NRI IT Infra 基盤は進化する
技術創発 －NRI情報技術レポート－
2013年10月発行

HTML５時代のWebアプリケーション開発の方向性

野村総合研究所
先端ITイノベーション部 主任テクニカルエンジニア

山城　俊介（やましろ しゅんすけ）

フロント技術に関する新技術の調査、アーキテクチャ検討、フレームワークの設計・開発の他、新技術の業務適用検討、実プロジェクトへの技術支援などを行っている。

要 旨

iPhone, Androidが世に登場し、以前にも増してユーザはアプリケーションの「使いやすさ」や「機能性」を求めるようになってきた。それはB2Cに留まらず、B2BやB2Eにおいても、強く求められるようになってきたと実感している。HTML5はさまざまなデバイス向けのユーザインタフェースを開発できるため、開発手段の有力な選択肢であるが、「使いやすさ」や「機能性」を求めていくとさまざまな課題が生じてくる。

本稿ではWebアプリケーション開発における、問題およびその背景、課題を挙げ、その対応策および技術動向を示し、今後どう取り組んでいくべきか考察を述べる。

キーワード：HTML5、JavaScript、アーキテクチャ、フレームワーク、スマートデバイス

1. はじめに

iPhone, Androidが世に登場し、以前にも増してユーザはアプリケーションの「使いやすさ」や「機能性」を求めるようになってきた。それはB2C (Business to Consumer) にとどまらず、B2B (Business to Business) やB2E (Business to Employee)においても、強く求められるようになってきたと実感している。スマートデバイス、PC、さらにはHybridcast[i]など、さまざまなデバイスが登場する現代において、HTML5は開発手段の有力な選択肢であるが、「使いやすさ」や「機能性」を求めていくとさまざまな課題が生じてくる。

本稿ではWebアプリケーション開発における、問題およびその背景、課題を挙げ、その対応策および技術動向を示し、今後どう取り組んでいくべきか考察を述べる。なお、以下で述べる「問題背景」・「問題」・「課題」・「対応策」の対応を図 1に示すので、参考にされたい。

2. Webアプリケーション開発が抱える問題・課題

(1) サーバサイドとの密結合による責務の重複・分散化

これまでのWebアプリケーション開発は、サーバサイドでHTMLを生成しクライアントに送信する形式が一般的であった。しかし、最近はAjax[ii]で動的にサーバサイドAPIを呼び出し、その結果を用いて画面内容を書き換えるなど、サーバサイド以外でHTMLを生成する

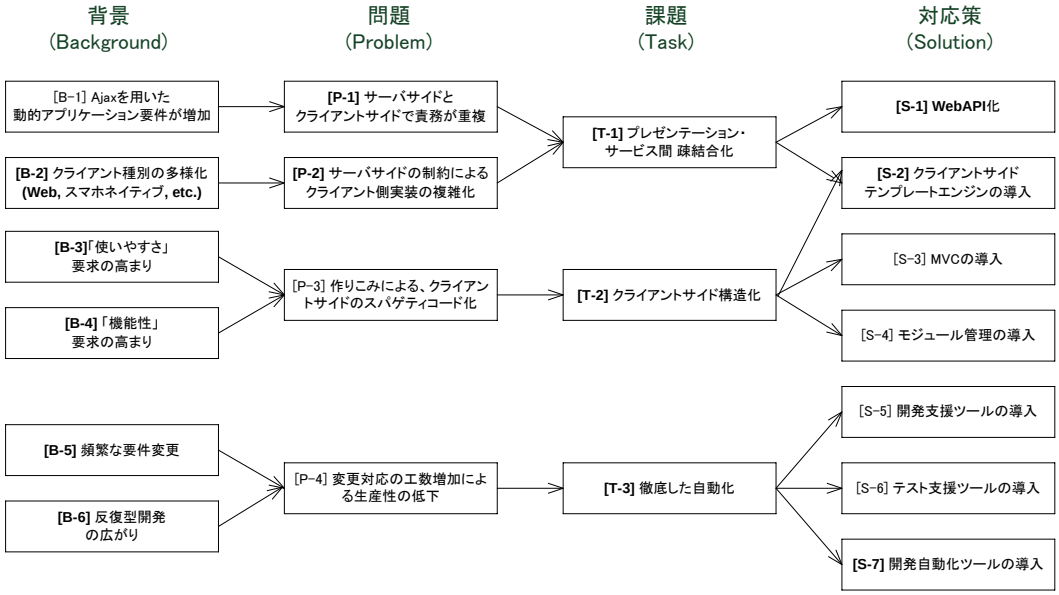

図 1 Webアプリケーション開発における背景・問題・課題・対応策

ケースが増えてきている([B-1])。この場合、HTMLを生成するロジックがサーバサイドとクライアントサイドの両方で併存することになり、双方を開発・保守しなければならない問題が発生する([P-1])。

また、当初Webアプリケーション開発であっても、途中でネイティブアプリケーション版に切り替える場合や、ネイティブアプリケーション版も並行して開発しなければならない場合など、クライアント種別は多様になることが多い([B-2])。このような場合にサーバサイド側でHTMLを生成することを前提とした構成だと、ネイティブアプリケーション版からサーバサイドAPIを呼び出すことが困難となる問題が発生する([P-2])。この問題には一般的には別途ネイティブアプリケーション用のAPIを用意するか、生成されたHTMLをスクレイピング[iii]して利用するなどして対応することになるが、開発・保守すべきコード量が増加するため、最善とはいえない。

これらの問題に対しては、サーバサイドとクライアントサイドで責務を明確にし、疎結合にすることが課題となる([T-1])。サーバサイドはあくまでサービスを提供し、クライアントサイドはプレゼンテーションを提供するのである。この課題を解決すれば、責務の重複をなくし、開発・保守するコードを最小限にとどめることができるようになる。

(2) JavaScriptのスパゲッティコード化

Webアプリケーションへの「使いやすさ」・「機能性」に対する要求は日々高まってきている([B-3], [B-4])。この要求に対応するために、多量のJavaScript・CSSを使い実現しているのが現状である。しかしJavaScriptにはモジュール管理機構が存在しないため、いとも簡単にスパゲッティ化してしまう問題がある([P-3])。そのため、JavaScriptを「そのまま」使いつづけると、生産性・保守性低下、また品質へも影響しかねない。prototype.jsやjQueryなど開発を便利にするライブラリが登場してきたが、あくまでコード量を減らすことができる程度で、アプリケーションの構造や開発の仕方そのものを変えるようなものではなかった。典型的なケースは、jQueryを利用しているが、コードがひとつのファイルに大量に記述され管理不能となるケースである。プロトタイプや開発初期は問題がないが、テスト・継続開発・保守が困難になり、徐々に開発スピード・品質を落とすこととなる。

この問題に対しては、クライアントサイドの構造化が課題となる([T-2])。これまでJavaScriptは構造化するほどの規模ではなかったが、近年は規模が増加しており、構造化による統制が必要なのである。

(3) 変更対応による生産性の低下

ユーザが直接触れることになるプレゼンテーション部分(画面・触り心地など)に対する要求は、頻繁に変更が入る傾向がある。「使いやすさ」・「機能性」への要求が高まっている昨今は特にその傾向がある([B-5])。このような状況では、画面デザイン→マークアップ→コーディング→ビルド→テストといった一直線のワークフローにはならず、このワークフローを繰り返し回すことで要件を固め、品質を向上していくのが一般的である([B-6])。このように頻繁に変更が入る状況下では、頻繁に変更が入らないことを前提とした開発体制・ツールを利用していると、各ステップにおける作業に多くの時間を割かれることになり、結果的に生産性が低下してしまうことになる([P-4])。

この問題に対しては、頻繁に変更が入るということを前提とした構成にする必要があり、そのためには「徹底した自動化」が課題となる([T-3])。

3. 課題対処に関する動向

ここでは、「2 Webアプリケーション開発が抱える問題・課題」で取り上げた問題・課題に対する、対処の動向をそれぞれ紹介する。

表 1 HTML5周辺のライブラリ/フレームワーク/ツール（例）

大分類	小分類	概要	プロダクトの例
HTML	エディタ	HTMLマークアップを支援するツール	Aptana Studio, Adobe Dreamweaver, Adobe Edgeシリーズ
	ビューア	HTMLレンダリング結果を確認するツール	各種ブラウザ, Adobe Edge Inspect CC
JavaScript	ライブラリ	便利ライブラリ	underscore, jQuery, Bacon.js, etc.
	MV* フレームワーク	MVC, MVVM等をサポートするフレームワーク(ライブラリ)	Backbone.js, Ember.js, Angular.js, Knockout.js, etc.
	UIフレームワーク	ユーザインタフェースを構築するためのフレームワーク	jQuery UI, jQuery Mobile, Twitter Bootstrap, etc.
	テンプレートエンジン	JavaScriptでJSPのようなテンプレートを実現するツール	Handlebars, Hogan.js, Mustache, Dust, jsRender, etc.
	モジュール管理機構	実行時、依存先のライブラリを事前にロードする等の管理をするフレームワーク	RequireJS, CommonJS, etc.
	一元化・難読化	複数のJavaScriptを一つのファイルへと圧縮するツール	r.js, uglify, etc.
CSS	コンパイラ	CSSよりも記述性の高い別言語で記述し、CSSを生成するコンパイラ	LESS, SCSS, Compass, etc.
開発環境	ビルドツール	開発者やCI環境において、ビルドを実施するツール	grunt
	パッケージマネージャー	Maven, Ivy, gemのようなパッケージマネージャー	bower
	開発ツールスタック	Railsのような開発の統合スタック	Yeoman
	デバッグ	ステップ実行等を支援するツール	ブラウザ同梱開発者ツール(F12等), weinre
テスト	単体テストサポート	JavaScriptロジックを単体テストするためのフレームワーク	QUnit, Jasmine, Sinon.js, etc.
	連結テストサポート	一連の処理を実行するためのブラウザエンジンおよび 処理実行フレームワーク	PhantomJS, SlimerJS, CasperJS, Selenium, etc.
JavaScript代替	altJS	JavaScript代替言語。コンパイラを用いて、JavaScriptへと変換する	TypeScript, JSX, CoffeeScript, Haxe, etc.

表 1には、本章で紹介するツール/ライブラリ/フレームワークを記載しているので、参考にされたい。なお、Webアプリケーション開発においては、W3C (World Wide Web Consortium) による標準化と並行してコミュニティ/ベンダによるツール/ライブラリ/フレームワークの開発が活発であり、これらトレンドが標準化へと還元されることが多いため、適切にフォローしていくことが大切である。

(1) クライアントサイドとサーバサイドの疎結合化を支えるアーキテクチャ/ライブラリ/フレームワーク

まず、サーバサイドをAPI化(WebAPI化)しそれを利用する動向がある([S-1])。Webアプリケーション開発で広く使われているRuby on RailsやPlay Frameworkなどでは、サーバサイドでHTMLを生成する機能を備えながらも、Web API(RESTインターフェイス)を生成する機能もサポートしており、クライアントサイドからこのWebAPIを利用するように構成すれば、サーバサイドとクライアントサイドを疎結合にすることができるようになる。

また、HTMLを動的生成するためのテンプレートエンジンに関しても、クライアントサイドで実行するJavaScriptライブラリ(Handlebarsなど)が登場している([S-2])。

このように、これまではサーバサイドでHTMLを生成する方式が主流であったが、動的ページ生成を含めた画面ロジックをJavaScriptに寄せ、サーバサイドAPIをREST化することで、クライアントとサーバサイドを疎結合にする方式が増えていくと考えられる。

(2) JavaScript構造化を支えるライブラリ/フレームワーク/ツール

① クライアントサイドMVCアーキテクチャ

複雑化してきたJavaScriptコードに対して、GUIアプリケーション開発で一般的なMVCアーキテクチャ(図 2)を取り込んだプロダクト(Backbone.js, Angular.jsなど)がトレンドとなってきている([S-3])。

これにより、ユーザ操作や画面上のイベントにどのように対処するか(Controller)、

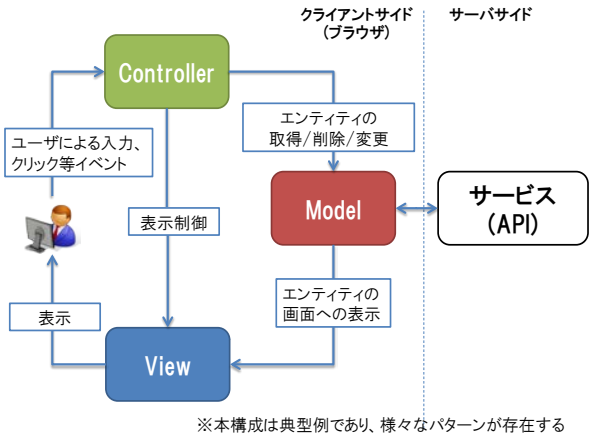

図 2 MVCアーキテクチャ(例)

データをどのように操作するか(Model)、データをどのように表示するか(View)に分けることができ、並行開発・単体テスト・構成管理面で、優位である。

② モジュール管理機構

「①クライアントサイドMVCアーキテクチャ」で述べたように複数の役割に分けた場合は、ソースコードに関しても分けて管理することが望ましい。しかし、ただ単純にファイルへと分割してしまうと、それぞれの依存関係管理やロード順序管理が煩雑になる(例えば、「A.jsというファイルはB.jsというファイルで必要としているため、B.jsよりも前にロードしておかなければならない」といったことを管理しなければならなくなる)。

これを解決するものとして、RequireJSやCommonJSなどのモジュール管理機構がある([S-4])。これらを使えば、Javaにおけるimportに似た感覚でコーディングでき、Javaと同様の構成管理が可能となり、煩雑さから開放される。

(3) 変更を前提とした自動化の仕組み

① ツールによる自動化

JavaやRuby同様JavaScript開発においても、コンパイルなどの定型的処理や、多量のライブラリ利用の必要性が出てきており、JavaやRuby同様にツール群が整備されてきている([S-5])。

まずは、make/Ant/Rake相当の gruntなどがある。これにより、JavaScript関連の定型的処理(例えばJavaScript難読化[iv]処理やテスト実行など)をバッチ化することが可能となる。

次に、JavaにおけるMaven/Ivy、Rubyにおけるgemのようなパッケージマネージャ bowerなどが登場し始めている。これにより、プロジェクトで必要となるライブラリを定義するだけで、必要なライブラリをダウンロード・配備することが可能となる。

また、Yeomanなど、Ruby on Railsのような「スタック」としてJavaScript開発環境全体をサポートするプロダクトが登場してきた。プロジェクト生成、クラス生成、テストコード生成、パッケージマネージャ/ビルドツールとの連携などが一元的に実行できる。各プロジェクトの標準化向けスタックを用意すれば、生産性の高い開発が可能となるだろう。

② テストの自動化

テストは大きく分けて、ブラウザを使うテストと、使わないテストがある。主に前者は単体テストで、後者は連結(結合)テストで用いられる。これら双方でテスト自動化するためのプロダクトがある([S-6])。

まず単体テストについては、QUnit, Jasmineといったフレームワークで実現可能である。これらは、Javaにおける JUnit、Rubyにおける RSpecに相当し、アプリケーション全体を動かさなくてもテストが可能であり、テスト実行にはgruntなどを用い、ブラウザではなくNode.jsなどのJavaScriptエンジンを利用する形を採る。ブラウザを用いないためDOM(HTMLのドキュメント構造; Document Object Model)が関連するコードのテストは通常実施できないが、フレームワークのモック機能を用いることで、DOM依存コードであってもテスト実行が可能となる。

次に連結(結合)テストについては PhantomJS, SlimerJSといった、ツールを用いる方法がある。これらは、「Headless browser」ないしは「Scriptable browser」と呼ばれ、「プログラムコードで自由に操作可能な画面なしブラウザ」と考えてよい。これらを用いると、ユーザの操作を模倣し、「どのようなリクエストが飛ぶのか」、「どのような画面が表示されるのか」を、自動的に確認できる。テスト操作をせずに画面を確認することができるため、後述のCIと連携させると、より力を発揮するものとなる。

③ CIによる自動化

CIとは、継続的インテグレーション(Continuous Integration)のことである。継続的インテグレーションを行うに当たり、リポジトリへのコミットやユーザ操作をトリガにして、テスト実行、プログラム統合を行うツールがあり、JenkinsやTravisCIが有名である。プログラム統合の先にある、検証/実行環境へのデプロイなども連続的に行うことがあり、この場合は特に継続的デリバリー(Continuous Delivery; CD)と呼ばれている[v]。

JavaScript開発環境においてもパッケージマネージャや、ビルドツールが整備されてきたことを受け、CI/CDが有効に実施できる環境となってきている([S-7])。これらをうまく組み合わせれば、「リポジトリにコミットし、すべての自動テストに成功した場合は、顧客の検証サーバへ自動的にデプロイされ、即座に顧客のフィードバックを受けられる状況にする」といったことが可能となる。

4. 考察 ～動向を踏まえたWebアプリケーション開発の考え方～

これまでに紹介した内容を振り返ると、いかにして疎結合化・構造化・自動化するかがポイントであることがわかる。これらは頻繁に変更されることが前提となるWebアプリケーションにおいて、高品質を維持するために必要なのである。

これを図 3で整理した。ワークフロー全体のサイクルをいかにして速く回すかが鍵となってくるのである。

一例として、次のような構成が考えられる。参考にされたい。

- Backbone.js(クライアントサイドMVCフレームワーク(ライブラリ))
- Handlebars(テンプレートエンジン)
- RequireJS(モジュール管理機構)
- Jasmine(テストフレームワーク)
- PhantomJS/CasperJS(テスト支援)
- jQuery Mobile(UIフレームワーク)
- Jenkins(CI)

5. まとめ

今回取り上げた技術要素は、あくまで現時点のものであり、今後数年で置き換わる可能性は否めない。例えば、W3CではWeb ComponentsというHTML/JavaScriptをコンポーネントとして管理する仕組みを整備中であり、これを前提としたものに一部置き換わっていくことも考えられる。

しかし、単に標準化されるのを待つのは得策ではない。Web Componentsについてもコミュニティやベンダの動きを注視し、標準化しているものである。そのため、Web Compone

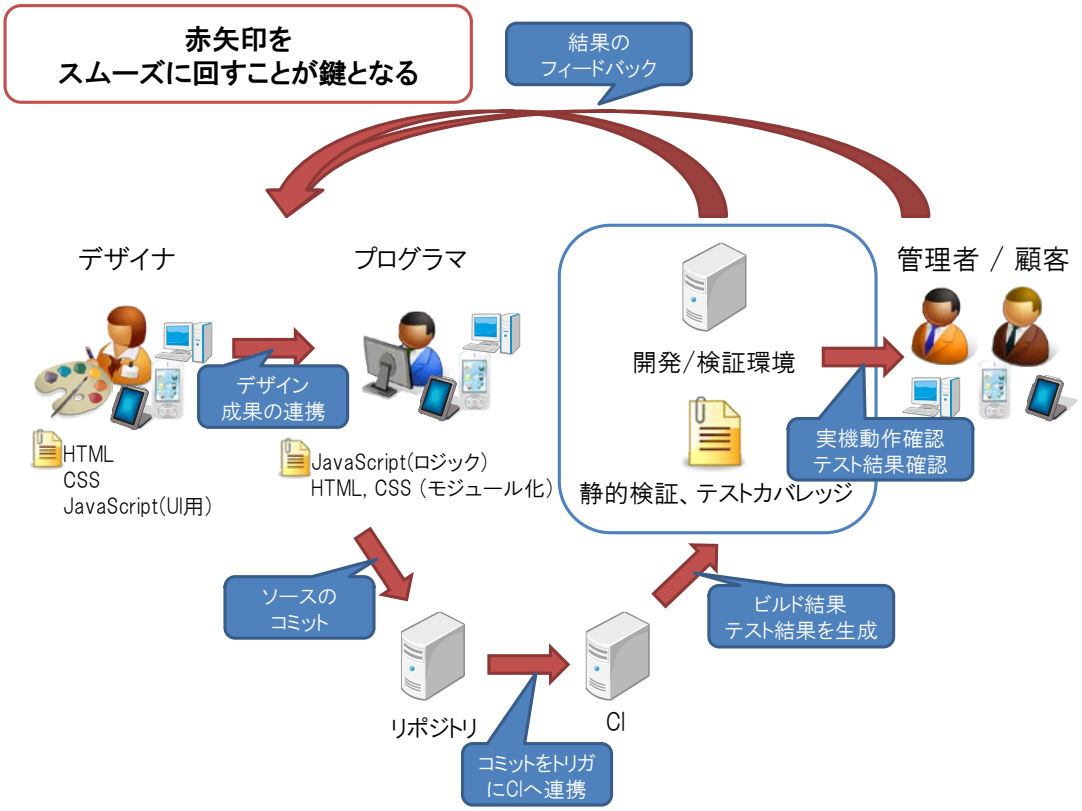

図 3 高速にワークフローサイクルを回すことが鍵

nts以前の構成で開発したものは多少書き換えが必要になるだろうが、アーキテクチャは再利用できることになるだろう。Webを取り巻く変化の速度はとても速く、常に追いつく覚悟で望まないと日々高まる要求に応えられない結果となるだろう。

NRIではこれらトレンドを常に取り込み、高品質なソフトウェア／サービスの開発／保守を継続できるよう、日々取り組んでいる。

---

[i] Hybridcast…放送と通信を連携させたテレビサービス。画面表現にHTML5を用いている(http://www.nhk.or.jp/hybridcast/online/)。

[ii] Ajax…Asynchronous JavaScript and XMLの略称であり、JavaScriptを用いてサーバサイドAPIに非同期的に通信することを指す。また、その結果を画面に反映させる部分も含める場合がある。

[iii] スクレイピング…HTML構造を解析し、目的となるデータを収集すること。

[iv] JavaScript難読化…JavaScriptコードを読みづらくすること。JavaScriptロジックは誰でも見られる状態であるため、読まれたくないコードについては、難読化をする場合がある。

[v] CIについての詳細は、技術創発「変わる開発自動化の価値」(2013年8月発行)を参照されたい。