2E-05

サーバサイドイメージレンダリングによるウェブUIコンポーネント・フレームワーク

内海 宏律† 菊地 大介† 浦邊 信太郎† 齋藤 邦夫† 手塚 大†
†株式会社 日立ソリューションズ東日本

1. はじめに

スマートフォンやタブレット型端末(スマートデバイス)の普及に伴い、業務システムをこれらの端末のWebブラウザから利用するニーズが増加している。しかし、スマートデバイスはPCと比較し性能が低くレンダリングに多くの計算量を必要とするページを描画すると性能問題が発生する。

2. スマートデバイスでのWebサービス利用の課題

そこでPCとスマートデバイス間の性能差とネットワーク環境の差に着目し、課題を整理した。

従来はModel-View-Controller(MVC)アーキテクチャのViewに相当するクライアント側でデータのレンダリングと表示を行っていた。HTML上でデータを可視化する際は、スマートデバイスに対応したJavaScriptライブラリなどで動的にDocument Object Model (DOM)を構築する方法やCanvas[1]上に画面を描画するなどの方法が用いられる。しかし、スマートデバイスに対して上記方式を用いると描画やスクロールなどのUI応答に時間がかかり、十分な操作性能が確保困難な状況となる。

また、スマートデバイスは3GやGSMなど、低速で不安定な通信環境下で利用される特徴がある。高速なLANで接続されたPCを想定したアプリケーションでは、各種デバイスでサポートされているXML形式でデータを一括転送することが多い。XMLはJSONなどと比較するとシリアライズ効率が悪く、データサイズが大きくなる傾向があるため特に低速回線環境では初期表示までの待ち時間が長くなる原因となる。以上のことから、課題は下記2点となる。

- スマートデバイスで描画負荷の高い画面を表示する際にも十分な操作性能を確保すること
- 低速で不安定な通信環境下で初期表示時間を短縮すること

3. サーバサイドイメージレンダリングの提案

3.1. 提案手法の概要

サーバ上のDBにデータが集約され、クライアントの要求に応じてデータを可視化、表示するアプリケーションを想定する。その際の一般的な処理フローを図1のように抽象化した。図1ではクライアントがMVCアーキテクチャでのViewに相当する。

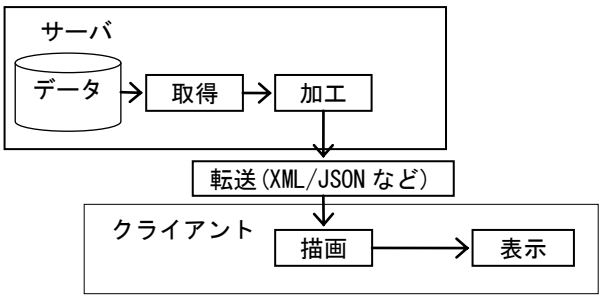

図 1 Webシステムの処理フロー

クライアントからのリクエストがあると、表示対象となるデータがModelとなるDBから取得され、Controllerがユーザの要求に応じてデータの加工などの処理を行う。その後、処理結果を表示するため、データをシリアライズし、Viewであるクライアントに転送する。クライアントは受け取ったデータをオブジェクトに復元した後、各オブジェクトの表示位置などを計算しながら画面を描画する。そこで、2章の課題を考慮し以下の手法を提案する。

- クライアントの負荷を下げるため、処理負荷が高い描画処理をサーバサイドで実行する
- 初期表示までに必要なデータ転送量を削減するため、画面表示に必要な領域のみを先に転送する

3.2. レンダリング処理のサーバサイド化

クライアントの処理負荷を低減するため、図2のように負荷の高い描画処理をサーバサイドで実行するサーバサイドイメージレンダリング方式を用いる。描画結果は画像データとしてクライアントに転送する。

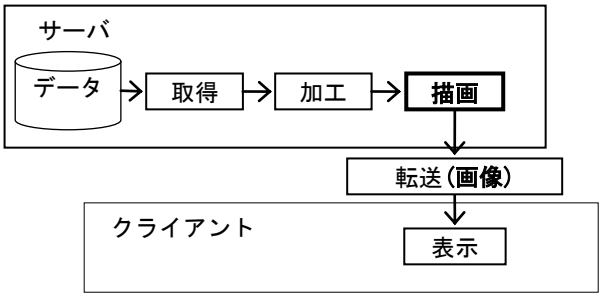

図 2 サーバサイドで描画を行う場合の処理フロー

これにより、クライアントの負荷は画像の表示のみとなり、クライアントの負荷が低減できる。また、

Web User Interface Component Framework using Server Side Image Rendering
Hironori Utsumi†, Daisuke Kikuchi†, Shintaro Urabe†, Kunio Saitou†, Masaru Tezuka†
†Hitachi Solutions East Japan, Ltd.

Copyright ©2015 Information Processing Society of Japan.
All Rights Reserved.

低速なクライアントで行っていた処理を高速なサーバ上で実行することで処理時間の短縮が実現できる。

3.3. 遅延処理による通信データ削減

初期表示までに必要な時間を短縮するため、画面全体ではなく、必要なデータのみを先に転送し、残りのデータは画面のスクロールなどに応じて遅延処理で転送する方式を用いる。転送すべきオブジェクトの分別はデータモデルの構造やレンダリングアルゴリズムなどに強く依存し、汎用化は容易ではない。提案手法ではサーバ上で描画処理を行うため、その後のデータ転送においてデータモデルなどを意識する必要がなくなる。提案手法では、転送データの画像化により、画像の座標情報のみでの汎用的な処理として遅延転送を実現できる。

4. 提案手法の評価

4.1. フレームワークの概要

これまでの検討をふまえ、サーバサイドで表示内容を描画し、それを画像としてクライアントに転送、表示するフレームワークを提案する。図 3 にその概要図を示す。

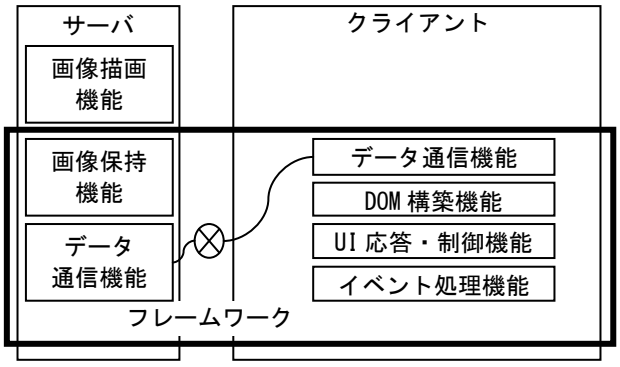

図 3 ウェブ UI コンポーネント・フレームワーク

フレームワークはアプリケーションが描画した結果を画像として受け取り、図 2 での転送以降の処理を提供する。また、ウェブ UI コンポーネントとして汎用的に利用されるスクロールやズーム機能をサポートすることで、アプリケーションを限定せずに広く適用できるようにした。さらに、ステートフルなプロトコルである WebSocket[2]を用い、サーバのメモリ上に画像を保持する。これにより、初回接続時に生成した画像をその後の遅延処理でも返すことが可能となり画像の再生成コストが不要となる。

4.2. 評価実験の設定

描画時間を比較するため、プロジェクトの進捗状況を可視化するガントチャートを例として描画時間を比較した。ガントチャートは作業項目(タスク)を行、時間を列として進捗状況を可視化する。各タスクには作業単位(アクティビティ)が矩形として描画される。データの規模は中規模プロジェクトを想定し表 1 のように設定した。なお、DOM 構築方式では dhtmlxGantt[3]を用いたが、ライブラリの制約上 100 アクティビティ(1 件/タスク)のみの描画となる。

表 1 テストデータの仕様

プロジェクト期間	6 ヵ月
タスク総数	100 行
アクティビティ総数	300 件(3 件/タスク)

描画時間の評価として、クライアントを Nexus7 2013 上の Chrome、サーバの CPU を Core i7-3930K とした環境でテストデータの描画にかかる時間を 10 回計測し平均値を求めた。また、データサイズの評価として、このデータを XML 形式および画質の劣化がない PNG 画像形式として描画した場合のサイズを測定した。データ全体の描画には約 3000 ピクセル四方の領域が必要であるが、初期表示領域として Full HD サイズの画面に相当する解像度でのデータサイズも測定した。なお、メモリ上でのデータモデルのサイズは 84kB であった。

4.3. 実験結果

各方式を利用した際の描画時間を表 2 に示す。高速なサーバで描画処理を行うことで処理時間の短縮が実現できる。また、Canvas 描画方式では UI 操作への応答に対し JavaScript による再描画が必要となるため、実際の体感速度は遅く感じる。

表 2 描画時間

DOM 構築方式	3357ms
Canvas 描画方式	238ms
サーバサイドレンダリング方式	138ms

画面の表示に必要となるデータサイズを表 3 に示す。画像化と表示対象領域のみの転送を併用することで初期表示に必要なデータサイズを大きく削減できることが分かる。

表 3 転送されるデータサイズ

XML データ	664kB
PNG 画像(全体)	308kB
PNG 画像(初期表示領域のみ)	56kB

5. おわりに

本稿ではクライアントの負荷を低減させるため、サーバサイドでレンダリングを行うウェブ UI コンポーネント・フレームワークを提案した。これにより複雑な構造を持つ画面をスマートデバイス上で高速に表示できるようになる。

参考文献

[1]HTML Canvas 2D Context | W3C,
http://www.w3.org/TR/2dcontext, Accessed 2014/12
[2]The WebSocket Protocol | IETF,
https://tools.ietf.org/html/rfc6455,
Accessed 2014/12
[3]dhtmlxGantt | Dinamenta, UAB,
http://dhtmlx.com/docs/products/dhtmlxGantt,
Accessed 2014/12

Copyright ©2015 Information Processing Society of Japan.
All Rights Reserved.