ソフトウェア工学 129－2
（2000．11．16）

JSPを用いたWebアプリケーション構築のためのフレームワークUJI

松塚 貴英* 野村 佳秀*

ビジネスアプリケーションをWebアプリケーションとして開発する際に、JSP/Servlet環境を使用する場面が多くなっているが、膨大な画面や処理を効率的に開発・保守できる手法が求められている。JSP/Servletによるアプリケーション構成法としてJ2EE Blueprintsが提案されているが、検討の結果、ページやロジック、データ間の連携が密になり保守性を損ねるという問題があることが分かった。そこで、我々がこれまで開発してきたJava Applet向けのプレゼンテーションフレームワークの技術をもとに、JSP向けのアプリケーションフレームワークを開発した。このフレームワークでは、JSPで表されるビュー部と、ビューが表示/入力の対象とするデータオブジェクト、データオブジェクトを操作するロジックをマッピング定義を介し疎に連携することにより、開発とメンテナンスの効率化を実現している。これまで、このフレームワークを利用してアプリケーションを開発することで、性能に影響を与えずに総コード量で10%、ロジックとしてメンテナンスする部分は85%以上の削減が可能であった。

UJI: a Web Application Framework using the JSP Environment

Takahide Matsutsuka, Yoshihide Nomura

Servlet and JSP become to be widely used for developing business applications in the web environment. In this situation, some methods are required for making a development and a maintenance easy since the number of screens or processes tends to be large. We recognized some problems in a web application develpment method in the J2EE Blueprints which is becoming a standard for using Servlet/JSP. So we made an application framework on them using our technology which is used for the development of Java applets. The framework separates JSPs and data and logics, and links them using some mapping definitions. In our evaluation, the framework decreased 10% of total code and 85% of presentation logic through the development of the application without decreasing a performance.

1 はじめに

ビジネスアプリケーションを構築する際のフロントエンドとして、Web環境を採用することが一般的になった。実装する際には、近年はASP[1]やJSP[2]といったスクリプティング環境を使用することが多い。これらのスクリプティング環境では、HTMLソース内に直接記述することによって手軽に開発が行なえる。しかし、実際に中規模～大規模のビジネスアプリケーションを構築する際は、画面数が多いことや、処理の種類が膨大であることなどから、ASPやJSPをそのまま使うわけではなく、何らかの標準化や開発省力化の手法が必要となる。

筆者らは、以前にアプレット環境においてサーバ・クライアントをコントロールするフレームワークUJI[1]を提案し、実アプリケーションにおいて有効性を確認している[3][4]。UJIでは、サーバとクライアントの接続制御を行なうとともに、プレゼンテーションのビューとロジックを分離し、データオブジェクトがその間を繋ぐというアプローチをとっている(図1)。

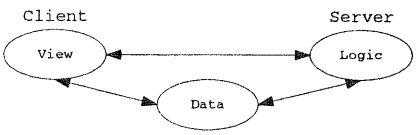

図1: アプレット環境のUJI構成

他にも、アプレットやWindows環境においてはSwing[5]やMFC[6]などのフレームワークが存在するが、いずれもWeb環境にそのまま適用できるものではない。

本論文では、UJIの考え方をWebアプリケーションにも適用することは有効であると考え、JSP環境上にフレームワークの構築を行なった。このJSP版UJI(以降UJIと表記)は、JSPで表されるビュー部と、ビューが表示/入力の対象とするデータオブジェクト、データオブジェクトを操作するロジックを疎に連携することにより、開発とメンテナンスの効率化を目指したフレームワークである。

*富士通研究所 ソフトウェア研究部
[1]UJIは、宇治茶に由来している。

2 J2EE Blueprints Web-Tierの評価

一般にビジネスアプリケーションでは、扱うデータ項目の数が多く、それにともなって画面の種類も多い。そのため、プレゼンテーションロジックでは大量のデータ項目と画面をいかに部品化し、整理して実装できるかが重要である。また、頻繁に画面仕様が変更されるため、画面表現とデータやロジックを分離して実装するとメンテナンスに有利となる。

J2EE Blueprints[9] では、プレゼンテーションを実装する Web Tier の実装方法として、JSP と JavaBeans を使用する方法を推奨している (図 2)。JSP には画面を構成する HTML と、JavaBean からデータを取得して表示するタグを記述し、JavaBean には JSP とやり取りするデータとロジックを記述する。こうすることにより、データやロジックは画面表現と分離され、画面仕様の変更によるカスタマイズが容易となる。

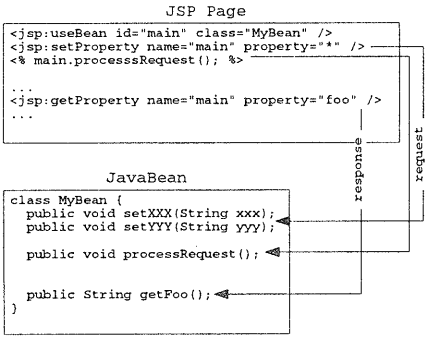

図 2: J2EE Blueprints の Web Tier

処理の流れとしては、次のようになる。

1. JSP ページが Web ブラウザからの HTTP リクエストを受け付ける。
2. JSP ページに対応した JavaBean を用意し、リクエストデータを設定する。
3. JavaBean のメソッド (通常は固定的に processRequest などのメソッド名を使用する) を呼出し、ビジネスロジックを処理する。
4. 再び JSP の処理において、JavaBean に設定されているデータを使用しながらページを表示する。

しかし、この方法には以下のような問題が存在するため、Blueprints の方法をそのまま利用して大規模なアプリケーションを開発することは困難である。

2.1 データとロジックの非分離

ビジネスアプリケーションでは、ロジック同士の分離を良くすることによってメンテナンス性が向上する。たとえば、利用者データに関する処理としては、ログイン、パスワード変更、利用者検索などがあるが、これらは同じデータを使用すること以外に互いに密接なつながりはないため、個別にメンテナンスを行ないたい。

しかし、Blueprints の方法ではデータとロジックが同一 JavaBean に実装されているため、これらの処理をそれぞれ別のクラスに分離することが困難となっている。仮にそれぞれのロジックを別のクラスに実装し、JavaBean 内からそれらを呼び出す構造をとったとしても、後述する振り分け処理の問題が発生する。

2.2 リクエストとレスポンスの非分離

ログイン画面から、ユーザ名とパスワードをリクエストし、そのレスポンスとして業務画面を表示するなど、リクエストとレスポンスでは表すデータもページも異なる場合が多いが、上記方法ではこの 2 者をうまく分離して実装することが困難となり、再利用性を阻害している。

ページに対応する JavaBean が 1 種類の場合は、その JavaBean 内にリクエスト用データとレスポンス用データを両方持たなければならない (図 2)。

ページにリクエスト用、レスポンス用と 2 種類の JavaBean を導入しても、JSP からリクエスト用、レスポンス用の両方の JavaBean を参照するため、このページは特定のリクエスト・レスポンスの組合わせでしか利用できない (図 3)。

2.3 振り分け処理の問題

構造上、いわゆる振り分け処理が 2 箇所に必要となる。

- Web ブラウザからのリクエスト (押されたボタンなど) に伴う処理ルーチンへの分岐部分
- 処理ルーチンから遷移後の画面を選択する部分

Blueprints の方法では、これらに関する特別の配慮がないため、Web ブラウザからのリクエストに対しては JavaBean 内で押されたボタンなどを

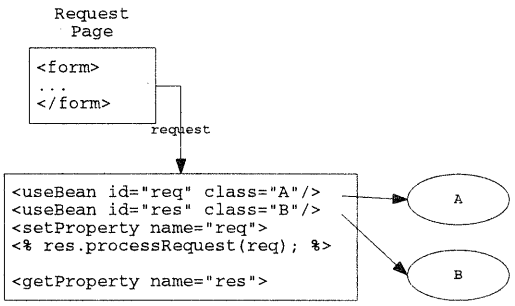

図 3: リクエストとレスポンスを両方記述した JSP

判断しなければならない。押されたボタンによる処理は全く異なるのが普通なので、振り分け処理により別メソッドを呼び出す等の処理が常に JavaBean 内に記述されることになり、処理の種類が増減した場合への対応が難しい。

また、JavaBean での処理後に表示する画面が変わる場合、JavaBean 内または JSP 内で表示する画面の変更を行なう必要がある。このとき、JavaBean に画面遷移を記述すると、画面情報が JavaBean に入ってしまい、画面とデータの連携が密になってしまう。一方 JSP に画面遷移を定義すると、画面間の連携が密になってしまう。いずれにしても再利用性を悪化させることになる。

3 UJI フレームワークの設計と実装

3.1 設計方針

前節の問題点を解決するためには、特別の仕組みなしに JSP や JavaBean を直接実装する Blueprints の方法では無理があるため、以下の方針でアプリケーションの設計を検討し、フレームワークを作成することにした。

- Blueprints では未分離となっているデータとロジックを分離する。
- 1 つの JavaBean が複数の画面情報を持たないようにする。同様に、1 つの JSP が複数の JavaBean の情報を持たなくても良いようにする。
- イベントによるロジックの選択と、処理結果による画面の選択は直接に行なわず、マッピング定義を通して間接的に行なえるような仕組みを設ける。

3.2 UJI による Web-Tier

前節の方針から、UJI で構成される Web-Tier は、図 4の構成とした。Blueprints での Web-Tier(図 2) に対して、次の点が異なる。

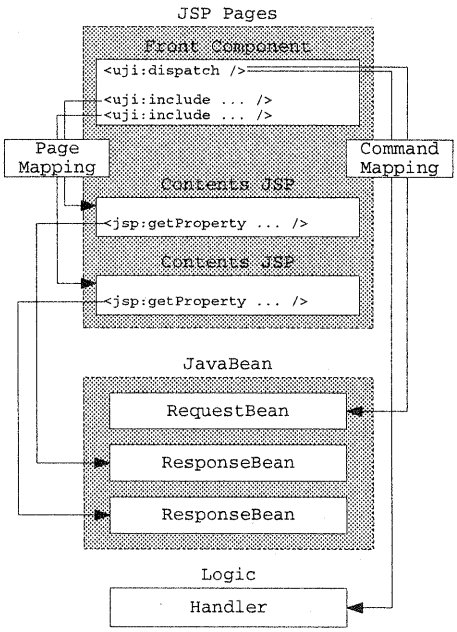

図 4: UJI で構成する Web-Tier

- JSP ページを分割し、複数のページを部品 (Contents JSP) として include 可能とした。各ページはそれぞれ異なる JavaBean を表示するようにし、異なるデータを分離して表せるようにした。また、これをサポートするためにフロントコンポーネントという制御ページを設けた。
- JavaBean はリクエスト用 (RequestBean) とレスポンス用 (ResponseBean) に分け、入力/表示するデータの種類に応じて用意できるようにした。
- ロジック部分は Handler としてデータから独立させた。
- フロントコンポーネントから Handler(ロジック) を呼び出したり Contents JSP を include したりする際に、それぞれ Command Mapping と Page Mapping というマッピングの仕組みを導入した。これにより、フロントコンポーネントを特定の JavaBean やロジックから非依存に記述できるようにした。

これらにより、2章の問題点に関し解決を図った。

3.3 実装環境

実装の際は移植性を考慮すると、どのServletコンテナ上でも動作可能であることが必要である。そのため、JSPのカスタムタグの仕組みを利用して実装を行なった。また、カスタムタグがサポートされているServlet 2.2/JSP 1.1の環境上で動作可能なものとした。

リファレンスプラットフォームとしては、Java2 SE 1.3[7]とTomcat 3.1[8]を使用した。

3.4 UJIの処理の流れ

UJIによるアプリケーションの処理の流れは、次のようになる。

1. フロントコンポーネントがWebブラウザからのHTTPリクエストを受け付ける。
2. フロントコンポーネント内に記述された<uji:dispatch>タグにより、RequestBeanにデータを設定する。
3. 引き続き、Command Mapに記述された定義にしたがってHandlerのメソッドを呼び出す。
4. Handler内ではRequestBeanからデータを取得し、ビジネスロジックを処理する。処理結果はResponseBeanに設定する。
5. フロントコンポーネント内に記述された<uji:include>タグにより、includeするべきContents JSPページを決定する。Contents JSPページ内でResponseBeanに設定されているデータを使用しながらページを表示する。

3.4.1 HTTPリクエストの受け付け

WebブラウザからのHTTPリクエストは、フロントコンポーネントが受け付ける。フロントコンポーネントは、Webブラウザからの要求を受付けるための専用の「入口」となるJSPページやServletである。特定の処理には依存せず、

- UJIフレームワーク処理の開始
  (<uji:dispatch>タグ)
- 表示ページのレイアウトとinclude
  (<uji:include>タグ)

の2つの処理を行なう(図5)。

フロントコンポーネントは特定の処理に依存するものであってはならない。なぜなら、特定の処理に依存してしまうと、各ページの<form>タグによるリクエスト先を変えなければならないことになり、ページ間で依存関係が発生して再利用性を阻害するためである。

```
<uji:dispatch />
<table>
<tr>
<td> <uji:include bean="left" /> </td>
<td> <uji:include bean="right" /> </td>
</tr>
</table>
```

図5: フロントコンポーネント

3.4.2 RequestBeanへのデータ設定

Webブラウザからのリクエストと、それがRequestBeanに設定される様子を図6に示す。フロントコンポーネントをリクエストに対して非依存にするためには、リクエスト内容が設定されるRequestBeanのクラスの情報を外部から入力する必要がある。UJIでは、これを"uji.bean"という名前で示されるリクエストパラメータに設定するようにした。

もともとリクエストする内容はWebブラウザに<form>タグ内の要素として表示されているため、formとRequestBeanオブジェクトが自然に対応づけられることになり、設計上も理解しやすい。

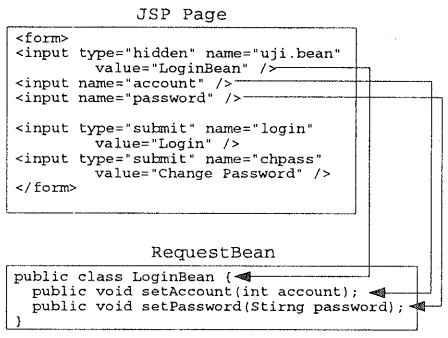

図6: formからRequestBeanへのデータ設定

3.4.3 Handlerの呼出し

Handlerを呼び出す際は、RequestBeanの種類だけで分岐するには情報が不足しているため、RequestBeanに対するアクション情報として押されたボタンの情報を併用することにした。分岐にはCommand Mapという定義を使用し、使用してい

る RequestBean のクラス名と押されたボタンの名前[2]から Handler のメソッド名を検索する (図 7)。

この方法により、リクエストに対応するロジックの決定方法が間接的なものとなるため、JSP 内に処理名 (processRequest など) を記述する必要がなく、振り分け処理も自動化される。また、複数の処理に同じ Handler を割り当てることもできるため、業務単位などで Handler を柔軟に設計できる。

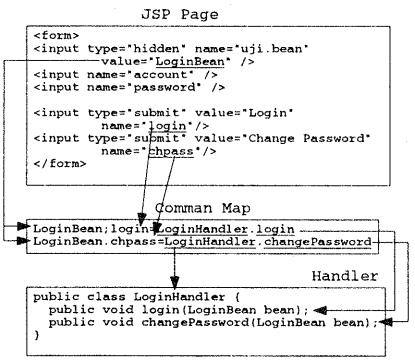

図 7: ハンドラへの分岐

3.4.4 Handler の記述

Handler 内では RequestBean が引数で渡され、そこからリクエスト情報を取得する。通常、そのデータを使用して EJB やデータベースにアクセスしてビジネスロジックを実行する。実行後の処理結果を表示するためには、データを ResponseBean に設定する。

図 4のように、一つの画面で複数の ResponseBean の組を使用できるようにする必要があるため、それぞれの ResponseBean はペイン名というキーで識別される。図 8は、"left" という名前のペインに UserBean オブジェクトを設定している様子を示す。

3.4.5 Contents Pege の include

Handler での処理が終了すると、フロントコンポーネントの<uji:include>タグにより Contents Page が include される。図 5の例では、画面全体をテーブルで左右に分け、それぞれ左側に "left"、右側に "right" というペイン名で指定されたオブジェクトを使用して include を行なう。

[2]実装上はボタン名と通常のテキストフィールドの名前は区別できないため、"uji.verb" リクエストパラメータでアクションを明示指定することもできるようにしてある。

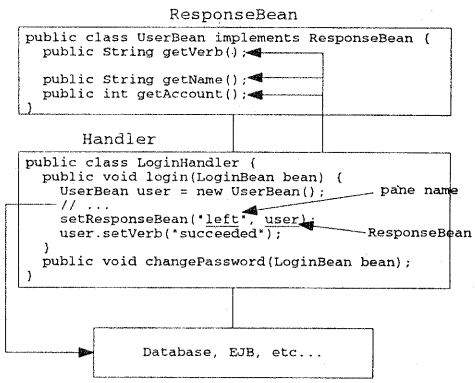

図 8: ハンドラ処理

include の際は、

- ResponseBean のクラス名
- ResponseBean から getVerb メソッドで取得される「表示方法」

が使用され、この 2 つのデータに合うページ名がページマッピングから決定され、表示に使用される。図 9の例では、UserBean に設定された verb が succeeded の場合と failed の場合でそれぞれ login-succeeded.jsp か login-failed.jsp が使用されることになる。

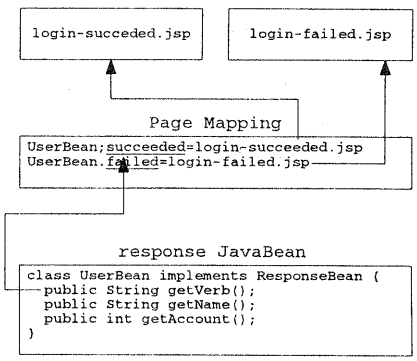

図 9: 表示ページの選択

各表示ページでは、Handler 内で設定された RequestBean が Request スコープで useBean できるようになっており、Blueprints での方法と同様に<jsp:getProperty>等を使用して表示を行なう。

4 考察

UJI フレームワークに関する考察として、記述に関する評価と基礎性能の評価を行なった。

4.1 記述に関する評価

UJI によるアプリケーションの再利用性を調査するために、Servlet で記述された業務処理のプレゼンテーション部分を UJI を利用したものに書換えた。

4.1.1 Blueprints との記述の相違

記述したアプリケーションの構成を図 10に示す。2 章で議論した Blueprints の構成と比較し、次の利点があった。

- JavaBean 内にはデータのみが存在し、データを引き出してデータベースに格納する、画面に表示するデータをデータベースから取出して JavaBean に設定するといったビジネスロジックはすべて Handler 内に実装された。
- 一つの JSP が参照する JavaBean は 1 種類のみとなり、他のページからのリクエストをマッピングする JavaBean の情報を入れる必要がなくなった。
- ハンドラのメソッド選択とページ選択はマッピングに記述され、振り分け処理が不要となった。

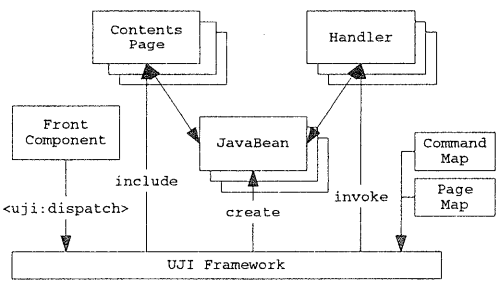

図 10: UJI によるアプリケーション構成

2 点目については、実際には 1 つの JavaBean が、RequestBean と ResponseBean の両方の働きをもつように実装された。この様子を図 11に示す。JSP A から JSP B にリクエストがあった場合、Blueprints では JSP B から JavaBean A に対してデータを設定する必要があるため、ここでクラス名の参照が発生してしまう。一方 UJI では、リクエスト内容をどの JavaBean にマッピングするかという情報を "uji.bean" リクエストパラメータが指定するため、このような参照がなく、依存関係を解消することができた。

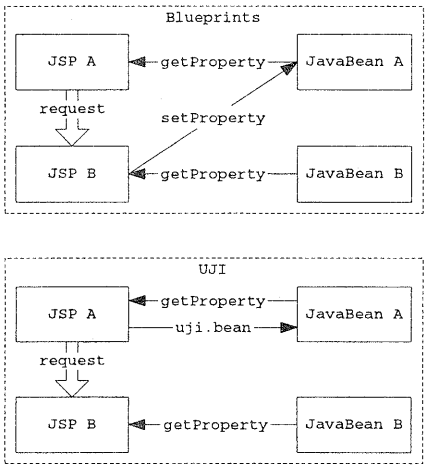

図 11: JSP と JavaBean の対応

さらに、Blueprints では直接言及していない利点として、ロジックの分類が柔軟になったという点がある。Blueprints のように JavaBean にロジックを実装するわけではないため、いくつかのリクエストの種類からなる業務処理を任意の粒度で Handler クラスとしてまとめられる。これにより、より業務の種類に忠実にロジックの分類ができた。

これらのことから、アプリケーション記述の際の部品化がより効率的に行なわれ、再利用性が向上していることが分かった。

4.1.2 記述量

もとの業務処理は Servlet で記述されていたため、開発行数の比較を行なった (図 12)。全体では UJI による開発量は Servlet に比べおよそ 1 割減である。しかし Servlet 版では、もともとロジック・データ・ビューの区別が曖昧であるため、表中でビューとなっている部分に、表示する項目を選択するための `if` 文による条件分岐が大量に実装されており、ロジックとしてメンテナンスする対象となっている。そのため、ロジックに対する変更が発生した場合にメンテナンスが必要な行数は

	Servlet	UJI [a]
フロントコンポーネント	64	21
マッピング		22
ロジック	**345**	**221**
データ	406	468
ビュー	**1288**	1239
計	2103	1971

[a]UJI の記述のうち、フロントコンポーネントとビューが JSP のコード量。他は Java のソースコード行数 (コメント除く) である。

図 12: 記述量比較 (ステップ)

		初回[a]	なし[b]	あり[c]
Servlet	データ設定	14	13	83
	ページ出力	264	219	216
	合計	278	232	299
JSP	データ設定	45	13	175
	ページ出力	296	204	216
	合計	341	217	391
UJI	データ設定	169	14	144
	ページ出力	461	231	240
	合計	630	245	384

[a]サーバ起動後最初のリクエスト (リクエストパラメータなし)
[b]リクエストパラメータなし
[c]リクエストパラメータあり

図 13: 性能比較 (単位:ms)

Servlet 版が (345+1288) 行であるのに対し、UJI 版では 221 行となり、大幅に削減されている。

4.2 性能

性能評価のために、次のような動作をするアプリケーションを Servlet, JSP, UJI のそれぞれで作成し、実行時間の計測を行なった。

処理は次の通り。

1. 55 個の<input>タグが含まれる form が存在するページからのリクエストを受け取り、リクエストデータを JavaBean に設定する。

2. この JavaBean を参照しながらページ (生成後サイズは約 32kB) を生成して出力する。

このうち、1~3 の処理を行なうもの (リクエストパラメータあり) と 3 のみを実行するもの (リクエストパラメータなし) の双方の実行時間を計測した (図 13)。

リクエストパラメータありの計測結果から、Servlet 環境に対し、JSP や UJI の環境では、ページ出力時間は同等で、データ設定 (リクエスト解析+JavaBean へのプロパティ設定) に時間がかかっている。これは、リクエストパラメータを解析する際に Servlet ではプログラムで画面固有に実装し、JSP は<jsp:setProperty>タグ、UJI は<uji:dispatch>タグによる汎用化された仕組みを利用しているためと考えられる。パラメータは少ないほど性能差は小さく、多くなるほど性能差が大きくなるといえる。

	初回	なし	あり
Servlet	95	94	72
JSP	86	94	55
UJI	73	94	63

図 14: 処理時間中の HTML 出力時間の割合 (単位:%)

また、全処理時間の中で HTML 出力が占める割合図 14から、処理時間の中でもっとも大きな比重を占めるのは HTML の出力時間であることがわかる。これらの結果から、出力する HTML のサイズが小さいほど Servlet と JSP/UJI の間の性能差が開き、大きくなるほど性能差は小さくなるといえる。

上記結果から、JSP と UJI の間の性能差はほとんどない。これは、UJI 固有の処理内容がハンドラ選択やページ選択などであり、JSP に比べてあまり多くなく、かつ軽い処理であるためと考えられる。

5 関連研究

Web のプレゼンテーション開発のためのフレームワークとしては、次のようなものが研究/開発されている。

5.1 SPFC

Java Apache Project で開発されている [10]。JFC(Swing) に近い記述方式で HTML ページを表現できるようにする。画面を TextField や Button といったオブジェクトから構成されたクラスとして記述することができるため、Swing の開発環境をそのまま利用できるようになる可能性があるが、逆に HTML エディタのような開発ツールの使用は困難となっている。

コマンド選択は EventListener を利用するため、UJI のようにマッピングのような仕組みを作りこ

むことは容易である。

5.2 Struts

Jakarta Projectで開発されている[11]。フロントコンポーネントとして機能するServletがUJIのRequestBeanに相当するActionFormクラスにデータを設定し、Handlerに相当するActionクラスを呼び出す。この点でUJIにかなり近い。

相違としては、以下の点が挙げられる。

- Actionの処理結果に応じて表示するJSPページが決定されるが、trueかfalseの2値しか返さないため、UJIのページマップで実現している柔軟なページ選択はできない。また、UJIがフロントコンポーネントで行なっている表示の部品化とレイアウトは行なわない。
- 処理結果はSession attributeとして設定されるため、不要になった場合のパージをユーザプログラムで行なう必要がある。

6 おわりに

ビジネスアプリケーションをWebアプリケーションとして開発する際には、大量のデータと画面を部品化するなど効率的に実装するための枠組みが必要となる。我々は、J2EEの手法を参考に、フレームワークUJIを開発した。UJIでは、画面を表すJSPとロジックを実装するHandlerが分離され、コマンドマッピング、ページマッピングの仕組みを導入することにより、JavaBeanオブジェクトを介して疎に連携される。

我々はUJIフレームワークをServlet/JSP環境上に実装し、アプリケーション実装による評価を行なった。Servletによる業務アプリケーションからの書き換えにおいては、ロジックがより整理され簡潔になるとともに、総記述量では10%程度、ロジックとしてメンテナンスする部分は85%以上の削減が可能であった。性能では、JSPでの記述に比べて殆んど変化がないことが分かった。

今後は、専用のタグに対応した効率的な開発環境が必要である。

参考文献

[1] Microsoft, ASP Technology Feature Overview, http://msdn.microsoft.com/workshop/server/asp/aspfeat.asp

[2] Sun Microsystems, Java Servlet API Specification, http://java.sun.com/products/servlet/

[3] 松塚貴英, 上原三八: 業務処理アプリケーションにおけるフレームワークの設計, 情報処理学会, ウィンターワークショップ・イン・高知論文集, pp.15–16(1999).

[4] T. Matsutsuka, K. Nagahashi, Y. Nomura, H.Hara: An Architecture to Develop Presentation Logic for Enterprise Business Applications, Proceedings of the Evolve 2000 Conference, Feb. 2000.

[5] Sun Microsystems, The Swing Connection, http://java.sun.com/products/jfc/tsc/

[6] Chuck Sphar, Learn Microsoft Visual C++ 6.0 Now, Microsoft Press, 1999.

[7] Sun Microsystems, Java 2 Platform Standard Edition, http://java.sun.com/j2se/

[8] The Jakarta Project, Tomcat, http://jakarta.apache.org/

[9] Sun Microsystems, Java 2 Platform Enterprise Edition Blueprints, http://java.sun.com/j2ee/blueprints/

[10] Java Apache Project, The Server Pages Foundation Classes, http://java.apache.org/spfc/

[11] The Jakarta Project, Struts, http://jakarta.apache.org/struts/