エンタープライズ向け Java 標準化の歴史

■企業情報システムを支える Java

Java で Web アプリケーションを開発する際にフレームワークを利用することは、すでに当たり前になっています。Java の Web 開発フレームワークは、細かい技術領域ごとに商用／オープンソースを問わず多数のプロダクトが入り乱れており、それらの特長／メリットもさまざまです。そこで、プロジェクトの特性に応じたフレームワークをその都度選択して組み合わせて使うことが一般的になっています。

一方で、基幹系アプリケーションを含む大規模なシステムの構築をミッションとするユーザー企業の情報システム部などでは、限られた人材で品質と生産性を安定的に維持するために、「自社のすべてのアプリケーション開発を支える『共通基盤』を整備したい」というニーズが根強くあります。この場合は、Java の Web アプリケーション開発のための基盤についてもその都度選択するのではなく、何らかの汎用的な標準を定めて体制や共通ライブラリなどの整備を行うことが求められます。

Java の場合、汎用的な利用を想定した標準技術としては、企業情報システムの構築に必要な API 仕様を公式の標準化プロセスによって定めた「Java EE」(Java Platform, Enterprise Edition)があります。しかし、これまでこの Java EE の標準機能だけでは開発に必要な機能がそろわず、上記のような『共通基盤』を検討するケースにおいても標準以外のフレームワークを別途ユーザー自身が選定することが一般的になっていました。とはいえ、Java のフレームワークが非常に多様であるために、汎用性を維持しつつ全体最適な標準を定めることに課題を抱えているケースが多く見られました。

ところが、上記のような Java の技術環境は、ここ 5 年ほどで大きく変わってきています。公式の標準である Java EE が、オープンソースの世界で発展してきた(非公式な)デファクト標準のフレームワーク技術を取り込んで標準規格化する、という構図が顕著になり、少なくとも従来のフレームワークが提供してきた主要な機能は Java EE 標準のみで十分にカバーできる環境がすでに整っています。

こうした流れから、今後の Java のフレームワークは、標準規格化されたフレームワーク技術の仕様に準拠して、汎用的な機能や設計ノウハウを提供することが一層求められていくと考えられます。前述のような『共通基盤』構築のケースでは、できる限りこうした公的な標準に準拠した技術を選択することで、「汎用的に長い間使える標準基盤」を整えることができるようになります(図1)。

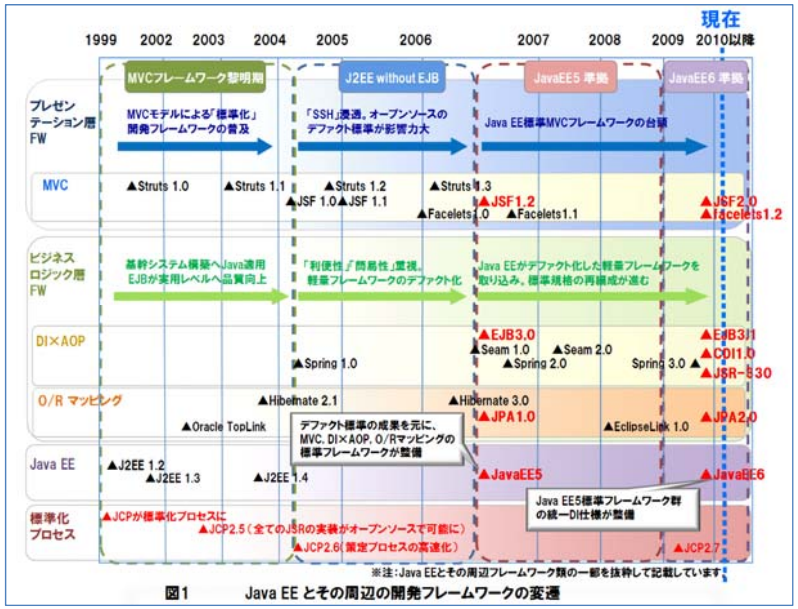

図1　Java EE とその周辺の開発フレームワークの変遷

■Java の仕様標準化プロセスの変遷

技術的な変遷を追う前に、まず Java の標準化プロセスについて簡単に確認しておきましょう。

現在の Java では、API の「仕様」と「実装」が明確に区別されています。仕様は実装とは独立して文書化されており、例えば、「Servlet」の仕様に従ってアプリケーションを作れば、Java EE 仕様に準拠して実装されたどのサーブレット・コンテナの上でも正しく動作することが期待できます。そして、Java EE 仕様の対応製品を作ることが、ベンダーだけでなくオープンソース・コミュニティによっても認められており、米 IBM、米 Oracle、米 Red Hat (JBoss)など複数のベンダーやコミュニティが、Java EE 対応アプリケーション・サーバー製品をリリースしています。こうした「仕様と実装の分離」とそのオープン性は Java の大きな特長であり、現在のエンタープライズ分野での繁栄の一因ともなっていると考えられます。

こうした Java の標準化プロセスは、どのように発展してきたのでしょうか。

まず、1999 年に標準化プロセスとして「JCP」(Java Community Process)が導入され、「JSR」(Java Specification Request)の提案をもとに、仕様の文書化と、そのお手本であるリファレンス実装(参照実装)を分けて開発する現在の仕組みが形作られました。またこの年、参照実装をオープンソース化する初の試みが行われ、サーブレット・コンテナの「Tomcat」が誕生します。その後、2002 年に発表された「JCP 2.5」によってすべての JSR のオープンソースによる実装が可能となりました。その後、2005 年には Java EE アプリケーション・サーバー全体をオープンソースで開発する「Project GlassFish」が始まり、仕様がリリースされるとともに、利用可能な参照実装がオープンソースで提供される流れが定着しました。

このように、Java は当初から仕様策定と参照実装のオープン性を意識しており、コミュニティの知見を標準仕様へと反映させる体制を意識的に作り上げてきました。そして、オープンソース・コミュニティから生まれた多くの開発技術が標準仕様へ還元されつつあります。

■エンタープライズ利用の道を開いた MVC フレームワーク

サーバーサイド Java の最も基本的な機能である Servlet と「JSP」(Java Server Pages)は、「JDBC」(Java Database Connectivity)や「JTA」(Java Transaction API)といったほかの基本的な API 群とともに 1999 年に「J2EE」(Java 2 Platform, Enterprise Edition) 1.2 にまとめられました。これによって、Java で企業システム向け Web アプリケーションを開発するための最低限の基盤が整いました。

しかし、この時点では Web アプリケーション開発のための素材が提供されただけであり、こうしたシンプルな API を使って実際の開発をするにはノウハウが必要でした。そこで、アプリケーションに一定の構造的な枠組みを与え、汎用的な機能を部品化することで設計とコーディングを簡単化する「開発フレームワーク」が登場します。特に、プログラムを Model、View、Controller の役割に分割して並行開発を可能とする MVC フレームワーク(図 2)がその中心となりました。

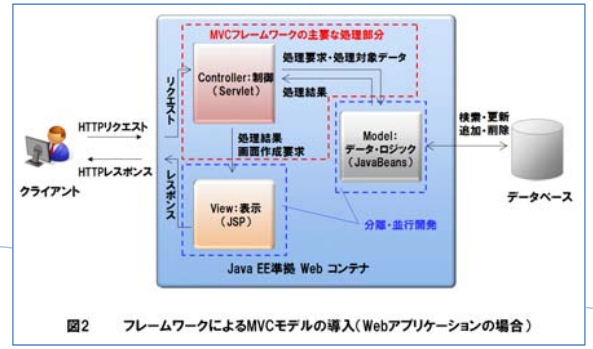

図2　フレームワークによるMVCモデルの導入(Webアプリケーションの場合)

この時期に多くのベンダーが MVC フレームワーク製品を開発・提供しましたが、オープンソースの世界では 2001 年に「Struts」が登場します。画面アクションごとにフォーム格納クラスやアクション・クラスを作成するというシンプルな構造であり、その取り組みやすさや無償であることなどから爆発的に普及が進みました。この後しばらく、Struts はオープンソース MVC フレームワークのデファクト標準として認知されるようになります。

これらの商用／オープンソースの MVC フレームワークは、Servlet＋JSP といった素の API だけでは難しかった Web アプリケーションの設計・開発を、一定のスキル・レベルでも安定して実施できるようにすることで、Java のエンタープライズ利用への道を開いたと言えます。ですが、この後しばらく、いずれかの MVC フレームワークが Java EE 標準として整理されることはありませんでした。

■軽量フレームワークの台頭 ～J2EE without EJB～

MVC フレームワークによって企業システムに Java が普及していくなか、Java EE は着々と企業システム開発に必要な機能を整備していきました。J2EE 1.3(2001 年)や J2EE 1.4(2003 年)では、「EJB」(Enterprise JavaBeans)(2.0、2.1)によってトランザクション管理やデータベース・アクセスといった機能を大規模／分散環境に提供する基盤が整えられました。

しかし、EJB は早くから重厚で難解な仕様が問題とされ、EJB を隠ぺい化してアプリケーション開発者に意識させないようにするためのデザイン・パターンやフレームワークが開発されました。

一方で、EJB が提供するような重厚な機能群を必要としない開発現場では、より簡単にアプリケーションを作成するための技術の研究が進みました。そして登場したのが「Spring」や「Seasar2」に代表される「DI コンテナ」(図 3)と、「Hibernate」や「TopLink」といった「O/R(Object/Rerational)マッピング・ツール」(図 4)です。これらはいずれもオープンソースのプロダクトとして登場し、EJB 2.x が実現していたことをより簡単に実現できる環境を作りました。

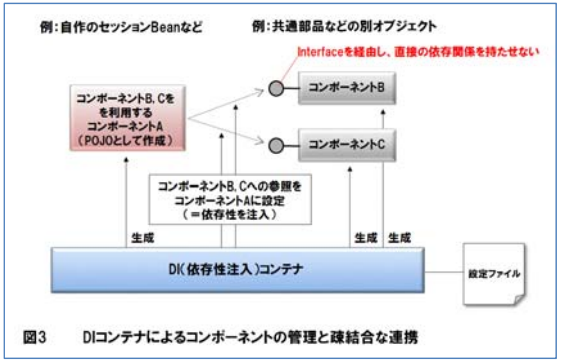

図3　DIコンテナによるコンポーネントの管理と疎結合な連携

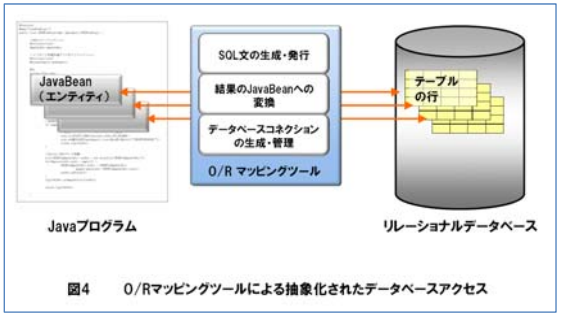

図4　O/Rマッピングツールによる抽象化されたデータベースアクセス

DI（Dependency Injection）コンテナは、EJB のような重量級のコンポーネント管理の仕組みを使わずにコンポーネント間の依存関係をフレームワーク側から制御し、アプリケーションを「POJO」（Plain Old Java Object、特定のフレームワークやコンテナが提供する機能を継承／実装しないクラスのこと）として開発するスタイルを確立しました。また、このDIを応用して、ログ出力や認証などの横断的処理を外部から差し込む「AOP」（Aspect Oriented Programming、アスペクト指向プログラミング）機能も提供されました。こうした技術で EJB の重要な機能であった宣言的トランザクション管理なども容易に実現できるようになったことから、「EJB コンテナは不要」という考えが加速しました。

O/R マッピング・ツールは、EJB 2.x が備えていたエンティティ Bean による Java オブジェクトと関係データベースの自動マッピング機能を、やはりシンプルな POJO ベースで可能にし、データベース・アクセス処理の抽象化をよりスマートに実現しました。

■フレームワークの組み合わせ利用が一般化

これらの軽量フレームワークが高い生産性を発揮したことから、2003 年～2006 年ごろには、MVC、DI×AOP、O/R マッピングの各機能について、それぞれデファクト化したいくつかの選択肢からプロダクトを選定し、組み合わせて利用することが一般的になりました。特に Struts-Spring-Hibernate といった組み合わせが有名になり、EJB コンテナを持たないサーブレット・コンテナ上の軽量フレームワーク開発が広まりました。この時期は、公的な標準である Java EE（J2EE）が相対的に影響力を弱めていたと言えるでしょう。

また、技術領域ごとに競争が促進されることで多数の優れたプロダクトが生み出される一方で、フレームワークの乱立による汎用的な標準基盤の整理の難しさという、冒頭で述べたような問題が顕在化してきたのもこのころと言って良いでしょう。

■標準規格の再編成 ～Java EE5～

軽量フレームワークによる開発が一般的になっていた 2006 年、Java EE 5 がリリースされました。それまでのバージョン呼称（J2EE 1.x）を改めたこのリリースは、J2EE 時代から大きな変化を遂げていました。特に、企業システム開発における技術標準の整理の観点から見ると、2 つの点が重要であったと筆者は考えています。1 つは、デファクト標準化した優れたオープンソースの技術を Java EE へ迅速に取り込む構図を作った点、もう 1 つは、MVC フレームワークの標準規格を定めた点です。

1 つ目のポイントについては、オープン性を指向する Java 本来の標準化プロセスがうまく機能した結果と言えると思います。EJB3 の仕様策定グループは、J2EE 不要を唱（とな）えてデファクト化していた DI コンテナの技術を大幅に取り込むとともに、オープンソース O/R マッピング・ツール「Hibernate」の作者・Gavin King 氏をメンバーに加えることでそのノウハウを標準規格として整理しました。

その結果、Java EE 5 に含まれる EJB 3.0 は、EJB コンテナ自体に DI×AOP 機能を備え、従来複雑だった EJB コンポーネントを POJO で開発できる軽量フレームワークとして刷新されました。また、POJO ベースの O/R マッピング仕様である「JPA」（Java Persistence API）も策定され、Java EE の主要機能が軽量フレームワークと同じ容易さで利用できるようになりました。こうした、デファクト標準と密接に連携した標準化体制は、Java EE 6 の策定でもそのまま生かされています。

2 つ目のポイントは、一見地味ですが、企業システム開発の標準基盤の整備に向けて大きな意味を持っていたと考えています。Java EE 5 では標準 MVC フレームワークとして「JSF」(JavaServer Faces)が選定されました。標準規格になれば、Java EE 対応のアプリケーション・サーバー製品は JSF の機能を内蔵することになります。すなわち、ユーザーは自分で何らかの MVC フレームワークを選定する必要がなくなるということです。

従来、スキル・レベルの統一が難しい企業のシステム開発では、MVC フレームワークが、開発を標準化／統制するための最も重要な技術要素でした。Servlet を中心に View 側と Model 側とでそれぞれ開発対象を定義し、それを軸に設計書や設計手順を標準化して、自社用の開発標準を構築していたのです。この部分が標準規格として整備されることは、冒頭で述べたような、企業におけるシステム開発の『共通基盤』の選択肢として、汎用的でライフ・タイムの長い技術を提供してもらえることを意味します。

これらの観点から見た Java EE 5 は、単なる開発技術の進歩というだけではなく、Java EE がデファクト標準の成果をもとに、アプリケーションの作り方に標準的な枠組みを与える方向へレイヤーを拡大していく(図 5)という、今後の Java 標準基盤の方向性を指し示してくれるものだったと言えると思います。

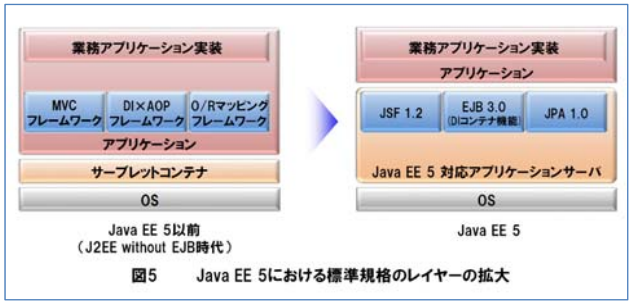

図5　Java EE 5における標準規格のレイヤーの拡大

ただし、Java EE 5 の時点では、まだ標準化が不十分な領域が残されていました。それを補完するための試みは、Java EE 5 のリリース時点ですでに始められていました。それが次に紹介する「JBoss Seam」です。

■フレームワークの統合とその標準化の流れ

Java EE 5 では、JSF、EJB3、JPA という技術領域ごとの標準仕様が規定されました。しかし、JSF と EJB3(JPA 含む)はもともと設計の土台が異なっており、それぞれ別のコンポーネント・モデルを持っています。こうした別々のフレームワークを併用するには、フレームワーク同士を結びつけるための手続きが必要です。

前述の Struts-Spring-Hibernate では、選定してきたこれらのフレームワークをつなぎ合わせる役割は主に DI コンテナの Spring が果たしていました。しかし、こうした組み合わせの技術は当然、標準化されたものではありません。違うフレームワークを選定した場合は、その仕様に応じて組み合わせ方を再検討する必要があります。こうした「フレームワークの組み合わせにかかるコスト」に対する問題意識は強く、すべてのフレームワーク機能を統一的にサポートするフルスタックのフレームワークの必要性が意識されるようになってきていました。

そこで、Java EE においてフルスタック・フレームワークを実現するために登場したのが「JBoss Seam」(以下、Seam と略す)です。Seam は、JSF と EJB3 のコンポーネントに統一的なアノテーションを付与することで、それらをすべて「Seam コンポーネント」として一元管理し、相互に DI を実行する機能を提供します。

Seam は「縫い合わせる」という名前の通り、フレームワーク間に統一的な DI×AOP の機能を提供することで、フレームワークを統合する役割を担っています。Seam は Java EE 5 標準フレームワーク群を統合するというその趣旨から、早くから Java EE 6 での標準仕様化が検討されていました。このほかにも「コンテキスト」と呼ばれるコンポーネントのライフ・サイクル管理機能などが今後の標準仕様の中核になると予想されていました。

NRI では 2006 年ごろから Java EE の拡大と統合の動きを見据えて次世代の Web 開発フレームワークの開発に取り組み、「ObjectWorks＋ R1.0」の Java EE 5 対応版としてリリースしています。これは、Java EE 5 をサポートするアプリケーション・サーバー上で Seam を使って Java EE 6 と同様の構成を作り、その上で企業システム開発向けに必要な部品群と、開発ノウハウを提供するフレームワークです。

■そして Java EE 6 へ

最後に、Java EE をめぐる直近の動きを見ておきましょう。

2009 年 12 月、Java EE 6 の仕様が最終リリースされました。その最大の特長の 1 つが、Java EE の標準規格となったフレームワーク群全体にわたる統一 DI 仕様を定めたことです(図 6)。この DI 仕様は「JSR-299:Context and Dependency Injection for Java EE Platform」(以下 CDI と略す)と呼ばれ、Seam のコンテキストと DI の仕様が元になっ

ています。これにより、Servlet, JSF, EJB3, JPA, JMS（Java Message Service）, JAX-WS（Java API for XML-Based Web Services), JAX-RS（同 RESTful Web Services）といった Java EE のコンポーネントがすべて、ひとつの DI 仕様のもとで統合できるようになりました。

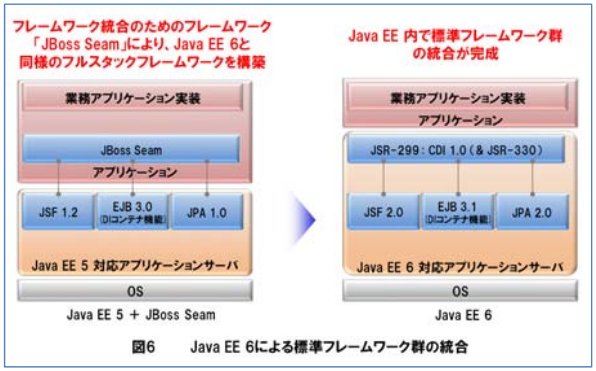

図6　Java EE 6による標準フレームワーク群の統合

同時に、「JSR-330： Dependency Injection for Java」で DI に用いるアノテーションの仕様が標準化されました。JSR-330 の策定には、「J2EE without EJB」時代にデファクト化した DI コンテナの Spring と Google Guice のコミュニティが携わっており、これらの DI コンテナと CDI とは、JSR-330 で定めた同じアノテーション仕様を共有しています。

CDI は、当初は「Web Beans」という名称で参照実装が開発されていましたが、最終的には「Weld」（溶接する）という名称になりました。Weld は Java EE 6 の参照実装アプリケーション・サーバーである GlassFish v3 にも搭載されています。

米 Sun Microsystems の Web サイトでは、GlassFish v3 を同梱（どうこん）した統合開発環境の「NetBeans 6.8」や「GlassFish Tools Bundle for Eclispe」が配布されており、実際に Java EE 6 を使ったアプリケーション開発を簡単に試してみることができます。

紹介した各種フレームワークが標準で搭載されていますので、フレームワーク・ライブラリを別途用意したりする必要はなく、例えばシンプルな Web アプリケーション開発プロジェクトに beans.xml（CDI の設定ファイル）を配置するだけで標準 DI×AOP 機能を利用することができるようになります。これこそ、フレームワークの標準仕様化の恩恵といえます。

また、EJB 3.1 では EAR ファイルではなく WAR ファイル内に EJB コンポーネントを格納できるなど、より簡単に Java EE アプリケーションを開発できる工夫が詰め込まれており、実際に試してみるとその手軽さに驚かれると思います。ぜひ試してみてください。

そして、Java EE 6 はリリースされましたが、現時点では、「Java EE 5 対応サーバー＋JBoss Seam」で構成された、Java EE 6 と同様のフレームワークが、企業システムでの実績を積み重ねている段階で、主要なアプリケーション・サーバー製品の対応が出そろうには、もう少し時間がかかるものと思われます。

【執筆者】 深津 康行
株式会社野村総合研究所（NRI） 基盤ソリューション事業本部 副主任システムコンサルタント。入社以来、Java/Web 系フレームワークの開発・導入支援や、開発標準化コンサルティングなどに従事。現在は NRI の SI フレームワーク「ObjectWorks＋」を中心に、開発基盤ソリューションの企画と顧客への提案を行っている。

※本文章は、2010 年 1 月[Think IT]に掲載されたものを再編集しています。