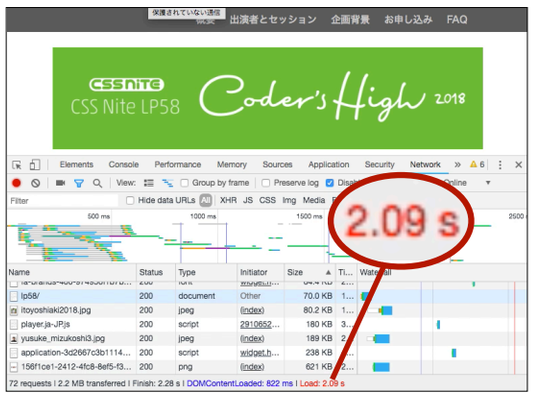
ブラウザキャッシュの効果

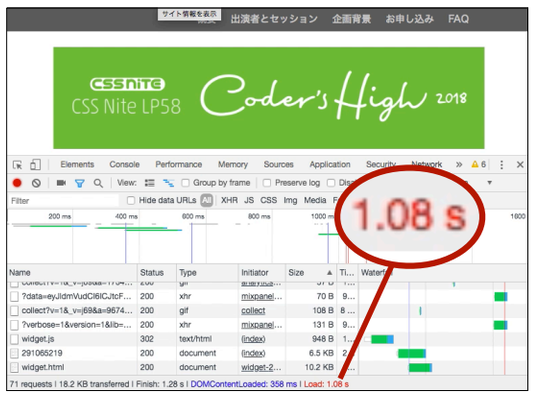
キャッシュ前　　キャッシュ後

キャッシュのあるなしで、どれぐらい速度が違うのか、
今回のイベントCSS Niteのページで試してみましょう。
極端な比較ですが、キャッシュを一切使わない場合と、使っている場合とで比較します。

ブラウザキャッシュの効果

キャッシュ前　　キャッシュ後





（Click）赤字のLoadを見ると分かりますが、キャッシュする前は2.09秒かかっていたものが、キャッシュを有効にすると1.08秒になりました。
およそ半分の時間になりました。

そこで、このキャッシュをうまく有効にするには、

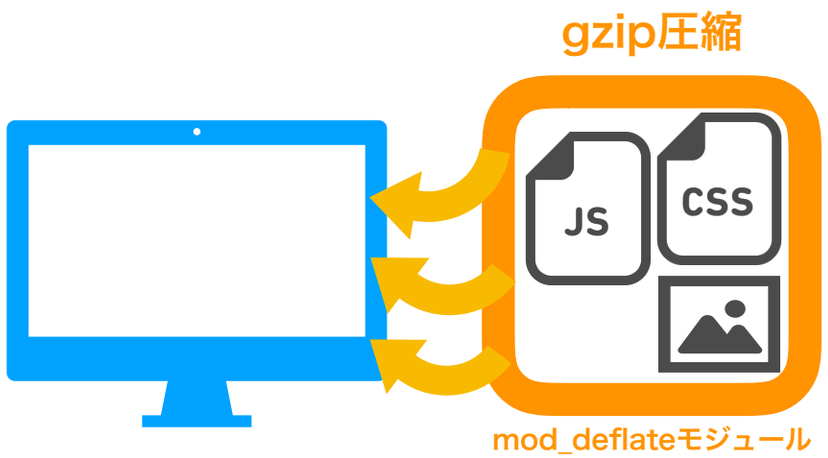
ブラウザにキャッシュさせる

```
# キャッシュ設定
<ifModule mod_expires.c>
  ExpiresActive On
  ExpiresDefault "access plus 7 days"
  ExpiresByType text/css "access plus 7 days"
  ExpiresByType text/javascript "access plus 7 days"
  ExpiresByType application/javascript "access plus 7 days"
  ExpiresByType application/x-javascript "access plus 7 days"
  ExpiresByType text/html "access plus 3 hours"
  ExpiresByType image/jpeg "access plus 1 month"
  ExpiresByType image/png "access plus 1 month"
  ExpiresByType image/gif "access plus 1 month"
</ifModule>
```

参考）Apache Module mod_expires
https://httpd.apache.org/docs/2.2/en/mod/mod_expires.html

コーダーの皆さんにもなじみ深い.htaccessというファイルがありますね。
このhtaccessに、ファイル形式ごとに、更新頻度に応じた有効期限をこのように記述するだけです。

mod_deflateでテキスト圧縮



さきほどは画像を圧縮しましたが、画像以外のファイル、例えばHTML、CSS、JavaScriptなども、圧縮してファイル容量を削減することができます。
mod_deflateモジュールのGzip圧縮で、サーバから転送するデータに対して圧縮をかけます。
（Click）

mod_deflateでテキスト圧縮

.htaccessに3行足すだけ

```
# 圧縮設定
<IfModule mod_deflate.c>
  AddOutputFilterByType DEFLATE text/html text/plain text/xml text/css text/javascript application/javascript
</IfModule>
```

引用元）Apache Module mod_deflate
https://httpd.apache.org/docs/2.2/en/mod/mod_deflate.html

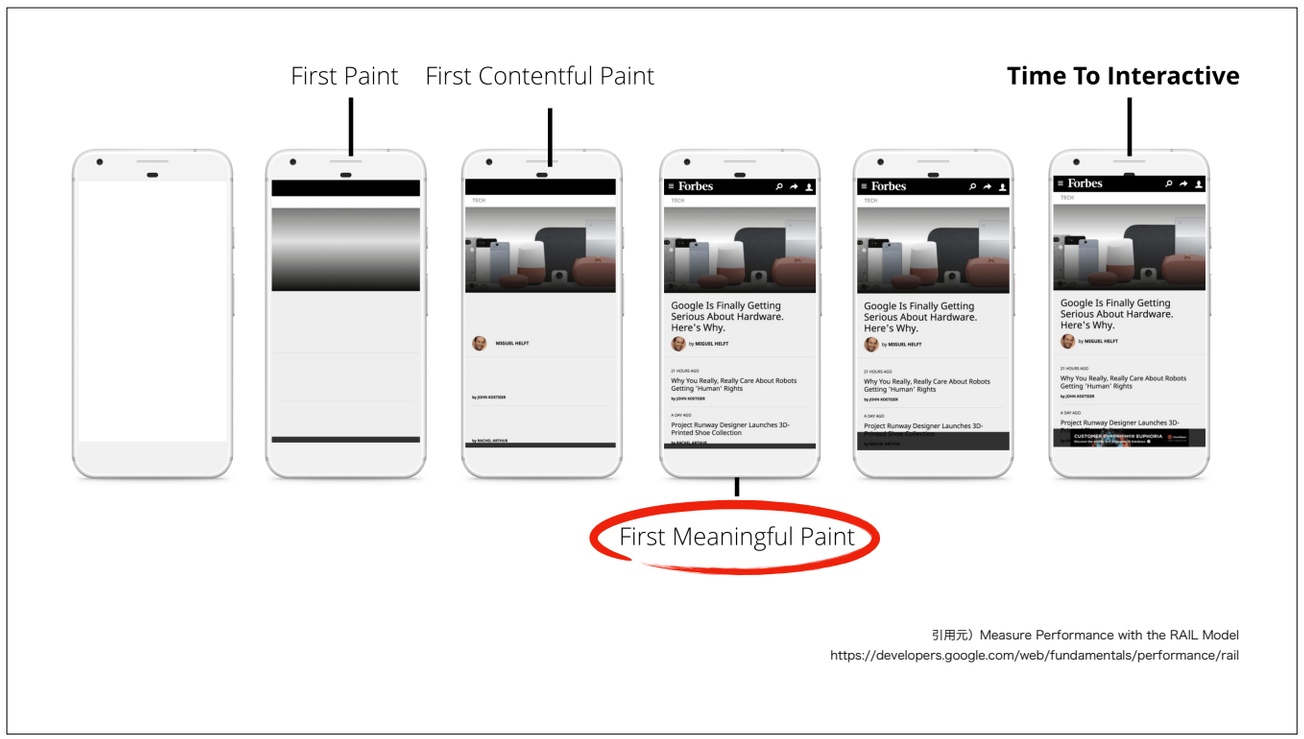
テキストを圧縮して配信するには、htaccessにこの3行を足すだけ！
圧縮したいテキストファイルの形式を指定します。
これで、テキストファイルのgzip圧縮が有効になります。

…さて、次のテクニックをご紹介する前に、

FMP

ファースト・ミーニングフル・ペイント

「FMP」ファースト・ミーニングフル・ペイント
という概念が大事になりますので、まずご説明します。



引用元）Measure Performance with the RAIL Model
https://developers.google.com/web/fundamentals/performance/rail

ファースト・ミーニングフル・ペイントとは、
「見る人にとって意味のある情報が表示されている状態」、その瞬間のことです。

左から、最初はまっ白、次に背景と、字も見え始めて、その次、赤丸の部分が「ファースト・ミーニングフル・ペイント」です。
できるだけ速く、この「見る人にとって意味のある情報が表示されている状態」にしよう、というのがこれからのお話になります。

参考）
https://developers.google.com/web/fundamentals/performance/rail
※英語にしないとみえない画像

JavaScriptの多くは、ページを表示する瞬間のファーストビュー、そして「ファースト・ミーニングフル・ペイント」には関わらない場合が多いです。
例えば、(click)一定時間たつとスライドショーが動いたり、スクロールしたらアニメーションで上に戻るボタンが出てくるようなものです。

しかしながら、ページをブラウザ上に描画するのは、通常はJavaScriptファイルを全部ダウンロードして、さらに解析し終わった後になります。

このため、JavaScriptを後回し、非同期にします。
まだ全てをダウンロード・解析し終わっていない時点でページの描画を開始してしまうことで、
ファーストビューを先に表示できます。

JavaScript非同期読込

6文字足すだけ

引用元）レンダリングを妨げる JavaScript を削除する
https://developers.google.com/speed/docs/insights/BlockingJS?hl=ja

具体的な方法として、async（エイシンク）属性を、今あるJavaScriptタグに書き足します。
(click)
直前の空白も含めて6文字足すだけ！
これだけで、簡単にスクリプトを非同期で読み込むことが出来ます。

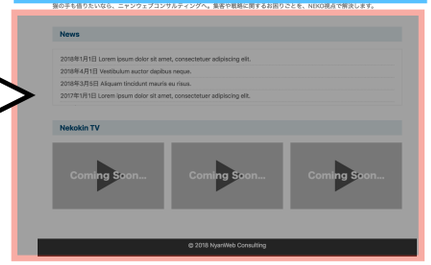
CSS非同期読込

ファーストビュー

FMPの影響範囲

下スクロールしないと表示されない領域

FMPに影響しない範囲



CSSファイルにも、JavaScriptと同じように、ファーストビュー・FMPには影響しないCSSがたくさん含まれています。
これも非同期にしてみましょう。

```
<!doctype html>
<html lang="ja">
<head>
<meta name="viewport" content="width=device-width,user-scalable=yes,maximum-scale=1">
<meta charset="UTF-8">
<title>NyanWeb Consulting | サンプルサイト</title>
<meta name="description" content="猫の手も借りたいなら、ニャンウェブコンサルティングへ。集客や戦略に関するお困りごとを、NEKO視点で解決します。">
<style>
html{font-size:62.5%;line-height:150%}body{font-family:-apple-system,BlinkMacSystemFont,"Helvetica Neue","Noto Sans Japanese","ヒラギノ角ゴ ProN W3",Hiragino Kaku Gothic
ProN,Arial,Meiryo,sans-serif;font-size:1.6rem;line-height:2;background:#665990}li{list-style-type:none}.globalHeader{padding:10px
0;background:#fff;border-bottom:solid 4px #F7C9CE;text-align:center;font-size:1.3rem;line-height:2}.globalHeader
.navigation{margin-top:10px}.globalHeader .navigation li{display:inline-block}.globalHeader .navigation li+li{margin-left:5px}.globalHeader
.navigation li
a{color:#F8ACAC}.inner_m{padding-left:20px;padding-right:20px;box-sizing:border-box}.goUp{display:none;position:fixed;bottom:30px;right:0}.goUp
img{width:100px;height:auto;opacity:0.8}body.top .sliderBackground{background:#FFF4E1;padding:0 30px;box-sizing:border-box;border-bottom:solid
4px #F8ACAC}body.top .slider img{width:100%;height:auto;max-width:980px}@media screen and
(min-width:768px){body{min-width:1020px}.globalHeader{text-align:left;font-size:1.6rem;line-height:2;padding:0}.globalHeader
.wrapper{width:940px;margin:auto;padding:10px 0;position:relative}.globalHeader .navigation{position:absolute;right:0;bottom:0}.globalHeader
.navigation li{display:block;float:left;background-color:#F7C9CE;border-radius:5px 5px 0 0;border-right:1px solid #F8ACAC}.globalHeader
.navigation li+li{margin-left:2px}.globalHeader .navigation li
a{text-decoration:none;color:#383838;display:block;width:100%;height:100%;padding:3px
20px}.inner_m{width:940px;padding-left:0;padding-right:0;margin-left:auto;margin-right:auto}.goUp{bottom:30px;right:15px}.goUp img{width:180px}}
</style>
</head>
<body class="top">
<header class="globalHeader">
<nav class="cf wrapper">
<a href="#" class="logo"><img src="img/common/logo.png" alt="NyanWeb Consulting"></a>
```

このサンプルコードのように、HTMLのheadタグ内に、ファーストビューで見えている部分のCSS「だけ」を直接書き込んでいます。

```
  <noscript id="deferred-styles">
    <link rel="stylesheet" type="text/css" href="small.css"/>
  </noscript>
  <script>
    var loadDeferredStyles = function() {
      var addStylesNode = document.getElementById("deferred-styles");
      var replacement = document.createElement("div");
      replacement.innerHTML = addStylesNode.textContent;
      document.body.appendChild(replacement)
      addStylesNode.parentElement.removeChild(addStylesNode);
    };
    var raf = window.requestAnimationFrame || window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame || window.msRequestAnimationFrame;
    if (raf) raf(function() { window.setTimeout(loadDeferredStyles, 0); });
    else window.addEventListener('load', loadDeferredStyles);
  </script>
</body>
</html>
```

引用元）CSS の配信を最適化する
https://developers.google.com/speed/docs/insights/OptimizeCSSDelivery?hl=ja

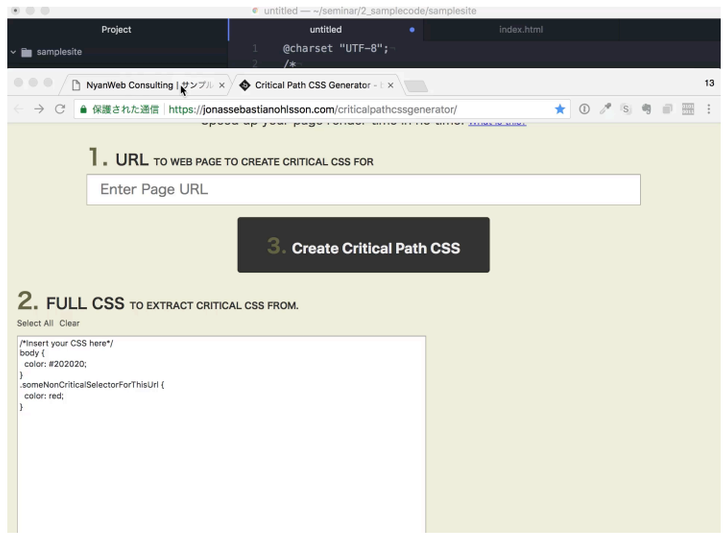
それ以外のスタイル、CSSはHTMLの末尾でJavaScriptを使って別途呼び出します。
このように、CSSの読み込みにJavaScriptを使うことで、先ほどの「JavaScriptの非同期読込」と同様に、CSSの読み込みを非同期、後回しにできます。

CSS非同期読込

こうすることで、ファーストビューに入る部分のCSS、ヘッドタグ内に書いてあるCSSをまず最優先で表示できます。

ただ、このファーストビューに影響するCSSというのが、CSSファイル内の一体どこなのか、切り分けが大変な時もあります。

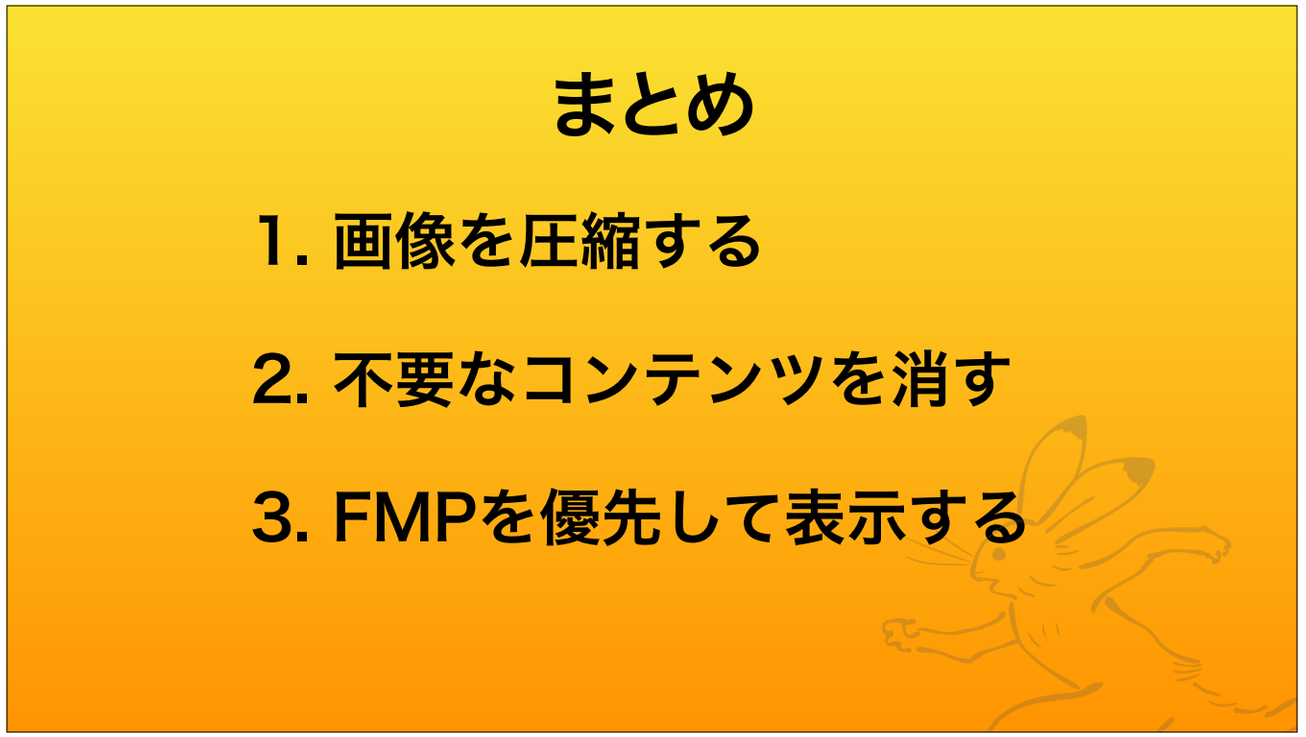
Critical Path CSS Generator



Critical Path CSS Generator
https://jonassebastianohlsson.com/criticalpathcssgenerator/

そういった場合に、便利な「Critical Path CSS Generator」というサイトがあります。
（Click）
このサイトに高速化したいページのURLを入力し、CSSファイルを直接コピペしてボタンを押すと、ファーストビューの範囲のCSSを推定して書き出してくれるので手助けになります。

これでテクニックのご紹介を終わります。
最後に、特に重要なポイントを3つにまとめます。



1つめ、画像はウェブページ上で大きな割合を占めるので、圧縮して、ファイル容量を少なくしよう。

2つめ、不要なコンテンツを消して、根本から表示を速くしよう。

3つめ、FMP。ファースト・ミーニングフル・ペイントを最優先に表示させよう。FMP以外の部分は非同期読込で後回しにします。

高速化の基本テクニックのひとつひとつは単純な工程です。
必要に応じて、うまく活用して役立てて行きましょう！

次はあなたもスピードUP！

ありがとうございました