ネットビジネスを支える EJB（Enterprise Java Beans）の設計留意点

―― ここが設計のポイント ――

江口　巧一　　石田　武弥

Java 2 Platform, Enterprise Edition（以下J2EE*1）と称す）によるシステム構築は，現在のWebアプリケーションの世界では主流となっている。筆者らは，コンサルタントとして今までに社内外のさまざまなJ2EE環境の開発プロジェクトに携わってきた。

プロジェクトのほとんどはQCDにおいて何らかの問題を抱えており―Rescue活動も主要な業務であるため―，筆者らを含めたコンサルティングチームのサポートにより何とか収まったという状況である。

本稿では，筆者らのコンサルティングの経験から，設計プロセスの中で最も重要なシステム設計時における問題点を取り上げ，その解決策を示す。

EJB設計の問題点の抽出

J2EEとはSun Microsystems社プログラミング言語「Java 2」の機能セットの一つで，企業の業務システムや電子商取引などで使われるサーバに必要な機能をまとめたものであり，Javaの標準機能に，サーバ用のAPplication Interface（以下APIと称す）や諸機能を付加したものである。J2EEはプラットフォームの機能に関する仕様，リファレンス実装，互換性テストパッケージ，ガイドラインといった要素から成り，Javaプログラムを部品化して組み合わせることができるようにする「Enterprise JavaBeans」（以下 EJB*1）と称す）や，サーバサイドプログラムとして動作するJavaプログラムである「Servlet」，Webページに Javaプログラムを埋め込んで内容を動的に生成する「Java Server Pages」（以下JSPと称す）などの技術を含む。また，XMLを操作するためのAPIも用意され，サーバ上でさまざまなソフトウェアを組み立てられるようになっている。

現在，主流となっているJ2EEに適合しているEJBは，以下のようなメリットがあることから，サーバ用アプリケーションに多く採用されている。

- システム拡張のしやすさ，メンテナンス性の向上
- 分散システムの中での階層の明確化

（図1に示すプレゼンテーション層，アプリケーション層，オブジェクトデータ層など）

- 共通のエンタープライズJava技術として相互利用性が拡大
- コンポーネントとしてのポータビリティが保証される

しかし，単に，EJBとして，作りこんだだけでは，このメリットを十分に利用できない。また，次項以降に記述する多くの問題を抱えるようになってしまう危険性がある。

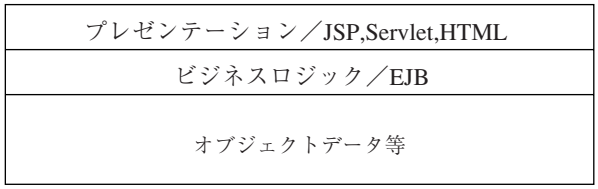

図1　J2EEでの処理階層

筆者らが関わったプロジェクトにおけるEJBの使われ方を見ると，以下のことが言える。

1. EJBのメリットを活かせず，却ってシステム開発のQCDを悪化させてしまっていることが多い。その理由として，システム開発期間が短く十分に設計時間が取れないということもあるが，本質的にはEJB設計における知識・理解が不足しているためと考えている。
2. EJB自体の作りは問題がないが，システムの中でのEJBの組み入れ方，使い方で，もう少し効率よく使える方法があると感じる設計がある。EJBコンポーネントを正しく設計することはできても，「そのEJBをどのように呼び出すか？」，「EJBの粒度（機能分割する大きさ）はどうするか？」などのシステムの中での使い方は，EJB設計とは別の設計知識（デザインパターンやフレームワーク）を必要とする。

以上述べたように設計の問題を大別すると以下のとおりである。

①EJB設計における知識・理解の不足からくる誤用。
②デザインパターンとフレームワークに関する知識不足。

＊1）J2EE，Java，EJBは，米国Sun Microsystems, Inc.の商標。

次項でこれらの詳細と解決策を述べていく。

設計者のEJB設計における誤用の原因

今まで筆者らが経験してきたプロジェクトの中で，EJB設計の初心者が起こしやすいEJBの誤った使い方の代表例を以下に紹介する。

(1) EJBコンポーネントモデルの誤った使用

一番多いのが，Statefull Session Bean と Stateless Session Beanの使い分けである。会話状態を一切管理する必要がないにも関わらず， Statefull Session Beanを使用しているケースが多い。誤った使い方は，バグを引き起こしたり，余計なオーバヘッド，リソース消費を招く。

(2) 適切なモデリングが行われていない

コンポーネント化や分散処理などのEJBのメリットを享受できないにも関わらず，無理矢理EJBで提供しているケースがある。オーバヘッドが大きいばかりか，バグも入りやすくなる。

(3) EJBの仕様で禁じられた操作を行っている

Containerでリソースの管理を行う関係で，EJB自身がスレッドの操作，Native libraryの使用，I/Oクラスを使用することは禁じられている。この制限を破ると，Containerの機能を損なう恐れがある。

EJBの仕様（用語解説[EJB]を参照）に基づくと，以下のように設計すべきである。

(1) はロジックを検討し，適切なBeanを導入すべきである。
(2) は，機能によって，通常のクラスで提供するほうが良い場合がある。
(3) はEJBの開発仕様をシステム開発者が守るべきである。

このように誤用が発生する原因は，明らかに設計者のEJBに対する知識不足である。

ではEJBに関する知識を習得するにはどうすれば良いのだろうか。

ＥＪＢ設計理解度習得の早道

EJBの仕様を良く理解するにはEJB仕様書に目を通すのが一番であるが，コンサルティング業務従事者はともかく，アプリケーション開発者（以下，AP開発者）には，そこまでの詳細知識は不要であり，目を通すのに必要な時間も馬鹿にならない（EJB 1.1の仕様書は英文で約300ページ）。

AP開発者に適した日本語でEJBを解説した書籍があれば良いのだが，数が少なく，内容的にも貧弱である。また，その他雑誌やWebでの情報もJSP/Servletの解説に比べて数が少なく，AP開発者が有益な情報を取得しにくい状態となっている。

そこで，筆者らが推奨するAP開発者のEJB設計知識習得の早道は，次の通りである。

①J2EE BluePrintsを熟読する

Sun*2) が提供している日本語ドキュメントである。ここには，EJBの基本的な使い方から，仕様書には載っていない注意点などもわかりやすく記載されている。

②サンプルプログラムを動かしてみる

上述のBluePrintsに付属しているサンプルプログラム（Java Pet Store SampleApplication ）を，自分で動かしながら，そのソースを眺めるのも有効である。このサンプルには，Session Bean，Entity Beanのソースコードも含まれており，一通り眺めることにより，EJBの使い方の大枠を体験できる。

また，JDC[2]（Java Developer Connection）のサイトでJ2EE エンタープライズアプリケーション開発の日本語版オンライントレーニングが公開されている。こちらもEJBを理解するには有効である。

デザインパターンとフレームワークの知識

前項でEJB設計における知識・理解度のアップについて述べたが，EJBコンポーネントを正しく設計することはできても，そのEJBのシステム内での振る舞いを理解することは難しい。その際に，スキルのない設計者であれば，誤ったシステム設計をする可能性が高い。それを避けるためには，デザインパターンとフレームワークの知識が必要となる。

デザインパターンについて

一般的なデザインパターンは，オブジェクト指向システムにおいて繰り返し現れる設計を体系的にまとめたもの（モデリング）であり，これを利用してソフトウェアの設計をすることにより，ソフトウェアの再利用性を高めることができる。

J2EEのデザインパターンとしては，現在代表的なものに，SJC（Sun Java Center）が提供しているJ2EE Patternsがある。これにはEJBだけではなく，JSP/Servletに関するパターンも収められている。

このデザインパターンには，さまざまなJ2EEシステムで共通に使われるデザインパターンが解説されており，Session BeanやEntity Beanの使い分け，粒度の考え方なども参考になる。しかし，実システムへの取り込みにはシステムの要件に合わせて注意深く選択する必要がある。

たとえば，この中のパターンには，EJBメソッド呼び出しがリモートであることを前提にして，このオーバヘッドを少なくするすることが，効果の1つであるものが幾つ

*2) Sunは，米国およびその他の国における米国Sun Microsystems, Inc.の商標または登録商標。

かある。しかし，実際のシステムでは，EJBは同一JVM内のJSP/Servletから呼ばれることが多く，またContainerにより呼び出しが最適化されることもあり，パフォーマンスを期待したこれらのパターンの適用は，実際に効果が出ないことがある。このように，デザインパターンを取り入れる際は，内容や効果を十分確認することが必要となる。これらの注意点を踏まえた上で，システムの成長を見越してデザインパターンを設計時に取り入れるよう努力することは，システム全体の品質を向上させることに役立つ。

フレームワークについて

デザインパターンに加え，認証機能，ログ出力機能や入力チェック処理などのように共通で使用できる処理をあらかじめ作成した，“フレームワーク”を用いることも，品質の高いシステムを構築する上で非常に有効である。

EJBの元々のコンセプトが，EJB内にはビジネスロジックのみを記述し，Containerがシステムで共通のリソース管理を行うというものであるため，Containerがある種のフレームワークの役割を果たしているが，まだ設計に対して自由度がある。

このため，フレームワークを提供し，ある程度の作り方を強制することで，設計者のスキルに影響されにくい品質を確保できる。また，共通で使用されるライブラリを提供することで，開発期間の短縮も実現できる。

公開されているフレームワークとしては，Jakartaプロジェクトの Strutsが存在する（Jakartaプロジェクトは世界各国のJava技術者たちが共同でサーバソリューションを開発・提供するオープンソースプロジェクトであり，Strutsはこの中のフレームワークに関するサブプロジェクトである）が，ServletとJSPが対象であるため，EJBでは使用できない。

したがって，現在のところEJBに適用できるパブリックなフレームワークは存在せず，商用ベースのものを利用するか，独自で開発するしかない。

我々は，現在独自にJ2EEのフレームワークを構築して実際のプロジェクトで活用し，成果を出している。以下でフレームワークのEJBに関する部分を一部簡単に紹介する。

図2は，EJBを組み入れるためのフレームワークのクラス図である。破線で囲まれた部分がフレームワークが提供する範囲で，実線で囲まれた部分が開発者が実装する範囲である。

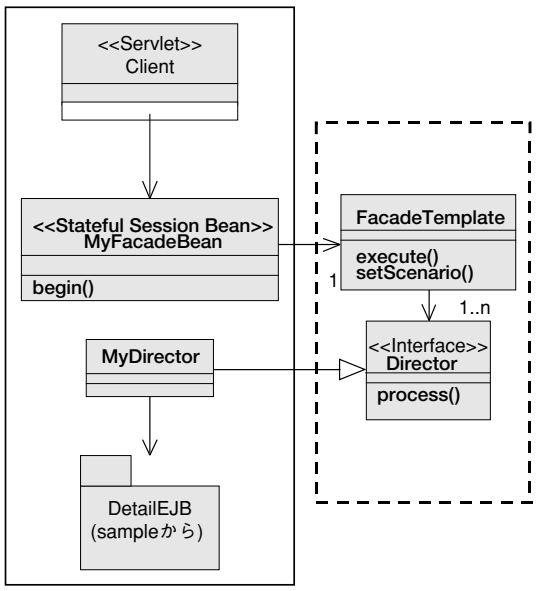

図2　EJBのフレームワーク

TIPS【基本用語解説】

［EJB］Enterprise Java Beans
米Sun Microsystemsが提唱するJavaベースでシステムを構築するための開発環境の標準仕様（Enterprise Java Beans Specification）である。
サーバ側でJavaをソフト部品化するためのコンポーネントモデル（Session BeanとEntity Beanの2種類）がある。

［Session Bean］
ユーザのセッションに対応するEJBコンポーネント。

［Entity Bean］
DBなどのリソースに対応するEJBコンポーネント。

［Stateless Session Bean］
クライアントとの会話状態を保持しないSessionBean。

［Stateful Session Bean］
クライアントとの会話状態を保持できるSessionBean。

［Container］
他の多くのコンポーネントの参照を保持できるコンポーネント。

［JVM］ Java Virtual Machine
Javaアプリケーションを環境に依存しないで動作させるための仮想システム。

［XML］ eXtensible Markup Language
World Wide Web ConsortiumのSGMLワーキンググループが仕様を作成したマークアップ言語。

商用フレームワークに対してプログラムを作成する場合，フレームワーク特有のインタフェースに従わなければならない。しかし，我々の開発したフレームワークの特徴は，汎用的なEJBもそのコードに手を入れる必要がなく，最小限のコーディングで，組み入れることができることである。

このため，サードベンダが提供するEJBも簡単に組み込むことができる。この機能を実現しているのがDirectorインタフェースである。汎用のEJBを組み入れる際は，EJBごとにDirectorインタフェースを継承した専用のクラスを作成することで，フレームワークへの組み込みが可能となる。

また，このフレームワークのもうひとつの特徴は，FacadeEJBクラス（図2ではMyFacadeBeanクラス）からのEJB呼び出しのフローをすべてXMLとして管理しているところである（Facadeとは他のコンポーネントや全く異なるインタフェースの集合に対し，共通のインタフェースを提供するクラスを定義することを言う）。

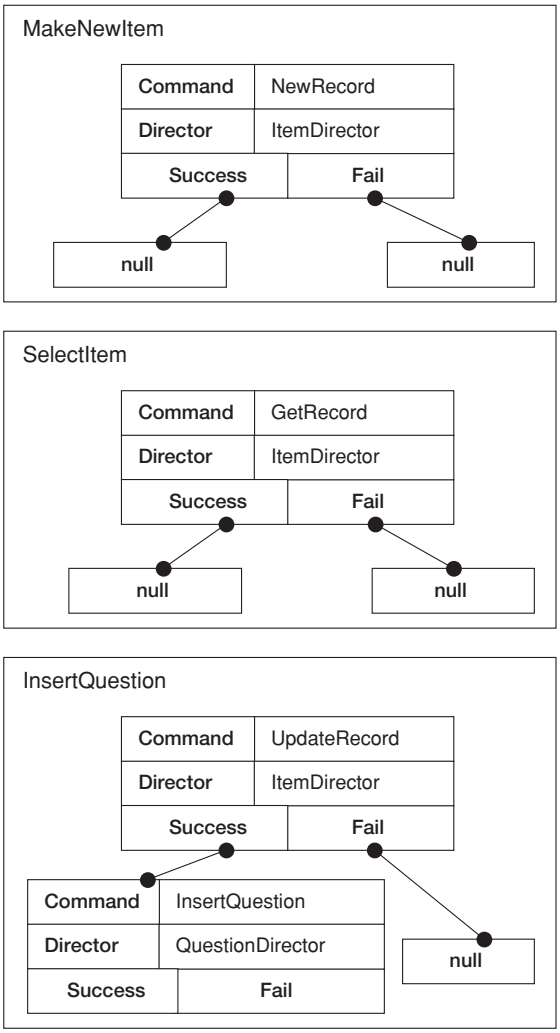

図3　FacadeEJBメソッドごとのフローデータ構造

開発者はまず，クライアントからの要求を単一点で受けるFacadeEJBを作成する。

通常であれば，この中でEJB呼び出しのフローを記述するのであるが，このフレームワークでは，別途XMLのデータとして管理されている。

図3が，FacadeEJBメソッドごとのフローデータ構造を表したものであり，ツリー形式として記述されている。

呼び出すEJBの順番や，返値の判断による処理の分岐などを全てこのツリー形式で記述しておく。この情報は，実行時にFacadeTemplateクラスに読み込まれ，実行される。これにより，EJB呼び出しのフローを仕様書の段階から簡単に実装に移行することが可能となり，管理やデバッグも容易になる。

現時点のフレームワークは，まだまだ，成長過程であり，常に成長している。したがって，最良のフレームワークを選択し，使用したプロジェクトごとにフィードバックを行っていく必要がある。

また，今回紹介できなかったが，EJB2.0で仕様化された
(1)　DB表間関連の取り込み
(2)　非同期処理機能の追加　等
常に最新の仕様を取り込みながら品質を上げていく必要もある。

あ　と　が　き

現在のEJB設計の問題点を述べた。システムデザインはプロジェクトの行方を左右する重要なキーである。

今回取り上げた問題以外にも，現在システムデザインには，さまざまな問題があり，今後もこれらの問題を解決するよう取り組んでいきたい。　◆◆

■参考文献

1）Sun javaBluePrints
　http://jdc.sun.co.jp/j2ee/blueprints.html
2）JDC（Java Developer Connection）
　http://jdc.sun.co.jp/index.html

●筆者紹介

江口巧一：Koichi Eguchi.ネットワークビジネスカンパニー ソリューション開発第1部
石田武弥：Takeya Ishida.ネットワークビジネスカンパニー ネットトランザクションコンサルティングチーム