部品化を指向した Web アプリケーション開発の枠組の設計と実装

1431110　山本竜太郎　　岩崎研究室

1　背景

1.1　Web アプリケーション

Web アプリケーションとは, Web 技術を用いて構築されたアプリケーションのことである. Web アプリケーションは, 基本的に Web ブラウザさえあれば環境を選ばず動作するという利点がある. 単純な Web アプリケーションでは, まず, クライアントである Web ブラウザがサーバに対してリクエストを送る. サーバはそのリクエストをプログラムで処理する. そして, その結果を HTML や CSS といった Web ブラウザで表示できる形でレスポンスとして返し, Web ブラウザがそれを表示するという流れで動作する.

しかし近年は, クライアント側でもプログラムが動作し, クライアント側とサーバ側双方のプログラムが連携しながら動作する方式をとるアプリケーションも多い. クライアント側で動作する JavaScript プログラムが非同期的にサーバにリクエストを送り, そのレスポンスに応じて動的にページの一部を書き換える Ajax [1] は, その実装手法の一例である. 本研究では, Web アプリケーションの中でもクライアントサイドで動作する部分に着目する.

1.2　Web アプリケーションの開発における問題

一般に Web アプリケーションのクライアントサイドは, HTML や CSS, JavaScript といった Web ページの記述に用いられる言語をそのまま用いて構築される. HTML は U ユーザインタフェース (UI) の論理構造, CSS は UI の見た目, JavaScript は UI の動きを定義するために用いる. これらの言語は, 言語仕様として名前空間を持たないため, 複雑な Web アプリケーション開発に利用すると問題が生じることがある.

例えば, アプリケーションを図 1 のように, 部品 A と部品 B に分けて開発することを考える. 部品 A の CSS と部品 B の CSS で同じ名前のセレクタを使用してしまったため, 定義の衝突が発生している. 本来部品 A は背景色が黒, 部品 B は背景色が赤となるはずが, 部品の読み込み順によってどちらの部品も背景色が黒となったり赤となったりする問題が発生しうる.

GUI アプリケーションを互いに独立した部品に分割して開発することは古くから知られている手法である [3] が, このような定義の衝突の問題があるため, 現状ではこの手法を単純に適用することは出来ない.

2　目的と方針

本研究では, 前節で述べた問題を踏まえ, 定義の衝突が発生しないように名前空間の概念を取り入れた Web アプリケーション開発の枠組の実現を目指す. この目的を実現

```
# 部品 A の HTML
<div class="wrapper">
...
</div>

# 部品 A の CSS
.wrapper { background: black; }

# 部品 B の HTML
<div class="wrapper">
...
</div>

# 部品 B の CSS
.wrapper {  background: red; }
```

図 1: 定義の衝突

するために, HTML, CSS, JavaScript で記述されたコードを読み込み, 名前空間を分離するためのコード変換を施した上で出力するライブラリを実装する.

3　設計

本研究で実装するライブラリの設計を図 2 に示す. まずライブラリは, HTML, CSS, JavaScript を用いて記述されたコードを, 定義の衝突が起こらないようにコード変換する. 例えば, CSS の定義の衝突が発生しないように, CSS セレクタを書き換えたり, HTML で定義されている DOM のクラス名を書き換えたりする. 同時に, 名前を書き換えた対象を書き換える前の名前で参照できるように、シンボルテーブルも生成する. そして, 変換した HTML, CSS, JavaScript とシンボルテーブルをまとめあげ, 単一の JavaScript コードへ変換する. 最終的に生成されるコードを JavaScript のみで記述されたコードとすることで, JavaScript が動作する Web ブラウザであればどのような Web ブラウザでも動作することを保証できる.

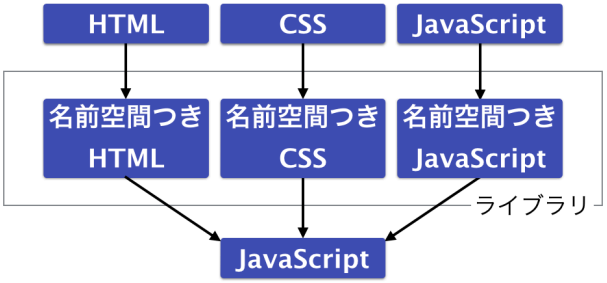

図 2: ライブラリの設計

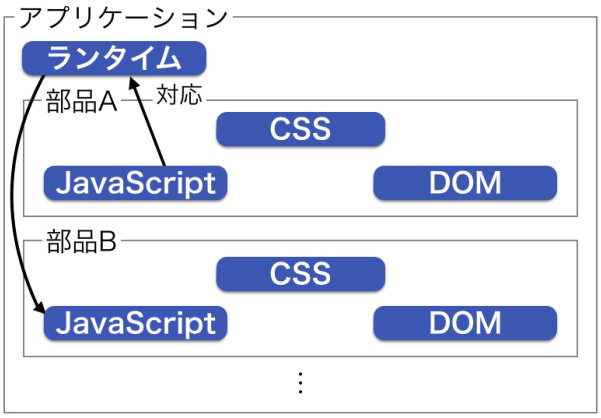

図 3: アプリケーションの動作

4　アプリケーションの動作

本研究の枠組みを用いて開発された Web アプリケーションがどのように動作するかを図 3 に示す．このアプリケーションは, 部品 A と部品 B という 2 つの部品から構成されている．部品 A および部品 B は, それぞれ自分の DOM と CSS と JavaScript を持っている．これらは部品ごとに独立しているため, 例えば部品 A の CSS によって部品 B の CSS が上書きされてしまうようなことは起こらない．また, 部品の DOM, CSS, JavaScript は他の部品からは参照できないため, 部品 A の JavaScript コードによって部品 B の DOM や CSS が書き換えられてしまうようなこともない．

一方で, 複数の部品が連携して動作するケースも考えられる．例えば, 部品 A で特定のイベントが発生した時, 部品 B を書き換えたいといったケースが考えられる．この時, 部品間の仲介を行うのがアプリケーションのランタイムである．ランタイムは, JavaScript で記述されており, 本システムで生成された JavaScript コードを解釈して適切に実行する役割を担う．ランタイムは, アプリケーションの開発者が自由に拡張することが出来る．アプリケーション開発者は, ランタイムに部品 A でのイベントと部品 B を対応付ける記述を行う．これをランタイムは解釈し, 部品 A でイベントが発生した時, それを自動的に部品 B に対してフォワーディングする．イベントを受け取った部品 B は, 自らを書き換える処理を実行することで, 部品 A のイベントに応じて部品 B を書き換えるという目的が達成される．このイベントと部品の対応関係は, 関数のように取り扱うことが出来, 関数合成のように複数の対応関係を合成し, より多くの部品を連携させることが出来る．

5　関連研究

Google Web Toolkit [2] は, Google が開発している Web アプリケーションフレームワークである．このフレームワークでは, UI を構成要素ごとに記述し, それを組み合わせることで Web アプリケーション開発を行う事ができ, 本研究と目指す方向が共通している．しかし, このフレームワークではアプリケーションを Java で記述する行うことを前提としており, Java の知識がないデザイナなどが利用するにはハードルが高い．本研究の手法は, HTML や CSS, JavaScript など既に Web アプリケーション開発で用いられている言語をベースとするため, 利用に当たってのハードルが低く抑えられている．

WebComponents [4] は, W3C が策定を進める Web アプリケーションの UI の部品化のための仕様である．既存の DOM や JavaScript に拡張を加え, 特定の DOM を外部から参照できないようにしたり, 特定の DOM に CSS の適用範囲を制限したり出来る．この仕様では, プリミティブな API しか定義されておらず, また DOM と CSS を分割して取り扱えるようにすることを目的としているため, JavaScript をどう扱うかについてはアプリケーションの開発者に委ねられている．本研究は, JavaScript の部品化を含めた枠組を実現することを目指している点が異なる．

6　現状と今後

現状は, アプリケーション記述に用いる言語の設計の検討を行っている段階である．これに伴い, 簡単なプロトタイプの実装も行っている．今後は設計が決まった部分から順に実装を進めていく予定である．

実装が完了次第, 評価を行う．評価の軸としては, アプリケーションを記述する上で十分な実行速度をもって実行できるかとアプリケーションの記述性が向上したといえるかが挙げられる．実行速度は, マイクロベンチマークを作成しネイティブに記述したアプリケーションと本研究で提案する手法で記述したアプリケーションの間にどの程度のオーバヘッドがあるかを測定し評価する．記述性は, 複数の部品が連携して動作するアプリケーションをネイティブに記述した場合と本研究で提案する手法で記述した場合で比較することで評価する．

対外発表は, 9 月に開かれるソフトウェア科学会全国大会での発表を予定している．

参考文献

[1] Jesse James Garrett. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications.
[2] http://www.gwtproject.org/
[3] Adele Goldberg and David Robson. Smalltalk-80: The Language and its Implementation. 1983.
[4] http://www.w3.org/TR/components-intro/