Java EE 上の SI 向けフレームワーク

■オープン・スタンダードだけでは足りないもの

Java による企業システム開発において現時点でのオープン・スタンダードである「Java EE 5」を補完するのが「JBoss Seam」(以下、Seam)ですが、こうした標準技術を習得したメンバーがある程度そろった開発体制であれば、Seam を使って標準フレームワーク群を組み合わせるだけで、生産性が高く今後の拡張にも柔軟に対応できるアプリケーション開発を実現できることと思います。

ところが一方で、スキルを持つ人材の確保が困難であったり、複数のベンダーやオフショアを活用して大規模な基幹系アプリケーションを開発するシステム・インテグレーション(SI)の現場では、こうしたオープン・スタンダードだけでは高い品質／生産性を安定的に維持するのに必要なものが欠けている場合があります。野村総合研究所(NRI)の SI における経験をもとに整理すると、欠けているものは大きく以下の 2 点に分類されると思います。

1. SI 案件向けにきめ細かな拡張機能群
2. 設計／開発を標準化するための仕組み

1 つ目は、端的に言えば、Java EE 仕様上の拡張部品群です。オープンソースを中心に整備されている Java EE は、汎用的に利用できる標準規格である以上、どうしても個々のプロジェクトで必要となるきめ細かな機能が網羅されていません。

特に、日本という地域性が特殊要件の源になっている部分があります。例えば、日本語の全角文字をチェックするためのバリデータ部品などは、英語圏を中心とするオープンソース・コミュニティではどうしても整備が追いつきません。また、アプリケーションの流量制御や閉塞制御といった機能についても、よりきめ細かい制御が望まれます。こうした、標準機能を補完する拡張的な機能群が、必要とされることが多々あります。

2 つ目は、一定の人材／スキルという制約の上で安定的にアプリケーション開発を行うための各種の仕組みです。これは「フレームワーク」が非常に大きな役割を発揮している領域ですが、SI においてはそれだけでは十分ではありません。

フレームワーク上で開発するアプリケーションのアーキテクチャーの標準化や開発時のルールなど、各種の取り決めがソフトウエア部品としてのフレームワークを補完する必要があります。SI においては、そうした各種の標準化やルールを内包した、広義のフレームワークが必要です。(ここでは、これを「SI フレームワーク」と呼ぶことにします。)

では、Java EE の基盤をフルに活用しつつ安定的に品質／生産性の高い開発を進めていくために、NRI が開発／提供している SI フレームワーク「ObjectWorks＋(オブジェクトワークス・プラス)」を紹介します。

■NRI の開発フレームワークへの取り組み ～ObjectWorks＋～

NRI は、Java サーバ・サイド開発の黎明期である 1999 年頃から、Java による Web アプリケーション開発のフレームワーク「オブジェクトワークス」を提供しています。オブジェクトワークスは、J2EE の拡張に合わせて EJB 対応、Web サービス対応といった標準機能への対応を重ねてきました。

2006 年に Java EE 5 が登場した際、NRI ではその技術仕様の標準化動向に対するインパクトの大きさと、今後の Java EE の技術標準の整理／統合の流れを見据え、オープン・スタンダードだけで従来のフレームワークの大部分をカバーする「Java EE 5」をベースに、フレームワークの新しいエディションを開発しました。この際に製品名称も改め、「ObjectWorks＋」としました(※従来の J2EE ベースのフレームワークも、同一ブランドで現在でも継続的に提供しています)。

ObjectWorks＋ Java EE 5 対応版(以下、単に ObjectWorks＋とします)は、Java EE 5 準拠のアプリケーション・サーバーを前提環境に、Seam を利用することで、Java EE 6 の統一 DI(Dependency Injection)基盤と同様の仕様で、アプリケーション開発全体の標準技術によるサポートを実現しています。

ObjectWorks＋の Web 開発フレームワークは、実行基盤(ObjectWorks＋/CORE、以下、CORE)と開発基盤(ObjectWorks＋/STUDIO、以下、STUDIO)に分かれます(図 1)。

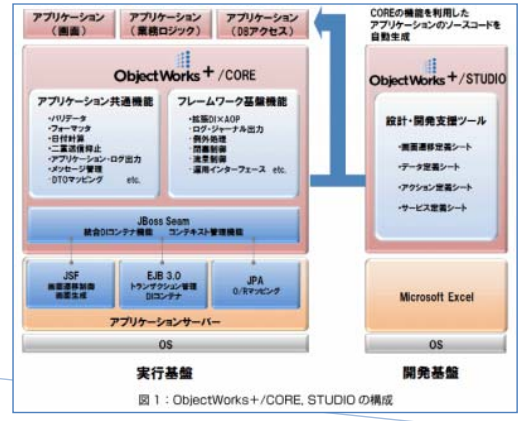

図 1：ObjectWorks＋/CORE, STUDIO の構成

前述の「オープン・スタンダードだけでは足りないもの」との対比で言えば、

ObjectWorks＋/CORE ： 拡張機能群
ObjectWorks＋/STUDIO ： 設計／開発の標準化ツール

となります。まず、CORE の特長を解説します。

（なお ObjectWorks＋は、核となる Web 開発フレームワーク機能のほかに、帳票出力、認証、ワークフローといった、業務システムで必要となる基本的な機能群を追加コンポーネントとして提供する、ミドルウエアの集合体でもありますが、ここでは Web 開発の領域にフォーカスして解説します）

■拡張機能［1］アプリケーション共通機能

CORE の提供する拡張機能の 1 つ目は、「アプリケーション共通機能」です。これは、文字通り、開発者が業務アプリケーションを実装する際に、共通部品として利用することになる機能群です。提供されている主なアプリケーション共通機能は以下の通りです。

・バリデータ
・フォーマッタ
・二重送信抑止
・コード・マスタ（テーブルのキャッシュ参照機能）
・各種 Java API（アプリケーション・ログ出力、メッセージ管理、日付計算、etc.）
・各種 JavaScript（クライアント・サイド二重送信抑止、キー抑止、etc.）

最も種類の多い拡張部品としては、バリデータ部品があります。ObjectWorks＋では、値のバリデーション情報を画面ファイル（JSP）のタグ内に記述する通常の JSF の方式と異なり、入力されたデータを格納する Java クラス（Model）にアノテーションとして記述する、「モデル・バリデータ」方式となっています。こうすることで、あるデータ項目のバリデーション情報を複数の画面ファイルへ分散させず、一元管理することができます（図 2）。

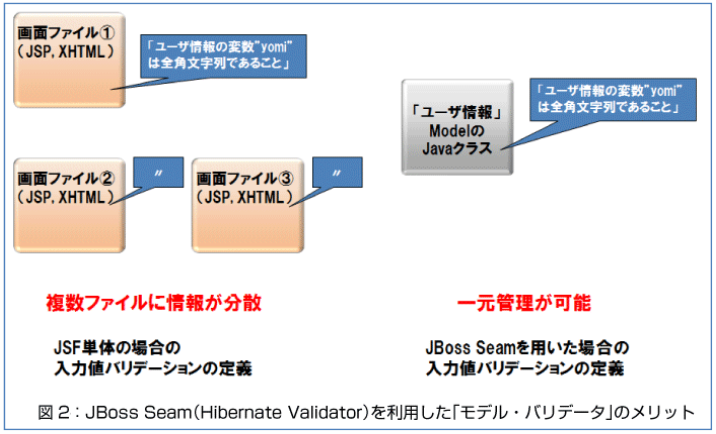

図 2：JBoss Seam（Hibernate Validator）を利用した「モデル・バリデータ」のメリット

CORE は、データのやり取りに用いる DTO（Data Transfer Object）のフィールド宣言部に付与できる、バリデーションのための各種のアノテーションを提供しています。

```
public UserInfo {
    @KSSCheckZenkaku // ← 変数”yomi”が全角文字列であることをチェックするバリデータ
    private String yomi;
    …
}
```

DTO にこのアノテーションを付与しておけば、画面ファイル側ではバリデーションをかけるためのカスタム・タグを記述するだけで、この仕様に沿ったバリデーションが起動する仕組みとなります。

この機能は、Seam が内包する O/R マッピング・ツール「Hibernate」の、「Hibernate Validator」という機能を利用しています。この Hibernate Validator も、Java EE 6 では「JSR-303：Bean Validator」として標準化された仕様です。つまり、ObjectWorks＋では、Java EE 6 以降に標準規格となる仕様に基づいたバリデーションを採用しています。

このほか、統一フォーマットに従ってログを出力できるアプリケーション・ログ出力部品や、メッセージを画面に返す手順を統一するためのメッセージ管理部品、画面側でキー抑止をするための JavaScript 部品など、大抵の業務システムで必要とされ、多くの場合ユーザーが自作する必要が生じていた、汎用的な機能群を提供しています。

■拡張機能［2］フレームワーク基盤機能

拡張機能の 2 つ目は、「フレームワーク基盤機能」です。これは、個々のアプリケーション開発者というよりは、アプリケーション全体の方式設計者や基盤設計者が使用する、基盤寄りの共通機能群です。以下が、主な提供機能です。

・拡張 DI×AOP 機能
・ログ・ジャーナル出力
・例外処理
・閉塞制御、流量制御
・運用機能（JMX インターフェース、運用画面、運用コマンド）

先頭の「拡張 DI×AOP 機能」は、すべての部品群の土台となる機能です。ObjectWorks＋では前述の通り、統一 DI×AOP 機能として Seam を活用し、標準的な DI コンテナの仕様に則って拡張部品群を提供しています。ですが、現在の Saem には AOP を実現するインターセプター（処理を横取りして共通機能を差し込むクラス）を設定ファイルから利用する機能がありません。こうした機能は、Seam をベースにした「JSR-299：Java Context and Dependency Injection（CDI）」で一部標準化されており、今後 Seam でも拡張されていくことが期待されています。

Seam は本来、アプリケーション・ソースコードへアノテーションを付与することで AOP を実現し、余計な設定ファイルを作成しなくても済むことをコンセプトの 1 つとしています。しかし、こうした方式はアプリケーションの生産性を高める一方で、部品利用の管理を難しくする側面も持ちます。SI の現場では、あえてこうした仕組みを用いず、従来どおり設定ファイルで設計情報を一元化した方が良い場面もあります。そこで ObjectWorks＋では、アプリケーション全体の基盤的な設計にかかわる「フレームワーク基盤機能」の多くを、こうした設定ファイルからも利用できるようにすることで、メンテナンス性を高く保つことを指向しています。

【主要な提供機能の内容（一部）】

★ログ・ジャーナル出力

統一したログ・フォーマットで基盤的に共通して出力すべきログを容易に出力できるようにしています。種類としては、

・ログ（トレース、稼働統計、SQL 稼働統計、処理時間超過アラート）
・ジャーナル（HTTP アクセス、インターセプター、Web サービス）

などがあります。特徴的な点としては、これらログに共通する「リクエスト ID」を与え、リクエストに対するすべての処理のひも付けができるうえ、認証の仕組みと組み合わせることで「ユーザーID」を埋め込み、ユーザーごとの操作履歴などをトレースすることができるようになることなどが挙げられます。

★流量制御／閉塞制御

同時リクエスト処理数をコントロールしたり、特定のサービスを一時的に閉塞したりする機能は、一般的な Java EE のアプリケーション・サーバー製品でも提供されています。アプリケーション・サーバーでは、HTTP セッション数による流量制御や URL 単位の閉塞など、比較的大きな粒度での制御が可能ですが、SI の現場ではより細かく、特定の Java クラスだけを制御したい、といった要件がある場合があります。

ObjectWorks＋では、ビジネス・ロジックを実装した Java クラスのクラス名称に対するネーミング・ルールに従って、特定の名称の Java クラスにだけ、流量／閉塞制御インターセプターをかけるといったことが可能になっています。

★運用機能

ObjectWorks＋では、これらの拡張機能を動的に操作するための運用機能も提供しています。フレームワーク基盤機能はいずれも、Java でシステムの管理／監視を行うための標準仕様である JMX（Java Management Extensions）のインターフェースを備えており、流量制御や閉塞制御などを動的に変更することができるようになっています。

JMX インターフェースを利用した運用機能として、GUI による操作が可能な「運用画面」（図 3）や、バッチファイル等による自動化に向いた「運用コマンド」を提供しています。

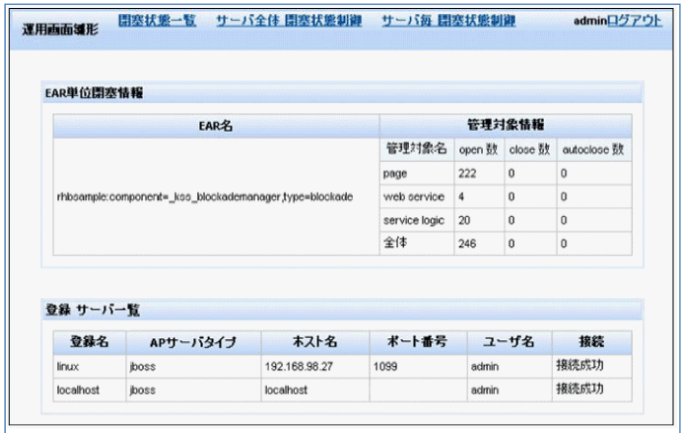

図 3：「フレームワーク基盤機能」の運用画面例

■開発ツール［1］アーキテクチャーの標準化

次に、ObjectWorks＋の「設計／開発を標準化するための仕組み」について解説します。

CORE は、アプリケーションを開発するための基盤としては、JSF（MVC）、EJB3（DI×AOP）、JPA（O/R マッピング）といった Java EE 5 標準フレームワーク群に Seam を組み合わせた標準スタックを採用しており、それ単体では、この上でどうアプリケーションの構造を設計するかは規定していません。開発者は、こうしたフレームワーク群の上で CORE の拡張機能群を活用し、自由にアプリケーションを設計することができます。

しかし、アプリケーションのクラス構造などを全く規定しないと、設計／開発にかかわる自由度が極端に大きくなってしまい、大規模な SI では開発のコントロールが効かなくなってしまう懸念があります。

そこで、設計／開発の標準化ツールである STUDIO では、Java EE 5 と Seam 上で CORE の機能を使って作成するアプリケーションに、アーキテクチャー上の標準化を導入する役目を果たしています。

具体的には、STUDIO の提供するアプリケーション設計情報の「定義シート」を作成することで、そこから、ある特定のアーキテクチャーに則ったアプリケーション・ソースコードを自動生成し、そのアーキテクチャーに従って必要な業務処理のコーディングを行うことをルール化しています（図 4）。

図 4 の下半分の、実行環境におけるアプリケーションのクラス構造を見ると、Java EE パターンとして一般的な、レイヤー構造を持っていることが分かると思います。アクション・クラスーサービス・クラス間で PL（プレゼンテーション・ロジック）と BL（ビジネス・ロジック）を分離し、レイヤー間のデータの受け渡しには DTO を用います。BL 側では、データの永続化に DAO（Data Access Object）層を設けています。

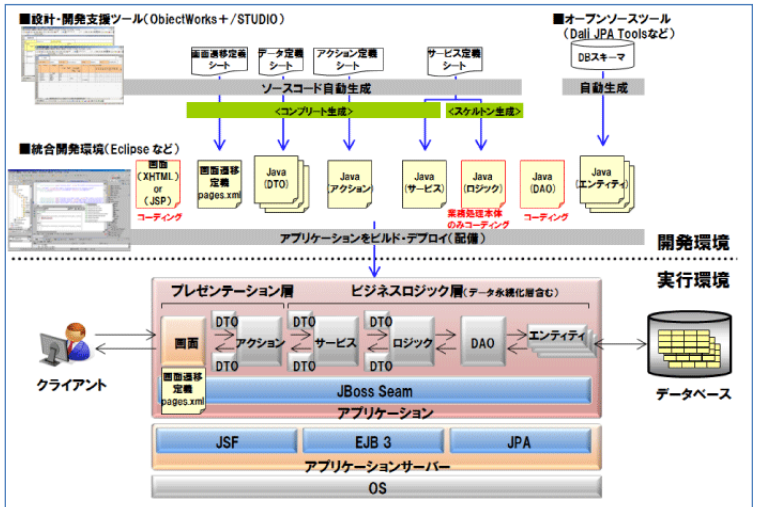

図４：定義シートの作成によってソースコードまで自動生成

Seam では元々、画面から直接エンティティへデータを渡し、DTO を必要としない設計パターンが意識されていますが、STUDIO が標準的に提供している定義シートが生成するアプリケーションは、図のように DTO を使ってデータを BL 層へ渡し、手続き的に業務処理のみをコーディングするような枠組みを与えています。

これは、マルチベンダーやオフショアを活用する大規模な SI では、アプリケーション開発者の設計スキルが統一されないために、手続き指向的に業務処理のみをコーディングするという分かりやすい開発方法が好まれる場合が依然として多く、そうした案件で設計／開発を標準化する仕組みとしての設計支援ツールが求められるケースが多いためです。

■開発ツール［2］定義シートによる設計・開発

STUDIO は、4 種類の定義シートを提供しています。定義シートはいずれも Microsoft Excel のスプレッド・シート形式で編集します。

(1) 画面遷移定義シート
(2) データ定義シート
(3) アクション定義シート
(4) サービス定義シート

(1)画面遷移定義シートは、Web アプリケーションの画面間の遷移と、その際に起動するアクションの情報を記述するシートです。ここで記述した情報は、JSF の画面遷移制御をラップしている、Seam の pages.xml という XML ファイルとして自動生成され、そのまま手を加えずに利用できます。

(2)データ定義シートは、DTO クラスの Java ソースコードを自動生成します。シートには、データ項目を一覧列挙するとともに、各データ項目がどのようなバリデーション属性(半角、全角、日付、数値範囲など)を持つかを定義する欄があり、ここを記述することで、自動生成した DTO に、前ページでも解説した「アプリケーション共通部品」のバリデータのためのアノテーションが付与されます。

(3)アクション定義シートは、画面遷移定義で定義したアクションが、BL 層のどのサービスを呼び出すかを定義します。このシートからは、サービス・クラスを呼び出すアクション・クラスのソースコードが自動生成されます。

(4)サービス定義シートは、アクション・クラスから呼び出されるサービス・クラスと、そのサービスが呼び出すロジック・クラスを定義します。このシートからは、各ロジック・クラスと、それらを順々に呼び出すサービス・クラスのソースコードが生成されます(ロジック層を設けず、サービス層に直接ビジネス・ロジックを記述することも可能です)。

(1)～(4)の定義シートから自動生成されたソースコードはほとんどすべて、そのままアプリケーション・アーカイブファイルに詰めて稼働させることができます。編集すべきファイルは、基本的には業務処理の本体を実装すべきロジック・クラスのみとなります。

このように STUDIO では、定義シートからのソースコード自動生成によって設計／開発の標準化と簡単化を実現すると同時に、アプリケーションの設計情報を、管理しやすいスプレッド・シートに集約することで、個々のアプリケーション開発者による設計パターン違反を抑止する役目を持っています。

また、STUDIO 定義シートのその他の役目として、「設計／開発単位を導入すること」が挙げられます。

STUDIO ではこうした定義シート類を、システム全体で 1 枚ではなく、Seam のコンテキストである「Conversation」ごとに 1 枚ずつ作成することをルール化しています。第 2 回で解説したとおり、Conversation は、画面遷移を伴う一連の処理など、複数のリクエストをまたいで維持されるスコープを表しますが、ObjectWorks＋ではこれを、ある一連の「業務」の単位と解釈して設計する標準化ルールを導入しています。

ObjectWorks＋による開発の際は、画面遷移定義の設計などに移るまえに、システムのサイト構成を、「業務単位」を意識した複数の Conversation に分割し、その Conversation ごとに STUDIO の定義シートを作成することとしています(図 5)。

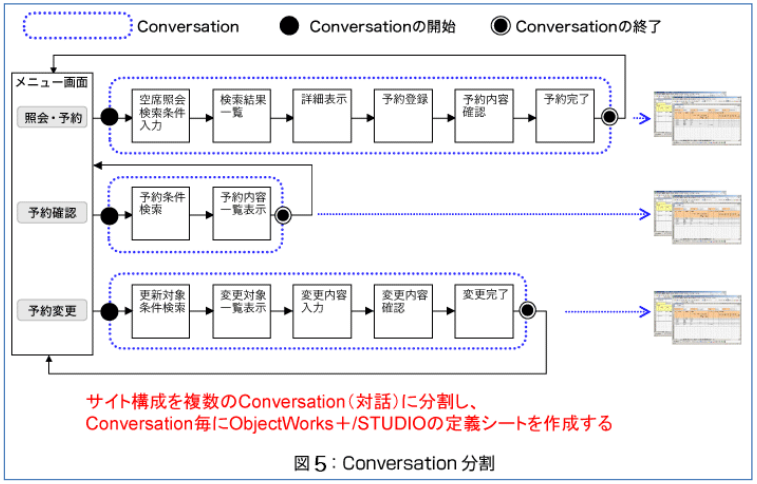

図 5：Conversation 分割

こうすることで、業務単位ごとに設計／開発を分離させることができ、不具合があった場合や仕様変更時などの影響範囲を Conversation 内に集約することができるなど、メンテナンス性を保ちやすくしています。

【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。