◆ Rubyの応用 ◆

4 Ruby on Railsと Webアプリケーション開発の変遷

一般

高橋征義（達人出版会）

Ruby と Web アプリケーション

Ruby は現在世界的に利用されているが，そこまで普及した大きな要因は Ruby on Rails（以下 Rails）という Web アプリケーションフレームワーク（以下 WAF）の活躍にあったことはよく知られている．本稿では，Rails についてのみではなく，Web アプリケーション開発の歴史的な背景も含め，簡単に紹介する．

Web の特徴とその開発手法の変遷

「Web アプリケーション」というソフトウェアカテゴリは大変若い．World Wide Web そのものの誕生が 1990 年であり，一般に普及したのは 1990 年代後半になってからである．

Web をシステム面から考えると，「サーバ・クライアント方式による，スケーラブルな分散型ハイパーメディアシステム」であるといえる．この「スケーラブル」という特徴を実現するため，ステートレスなプロトコルである HTTP，ユニークな名前付けを可能とする URL があり，またハイパーメディアとして HTML と MIME タイプを駆使している．

この特徴の中でも，HTTP がステートレスであったことと，HTML というプレーンテキストによる簡素なマークアップ記法が使われたことの意味は非常に大きかった．この 2 つの特徴を利用して，標準入出力と環境変数のみを使ってインタラクティブなサービスを Web で提供するための仕組みである CGI が作られた．1990 年代当時，Mac や Windows 95 などの OS で GUI（グラフィックユーザインタフェース）アプリケーションが開発・利用されることは当たり前となっていたが，それに比べると CGI を使ったアプリケーション開発は非常に単純であり，かつ OS に依存しない高いポータビリティを確保できた．加えて，世界中からサービスを利用できる，という利便性は何物にも変えがたかった．HTML の表現力の乏しさは当時から課題ではあったが，むしろそれが良い意味での制約となり，開発のハードルを下げることに貢献した．

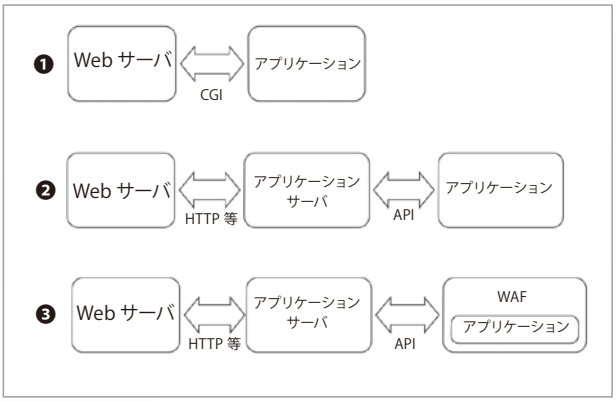

図 -1　Web アプリケーション開発の変遷

Web アプリケーション開発の流れを大きくまとめてみる（図 -1）．

1. ライブラリ等を使った CGI
2. アプリケーションサーバ
3. アプリケーションサーバ＋ WAF

CGI アプリケーションは，シェルスクリプトでも簡単に記述できるため，UNIX 系の OS では簡単に導入された．また，シェルスクリプトよりも高機能なテキスト処理が利用でき，ライブラリ化も容易な Perl が流行した．

しかし，CGI では新規リクエストがあるたびに新

しいプロセスを作るため，その起動コストが問題となった．この問題を解決するには，何かしらの永続的なプロセスを用意し，Web サーバとそのプロセスが何かしらの通信を行うことで処理を実現することが必要となった．そこでアプリケーションサーバが導入された．

特に Java の場合，Java のプロセスの起動コストは非常に大きかったため，アプリケーションサーバの導入が必須であった．そこで，Java サーブレットの仕様とともに，Tomcat などのサーブレットコンテナが広まった．また，CGI ではストレージとしてファイルシステムを使うことが多かったが，Java では JDBC の利用により，データベースサーバも別途用意するようになり，「Web サーバ + アプリケーションサーバ +DB サーバ」という 3 層モデルのパターンが生まれた．

一方，スクリプト言語においては，CGI の資産を利用すること，また CGI 時代にはレンタルサーバなどサーバ利用に対する制限の厳しい環境も多かったためか，サーバプロセスはあまり導入されなかった．その代わり，折衷案的な対応として，FastCGI のように，単純に CGI のプロセスの永続化を図る仕組みなども利用された．

もっとも，アプリケーションサーバが導入されても，同時に WAF が生まれたわけではない．Web アプリケーション開発の文脈でフレームワークの概念を普及させたのは何よりも Struts の登場が大きかった．

WAF が流行ったのは，Web にはできることが限られたためでもある．HTTP は，1 つのリクエストに 1 つのレスポンスが対応する，単純なプロトコルである．また GUI の部品としても，フォームなどに使える要素は限られており，JavaScript 等によるクライアントアプリケーション側でできる操作は限定的なものでしかなかった．

このように，ある程度決まりきったことしかできない制約の中で，その制約を利用して省力化を図りつつ，アプリケーションごとの自由度を保ちたい，というのが Web アプリケーション開発の条件であった．Struts にしろそれ以外の WAF にしろ，このような制約を前提に開発されていた．

Struts は，WAF を一般的にし，Tomcat+Struts の組合せは Java による Web アプリケーション開発を広めた．スクリプト言語による Web アプリケーション開発でも，Strtus を参考にした WAF が登場したりもしたが，Struts と同様に，あるいはそれを越えて普及する WAF が生まれるまでには至らなかった．そのため，簡易なアプリケーションであれば Perl や PHP などのスクリプト言語で，本格的なアプリケーションは Java で，といった風潮も感じられた．

そのような流れを大きく変えることになったのが，Rails である．

Rails の登場

Rails は 2004 年に公開された．発表時点では，いくつかある新しい WAF の 1 つ，といった位置づけに過ぎなかったが，2005 年頃には Ruby とも関係のない，Web アプリケーション開発の文脈で「Java の 10 倍の生産性」という扇情的なキャッチコピーや，「15 分でブログを作るデモ動画」などのキャッチーな宣伝も伴って，一躍有名となった．

それ以来，Ruby 製の WAF としてはもちろん，WAF 全体を見渡しても，Rails は大きな影響を与え続けている．Rails 2 の時代には，Rails への対抗馬として，モジュール性を高めた新たな WAF，merb が現れたが，Rails 3 では Rails と merb が統合するという驚きの判断が行われ，双方の長所を併せ持つ新 Rails が誕生し，今に至っている．

◆ Rails の特徴

Rails について説明する際，Rails 作者の David は「Opinionated」という言葉をよく使っていた．この表現の通り，Rails は非常に自己主張の強いフレームワークであった．Rails を理解するにはほかのフレームワークと比較すると分かりやすいだろう．

Rails の特徴をいくつか挙げる．

1. フルスタック MVC（Model View Controller）フレームワーク
2. 疎結合を避け，規約を重視することによる開発効率の最大化
3. メタプログラミングやジェネレータの利用による開発効率の最大化

最初の特徴として，Rails はフルスタックの MVC フレームワークであることを前面に押し出していた．当時のフレームワークはさまざまなライブラリを組み合わせて自由にカスタマイズできることを売りにするものも少なくなかったが，Rails は Model のための O/R マッパーや View のためのテンプレートライブラリなど，必要なものを全部取り込み，それだけでアプリケーションが開発できるようになっていた．

また，Web アプリケーションを開発するにあたってアーキテクチャを考える際，当時よく参照されていた PoEAA こと『エンタープライズアプリケーションアーキテクチャパターン』（マーチン・ファウラー）を強く意識していた．O/R マッパーが「ActiveRecord」という名前なのも，この書籍の Active Record パターンを元にして命名されている．先ほど触れた MVC パターンも，本書の Web プレゼンテーションパターンとして紹介されている．

もっとも，PoEAA では Active Record パターンよりも Data Mapper パターンの方がより高度なものであるとして推奨している傾向があるのだが，David はあくまで Active Record パターンを拡張したものの方がシンプルかつ必要十分であるとしている．このような割り切りの良さは Rails の随所に見られる．

2 番目の特徴は 1 番目の特徴にも関係しているが，これは Struts などで課題になっていた，主に XML での設定等のカスタマイズ性に対する Rails の回答でもある．

Struts のカスタマイズ性は，Java の多様なライブラリの利用と，XML による柔軟な設定によって担保されていた．Java はコンパイル言語であるため，ライブラリ等はバイナリの形で再利用される．コンパイル済み Java バイトコードに対する操作などはまだ広く用いられていなかった当時では，拡張ポイントとしては XML の設定用言語と Java の高度な知識を必要とした．そのため，自由に設定ができないか，あるいは大量の設定項目が用意されてしまうといった問題が起こりがちだった．

Rails はこれに対し，最低限の設定ファイルを YAML の形で持つ以外は，規約によって設定を行わずに済ませる，といった方法を打ち出した．Model/View/Controller のファイルの場所も決まっており，またコントローラのクラス名と URL とのマッピング，DB のテーブルとクラス名といったものも標準で対応づけさせた（しかもなぜかモデルやコントローラのクラス名は単数形なのにテーブル名や URL は複数形，といった無意味に凝った対応付けであった）．

また，この実現にはメタプログラミングも積極的に使われた．とりわけ ActiveRecord のそれは極端で，プログラム内ではクラスを定義するだけで，実行時に DB からスキーマ情報を取り出し，フィールド名を元にしたメソッドが動的に定義されるようになっていたり，また 1 対多のテーブル間のリレーションシップなども「has_many:users」のような自然言語文のような行が 1 行だけで表現できるようになっていた．

同時に，アプリケーションファイル群の雛形だけではなく，モデルクラスやビューテンプレート，テストコード等についても自動生成するためのジェネレータが用意されていた．モデルのテーブル名，各フィールド名とその型を指定すると，モデル，ビュー，コントローラ，テストなどまとめて生成する scaffold も導入され，前述の 15 分デモのような場面でも多用された．

Ruby の理由

Rails が Ruby で作られたことについては，ある種の偶然の結果という側面はある．しかし，Ruby だからこそできたところもあるのではないだろうか．実際，David は「Ruby has just a deep emotional appeal of how beautiful you can write some-

thing.」などと発言するなど，Ruby については最大限の賛辞を隠さない．

前節で「Rails の特徴」として挙げたメタプログラミングは，Ruby の言語的特徴と強く結びついている．たとえばスクリプト言語の中でも比較的動的な特性が少ない PHP では，Rails のように既存のクラスに対して自由にメソッドを追加することができなかった．

また，Rails での「設定」は，YAML ファイルだけではなく Ruby で書かれたコードそのものも含んでいる．これは今回の特集での別記事で書かれているドメイン特化言語（DSL）の影響下にある．このような意味で，Ruby でなければ Rails はこのようにはならなかったということは言えるだろう．

もっとも，メタプログラミングや DSL には常に「やり過ぎ」の危険が伴う．メタプログラミングを実現するために初期の Rails では monkey patching が多用された．monkey patching によるメソッドの書き換えなどは，単にコードが分かりづらくなるだけではなく，複数のパッチが干渉したり，バージョンが変わると動作しなくなる等の弊害も生まれることがあった．

このような，ある意味で乱暴な手法は，プログラミング言語によっては可能であっても積極的には行うべきではない，という文化的な判断が下されるコミュニティもあるだろう．その点，Ruby ではある程度の危険なことであっても許される文化があった．このような寛容な態度がなければ，Rails のようなプロダクトは決して生まれることはなかっただろう．

Rails と Ruby の将来

2015 年現在においても，Rails が代表的な WAF の位置を確保し続けている．しかしながら，Rails の将来はそれほど明るいわけではない．

10 年にも渡って Rails が WAF のトップを走り続けていられるのは，もちろん Rails とその開発チームの素晴らしさの賜物である．しかしながら，Web アプリケーションという分野そのものの位置づけが大きく変わり，現在はそもそも WAF が主戦場ではなくなった，すなわち積極的に WAF を開発する時代ではなくなっている，という要因もあるのではないだろうか．

現在のトレンドはスマートフォンアプリやクライアントサイドアプリケーションといった，クライアント側でのアプリケーション開発にある．スマートフォンアプリであれば，サーバ側は単に API の提供のみとなり，Rails のようなフルスタックな WAF はオーバースペックになりかねない．むしろ，高速・省リソースでリクエストをさばける Erlang や Go のような言語や，その上で作られた薄いフレームワークの方が好まれるようになるかもしれない．もちろん，Rails 側もそのような変化に対応するべく，Rails 5 では API サーバ制作用の Rails API gem が Rails 本体に取り込まれたり，WebSocket を使うための Action Cable を導入が予定されている．

冒頭に述べた通り，Web アプリケーションは大変若く，またその変化も激しい．Rails はその Web アプリケーション開発に大きな役割を担うこととなり，そのため Rails 自体も激しく変化してきた．その変化は Ruby そのものにも及んだが，Ruby と Rails は互いに良い影響を与え合い続けてこれたことが両者の成功の要因の 1 つだろう．今後もその幸せな関係が続くことを期待したい．

（2015 年 8 月 21 日受付）

高橋征義　takahashim@tatsu-zine.com

北海道大学工学部情報工学科卒業．（株）達人出版会代表取締役，一般社団法人日本 Ruby の会代表理事．