JBoss Seam ～JavaEE は統合フレームワークへ～

■統合的なフレームワークを目指す Java EE

誕生当初から次代の Java EE(このころは、まだ J2EE と呼ばれていた)の中核を担う技術として注目を集めていた JSF 1.2 がプレゼンテーション層のフレームワークとして正式に採用され、Java EE の本体とも言える EJB は DI コンテナの機能を取り込み、生産性を飛躍的に向上させた EJB 3.0 として大きく生まれ変わりました。

また、それまで CMP Entity Bean として EJB 仕様の一部として存在していた O/R マッピング機能は、Java EE とは独立して発展し続けてきた Hibernate や TopLink といったオープンソースの O/R マッピング・ツールに歩み寄る形で新たに策定され、EJB とは独立した API である JPA 1.0 に役割を委ねることになりました。

このように機能の統廃合や、Java EE 以外のコミュニティのノウハウの取り込みを行うことによって、JSF 1.2 – EJB 3.0 – JPA 1.0 というスタックを築き上げた Java EE 5 ですが(図 1)、既に米 JBoss の Gavin King 氏には、これらのスタックをシームレスにつなげ、統合的な Web 開発フレームワークとして提供するという、次の Java EE 6 に向けた青写真がありました。

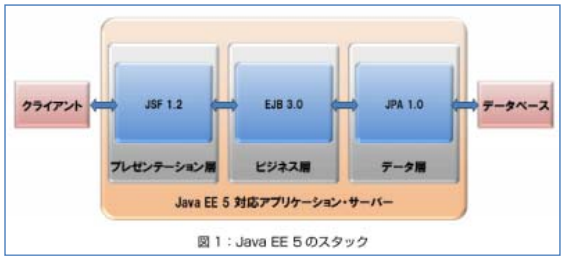

図 1：Java EE 5 のスタック

この青写真を具現化したものが、同氏が開発した JBoss Seam(以下、Seam)という Web 開発フレームワークであり、それを基にして同氏が策定し、Java EE 6 に採用されることになった「JSR-299:Context and Dependency Injection for Java EE Platform」(以下 CDI)です。

では、今後の Java EE の歩みにおいて大きな鍵となるであろう、Seam についてご紹介します。

■JBoss Seam とは

Seam は、JBoss Application Server などと同様に、JBoss コミュニティ(http://www.jboss.org/)で開発されているオープンソース・プロダクトです。ライセンス形態は LGPL(GNU Lesser General Public License)であり、無償での入手、利用が可能ですが、米 Red Hat(2006 年に米 Red Hat が米 JBoss を買収)の有償サポートを受けることも可能です。

Seam は、JBoss Application Server はもちろん、それ以外の Java EE 5 に準拠したアプリケーション・サーバー上でも動作させることが可能な Web 開発フレームワークであり、以下のような特徴を持っています。

- 付属ツールの seam-gen によってアプリケーションのひな型を作成でき、簡単に開発が始められる。
- JSF と EJB をシームレスにつなぐ機能を持つ。
- 独自のコンポーネント・モデルによって、ステートフルな Web アプリケーション開発を容易にする。

ここではまず、開発環境について簡単に紹介します。

前述のように、Seam はさまざまな環境で動作させることが可能ですが、一番簡単に動作させるためには、JBoss Application Server を使用するのが良いでしょう。JBoss Application Server には DBMS(HSQLDB)もバンドルされているので、Seam と JBoss Application Server に加えて、JDK を用意すれば、Seam のすべての機能を確認できます。

参考までに、今回、動作確認に用いたソフトウエアは以下の通りです。

・JBoss Seam 2.2.0
(http://sourceforge.net/projects/jboss/files/JBoss%20Seam/2.2.0.GA)
・JBoss Application Server 5.1.0
(http://sourceforge.net/projects/jboss/files/JBoss/JBoss-5.1.0.GA/)
・JDK 5.0 UPDATE 21

Seam と JBoss Application Server は、適当な位置に解凍するだけで構いません。JDK はインストール後、環境変数「JAVA_HOME」を適切に設定します。事前準備はこれだけです。

■seam-gen を使ってアプリケーションのひな型を作成する

Ruby on Rails という Ruby 言語で開発された Web 開発フレームワークのことをご存じの方も多いかと思います。Ruby on Rails が scaffold という機能を使ってアプリケーションのひな型をコーディングレスに作成できるように、Seam でも seam-gen を使ってアプリケーションのひな型をコーディングレスに作成できます。

まず、解凍された Seam のディレクトリ直下に、seam あるいは、seam.bat というコマンドが用意されていますので、コマンド・プロンプトから、

```
$ ./seam setup
```

を実行すると、対話形式で環境セットアップを行うことができます。プロジェクトのホーム・ディレクトリ（以降、「${PROJECT_HOME}」）と JBoss のホーム・ディレクトリ（「以降、${JBOSS_HOME}」）を適切に指定し、デプロイ形式は EJB を使用するので「ear」を選択し、それ以外はデフォルトのままで構いません。

セットアップが完了した後、

```
$ ./seam create-project
```

を実行すると、ひな型アプリケーションの含まれたプロジェクト・ディレクトリが「${PROJECT_HOME}/myproject」に作成されます。

```
$ ./seam deploy
```

を実行すると、プロジェクトのビルド、およびアプリケーション・サーバーへのデプロイが実行されますので、アプリケーション・サーバーを起動し「http://localhost:8080/myproject/」にアクセスしてみてください。「Welcome to Seam!」という画面が表示されれば成功です。

これだけでは味気ないので、フォーム・アプリケーションのひな型も作成してみましょう。アプリケーション・サーバーを停止して、先ほどと同じプロンプトから、

```
$ ./seam new-form
```


を実行します。Seam コンポーネントの名前を聞いてくるので、ここでは「hello」としましょう。あとはデフォルトのままで構いません。コマンドが完了すると、先ほどのプロジェクト・ディレクトリに、リソースが追加されます。

先ほどと同じ手順で、ビルド、デプロイを行い、アプリケーション・サーバーを起動して「http://localhost:8080/myproject/hello.seam」にアクセスしてみてください。表示された画面のフォームに文字列を入力し、「hello」ボタンをクリック、入力した文字列が上部のステータス・メッセージに表示されれば成功です（図 2）。

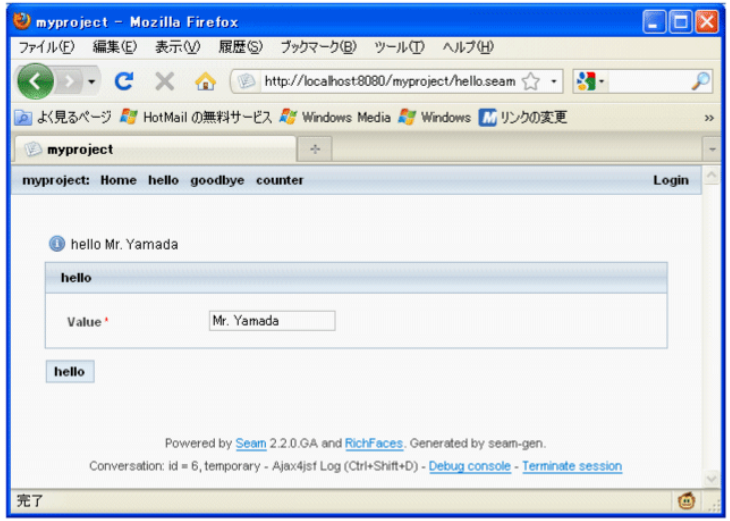

図２：ひな型アプリケーションの画面 1

scaffold と同様に、CRUD（Create, Read, Update, Delete）アプリケーションのひな型を seam-gen を使って作成することもできます。このようにして作成されたアプリケーションのひな型は、これらを基にして開発を始めることで、開発効率を上げることができるとともに、Seam の機能を把握するためにも大いに役に立つことと思います。

■JSF と EJB をシームレスにつなぐ

Seam の名の通り、JSF と EJB をシームレスにつなぐことができるということが、Seam の大きな特徴であると言えます。この点について、Seam が果たす役割について説明する前に、まずは Seam を用いない場合の開発について整理してみます。

JSF では、Managed Bean と呼ばれる Java Beans を媒介にして、ビジネス層へのデータの受け渡しを行います。もちろん、Managed Bean 自体がビジネス・ロジックを持っても構わないのですが、ここで注意しなければならないのは、Managed Bean として登録できるのは、あくまで Java Beans であり、EJB ではないということです。ビジネス・ロジックを EJB として実装する場合には、やはり、パラメータの受け渡しのみを目的とした Managed Bean を作成せざるを得ないということになります。また、Managed Bean の作成には、faces-config.xml の記述という煩わしい作業がつきまといます。

Seam を使用した場合、これらの作業はどう変わってくるのでしょうか。先ほど作成したフォーム・アプリケーションのひな型を見てみましょう。

まず、「${PROJECT_HOME}/myproject/resources/WEB-INF/faces-config.xml」のどこを見ても、Managed Bean に関する記述は見当たりません。次に、「 ${PROJECT_HOME}/myproject/src/hot/com/mydomain/myproject/action/HelloBean.java」を開くと、ソースの 13 行目に「@Name("hello")」というアノテーションが付加されているのが分かります。

結論から言いますと、Seam を使用した場合、Managed Bean を登録するための faces-config.xml の編集作業は必要ありません。@Name アノテーションが付加されたクラスは、Seam コンポーネントと呼ばれ、Managed Bean の役割を果たすようになります。要するに、@Name アノテーションは<managed-bean-name>と同じ意味を持つというわけです。

また、このソースでは省略されていますが、「@Scope(SESSION)」というようなアノテーションを付加することによって、若干意味合いは異なりますが、<managed-bean-scope>と同様の設定を行うことも可能です(クラスのソース自体に記述を行うわけですから、当然<managed-bean-class>の設定は必要ありません)。

もう 1 つ、このソースで注目すべき個所は、12 行目の@Stateful アノテーションです。そう、このクラスは EJB なのです。Seam では、EJB も Java Beans と同様に Seam コンポーネントとして登録することができ、JSF からのパラメータを直接受け取ることができるのです(図 3、4)。


```
...
12  @Stateful                                   ① StatefulSessionBean であることの宣言
13  @Name("hello")                              ② Seam コンポーネント名の指定
                                                (JSF中のEL 式で変数名として用いられる)
14  public class HelloBean implements Hello
15  {
...
20      private String value;                   ③表示文字列を格納するプロパティ
                                                (JSF中の inputText タグの value 属性で指定)
21
22      public void hello()                     ④「hello」ボタンのクリック時に実行されるメソッド
23      {                                       ( JSF中のcommandButtonタグの action 属性で指定)
...
27      }
...
32      public String getValue()                ⑤③のアクセサメソッド
33      {
34          return value;
35      }
36
37      public void setValue(String value)
38      {
39          this.value = value;
40      }
...
45  }                                           ${PROJECT_HOME}/myproject/src/hot/com/
                                                mydomain/myproject/action/HelloBean.java
```

図 3：Seam コンポーネントのソース 1

```
...
12  <ui:define name="body">
13
14      <h:form id="helloForm">
...
21              <h:inputText id="value" required="true"
22                  value="#{hello.value}"/>        ①表示文字列を格納するプロパティ
                                                    (HelloBean.javaのvalueプロパティ)
...
30          <h:commandButton id="hello" value="hello"
31              action="#{hello.hello}"/>           ②「hello」ボタンのクリック時に実行されるメソッド
                                                    (HelloBean.javaのhelloメソッド)
...
34      </h:form>
35
36  </ui:define>
...                                                 ${PROJECT_HOME}/myproject/view/hello.xhtml
```

図 4：JSF のソース 1

このように、Seam はアノテーションを活用することによって、faces-config.xml を記述する手間を減らし、さらには EJB 呼び出しを隠ぺいし、Managed Bean と EJB の役割を Seam コンポーネントとして統合することで、冗長な Managed Bean を作成する手間を省いてくれるのです。これが、JSF と EJB をシームレスにつなぐ Seam の果たす大きな役割です。

次に、ステートフルな Web アプリケーション開発を効率化する Seam 独自のコンポーネント・モデルについて解説します。

■JBoss Seam のコンポーネント・モデル

Seam はコンテキスト依存(contextual)コンポーネント・モデルという独自のコンポーネント・モデルを持っています。このコンポーネント・モデルは、従来の Java EE 上でステートフルな Web アプリケーションを開発するうえでの問題を解決し、さらに対話(conversation)という新たな概念によって、ステートフルな Web アプリケーション開発を飛躍的に効率化させるために導入されました。以下では、この Seam 特有のコンポーネント・モデルについて紹介します。

まず、前述の「@Scope(SESSION)」というアノテーションは、一体「何の」スコープを指し示しているのでしょうか。その答えがコンテキストです。

Seam は Application、Session、Conversation など(最初の 2 つは Servlet API でなじみ深いものだと思います)のさまざまなスコープを持ったコンテキストを持っており、すべての Seam コンポーネントのインスタンスは、いずれかのコンテキスト上で管理されています。Seam コンポーネントは、それぞれのタイミングでインスタンス化(初回の呼び出し時に作成、明示的に作成)およびコンテキスト上で管理され、コンテキストのスコープが終了(セッションの終了、リクエストの終了)するとともに破棄されます。

Seam コンポーネントは、コンテキスト上で、コンテキスト変数と呼ばれる名前つき変数(@Name アノテーションで指定した名前)の中に格納され、EL 式(Expression Language)の中から変数名で参照することができます。

また、インジェクション用のアノテーション(@In)を使用することによって、指定したプロパティへコンテキスト変数から値を差し込むことができ、逆にアウトジェクション用のアノテーション(@Out)を使用することによって、指定したプロパティからコンテキスト変数へ値を差し込むこともできます。このような機能をバイジェクションと呼んでいます。

インジェクションはコンポーネント・ツリーを構築するための重要な機能ですが、インジェクションで重要なことは、そのプロパティを持つコンポーネントがインスタンス化され

た時に行われるのではなく、そのインスタンスが呼び出される際に毎度行われるということです(ちなみにアウトジェクションはインスタンスの呼び出しが終了した際に行われます)。この仕組みによって、スコープの異なるコンポーネントがインジェクトされる場合でも、整合性を損なうことなくコンポーネントを構築でき、faces-config.xml において長さの異なるスコープの Managed Bean を<managed-property>に設定した場合のような問題が起こらないようにしています。

Seam のコンテキストのなかでも、特に特徴的な概念が、Conversation(対話)です。

Conversation は、Seam のコンポーネント・モデルの中核をなす概念であり、今までの Java EE 開発にはなかった新たな概念です。対話コンテキストのスコープは複数のリクエストにまたがりますが、Session スコープよりは短く、ユーザーが 1 つのウィンドウの中で行う、不可分な作業単位を表します。

例えば、ユーザー登録機能が複数ページに亘って行われるようなアプリケーションにおいて、最初の画面のサブミットから、最後の画面のサブミットまでは、対話であると考えることができます。こう書くと今まで Session スコープで管理していたほとんどのものは、対話スコープで管理すべきなのではないかと思われるのではないでしょうか。

アプリケーションのひな型を使って、具体的なイメージをつかんでみましょう。

```
------------------------------------------------------------------------
$ ./seam new-conversation
------------------------------------------------------------------------
```

を実行します。フォーム・アプリケーションの場合と同じように、Seam コンポーネントの名前を聞いてきますので、「counter」と入力しましょう。あとはデフォルトで構いません。

先ほどと同じ要領で、ビルド、デプロイを実行した後、「http://localhost:8080/myproject/counter.seam」にアクセスすると、数値「0」が表示されたパネルと、「Begin」ボタンがページに表示されます。「Begin」ボタンをクリックすると、「Increment」ボタンと「End」ボタンが表示されます。「Increment」ボタンをクリックする度に、パネルに表示された数値は増加していき、「End」ボタンをクリックするとトップのページに遷移します。上部の「counter」リンクを選択し、再び先ほどの画面に戻ると、画面は初期状態に戻っています(図 5)。

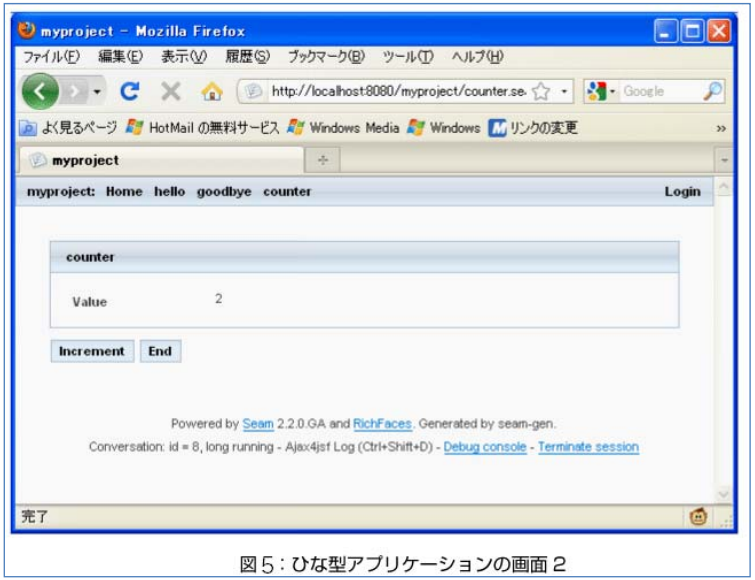

図５：ひな型アプリケーションの画面２

この単純なアプリケーションにおいて、「Begin」ボタンをクリックしてから「End」ボタンをクリックするまでの間、パネルに表示される数値は、対話コンテキストで管理されています。

「${PROJECT_HOME}/myproject/src/hot/com/mydomain/myproject/action/CounterBean.java」を見てみましょう。begin()、increment()、end()と、各ボタンに対応するアクション・メソッドが並んでいますが、begin()メソッドの上には@Begin アノテーションが、end()メソッドの上には@End アノテーションが付加されているのが分かります。対話コンテキストは、このようにアノテーションによって開始と終了を宣言することにより管理します。HTTP Session を自前で管理しなくても、アノテーションを付加するだけで、このようなアプリケーションが作れるのです（図 6、7）。


```
...
12  @Stateful
13  @Name("counter")
14  public class CounterBean implements Counter,
    Serializable
15  {
...
18      private int value;
19
20      @Begin
21      public String begin()
...
28      public String increment()
...
37      @End
38      public String end()
...
53  }
```

①Seamコンポーネント名の指定
（JSF中のEL式で変数名として用いられる）

②カウンター値を格納するプロパティ
（JSF中のEL式で指定）

③「Begin」ボタンのクリック時に実行されるメソッド
対話が開始する
（JSF中のcommandButtonタグのaction属性で指定）

④「Increment」ボタンのクリック時に実行されるメソッド
カウンターを1増やす
（JSF中のcommandButtonタグのaction属性で指定）

⑤「End」ボタンのクリック時に実行されるメソッド
対話が終了する
（JSF中のcommandButtonタグのaction属性で指定）

${PROJECT_HOME}/myproject/src/hot/com/mydomain/myproject/action/CounterBean.java

図６：Seam コンポーネントのソース２

```
...
11  <ui:define name="body">
...
19          <span class="value">#{counter.value}</span>
...
27      <h:form id="counter">
28          <div class="actionButtons">
29              <h:commandButton id="begin" value="Begin"
30                  action="#{counter.begin}" rendered="#{not conversation.longRunning}"/>
31              <h:commandButton id="inc" value="Increment"
32                  action="#{counter.increment}" rendered="#{conversation.longRunning}"/>
33              <h:commandButton id="end" value="End"
34                  action="#{counter.end}" rendered="#{conversation.longRunning}"/>
35          </div>
36      </h:form>
37
38  </ui:define>
...
```

①カウンター値を格納するプロパティ
（CounterBean.javaの valueプロパティ）

②「Begin」ボタンのクリック時に
実行されるメソッド
（CounterBean.javaの beginメソッド）

③「Increment」ボタンのクリック時に
実行されるメソッド
（CounterBean.javaの incrementメソッド）

④「End」ボタンのクリック時に
実行されるメソッド
（CounterBean.javaの endメソッド）

${PROJECT_HOME}/myproject/view/counter.xhtml

図７：JSF のソース２

とはいえ、この程度であれば、HTTP Session で管理しても大した手間ではないのではないかと思われる方もいらっしゃるかもしれません。そのような方は、アプリケーションを複数のタブで開いて動作させてみてください。それぞれのタブで開かれたカウンタ

ーが互いに独立して数を増やしていくのが確認できると思います。Seam では、同一セッション上で並行して進行するような対話にも簡単に対応できるのです。

■Weld、そして JBoss Seam 3 へ

ここまで、Seam の持つ 3 つの特徴を中心に説明してきましたが、特に後半の 2 つ、

- JSF と EJB をシームレスにつなぐ機能を持つ。
- 独自のコンポーネント・モデルによって、ステートフルな Web アプリケーション開発を容易にする。

は、2009 年 12 月にリリースされた Java EE 6 に採用されている仕様、CDI の中でも中核をなす重要な機能になります。アノテーションや API については仕様の策定にあたって調整が行われたため、Seam の機能がそのまま Java EE 6 として採用されているわけではありませんが、概念や機能の大枠は Seam のそれと大差はありません。

また、現在 Seam コミュニティでは、次期バージョンである Seam 3.0 に向けて開発が行われている最中ですが、CDI のリファレンス実装（参照実装）である Weld がそのまま Seam 3.0 のコアとして採用され、JavaEE 6 準拠の Web 開発フレームワークとなる予定です。

Seam は、CDI の部分以外にも、Seam-gen をはじめとするツール、ユーティリティ群を豊富に備えており、Java EE 準拠の Web 開発フレームワークとして、最も実用に耐えうる、現実的なものであるといえます。

Seam のもっと詳しい内容や技術については、NRI の開発エキスパート達が執筆した「JBoss Seam」の入門書『オープンソース徹底活用 JBoss Seam による Web アプリケーション開発』（http://www.shuwasystem.co.jp/products/7980html/2319.html）を、ぜひご一読ください。実用的なサンプルアプリケーションの作成方法を交えながら、Java EE 6 時代の中核となる標準技術を解説しています。

【執筆者】 佐野 大輔
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。前職での大企業向け Web 型グループウエアの開発経験を経て、2008 年 8 月より現職。NRI の SI フレームワーク「ObjectWorks＋」関連製品の導入コンサルティングを行っている。

※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。