TERASOLUNA Server Framework for Java (Web 版)

1.TERASOLUNA Server Framework for Java（Web版）とは

NTT データで開発された Web アプリケーションフレームワークであり、プレゼンテーション層には Apache Struts、データパーシステンス層に Apache iBATIS、それらを統括する IoC/DI（Inversion of Control ／ Dependency Injection）フレームワークとして Spring Framework を用いて構成されています。特徴としては以下のことがあげられます。

1.1. 設定ファイルによる開発

Struts、Spring および iBATIS を用いることにより、設定ファイルベースでアプリケーションを管理することができます。これにより、仕様変更に強いアプリケーションを実現することができます。

1.2. トランザクション管理

従来の JDBC では DB 接続やトランザクション処理を複雑に記述していましたが、TERASOLUNA フレームワークを使用することにより、自動化したり簡潔に記述することができるため、コミットミス、ロールバックミスなどを防ぐことができます。

1.3. 拡張されたアクション

TERASOLUNA フレームワークによって拡張されたエンタープライズアプリケーションで使用する標準的な機能を実現する以下のようなアクションクラスを利用することにより、新たに個々のアクションクラスを実装する必要がありません。

- 指定されたセッションを削除するアクション
- 帳票の一時保存用などに使用できるディレクトリを作成するアクション
- ビジネスロジックを実行するアクション
- ログオフ処理を行うアクション

1.4. 拡張性

ベースにしているフレームワークの拡張性を損なうことなく拡張しているため、独自に機能を拡張することが容易です。

1.5. ユースケーススコープ

ベースとしている Struts のスコープとしてはページ、リクエスト、セッション、アプリケーションの４つがありますが、TERASOLUNA フレームワークではそれを独自に拡張し、より業務に適したユースケース単位でのセッション管理を実現することができます。

1.6. セッション自動削除

セッションの削除漏れに伴うメモリ枯渇のリスクを下げ、より安全に Web アプリケーションが

運用できます。

1.7. 認証・アクセス制御

ログイン情報の管理機能が実装されているため、認証や権限処理が容易になります。

上記の特徴はベースとしているフレームワーク（Struts、Spring、iBATIS）から得られる特徴も含まれているため、TERASOLUNA Server Framework for Java（Web 版）固有の特徴である、1.5 ユースケーススコープおよび 1.6 セッションの自動削除を取り上げ説明いたします。

2. TERASOLUNA のスコープ

2.1. 標準的な Web アプリケーションのスコープ

Servlet API を使用した標準的な Web アプリケーションでは以下の４つのスコープがあります。

- ページスコープ
  同じ JSP ファイルの範囲で有効なスコープになります。
- リクエストスコープ
  単一のリクエスト内で有効なスコープになります。削除する必要もなく、メモリの圧迫リスクもありません。業務データが javax.servlet.http.HttpServletRequest に保存されます。
- セッションスコープ
  同じコンピュータからのリクエストの範囲で有効なスコープになります。時間でその有効期限を設定するなど、そのスコープをコントロールすることができます。ユーザーアクセスごとに生成されるため、有効時間の設定によってはメモリが圧迫される可能性が高くなるため､意識的に削除する必要があります｡業務データが javax.servlet.http.HttpSession に保存されます。例としてショッピングサイトではショッピングカートなどのデータをこのスコープで管理します。
- アプリケーションスコープ
  Web アプリケーションを起動してから終了するまでの範囲で有効なスコープになります。各ユーザーアクセスで共有して使用することができます。業務データが javax.servlet.ServletContext に保存されます。

以上の４つのスコープは、システム側からみたスコープであり、業務から見たデータの有効範囲ではありません。セッションスコープにいろいろなデータを保管していくことがもっとも便利な方法ではありますが、これですと多量のアクセスがあった場合など、メモリが枯渇する可能性が高くなります。ですので、業務の観点では、リクエストスコープとセッションスコープの中間的なスコープでデータを管理できる方法が求められています。そのために、よく行う方法としては、セッションスコープでデータを保管し、トップ画面に戻るタイミングなどで不要なデータを削除したり、あるいはセッションスコープを使用禁止にして、各画面に hidden 属性を付けてクライアントに保存させる方法があります。そのような応用をして、各 Web アプリケーションは業務で使用できるようにしています。

2.2. ユースケーススコープ

TERASOLUNA Framework では通常のセッションスコープを独自に拡張し、より業務に適したユースケース単位でのセッション管理を実現することができます。ユースケーススコープとは、

セッションスコープの一種です。通常のセッションスコープは明示的に削除しなければ、メモリ上から取り除かれません。TERASOLUNA Framework 独自の機能により、接頭語として『__』を付けられたアクションフォームはセッション中に１つしか保存されないようになります。新しく作成されるアクションフォームの名称に『__』が付けられている場合、『__』が付けられている全てのセッションスコープのアクションフォームを削除することにより、それを実現しています。したがって、セッションスコープとリクエストスコープの中間的なスコープの役割を果たしています。前述の通常の Web アプリケーションでの問題であった、セッションスコープのメモリが枯渇する恐れも解決され、手動でセッションから削除する必要もありません。業務からみた理想のセッション管理が実現できます。それでは、３章で環境設定、４章でセッションスコープを用いたサンプル、５章でユースケーススコープを用いたサンプルを取り上げ、TERASOLUNA Framework 独自のセッション管理を説明します。

3. 環境設定

3.1. 前提条件

Windows XP Professional ServicePack 3 環境下で動作確認しております。

3.2. Eclipse（Java）のインストール

3.2.1. Eclipse のダウンロード

Eclipse 3.4.2 Ganymede SR2 をベースに日本語化され、複数のバージョンの JRE も Tomcat も一緒に含まれている Pleiades All in One Java（JRE あり）を用いて作業を行います。

http://mergedoc.sourceforge.jp/

で最新版がダウンロードできます。

現在、『pleiades-all-in-one-java-jre_20090426.zip』が最新版になります。

3.2.2. 圧縮ファイルの解凍

ダウンロード後、任意のフォルダに解凍します。

3.2.3. Eclipse の起動

解凍されたフォルダ¥eclipse ¥eclipse.exe を実行し、Eclipse を起動します。

3.3. ブランクプロジェクトのインポート

3.3.1. ブランクプロジェクトのダウンロード

TERASOLUNA Server Framework for Java（Web 版）は最新バージョンである Ver2.0.2.0 を利用します。

http://sourceforge.jp/projects/terasoluna/releases/39697/

からダウンロードすることになります。ここからブランクプロジェクトである『terasoluna-server4jweb-blank_2.0.2.0.zip』をダウンロードします。

3.3.2. 圧縮ファイルの解凍

ダウンロードした『terasoluna-server4jweb-blank_2.0.2.0.zip』を任意のフォルダに解凍します。

3.3.3. Eclipse のプロジェクトインポート

Eclipse の［ファイル］ → ［インポート］をクリックします。

［一般］ → ［既存プロジェクトをワークスペースへ］を選択し、［次へ(N)］をクリックします。

ルート・ディレクトリーの選択の［参照］をクリックし、3.3.2 で解凍したフォルダ『terasoluna-spring-thin-blank』を指定し、プロジェクトをワークスペースにコピーにチェックを入れて［完了(F)］をクリックします。

3.4. Tomcat の設定

3.4.1. ブランクプロジェクトの起動

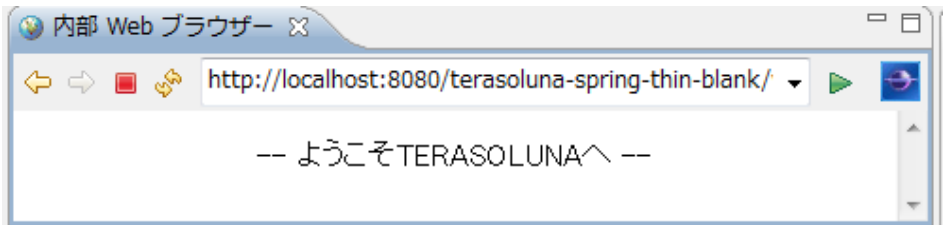
パッケージ・エクスプローラーのパッケージ名『terasoluna-spring-thin-blank』を右クリック → ［実行(R)］ → ［サーバーで実行］をクリックします。
［サーバーのタイプを選択(S)］の［Apache］ → ［Tomcat v5.5 サーバー］を選択し、［次］をクリックします。
［Tomcat インストール・ディレクトリ(D)］の［参照］をクリックし、インストールした Pleades フォルダ配下の「tomcat5.5」を選択します。［JRE(J)］は jre1.5 を選択します。
以上の設定を行った後、［完了］をクリックします。

3.4.2. ブランクプロジェクトの動作確認

ブラウザで

```
http://localhost:8080/terasoluna-spring-thin-blank/
```

を開きます。



『-- ようこそ TERASOLUNA へ --』画面が表示され、動作確認ができました。

4．通常のセッションスコープを使った例

4.1．サンプルのダウンロードについて

今回のサンプルは以下のリンクからダウンロードできます。

＜サンプルダウンロード＞

http://www.bbreak.co.jp/technique/doc/terasoluna/terasoluna_web1.zip

※このダウンロードファイルは3.3.ブランクプロジェクト『terasoluna-server4jweb-blank_2.0.2.0.zip』との差分になっております。

4.2．サンプルの概要

通常のセッションスコープを使った例として、簡易的なオークションサイトを用います。

ゲームの内容は以下になります。

4.2.1．出品者が最低落札価格を登録する。

最低落札価格がセッションに登録されます。

4.2.2．最低落札価格を確認する。

4.2.3．出品完了。

4.2.4．落札者が入札価格を入力する。

入札価格がセッションに登録されます。

4.2.5．入札価格を確認する。

4.2.6．最低落札価格と入札価格を表示します。

セッションに登録されている両方の価格が表示されます。

4.3．ソースの編集・作成

ブランクプロジェクトを元に、編集・作成を行います。

4.3.1．struts-config.xml の編集

terasoluna-spring-thin-blank¥webapps¥WEB-INF¥struts-config.xml に以下の内容を編集します。

```
※　省略

<!-- ==================================== フォームビーン定義 -->
  <form-beans>

    <form-bean name="_sellingForm"
      type="jp.terasoluna.fw.web.struts.form.DynaValidatorActionFormEx">
      <form-property name="price"
        type="java.lang.String" />
    </form-bean>

    <form-bean name="_biddingForm"
      type="jp.terasoluna.fw.web.struts.form.DynaValidatorActionFormEx">
      <form-property name="price"
        type="java.lang.String" />
    </form-bean>

  </form-beans>

※　省略

<!-- ==================================== アクションマッピング定義 -->
  <action-mappings type="jp.terasoluna.fw.web.struts.action.ActionMappingEx">

    <action path="/welcome" parameter="/welcome.jsp"/>

    <!-- 出品登録 -->
    <action path="/sellingInput" parameter="/sellingInput.jsp"/>

    <!-- 出品登録 BL -->
    <action path="/sellingInputBL" name="_sellingForm" >
      <forward name="success" path="/sellingConfirm.do" />
      <forward name="failure" path="/sellingInput.do" />
    </action>

    <!-- 出品確認 -->
```

```
    <action path="/sellingConfirm" parameter="/sellingConfirm.jsp"/>

    <!-- 出品完了 -->
    <action path="/selling" parameter="/selling.jsp"/>

    <!-- 入札入力 -->
    <action path="/biddingInput" parameter="/biddingInput.jsp"/>

    <!-- 入札登録 BL -->
    <action path="/biddingInputBL" name="_biddingForm" >
      <forward name="success" path="/biddingConfirm.do" />
      <forward name="failure" path="/biddingInput.do" />
    </action>

    <!-- 入札確認 -->
    <action path="/biddingConfirm" parameter="/biddingConfirm.jsp"/>

    <!-- 入札完了 -->
    <action path="/bidding" parameter="/bidding.jsp"/>


  </action-mappings>

※  省略
```

4.3.2. moduleContext.xml の編集

terasoluna-spring-thin-blank¥webapps¥WEB-INF¥moduleContext.xml に以下の内容を編集します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
```

```
      xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-2.5.xsd">

<!-- モジュール固有の Bean 定義 -->

  <!-- 共通定義のインポート -->
  <import resource="commonContext.xml" />

<!-- ===================================== アクション・業務ロジック定義 -->
  <!-- アクション定義 -->
  <bean name="/welcome" scope="singleton"
    class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

  <!-- 出品登録 -->
  <bean name="/sellingInput" scope="singleton"
    class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

  <!-- 出品登録 BL -->
  <bean name="/sellingInputBL" scope="singleton"
    class="jp.terasoluna.fw.web.struts.actions.BLogicAction" >
    <property name="businessLogic" ref="priceInputBLogic" />
  </bean>

  <!-- 出品確認 -->
  <bean name="/sellingConfirm" scope="singleton"
    class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

    <!-- 出品完了 -->
  <bean name="/selling" scope="singleton"
    class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

  <!-- 入札入力 -->
  <bean name="/biddingInput" scope="singleton"
```

```
    class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

  <!-- 入札登録 BL -->
  <bean name="/biddingInputBL" scope="singleton"
    class="jp.terasoluna.fw.web.struts.actions.BLogicAction" >
    <property name="businessLogic" ref="priceInputBLogic" />
  </bean>

  <!-- 入札確認 -->
  <bean name="/biddingConfirm" scope="singleton"
    class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

  <!-- 入札完了 -->
  <bean name="/bidding" scope="singleton"
    class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />


  <!-- 価格登録 BL -->
  <bean id="priceInputBLogic" scope="singleton"
    class="jp.co.bbreak.terasoluna.websample.blogic.PriceInputBLogic">
  </bean>

</beans>
```

4.3.3. blogic-io.xml の編集

terasoluna-spring-thin-blank¥webapps¥WEB-INF¥blogic-io.xml に以下の内容を編集します。

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE blogic-io PUBLIC "-//NTTDATA//DTD TERASOLUNA for Spring blogic-io
1.0//JA"
                          "dtd/blogic-io.dtd">
```

```
<!-- ビジネスロジック入出力定義 -->
<blogic-io>

  <action path="/sellingInputBL">
    <blogic-params bean-name="jp.co.bbreak.terasoluna.websample.dto.PriceDto">
      <set-property property="price" blogic-property="price" source="form" />
    </blogic-params>
    <blogic-result>
      <set-property property="price" blogic-property="price" dest="session" />
      </blogic-result>
  </action>

  <action path="/biddingInputBL">
    <blogic-params bean-name="jp.co.bbreak.terasoluna.websample.dto.PriceDto">
      <set-property property="price" blogic-property="price" source="form" />
    </blogic-params>
    <blogic-result>
      <set-property property="price" blogic-property="price" dest="session" />
      </blogic-result>
  </action>

</blogic-io>
```

4.3.4. welcome.jsp の編集

terasoluna-spring-thin-blank¥webapps¥ welcome .jsp を編集します。

```
※　省略
-- ようこそ TERASOLUNA へ --<BR>
<BR>
<center>
<table border="0">
  <tr>
    <td><html:link page="/sellingInput.do">出品登録へ</html:link></td>
  </tr>
</table>
</center>
```

```
</div>
</body>
</html:html>
```

4.3.5. sellingInput.jsp の作成

terasoluna-spring-thin-blank¥webapps¥sellingInput.jsp を作成します。<body>タグ以外は welcome.jsp と同じ内容です。

```
<body>
<div style="text-align: center">
-- 出品する皆様へ --<BR>
<BR>
最低落札価格を入力して、<BR>
「出品」をお願いします。<BR>
<BR>
<center>
<ts:body>
  <ts:form action="/sellingInputBL">
    <table border="1">
      <tr>
        <td>最低落札価格</td>
      </tr>
      <tr>
        <td><html:text property="price" /></td>
      </tr>
      <tr>
        <td ><ts:submit value="出品確認へ" /></td>
      </tr>
    </table>
  </ts:form>
</ts:body>
<BR>
<table border="0">
  <tr>
    <td><html:link page="/welcome.do">トップ画面へ</html:link></td>
  </tr>
```

```
</table>
</center>
</div>
</body>
```

4.3.6. sellingConfirm.jsp の作成

terasoluna-spring-thin-blank¥webapps¥sellingConfirm.jsp を作成します。<body>タグ以外は welcome.jsp と同じ内容です。

```
<body>
<div style="text-align: center">
-- 出品する皆様へ --<BR>
<BR>
この最低落札価格でよろしいですか？<BR>
これでよろしければ、［出品］を<BR>
変更するのであれば、［戻る］を<BR>
クリックしてください。<BR>
<BR>
<center>
<table border="1">
  <tr>
    <td>最低落札価格</td>
  </tr>
  <tr>
    <td><bean:write name="sellingForm" property="price" scope="session" /></td>
  </tr>
  <tr>
    <td ><html:link page="/selling.do">出品</html:link></td>
  </tr>
</table>

<BR>
<table border="0">
  <tr>
```

```
    <td><html:link page="/sellingInput.do">戻る</html:link></td>
  </tr>
</table>
</center>
</div>
</body>
```

4.3.7. selling.jsp の作成

terasoluna-spring-thin-blank¥webapps¥selling.jsp を作成します。<body>タグ以外は welcome.jsp と同じ内容です。

```
<body>
<div style="text-align: center">
-- 出品する皆様へ --<BR>
<BR>
この最低落札価格で登録されました。<BR>
［入札価格入力へ］をクリックして<BR>
担当を交代してください。<BR>
<BR>
<center>
<table border="1">
  <tr>
    <td>最低落札価格</td>
    <td><bean:write name="sellingForm" property="price" scope="session" /></td>
  </tr>
</table>
<BR>
<table border="0">
  <tr>
    <td><html:link page="/biddingInput.do">入札価格入力へ</html:link></td>
  </tr>
</table>
</center>
</div>
```

```
</body>
```

4.3.8. biddingInput.jsp の作成

terasoluna-spring-thin-blank¥webapps¥biddingInput.jsp を作成します。<body>タグ以外は welcome.jsp と同じ内容です。

```
<body>
<div style="text-align: center">
-- 入札する皆様へ --<BR>
<BR>
入札価格を入力して、<BR>
［入札確認へ」をクリックしてください。<BR>
<BR>
<center>
<ts:body>
  <ts:form action="/biddingInputBL">
    <table border="1">
      <tr>
        <td>入札価格</td>
      </tr>
      <tr>
        <td><html:text property="price" /></td>
      </tr>
      <tr>
        <td ><ts:submit value="入札確認へ" /></td>
      </tr>
    </table>
  </ts:form>
</ts:body>
<BR>
</center>
</div>
</body>
```

4.3.9. biddingConfirm.jspの作成

terasoluna-spring-thin-blank¥webapps¥biddingConfirm.jspを作成します。<body>タグ以外はwelcome.jspと同じ内容です。

```
<body>
<div style="text-align: center">
-- 入札する皆様へ --<BR>
<BR>
この入札価格でよろしいですか？<BR>
これでよろしければ、［入札］を<BR>
変更するのであれば、［戻る］を<BR>
クリックしてください。
<BR>
<center>
<table border="1">
  <tr>
    <td>入札価格</td>
  </tr>
  <tr>
    <td><bean:write name="biddingForm" property="price" scope="session" /></td>
  </tr>
  <tr>
    <td ><html:link page="/bidding.do">入札</html:link></td>
  </tr>
</table>
<bean:write name="sellingForm" property="price" scope="session" />
<BR>
<table border="0">
  <tr>
    <td><html:link page="/biddingInput.do">戻る</html:link></td>
  </tr>
</table>
</center>
</div>
```

```
</body>
```

4.3.10. bidding.jsp の作成

terasoluna-spring-thin-blank¥webapps¥bidding.jsp を作成します。<body>タグ以外は welcome.jsp と同じ内容です。

```
<body>
<div style="text-align: center">
-- オークション結果 --<BR>
<BR>
<center>
<BR>
<table border="1">
  <tr>
    <td>最低落札価格</td>
    <td><bean:write name="sellingForm" property="price" scope="session" /></td>
  </tr>
  <tr>
    <td>入札価格</td>
    <td><bean:write name="biddingForm" property="price" scope="session" /></td>
  </tr>
</table>
<BR>
<table border="0">
  <tr>
    <td><html:link page="/welcome.do">トップ画面へ</html:link></td>
  </tr>
</table>
</center>
</div>
</body>
```

4.3.11. PriceInputBLogic.java の作成

terasoluna-spring-thin-blank¥sources¥jp¥co¥bbreak¥terasoluna¥websample¥blogic¥PriceInputBLogic.java を作成します。

```
package jp.co.bbreak.terasoluna.websample.blogic;

import jp.co.bbreak.terasoluna.websample.dto.PriceDto;
import jp.terasoluna.fw.service.thin.BLogic;
import jp.terasoluna.fw.service.thin.BLogicResult;

public class PriceInputBLogic implements BLogic<PriceDto> {

    public BLogicResult execute(PriceDto param) {

        BLogicResult result = new BLogicResult();
        result.setResultObject(param);
        result.setResultString("success");

        return result;
    }
}
```

4.3.12. PriceDto .java の作成

terasoluna-spring-thin-blank¥sources¥jp¥co¥bbreak¥terasoluna¥websample¥dto¥PriceDto.java を作成します。

```
package jp.co.bbreak.terasoluna.websample.dto;

public class PriceDto {

        private String price;

        public String getPrice() {
                return price;
        }

        public void setPrice(String price) {
```

```
                this.price = price;
        }
}
```

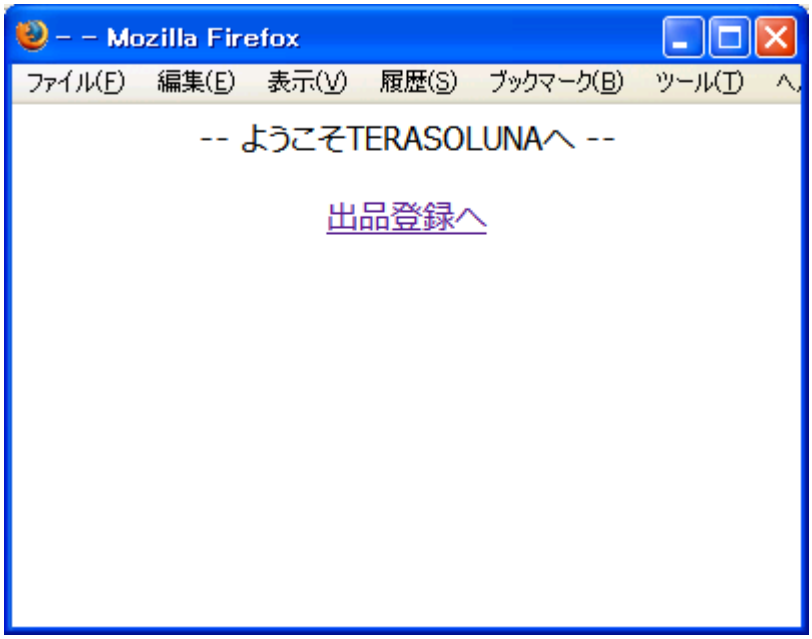
4.4. オークションサイトの実行

パッケージ・エクスプローラーのパッケージ名『terasoluna-spring-thin-blank』を右クリック → ［実行(R)］ → ［サーバーで実行］をクリックします。

ブラウザで以下の URL を開きます。

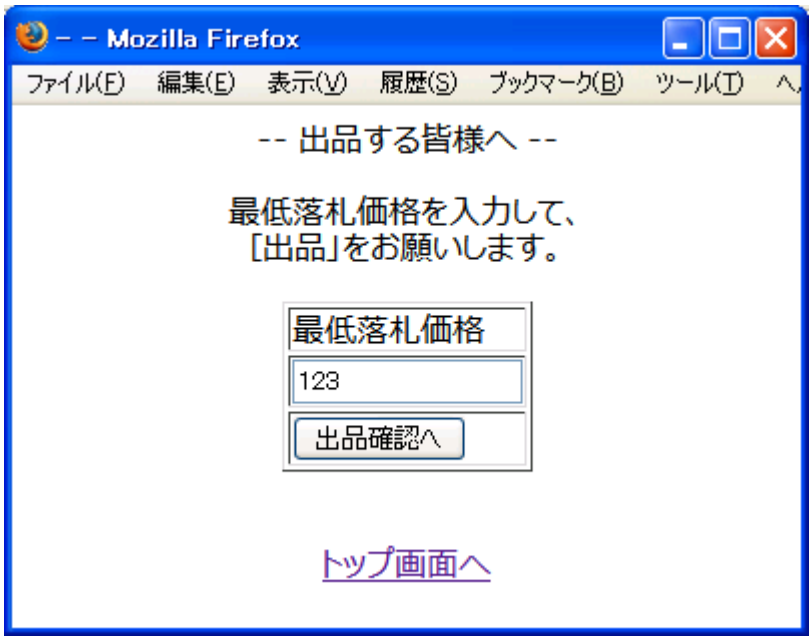
```
http://localhost:8080/terasoluna-spring-thin-blank/
```

4.4.1. ウェルカム画面



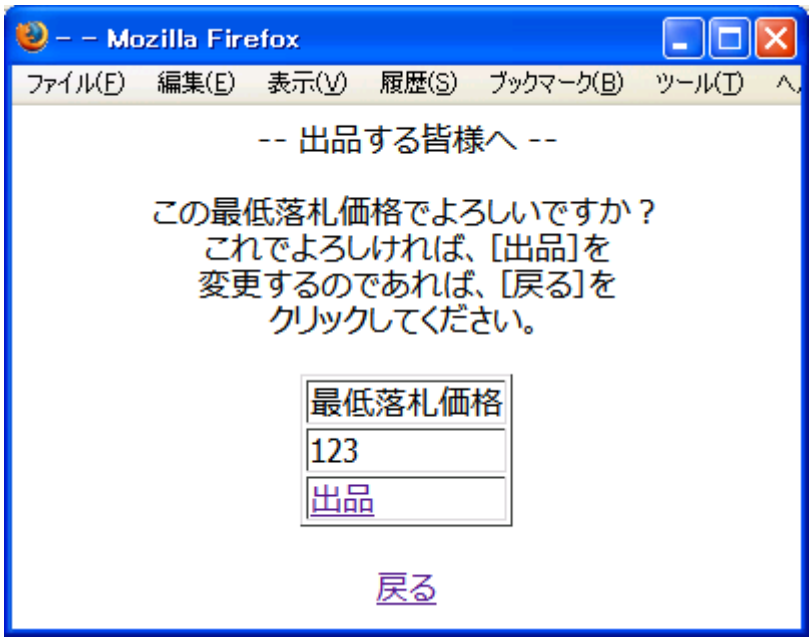
出品登録へをクリックします。

4.4.2. 出品登録画面



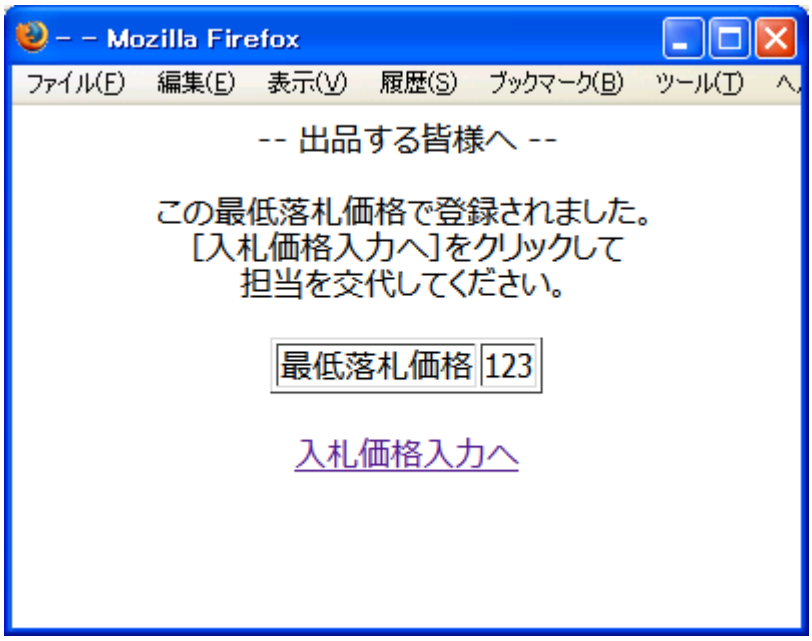
最低落札価格に金額を入力し、出品確認へをクリックします。

4.4.3. 出品確認画面



最低落札価格を確認し、出品をクリックします。

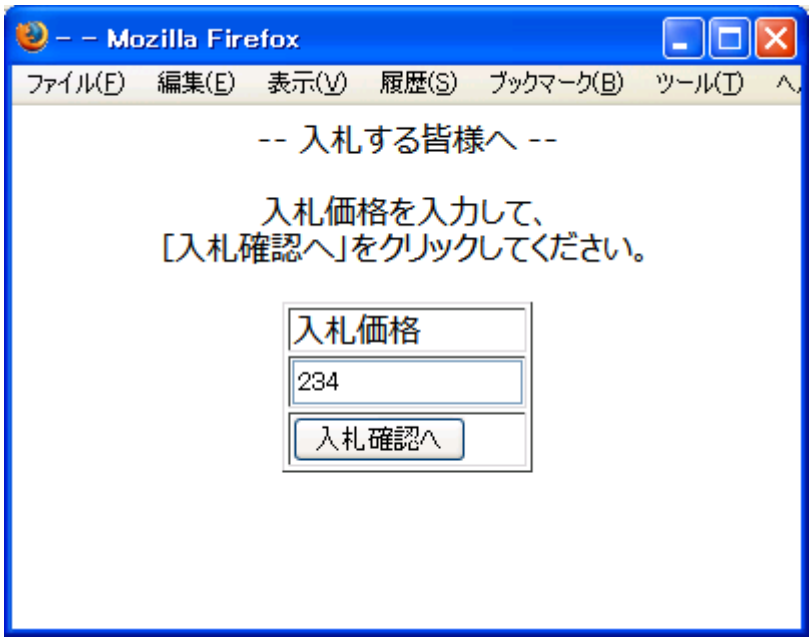
4.4.4. 出品完了画面



出品が登録されました。

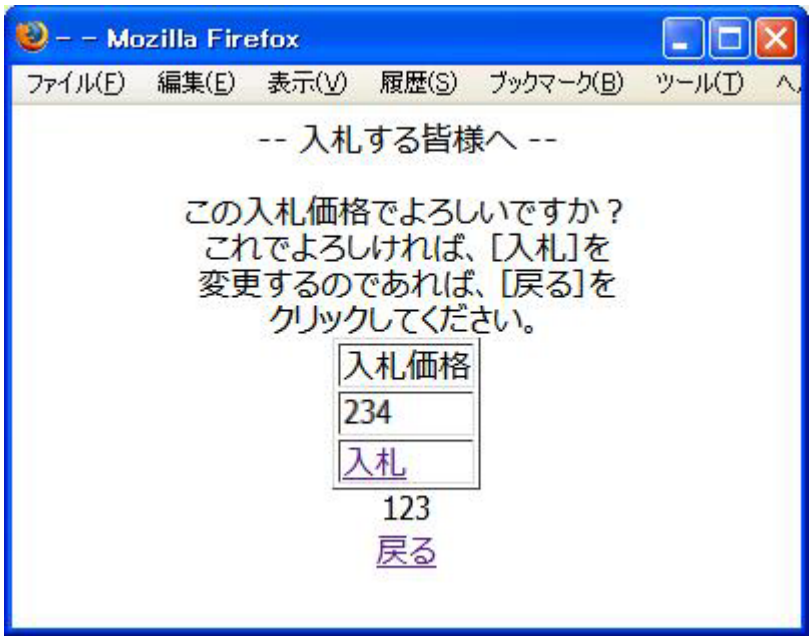
4.4.5. 入札価格入力画面

入札価格入力へをクリックします。



落札できそうな金額を入札価格入力欄に入力し、出品確認へをクリックします。

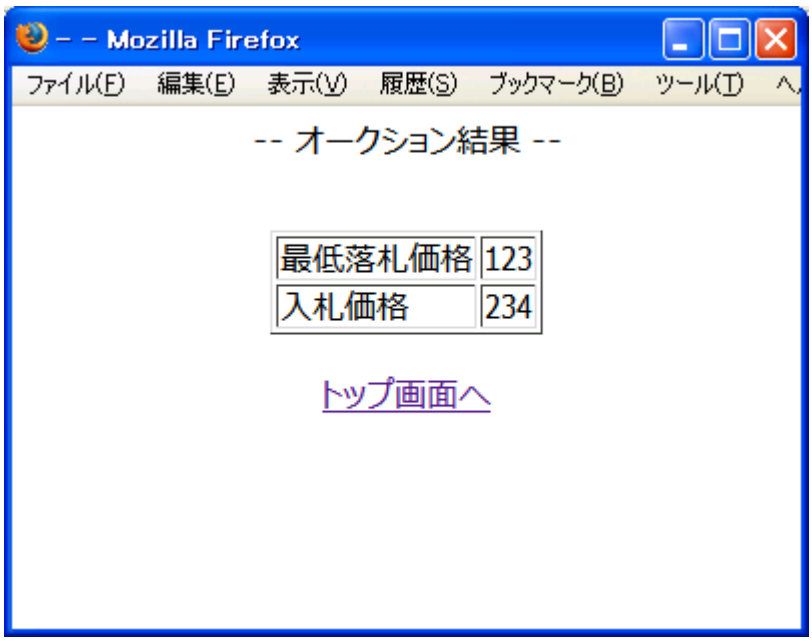
4.4.6. 入札価格確認画面



本来は不要な情報ですが、セッションに最低落札価格が含まれていることを表すために、「入札」と「戻る」の間に表示するようにしております。

入札価格を確認し、入札をクリックします。

4.5. 入札完了画面



入札が完了しました。

セッションに保存されている最低落札価格と入札価格が表示されることを確認できました。した。

4.6. セッションスコープの利点と弱点

簡単にデータを入れたり出したりできることは非常に便利ですが、本来表示されてはいけない内容を表示してしまったり、削除し忘れてメモリを枯渇させてしまったりなどの弱点があります。

5．ユースケーススコープを使用した場合

5.1．サンプルのダウンロードについて

今回のサンプルは以下のリンクからダウンロードできます。

＜サンプルダウンロード＞

http://www.bbreak.co.jp/technique/doc/terasoluna/terasoluna_web2.zip

※このダウンロードファイルは3.3.ブランクプロジェクト『terasoluna-server4jweb-blank_2.0.2.0.zip』との差分になっております。

5.2．セッションスコープからユースケーススコープの対応

4で使用したFormBeanを『_』を付けることにより、ユースケーススコープに対応させます。

sellingForm → _sellingForm

biddingForm → _biddingForm

と変更します。

変更するファイルは、struts-config.xml と sellingConfirm.jsp、selling.jsp、biddingConfirm.jsp、bidding.jspになります。

5.3．オークションゲームの実行

最後の入札確認の画面までは今までと同じように進んでいきます。しかし、その後の入札完了の画面は以下のようになりました。

ログを確認しますと、以下のエラーメッセージが表示されています。

```
[2009/05/08 16:05:35][ERROR][[jsp]] サーブレット jsp の Servlet.service()が例外を投げました
javax.servlet.jsp.JspException: スコープ session に Bean _sellingForm がありません
```

ユースケーススコープに対応させたため、『_biddingForm』が作成されるときに『_sellingForm』がセッションから削除されたために、存在しないエラーとなったわけです。

6. まとめ

ユースケーススコープを使うことにより、そのデータの生存期間が明確になり、本来表示されてはいけない内容を表示してしまったり、削除し忘れてメモリを枯渇させてしまったりなどのセッションスコープの弱点をカバーすることができますが、今回の例でもわかるように、セッション上にないため、要件を満たすためにデータを保管する代替案を検討しなければならなくなります。しかし、ユーザー情報だけセッションスコープに保存して、それ以外の業務データはユースケーススコープに保存するというすっきりしたセッション管理になります。この点だけとっても、非常に使い勝手のよいフレームワークといえるでしょう。

開発部
荒川　正義