付録 B　Apache Wicket

B.1　Java Web アプリケーションフレームワーク

Servlet や JSP を使ってある程度の規模の Web アプリケーションを作成するときは、定型の処理が大部分を占める。例えば、フォームに入力されたデータをチェックしてオブジェクトにまとめる、データベースから取得したデータをオブジェクトにまとめる、セッションにデータを保存し再取得する、などという処理である。

そこで、Web アプリケーションフレームワークが考え出された。共通の定型部分を “枠組み”（フレームワーク）として提供し、個々の Web アプリケーションの開発者にはコードを “枠組み” に、はまるかたちで作成してもらうことで、開発の効率向上を図る。さらに多くのフレームワークは、MVC アーキテクチャーを採用して、プログラマーとデザイナーの分業を可能にした。あまたの Java 用 Web アプリケーションフレームワークがあるが、そのような Web アプリケーションフレームワークの代表が Apache Struts (`http://struts.apache.org`) である。処理は Java プログラム、表示は JSP、設定は XML、に分離して記述できるように設計されている。Apache Struts は Java の Web アプリケーションフレームワークとして、現在では “枯れた” プラットフォームになっている。

B.2　従来型フレームワークの問題点

一方で、設定ファイルが複雑化し、“XML 地獄” と呼ばれる状況が起きている。プログラミング言語ならば簡単にできるような設定が、XML ではできなかったり、複雑になったりする。また、プログラミング言語では当然可能な間違いのチェックが、XML のコードではできない。XML ではなく Java アノテーション (annotation) で設定を記述するフレームワークもあるが、これらの問題は完全には解決しない。

表示についても、JSP の中に EL (Expression Language) や タグライブラリ (tag library) のような “プログラミング言語もどき” が必要になってしまい、結局二重手間になってしまっている。

また、オブジェクト指向らしい設計になっていないので、同じ Web アプリケーションでも、アプレット（の自然な書き方）とはまったく違う記述をする必要がある。

B.3　Apache Wicket

そこで、これらの点を反省した、新しい Java 用 Web アプリケーションフレームワークがいくつか現れてきた。Apache Wicket もその一つである。

設定は Java プログラム中で行う、表示も JSP ではなく通常の HTML 形式を用意し、それを書き換える Java プログラムを記述する、オブジェクト指向で設計されアプレットに近いかたちでプログラムを記述することができる、などの特徴がある。

B.4　Wicket のプログラム例

とはいえ、Wicket を使いこなすにはコンポーネント（Component — 各 HTML タグに相当するクラス）の API を知る必要がある。とりあえず HTML タグを知っていれば何とかなる生の Servlet API に比べると、多少はじめのハードルが高い。

以下では、コンポーネントのうち、ほんの一部を使った、Apache Wicket のプログラム例を紹介する。そのほかのコンポーネントや、より詳しい Wicket コンポーネントの使用法は Wicket プロジェクトの Web ページ（`http://wicket.apache.org`）の左バナーから “API Docs” や “Learn” の下の “Examples”, “Components” などで調べることが出来る。

B.5　Wicket プロジェクト

Wicket のプロジェクトは、通常 Apache Maven というプロジェクト管理ツールを使って生成する。この方法については、別資料で解説する。

プロジェクトを生成すると、ファイル階層の中で、`src/main/java` の下の、生成時に指定したパッケージに応じた場所に、`WicketApplication.java`, `HomePage.java`, `HomePage.html` というファイルができているはずである。アプリケーションに対する様々な設定を行うためのクラスが、この中の `WicketApplication` である。

`WicketApplication` クラスの `init` メソッドでアプリケーションで使用する文字エンコーディングを指定する。

```
    getMarkupSettings().setDefaultMarkupEncoding("UTF-8");
    getRequestCycleSettings().setResponseRequestEncoding("UTF-8");
```

の 2 行を（`// add your configuration here` の下に）追加しておく。もちろん別のエンコーディング (Windows-31J など) を使用することもできる。

B.6　WebPage クラス

個々のページは、`HomePage.java` と `HomePage.html` のように同じディレクトリーに置かれた（拡張子だけが異なる）同名の Java ファイル（Java クラス）と HTML ファイルの組からなっている。

`HomePage.html` の一部を次のように書き換えてみる。

```
 ...
<div id="bd">
  <h2>Current Time</h2>
  <p>ただいま <span wicket:id="date">Date</span> です。</p>
</div>
 ...
```

wicket:id という属性は、Wicket の Javaプログラムから操作するために使われる。

ページに対応する Java クラスは WebPage というクラスを継承して作成する。

```
 ...   // import 文は掲載省略

public class HomePage extends WebPage {
  private static final long serialVersionUID = 1L;
  public HomePage(final PageParameters parameters) {
    super(parameters);

    add(new Label("date", Calendar.getInstance()
                                  .getTime().toString()));
  }
}
```

Label は HTML 中の文字列に対応するコンポーネントである。このコンストラクターの第 1 引数が対応する wicket:id、第 2 引数が、置き換える文字列である。このオブジェクトをページに add することにより、ページを表示したとき、wicket:idが "date" の部分の中身が、現在の日付（Calendar.getInstance().getTime().toString()）に置き換わる。つまり、次のような HTML が生成される。

```
  <p>ただいま <span>Wed Jan 23 12:34:56 JST 2013</span> です。</p>
```

B.7 Wicket の実行

Wicket のプロジェクトをテストするときは、src/test/java の下のパッケージに応じた場所にある Start.java を通常の “Java アプリケーション” として実行する。ブラウザーで http://localhost:8080 にアクセスすると、HomePage の出力結果を見ることができる。（実運用するときは war ファイルをエクスポートして、Web アプリケーションコンテナーに配置する。）

B.8 フォーム・コンポーネント

次に、フォームを使った例を紹介する。その前に、HomePage にこの例へのリンクを付け加えておく。

HomePage.html へ追加

```
<a wicket:id="counterLink">Counter</a>
```

HomePage.java へ追加

```
add(new BookmarkablePageLink<Void>("counterLink", Counter.class));
```

Counter ページの HTML ファイルと Java ファイルを次に示す。

Counter.html

```
...
<div id="bd">
  <h2>累計カウンター</h2>
  <span>整数を入力してください。 </span>
  <form name="submitForm" wicket:id="submitForm">
    <input type="text" wicket:id="inputText" />
    <input type="submit" />
  </form>
  累計は <span wicket:id="total">-1</span> です。
</div>
...
```

Counter.java

```
...

public class Counter extends WebPage {
  private static final long serialVersionUID = 1L;
  private int counter = 0;

  public Counter(final PageParameters parameters) {
    super(parameters);

    final TextField<Integer> inputText
      = new TextField<Integer>("inputText",
                               new Model<Integer>(0),
                               Integer.class);
    final Label total = new Label("total", "0");
    Form<Void> submitForm = new Form<Void>("submitForm") {
      private static final long serialVersionUID = 1L;

      @Override
      protected void onSubmit() { // 「送信」を扱う
        int number = inputText.getModelObject();
        counter += number;
        total.setDefaultModelObject(counter);
      }
    };
    submitForm.add(inputText);
    add(submitForm);
    add(total);
  }
```

```
29 }
```

この Wicket ページは、テキストフィールドに入力した整数の累計を表示していくだけ、という単純な機能を提供している。

まず、セッションがまったく明示的に扱われていないことに注目する。状態を保存するために、通常のフィールド（インスタンス変数）counter が使われている。しかし実際に試してみると、別セッションどころか、同一ブラウザーの別のタブからアクセスしても、異なる Wicket ページの結果が混ざらず、ページごとに累計が計算される。これは舞台裏で極めて巧妙にセッションが使われているからである。

HTML の form タグに対応しているのは、Form というクラスを継承する匿名クラスである。コンストラクターの第 1 引数はやはり対応する wicket:id である。このクラスの onSubmit というメソッドに「送信」ボタンをクリックしたときの処理を記述する。

テキストフィールド（<input type="text" ... >）に対応するのは、TextField というクラスである。このコンストラクターに第 2 引数として渡されている Model は入力されたデータを保持するためのオブジェクトである。ここに IModel インタフェースを実装する別のクラスのオブジェクトを使用して、データを別のオブジェクトに持たせるようにすることもできるが、詳しい説明は割愛する。この Model と TextField の型パラメーターとして、Integer を指定し、TextField のコンストラクターに Integer.class を渡すことにより、テキストフィールドに入力された文字列が整数に変換できないときは、入力を無視する。

テキスト入力に入力された値は、TextField クラスの getModelObject メソッドで取り出す。また、ラベルの文字列は、Label クラスの setDefaultModelObject メソッドで変更する。結局、同じような動作をするアプレットとほとんど同じような構成になっている。

B.9 Quiz

Quiz の Wicket 版を紹介する。問題のデータをテキストファイルから読み込み、過去に正解した問題数などによって、表示を変更する Web ページである。このために正解した問題数や現在何問めを実行しているかを記憶しておく必要がある。Wicket 版の場合、通常のフィールドにこのような情報を記憶しておく。

なお、HomePage に Counter のときと同じようにこのページへのリンクを付け加えておく。

HomePage.html へ追加

```
<a wicket:id="quizLink">Quiz</a>
```

HomePage.java へ追加

```
add(new BookmarkablePageLink<Void>("quizLink", Quiz.class));
```

Quiz.html

```
...
 <div id="bd">
     <span wicket:id="message">Message will be here.</span>
     <span wicket:id="text">Text will be here.</span>
     <form name="submitForm" wicket:id="submitForm">
         <div wicket:id="radio"></div>
         <input type="submit" />
     </form>
 </div>
...
```

Quiz.java

```
...

public class Quiz extends WebPage {
  private static final long serialVersionUID = 1L;
  int number = 0, score = 0;
  ArrayList<String[]> questions = new ArrayList<String[]>();
  ArrayList<String[]> results   = new ArrayList<String[]>();
  String answer;
  Label messageLabel, textLabel;
  RadioChoice<String> radio;
  Form<Void> submitForm;

  public Quiz(final PageParameters parameters) {
    super(parameters);
    try {
      loadQuiz();
    } catch (IOException e) { /* 本来は何かエラーを表示する */ }

    messageLabel = new Label("message", new Model<String>());
    // タグをエスケープしない
    messageLabel.setEscapeModelStrings(false);
    add(messageLabel);
    textLabel = new Label("text", new Model<String>());
    add(textLabel);
    submitForm = new Form<Void>("submitForm"){
      private static final long serialVersionUID = 1L;

      @Override
      protected void onSubmit() {
        String answer = radio.getModelObject();
        String[] tokens = questions.get(number++);
        String right
          = tokens[Integer.parseInt(tokens[tokens.length-1])];
        String check;
        if (answer.equals(right)) {
          score++;
          updateMessage("<p>正解です。</p>");
```

```
38            check = "○";
39          } else {
40            updateMessage(String.format(
41              "<p>残念でした。あなたの選択は%s、正解は%sでした。</p>",
42              answer, right));
43            check = "×";
44          }
45          results.add(new String[] {
46            ""+number, tokens[0], answer, right, check });
47        }
48      };
49      radio = new RadioChoice<String>("radio", new Model<String>(),
50                                      new ArrayList<String>());
51      submitForm.add(radio);
52      add(submitForm);
53
54      updateMessage(
55        "<p>ようこそ、Quiz␣へ！<br/>では最初の問題です。</p>");
56    }
57
58    private void loadQuiz() throws IOException {
59      ServletContext servletContext
60        = WebApplication.get().getServletContext();
61      File f = new File(
62                  servletContext.getRealPath("/WEB-INF/quiz.txt"));
63       ...   // 省略
64    }
65
66    private void updateMessage(String message) {
67      if (number >= questions.size()) {
68        Result resultPage = new Result(null);
69        message += String.format(
70          "<p>これで  QUIZ␣は終わりです。正解は␣%d␣問でした。</p>",
71          score);
72        resultPage.messageLabel.setDefaultModelObject(message);
73        resultPage.listView.setModelObject(results);
74        setResponsePage(resultPage);
75      } else {
76        messageLabel.setDefaultModelObject(message);
77        String[] tokens = questions.get(number);
78        textLabel.setDefaultModelObject(
79          String.format("第␣%d␣問:␣%s", number+1, tokens[0]));
80        List<String> options = new ArrayList<String>();
81        for (int i=1; i<tokens.length-1; i++) {
82          options.add(tokens[i]);
83        }
84        radio.setChoices(options);
85      }
86    }
```

```
87 }
```

本質的な部分は生の Servlet API を使ったときの例と変わっていないが、やはり、セッションを明示的に扱う部分はない。

ラジオボタンに対応する RadioChoice というコンポーネントを使用している。選択肢を設定するのに、このクラスの setChoices というメソッドを使用する。

クイズの問題が書かれたファイルにアクセスするために、ServletContext クラスが必要だが Quiz クラス自体は、 Servlet ではない（HTTPServlet を継承していない）。つまり、getServletContext メソッドを持っていない。そこで ServletContext クラスのインスタンスを、WebApplication クラスの get クラスメソッド経由で取り出している。

すべての問題を解いたあと、この Wicket 版では別のページ（Result）に移動して結果を表示することにしている。ページを移動するためには、ページオブジェクト、つまり WebPage クラスのサブクラスのインスタンスを作成して、setResponsePage メソッドに渡せば良い。

ページオブジェクトを作成してから、メソッドを呼び出してデータを受け渡すことができるので、ここでもセッションを明示的に使う必要がない。

B.10　リストビュー

クイズの結果を表形式で表示する Result クラスはリストビュー（ListView）というコンポーネントを使用している。

Result.html

```
1  ...
2   <div id="bd">
3     <h2>Result</h2>
4     <span wicket:id="message">Message will be here.</span>
5     <table border="1">
6       <tr>
7         <th>番号</th>
8         <th>問題</th>
9         <th>あなたの選択</th>
10        <th>正解</th>
11        <th>○×</th>
12      </tr>
13      <tr wicket:id="resultListView">
14        <td wicket:id="no"></td>
15        <td wicket:id="text"></td>
16        <td wicket:id="choice"></td>
17        <td wicket:id="answer"></td>
18        <td wicket:id="check"></td>
19      </tr>
20    </table>
21  </div>
22 ...
```

Result.java

```
...

public class Result extends WebPage {
  private static final long serialVersionUID = 1L;
  Label messageLabel;
  ListView<String[]> listView;
  public Result(final PageParameters parameters) {
    super(parameters);

    messageLabel = new Label("message", new Model<String>());
    // タグをエスケープしない
    messageLabel.setEscapeModelStrings(false);
    add(messageLabel);
    listView = new ListView<String[]>("resultListView",
                new ArrayList<String[]>()) {
      private static final long serialVersionUID = 1L;

      @Override
      protected void populateItem(ListItem<String[]> item) {
        String[] row = item.getModelObject();
        item.add(new Label("no", row[0]));
        item.add(new Label("text", row[1]));
        item.add(new Label("choice", row[2]));
        item.add(new Label("answer", row[3]));
        item.add(new Label("check", row[4]));
      }
    };
    add(listView);
  }
}
```

リストビューは任意の要素の繰り返しを作成するために使用される。この例では、wicket:id が "resultListView" の tr 要素が、ListView コンストラクタの第 2 引数のリストの要素数だけ繰り返されることになる。最初作成されるとき、このリストは空だが、あとで Quiz クラスから setModelObject メソッドで変更されることになる。

繰り返される個々の HTML 要素の内部には populateItem メソッドでコンポーネントを追加する。populateItem に引数として渡される ListItem オブジェクトから getModelObject メソッドで ListView の ModelObject（コンストラクターの第 2 引数）の個々の要素を取り出す。これをもとにコンポーネントを生成し、ListItem オブジェクトに add する。この例では、5 つの Label オブジェクトを生成している。

キーワード :

Web アプリケーションフレームワーク, MVC アーキテクチャー, Apache Strut, Apache Wicket, Apache Maven, ページ, コンポーネント, リストビュー,