Web-based Signage
コンテンツ開発ガイドライン

Release Candidate Rev.

2016 年 2 月 15 日

作成： Web プラットフォーム性能ベンチマーク検討会

発行： 慶應義塾大学 SFC 研究所 先端ウェブアーキテクチャ・ラボ

内容

1. コンテンツ開発ガイドラインの概要.....4
1.1. ガイドラインの範囲.....4
1.2. ガイドラインの方針.....5
2. 開発をはじめる前に.....6
2.1. 開発対象の全体像と特性の把握.....6
2.2. 開発プロセスについて.....8
3. ノウハウ集.....9
3.1. 表記ルールについて.....9
3.2. コンテンツの構成と配置.....10
3.2.1. jQuery 等巨大ライブラリの利用を避ける.....10
3.2.2. JavaScript/CSS ファイルは結合し、ミニファイする.....10
3.2.3. 部品利用する画像は CSS Sprite 等を用いて 1 ファイルとする.....10
3.3. コンテンツ記述時.....11
3.3.1. 動的な DOM 生成時は DocumentFragment を利用する.....11
3.3.2. CSS の gradient/box-shadow/border-radius を控えめにする.....11
3.3.3. フォントがキャッシュに収まるように文字スタイルを絞る.....11
3.3.4. リフローが与える影響を考慮する.....12
3.3.5. マルチコア環境では Web Worker を活用する.....12
3.3.6. GPU の活用.....12

1. コンテンツ開発ガイドラインの概要

本資料は、Web-based Signage 普及を目的に、コンテンツ制作者向けにハードウェア上でもっとも効果的に動作するコンテンツを制作するためのガイドラインを示す。

1.1. ガイドラインの範囲

Web-based Signage は、コンテンツを管理する CMS（コンテンツ登録やプレイリストなどを管理）、コンテンツを配信する CDN（配信クラウドとネットワーク）、そして、コンテンツを再生する端末から構成される。本ガイドラインは、これら Web-based Signage システム全体のうち、コンテンツ部分を対象とする。

Web-based Signage システムは、以下の要素で構成される。

ハードウェア

物理的に端末を構成する構成要素を指す。具体的には、パネル、SoC（CPU, GPU, メモリなど）、入出力インタフェース（HDMI, USB, 電源など）で構成される。

オペレーティングシステム

Web-based Signage として端末ハードウェアを利用するためにインストールされる基本ソフトを指す。具体的には、Windows, Android, iOS, Linux, FreeBSD, Firefox OS などを指す。

ブラウザーランタイム

HTML, CSS, JavaScript などの Web 標準技術を動作させることができるアプリケーション基盤を指す。具体的にはウェブブラウザーを指すが、必ずしも、一般的なウェブブラウザーである必要はない。例えば、Android の WebView など、ランタイムとして用意された環境も含む。

JS プレーヤー

プレイリストに基づいてコンテンツを再生するためのアプリケーションを指す。Web-based Signage では、ブラウザーランタイム上で動作するウェブアプリケーションとして作られる。具体的には、HTML, CSS, JavaScript を駆使して開発される。

コンテンツ

コンテンツは、JS プレーヤー上で表示または実行され、サイネージ端末の前にいるユーザーに見えるものである。コンテンツは、単体画像、動画、または、ウェブアプリケーションとして制作される。中には、これらコンテンツを表示しつつ、音声を再生するものも存在する。

1.2. ガイドラインの方針

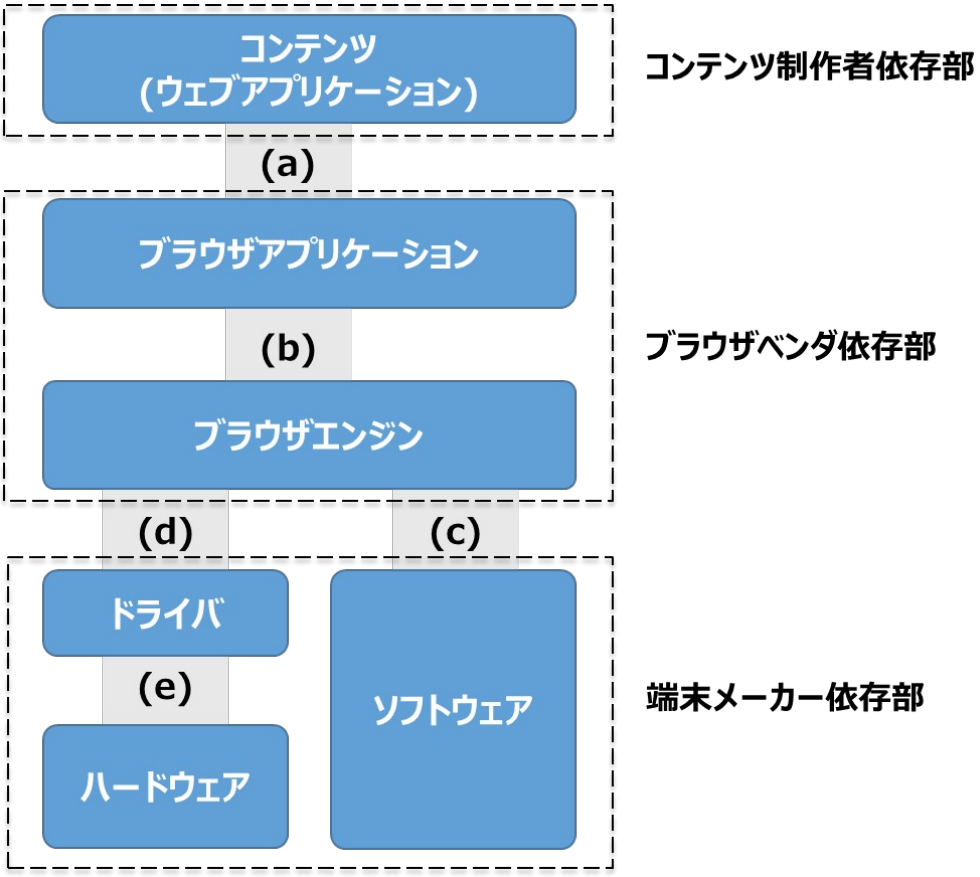
PC 等に比べ、RAM/ROM/CPU 等の資源が少ない組み込み機器上で動作するブラウザーランタイム、JS プレーヤーのパフォーマンスを引き出すための指針を示す。具体的なコード例を避け、コンテンツ制作者の創意工夫の余地を残しながら、原則として従うべきグッド/バッドノウハウを紹介する。

2. 開発をはじめる前に

組み込み機器向けにコンテンツ(ウェブアプリケーション)を制作する上で、表示対象となる端末・ブラウザの特性を把握することが非常に重要である。本章では、開発をはじめる前にどういった端末情報を把握すべきかについて記述する。

2.1. 開発対象の全体像と特性の把握

参考情報としてブラウザ搭載組み込み機器の概念図を以下に示す。この図の中の(a)以外については、コンテンツ制作者が関与できない部分のため、『Web-based Signage 試験結果表』を事前に入手し、表示対象となる端末・ブラウザの特性を把握することを推奨する。



(a) HTML コンテンツ

唯一コンテンツ制作者がコントロールできる部分。コンテンツ制作者は(b)~(e)の特性を考慮した企画・デザインを行うことが肝要である。『Web-based Signage 試験結果表』に記されている特性を理解した上でコンテンツの内容を検討し、表示対象となる端末・ブラウザに最適なコンテンツ設計・実装をおこなう必要がある。

(b)ブラウザ内部

「ブラウザ」は前述の図の様に、大きく分けてアプリケーション、エンジン、端末との結合部分と 3 つに分けることができる。ブラウザアプリケーションとは、主にブラウザエンジンの起動や終了、メニュー、ブックマークなどの UI 表示などのコンテンツ実行以外の付帯部分、さらには端末側からのキー入力やタイマーなどの OS やハードウェアからの割り込みをきっかけにしてブラウザエンジンに対して制御命令をだす部分である。ブラウザアプリケーションは、上述の様にコンテンツ以外の外部の入力をブラウザに伝える役割をしている。その関係で、同じブラウザエンジンを利用していても、ブラウザアプリケーション次第では異なる挙動となる。後述の(c)(d)による影響が大きい場合もあるが、表示対象となる端末に、特に気をつけるべき動作制限が公開情報としてないかを事前に調査しておくことを推奨する。なお、ブラウザベンダによっては PC 上で動作するシミュレータを用意していることもあり、必要に応じて入手・活用することを推奨する。

(c) ブラウザ移植部（ソフトウェア実装）

ブラウザエンジンの設計次第ではあるが、ハードウェアを利用できそうな機能は、ブラウザエンジンの外で処理を行えるように切り出されている場合がある。端末の仕様によるが、これらの切り出された機能がハードウェアに結合されている場合と、ソフトウェアで実装されている場合がある。『Web-based Signage 試験結果表』を参照することで、これらの特性がつかめてくると思うが、後述の GPU アクセラレーションが無効、もしくはほぼ効いていないような場合は、これらの機能はなるべく利用を控えることを推奨する。例えば、アニメーションなどの描画性能は GPU アクセラレーションが無効な場合は CPU やメモリバス速度に依存してくる。この場合、アニメーション、グラデーション、アルファ値の操作などが高負荷処理となる。

(d)ブラウザ移植部（ハードウェア結合）

GPU アクセラレーションが有効な場合は、ブロック要素のレイヤー化などが利用できる可能性があるため、より滑らかなアニメーションが期待できる。レイヤー化の詳細については、後述の「3. ノウハウ集」を参照とするが、レイヤー化を意識したコンテンツ制作をおこなう必要

があるため、企画・デザインの検討に入る前に表示対象の端末が GPU アクセラレーションに適応しているかどうか調査しておくことを推奨する。

(e)デバイスドライバーハードウェア間

(c)(d)にて記述したハードウェア・アクセラレーションが有効な場合においても、デバイスドライバとハードウェアの間の結合品質によっては、期待どおりのパフォーマンスが出ないことがある。表示対象の端末で、このような既知の問題が存在するかを企画・デザインの検討に入る前に調査しておくことを推奨する。

2.2. 開発プロセスについて

開発をはじめる前に対象となる端末を入手し、開発中のコンテンツを随時テストできる環境を構築することを強く推奨する。開発効率を優先して一般的な PC ブラウザにて開発を進めることについては否定しないが、出来るだけ実際の端末上で動作確認をおこなうことを推奨する。実機での動作は、PC の動作と比べて動作速度や見え方などが異なる場合も多く、コンテンツ作成時から実機で確認していくことでコンテンツ作成の手戻りを防ぐことが出来る。たとえば、ブラウザエンジンの種類や端末毎にサポートする機能が微妙に異なっていることや、利用可能なメモリ量にも違いがある場合があり、端末上でのみエラーが発生することもある。また、動作速度の違いにより期待する UX が実現できないこともある。一般的な PC ブラウザやブラウザベンダが提供する PC シミュレータ等を活用し、効率的に開発を進めつつ、定期的に対象となる端末上で動作確認をおこなうことが重要である。

3. ノウハウ集

3.1. 表記ルールについて

以降に記述する各ノウハウについて、そのノウハウにより良化が見込まれる項目や、逆に悪化する可能性のある項目(トレードオフ)を以下のタグ表記により示すものとする。

良化の可能性あり

悪化の可能性あり

項目：

[CPU] … CPU 負荷に影響するため CPU スペックが低い端末での効果が期待できる項目

[通信] … 通信負荷に影響するため、主にコンテンツ表示までの速度の改善が見込める

[メモリ] … メモリ使用量が減るため、メモリ搭載量の少ない端末でのメモリ不足回避が見込める

[描画] … 描画処理の負荷が軽減されるため、UX の改善に繋がる可能性がある

[フォント] … 文字描画処理の負荷に影響するため、文字を多用するコンテンツでの改善が見込める

[保守性] … 成果物の運用・メンテナンス時の手間の削減が見込める

表記例：

通信 メモリ

通信処理の高速化が見込めるが、トレードオフとしてメモリ使用量が増えるという意味。

3.2. コンテンツの構成と配置

3.2.1. jQuery 等巨大ライブラリの利用を避ける

CPU　通信　メモリ　描画

高度なアニメーション処理が必要とされる場合を除き、原則としてミニファイした状態で数百 KB 以上の容量をもつライブラリの利用を避ける。高度な用途向けに設計されたライブラリは一般的に処理負荷が高く、シンプルな処理に利用すると、スムーズなコンテンツ再生の妨げとなることが多い。また、巨大ライブラリを利用しないことで、通信処理や JS 解析処理の軽減により起動高速化やメモリ使用量の削減などの効果も期待できる。

3.2.2. JavaScript/CSS ファイルは結合し、ミニファイする

通信　保守性

JavaScript ファイルや CSS ファイルはリクエスト回数の削減、および処理系の負荷軽減を目的とし、ファイルを結合したのち、ミニファイ(難読化でも同様の効果)する。これらを HTML ファイル中に埋め込むことで、さらにリクエスト回数を削減するという手法もあるため、開発規模や保守性などを考慮した上で適切な手法を選定することを推奨する。

3.2.3. 部品利用する画像は CSS Sprite 等を用いて 1 ファイルとする

通信　メモリ　保守性

部品として利用する小さい画像については CSS Sprite(1 枚の画像の特定領域のみを padding, overflow: hidden 等で表示する技法)等を用いてできるだけ 1 ファイルにまとめる。これにより、リクエスト回数を削減することができる。ただし、利用頻度がそれほど高くない画像を含んでしまった場合にメモリ使用量が単体ファイル利用時よりも増えてしまう場合があるため、対象とする画像の選定は慎重におこなう必要がある。

3.3. コンテンツ記述時

3.3.1. 動的な DOM 生成時は DocumentFragment を利用する

CPU 描画

DOM ツリーの操作にはそれに対応するメモリ操作や再描画処理が伴うため、Native API の appendChild()を繰り返し呼び出して HTML を生成することや、jQuery の $.append() や $.appendTo()を繰り返し呼び出して HTML を生成することは避けた方がよい。代わりに DocumentFragment により仮想的な DOM ツリーを構築してから、DOM ツリーに appendChild することで再描画を最低限に抑制することができる。DocumentFragment が利用できない場合は String で出力する（innerHTML 等）などの手法でも実現できる。

3.3.2. CSS の gradient/box-shadow/border-radius を控えめにする

描画

大きな画像に対する CSS での飾り付け処理は、ブラウザーランタイムにとって非常に負荷の高い処理となるため基本的には避ける。動的にトランジションしていく場合など、特別の理由がない限り元の画像のほうで付与しておく。特に GPU アクセラレーションの有無により、大きくパフォーマンスが異なる可能性があるため、企画・デザイン検討段階で、端末性能を正しく把握するためにコンテンツのプロトタイプを用意し動作確認を実施することを推奨する。

3.3.3. フォントがキャッシュに収まるように文字スタイルを絞る

フォント

環境に依存するが、基本的にフォント描画は処理コストは高い。一度描画したフォントを使いまわし、キャッシュに残るように工夫する。特に文字列をアニメーションさせる場合など、頻繁にフォント描画が発生するようなコンテンツでは効果が得られる可能性がある。

3.3.4. リフローが与える影響を考慮する

CPU 描画

親要素に対する class / style の変更は、子孫要素へも影響があるため、再計算・再描画処理が大量に発生してしまう場合がある。特に深い階層をもつ要素ではパフォーマンスへの影響が甚大なため、コンテンツ制作者はリフロー発生を意識した設計・実装をすること。

3.3.5. マルチコア環境では Web Worker を活用する

CPU

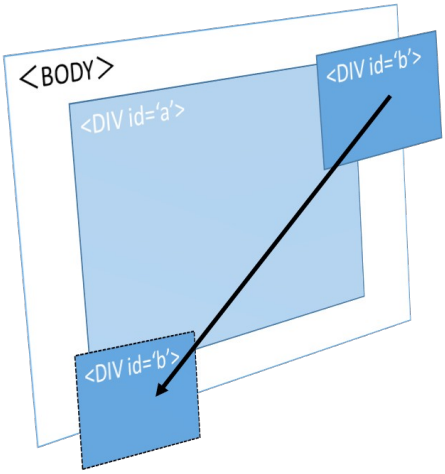
DOM 処理等の長大な JavaScript 処理を複数実行する必要がある場合かつ、性能要件における「マルチ対応型」に適合する環境であれば、積極的に Web Workers を活用する。

3.3.6. GPU の活用

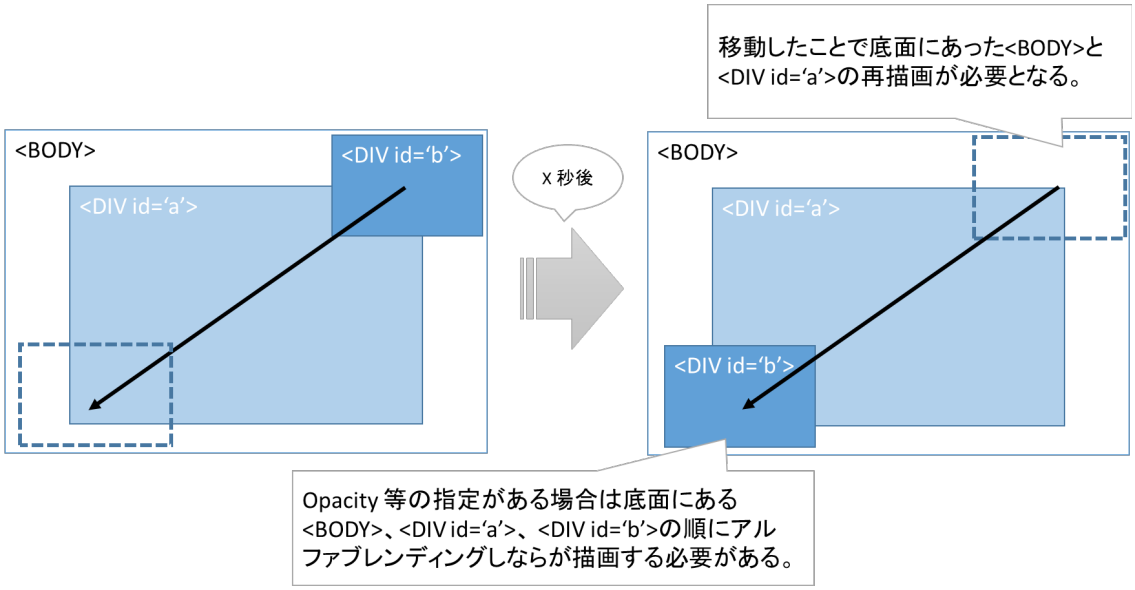
CPU 描画

ブラウザエンジンの種類やその移植方法によっても活用の仕方が異なるため一意に説明することは難しいため、ここでは Webkit 系のブラウザについて説明する。Webkit では Accelerated-Compositing と呼ばれる機能があり、この機能が有効化されたブラウザでは、指定したブロック要素を別レイヤー化することでアニメーションの高速化ができる可能性がある。Accelerated-Compositing はその名のとおり、「合成処理」を高速化するというものである。具体的には 1 枚の絵として生成していたコンテンツを複数のレイヤーに分けてレイヤー同士を GPU で合成(ブレンディング)することで高速化を図るというものである。

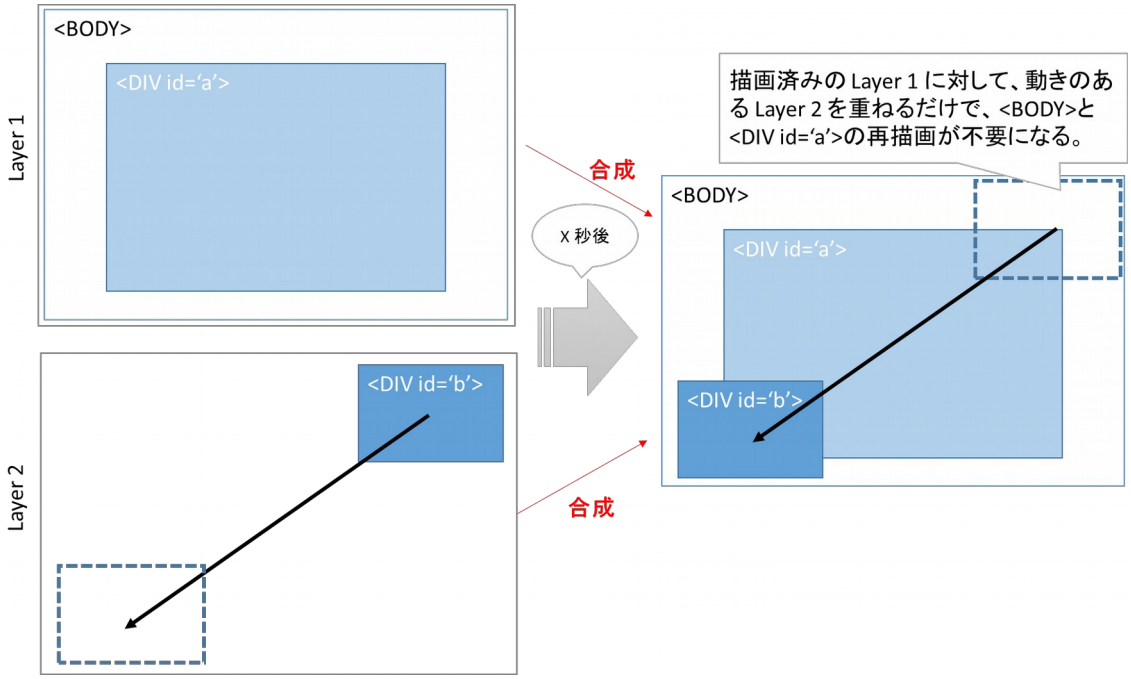
以下では <BODY><DIV id='a'>の要素の上で<DIV id='b'>がアニメーションする場合を例にレイヤー化の概念を説明する。



まず、レイヤー化ができない場合、以下の図のようにアニメーションが発生する度に、DOM ツリーを辿って何度も再描画する必要があり、描画処理に時間がかかってしまうためアニメーションのフレームレートが上がらない場合が多い。



一方でレイヤー化できる場合、<BODY><DIV id='a'>と<DIV id='b'>の描画レイヤーを分離し、それぞれのレイヤーを GPU で合成することで、再描画処理の軽減が見込める。



但し、レイヤーの多用は VRAM を著しく消費するため、端末によっては正常な描画ができなくなる場合があるため注意が必要である。加えて、レイヤーの多用により、本来レイヤー化して描画を分離したい要素が意図通りに分離できない事態も発生する可能性がある。

一般的にCSS Animations、CSS Transitions などを指定することによりレイヤー化することができる。他にもtranslate3d(0,0,0)などの指定により強制的にレイヤー化するハック的な手法もあるため、対象となる端末上で事前に検証しておくことを推奨する。