Struts、オープン・ソースMVC実装

サーブレットおよびJSPフレームワークを使用した大規模Webサイトでの複雑な機能の管理

Malcolm Davis

2001年 2月 01日

この記事では、Strutsについて説明します。Strutsは、サーブレットとJavaServer Pages (JSP) テクノロジーを使用するModel-View-Controller (MVC) の実装です。Strutsは、Webプロジェクトの中での変更を制御したり、分業化を促進したりするのに役立ちます。Strutsを採用したシステムを実装しない場合でも、この記事のいくつかのアイデアは、今後のサーブレットやJSPページの実装において参考になることでしょう。

概要

最近では、小中学校の生徒でもインターネットにHTMLページを公開するようになってきました。しかし、学校の生徒のページとプロが開発するWebサイトとでは、雲泥の差があります。ページ・デザイナー (またはHTML開発者) は、色、顧客、製品フロー、ページ・レイアウト、ブラウザーの互換性、イメージの作成、JavaScript、など、さまざまな要素を考慮しなければなりません。見栄えの良いサイトをまとめるにはかなりの作業が必要になりますが、ほとんどのJava開発者は、ユーザー・インターフェースの作成よりも見栄えの良いオブジェクト・インターフェースを作成することのほうに関心があります。JavaServer Pages (JSP) テクノロジーは、ページ・デザイナーとJava開発者をつなぐものとなります。

大規模Webアプリケーションでの作業をしたことがあるなら、変更の語の意味はよく理解できることでしょう。Model-View-Controller (MVC) は、変更制御のための設計パターンです。MVCは、インターフェースを、ビジネス・ロジックおよびデータから分離します。Strutsは、実装の一部として、J2EE仕様からのServlets 2.2およびJSP 1.1タグを使用したMVC実装です。Strutsを使ってシステムを実装することはないかもしれませんが、Strutsについて調べるなら、今後のサーブレットおよびJSPの実装の参考になるでしょう。

この記事では、まず比較的親しみやすい要素を使用したJSPファイルの例を示し、そのようなページのメリットとデメリットについて説明します。次に、Strutsについて説明し、Strutsによってどのようにプロジェクトの中での変更が制御されたり分業化が促進されたりするかを示します。最後に、簡単なJSPファイルを、ページ・デザイナーと変更を考慮に入れて作り直します。

JSPファイルはJavaサーブレット

JavaServer Pages (JSP) ファイルは、サーブレットのもう1つの見方以上の何でもありません。JSPファイルの概念により、JavaサーブレットをHTMLページとしてみることができます。そのような見方により、Javaコードだとよく使われる見苦しい`print()`ステートメントを使う必要はまったくなくなります。JSPファイルは、プリプロセスにより`.java`ファイルに変換され、その後、コンパイルされて`.class`ファイルになります。Tomcatを使っているなら、プリプロセス後の`.java`ファイルは`work`ディレクトリーに入っており、それを見ることができます。コンテナーによっては、別の場所に`.java`ファイルと`.class`ファイルが格納されます。その場所はコンテナーによって違います。図1は、JSPファイルからサーブレットへの流れを示します。

図1. JSPファイルからサーブレットへの流れ

(Microsoft Active Serve Page (ASP) とはかなり違います。ASPはコンパイル後、別個のファイルではなくメモリーに入れられます。)

簡単な自己完結型JSPファイル

小規模なJSPアプリケーションでは、データ、ビジネス・ロジック、およびユーザー・インターフェースが1つのコード・モジュールにまとまっていることがよくあります。さらに、一般にアプリケーションには、アプリケーションのフローを制御するロジックが含まれています。リスト1と図2に示すのは、ユーザーがメーリング・リストに参加するための簡単なJSPファイルです。

リスト1.`join.jsp` -- 簡単な要求/応答JSPファイル

```
<%@ page language="java" %>
<%@ page import="business.util.Validation" %>
<%@ page import="business.db.MailingList" %>
<%
String error = "";
String email = request.getParameter("email");
// do we have an email address
if( email!=null ) {
    // validate input...
    if( business.util.Validation.isValidEmail(email) ) {
        // store input...
        try {
            business.db.MailingList.AddEmail(email);
        } catch (Exception e) {
            error = "Error adding email address to system.  " + e;
        }
        if( error.length()==0 ) {
%>
            // redirect to welcome page...
            <jsp:forward page="welcome.html"/>
<%
        }
    } else {
        // set error message and redisplay page
        error = email + " is not a valid email address, please try again.";
    }
} else {
    email = "";
}
```

```
%>
<html>
<head>
<title>Join Mailing List</title>
</head>
<body>
<font color=red><%=error%></font><br>
<h3>Enter your email to join the group</h3>
<form action="join.jsp" name="joinForm">
    <input name="email" id="email" value=<%=email%>></input> <input type=submit value="submit">
</form>
</body>
</html>
```

図2. 簡単な要求/応答の場合、このJSPファイルはデータを設定し、次のページへのフローを制御し、HTMLを作成する

このメーリング・リストJSPファイルは自己完結型であり、1つのモジュールですべてを実行します。このJSPファイルに含まれていないのは、検証のための実際のコード (`isValidEmail()` に含まれている) と、データベースに電子メール・アドレスを入れるコードです。(`isValidEmail()` メソッドを再使用可能コードに入れることは当然のように思えますが、私は`isValidEmail()` をページに直接入れた例も見てきました。) 単一ページのアプローチの利点は、わかりやすく、初めての場合に作成が簡単であるということです。さらに、グラフィック開発ツールがあるので、手軽に利用できます。

`join.jsp` の実行内容

1. 初期入力ページを表示する。
2. フォーム・パラメーターのうち`email` の値を読む。
3. `email` アドレスを検証する。
4. `email` のアドレスが有効なら、
    - そのアドレスをデータベースに追加する。
    - 次のページを表示する。
5. `email` のアドレスが無効なら、
    - エラー・メッセージを設定する。
    - もう一度`join.jsp` を表示し、エラー・メッセージを表示する。

単一ページ・アプローチの特徴

- HTMLとJavaの結合
  JSPファイルをコーディングする人は、ページ・デザイナーであり、かつJava開発者でなければなりません。その結果のJavaコードがひどいものになったり、見苦しいページになったり、場合によってはその両方になったりします。
- JavaとJavaScriptの混乱
  ページが大規模なものになってくると、JavaScriptを使うことになる場合がしばしばあります。ページにJavaScriptを使用する場合、そのスクリプトとJavaコードとが混乱の原因になることがあります。たとえば、クライアント側で`email` フィールドの検証のためにJavaScriptを使用する場合がそうです。

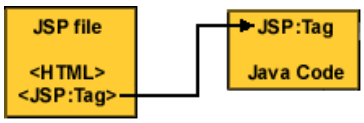
- 内部フロー・ロジック
  アプリケーションのフロー全体を理解するためには、すべてのページを見る必要があります。100ページからなるWebサイトがスパゲッティー・ロジックでできているところを想像してみてください。
- デバッグの難しさ
  見かけが悪いだけでなく、HTMLタグ、Javaコード、そしてJavaScriptコードがすべて1つのページに入れられているため、問題点がある場合にデバッグするのが困難です。
- 強い結合
  ビジネス・ロジックやデータを変更すると、それに関係するすべてのページを修正することが必要になります。
- 美学
  大規模なページの場合、このようなコーディング方法は見た目にも扱いにくくなります。私がMicrosoft ASP開発に携わっていた時、1000行にもなるページをよく見かけました。構文が色分けされているとしても、読みにくくわかりにくいことは確かです。

このHTMLにこれ以上Javaコードはいらない

リスト1では、Javaコードの中にたくさんのHTMLが含まれるかわりに、1つのHTMLファイルの中にたくさんのJavaコードが含まれています。この点では、ページ・デザイナーにJavaコードを生成してもらうという点を除き、あまり多くのことは達成されていません。しかし、すべてが失われるわけではありません。JSP 1.1には、タグと呼ばれる新しい機能があります。

JSPタグは、単にJSPファイルからコードをくくり出すための手段です。タグ・コードがサーブレットに含まれるということから、JSPタグをJSPファイルのマクロと考える人もいます。(マクロと見る観点も、ほとんど正しいと言えます。) 私としては、Javaコード中にHTMLのタグが出現してほしくないのと同じ理由で、JSPファイルの中にJavaコードが出現してほしくありません。JSPテクノロジーの優れた点は、ページ・デザイナーがJavaコードのことを気にすることなくサーブレットを生成できるということです。タグを使えば、Javaプログラマーは、JavaコードをHTMLのように扱うことによって、JSPファイルを拡張できます。図3は、JSPページからコードを取り出してJSPタグに入れるという、一般的な概念を示しています。

図3. JSPタグ



リスト2は、Strutsタグの機能を示す例です。リスト2では、HTMLの通常の`<form>` タグを、Strutsの`<form:form>` タグに置き換えています。リスト3は、ブラウザーが受け取ることになるHTMLです。ブラウザーはHTMLの`<form>` タグを受け取っていますが、さらにJavaScriptなどの付加的なコードも受け取ります。その付加的なJavaScriptでは、`email` というアドレス用フィールドにフォーカスを設定しています。サーバー側の`<form:form>` タグ・コードによって、適切なHTMLが作成され、ページ・デザイナーの手によることなくJavaScriptが抽出されます。

リスト2. Strutsのformタグ

```
<form:form action="join.do" focus="email" >
    <form:text   property="email" size="30" maxlength="30"/>
    <form:submit property="submit" value="Submit"/>
</form:form>
```

リスト3. ブラウザーに送られるHTML

```
<form name="joinForm" method="POST" action="join.do;jsessionid=ndj71hjo01">
    <input type="text" name="email" maxlength="30" size="30" value="">
    <input type="submit" name="submit" value="Submit">
</form>
<script language="JavaScript">




<!--
    document.joinForm.email.focus()
// -->
</script>
```

JSPタグに関する注意事項

- JSPタグには、JSP 1.1以降を実行するコンテナーが必要です。
- JSPタグはサーバー側で実行されます。HTMLタグのようにクライアント側で解釈されることはありません。
- JSPタグは、真の意味でコードの再利用の機能を提供します。
- HTMLとJavaScriptは、JSPの`include`のメカニズムを使うことによっても、ページに追加できます。しかし、開発者には巨大なJavaScriptライブラリー・ファイルを作成する傾向があり、それらのライブラリーがそのJSPファイルにインクルードされることになります。その結果、必要をはるかに超えるHTMLページがクライアントに戻されることになります。`include`は、ページのヘッダーやフッターなど、HTMLの断片のために使用するのが正しい用法です。
- JSPタグは、Javaコードを抽出することによって、開発の役割の分業化を促進してきました。

Model-View-Controller (MVC)

JSPタグで解決されるのは、問題の一部にすぎません。検証、フロー制御、およびアプリケーションの状態の更新の問題がまだ残っています。ここで、MVCが役に立ちます。MVCでは、問題が3つのカテゴリーに分割され、それにより単一モジュールのアプローチに伴ういくつかの問題点が解決されます。

- **モデル**
  モデルには、アプリケーションの中心となる機能が含まれています。モデルは、アプリケーションの状態をカプセル化します。そこに含まれる機能が状態だけの場合もあります。ビューまたはコントローラーについては、何も認識しません。
- **ビュー**
  ビューは、モデルの表現を提供します。それは、アプリケーションの外観です。ビューからモデルの取得ルーチン（getter）にアクセスすることは可能ですが、設定ルーチン（setter）については何も認識しません。さらに、コントローラーについても認識しません。モデルに変更があった場合は、必ずビューに通知される必要があります。
- **コントローラー** (Controller)
  コントローラーは、ユーザー入力に対して応答します。これは、モデルを作成し、設定します。

MVCモデル2

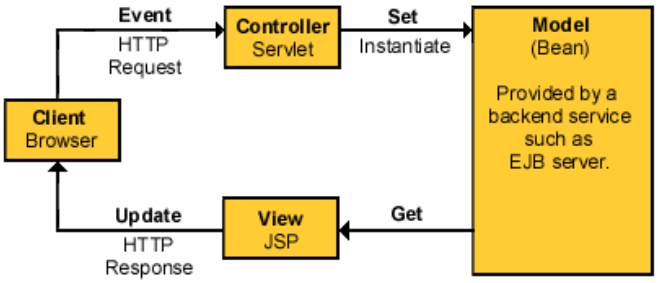
Webは、ソフトウェア開発者にとっていくつかの課題をもたらしました。特に、クライアントとサーバーの間のステートレスな接続ということがあります。このステートレスな動作のため、モデルがビューに変更を通知することは困難です。Webにおいては、アプリケーションの状態の変更を検出するために、ブラウザーがサーバーに再度問い合わせる必要があります。

もう1つの特筆すべき変更は、ビューの実装に、モデルまたはコントローラーとは違うテクノロジーが使用されているということです。もちろん、Javaコードを (またはPERL、C/C++ など何でも) 使用してHTMLを生成することは可能です。そのようなアプローチには、いくつかの欠点があります。

- JavaプログラマーはHTMLではなくサービスの開発をすべきです。
- レイアウトを変更するたびにコードを変更することが必要になります。
- サービスの利用者が、それぞれ特有の必要に従ってページを作成することが可能でなければなりません。
- ページ・デザイナーが、ページの開発に直接関係することができなくなります。
- コード中にHTMLを組み込むのは、見苦しい方法です。

Webについては、MVCの古典的な形式を変更する必要がありました。図4は、MVCをWebに応用したもの (MVCモデル2またはMVC 2) を示しています。

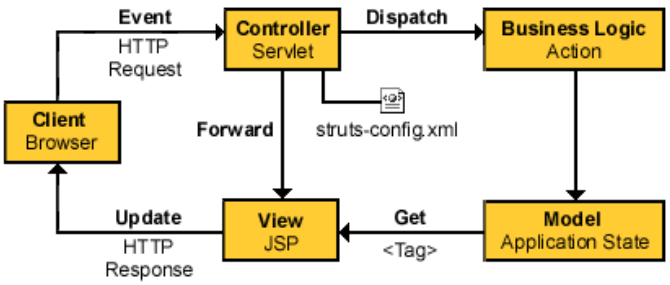
図4. MVCモデル2



Struts、MVC 2の実装

Strutsは、再使用可能なMVC 2設計を構成するクラス、サーブレット、およびJSPタグの集まりです。この定義は、Strutsがライブラリーではなくフレームワークであることを示唆しています。しかし、Strutsには、フレームワークからは独立した充実したタグ・ライブラリーやユーティリティー・クラスも含まれています。図5に、Strutsの概要を示します。

図5. Strutsの概要



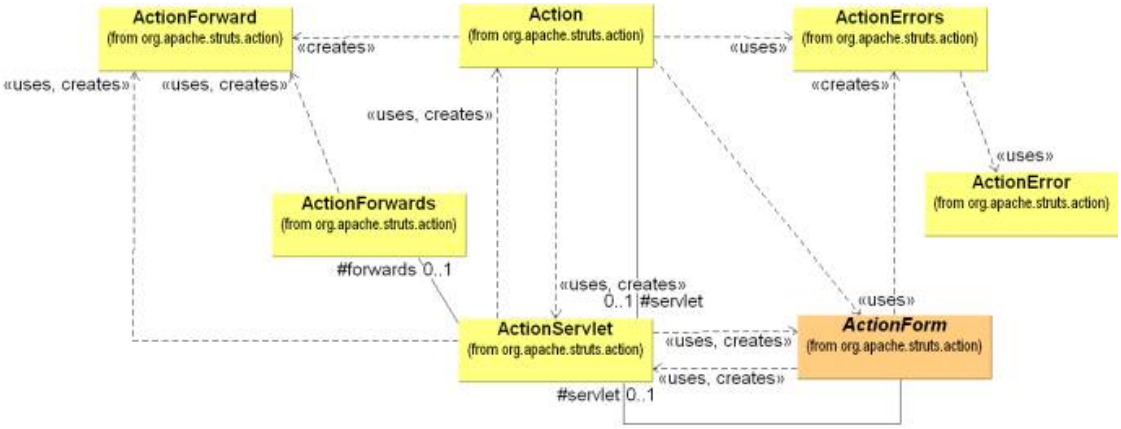
Strutsの概要

- **クライアント・ブラウザー**
  クライアント・ブラウザーからのHTTP要求により、イベントが作成されます。Webコンテナーは、HTTP応答を戻します。
- **コントローラー**
  コントローラーは、ブラウザーからの要求を受け取り、どこにその要求を送るべきかを決定します。Strutsの場合、コントローラーは、サーブレットとして実装されたコマンド・デザイン・パターンです。コントローラーの設定には`struts-config.xml` ファイルが使用されます。
- **ビジネス・ロジック**
  ビジネス・ロジックは、モデルの状態を更新し、アプリケーションのフロー制御を支援します。Strutsの場合、このことは、実際のビジネス・ロジックに対するラッパーとして`Action` クラスを使用してなされます。
- **モデル状態**
  モデルは、アプリケーションの状態を表しています。アプリケーションの状態は、ビジネス・オブジェクトによって更新されます。ActionForm beanは、永続的レベルではなく、セッション・レベルまたは要求レベルでのモデル状態を表しますJSPファイルは、JSPタグを使うことによって、ActionForm beanから情報を読みます。
- **ビュー**
  ビューは、単なるJSPファイルです。フロー・ロジック、ビジネス・ロジック、モデル情報は含まれず、タグが含まれるだけです。Velocityなどのフレームワークと比較した場合に、タグはStrutsを特徴付けるものの1つです。

Strutsの詳細

図6に示すのは、`org.apache.struts.action` パッケージのUMLダイアグラムの一部です。図6には、`ActionServlet` (コントローラー)、`ActionForm` (フォームの状態)、および`Action` (モデルのラッパー) の間の最小限の関係が示されています。

図6. コマンド (ActionServlet) とモデル (Action & ActionForm) の関係を示すUMLダイアグラム



`ActionServlet` クラス

関数マッピングを使用していた時代のことをご存じでしょうか? その場合には、入力イベントを関数へのポインターに対応付けていました。賢明な人であれば、構成情報をファイルに入れて、実

行時にファイルをロードするかもしれません。関数ポインターの配列は、過去のCの構造化プログラミングの古き良き時代の方法です。

現在では、Javaテクノロジー、XML、J2EEなど、さらに優れた技術があります。Strutsのコントローラーは、イベント (一般にはHTTPポスト) をクラスに対応付けるサーブレットです。どういうことになるか、考えてみてください。コントローラーは構成ファイルを使用するので、値をハードコーディングする必要はありません。時代は変わっても、基本的なことは同じです。

`ActionServlet` はMVC実装のコマンド部であり、フレームワークの中心となるものです。`ActionServlet` (コマンド) は、`Action`、`ActionForm`、および`ActionForward` を作成し使用します。前述のように、コマンドの構成には`struts-config.xml` ファイルが使用されます。Webプロジェクトの作成中、`Action` と`ActionForm` は、特定の問題スペースを解決するために拡張されます。`struts-config.xml` のファイルによって、拡張クラスの使用方法が`ActionServlet` に対して指示されます。このアプローチには、いくつかの利点があります。

- アプリケーションのロジックのフロー全体が階層構造のテキスト・ファイルになります。特に大規模なアプリケーションの場合入力は、ずっと簡単に表示できてわかりやすくなります。
- ページ・デザイナーは、Javaコードを気にすることなくアプリケーションのフローを理解できます。
- Java開発者は、フローが変更になってもコードを再コンパイルする必要がありません。

`ActionServlet` を拡張することによって、コマンドの機能を追加できます。

`ActionForm` クラス

`ActionForm` は、Webアプリケーションのセッションの状態を管理します。`ActionForm` は、各入力フォーム・モデルごとにサブクラスを作成するための抽象クラスです。「入力フォーム・モデル」という語を使用していますが、これは、HTMLフォームによって設定または更新されるデータの一般概念が`ActionForm` によって表されるということです。たとえば、HTMLフォームによって設定される`UserActionForm` があるとします。Strutsフレームワークでは、次のようになります。

- `UserActionForm` が存在するかどうかを調べ、存在しないなら、そのクラスのインスタンスを作成します。
- Strutsは、HttpServletRequestの対応するフィールドを使って`UserActionForm` の状態を設定します。あの`request.getParameter()` 呼び出しを使う必要はもうありません。たとえば、Strutsフレームワークは要求ストリームから`fname` を取り出して、`UserActionForm.setFname()` を呼び出します。
- Strutsフレームワークは、`UserActionForm` の状態を更新してから、ビジネス・ラッパー`UserAction` にそれを渡します。
- それを`Action` クラスに渡す前に、Strutsは、さらに`validation()` メソッドを`UserActionForm` に対して呼び出すことにより、フォーム状態の検証を実行します。
  注: これは、場合によっては賢明な方法とは言えないことがあります。他のページやビジネス・オブジェクトの中で`UserActionForm` を使うことがあり、その検証は異なっているかもしれません。状態の検証は、`UserAction` クラスでするほうがよいかもしれません。
- `UserActionForm` は、セッション・レベルで管理できます。

注:

- `struts-config.xml` ファイルは、HTMLフォーム要求をどの`ActionForm` に対応付けるかを制御します。
- `UserActionForm` は、複数の要求を対応付けられます。
- `UserActionForm` は、ウィザードなどにおいて複数のページにわたって対応付けられます。

`Action` クラス

`Action` クラスは、ビジネス・ロジックのラッパーです。`Action` クラスの目的は、`HttpServletRequest` をビジネス・ロジックに変換することです。`Action` を使うには、サブクラス化をし、`process()` メソッドを上書きします。

`ActionServlet` (コマンド) は、`perform()` メソッドを使用することによって、パラメーター化されたクラスを`ActionForm` に渡します。ここでも、`request.getParameter()` を呼び出す必要はもうありません。ここでイベントが発生する時点までに、入力フォーム・データ (またはHTMLフォーム・データ) は、すでに要求ストリームから取り出されて`ActionForm` クラスに変換されています。

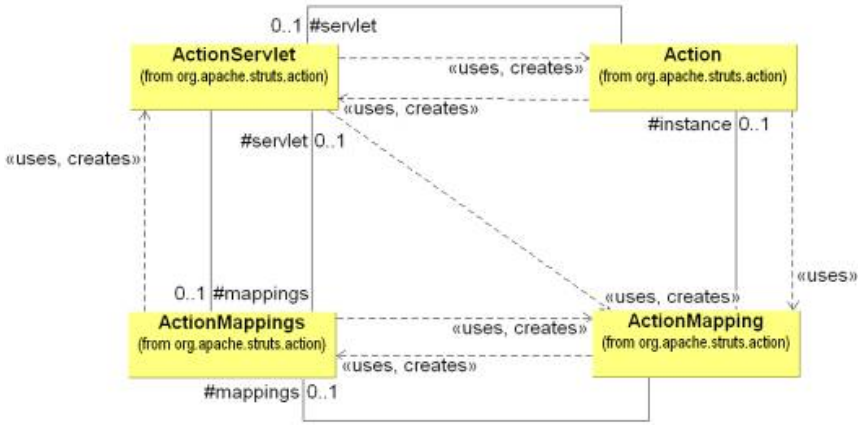
注:`Action` クラスを拡張する場合は、考えすぎないようにしてください。`Action` クラスで制御するのは、アプリケーションのロジックではなくアプリケーションのフローです。ビジネス・ロジックを別個のパッケージまたはEJBにすることにより、柔軟性と再利用の可能性が高くなります。

`Action` クラスに関する別の考え方は、アダプター (Adapter) デザイン・パターンとみなすことです。`Action` の目的は、クラスのインターフェースをクライアントが予期する別のインターフェースに変換することであり、インターフェースの非互換性のために動作しないクラスがうまく動作するようにすることです (Design Patterns - Elements of Reusable OO Software、Gof著、による)。このインスタンスのクライアントは、特定のビジネス・クラス・インターフェースについて何も知らない`ActionServlet` です。このようにして、Strutsは、それが認識するビジネス・インターフェース`Action` を提供します。`Action` を拡張することによって、ビジネス・インターフェースがStrutsのビジネス・インターフェースと互換性のあるものになります。(興味深いのは、`Action` はインターフェースではなくクラスである、という点です。`Action` はインターフェースとして出発しましたが、時間の経過と共にクラスに変更されました。完ぺきなものは何もありません。)

`Error` クラス

図6のUMLダイアグラムには、`ActionError` と`ActionErrors` も含まれています。`ActionError` は、個々のエラー・メッセージをカプセル化したものです。`ActionErrors` は、ビュー (View) がタグを使用してアクセスできる`ActionError` クラスのためのコンテナーです。`ActionError` は、Strutsでエラーのリストを記録するための手段となるものです。

図7. コマンド (ActionServlet) とモデル (Action) の関係を示すUMLダイアグラム



`ActionMapping` クラス

入力イベントは普通HTTP要求の形式であり、それはサーブレット・コンテナーにより`HttpServletRequest` に変換されます。コントローラーは入力イベントを見て、要求を`Action` クラスにディスパッチします。`struts-config.xml` により、コントローラーがどの`Action` クラスを呼び出すかが決定されます。`struts-config.xml` の構成情報は、一連の`ActionMapping` に変換され、それが`ActionMappings` のコンテナーに入れられます。(s で終わるクラスはコンテナーです。)

`ActionMapping` には、特定のイベントを特定の`Actions` にどうマッピングするかについての情報が含まれています。`ActionServlet` (コマンド) は、`perform()` メソッドを使うことによって、`ActionMapping` を`Action` クラスに渡します。これにより`Action` は、フロー制御のための情報を入手できます。

`ActionMappings`

`ActionMappings` は`ActionMapping` オブジェクトの集まりです。

メーリング・リストのサンプルの改訂版

`join.jsp` を悩ます問題がStrutsによって解決されるようすを見てみましょう。改訂版は2つのプロジェクトで構成されています。第1のプロジェクトは、Webアプリケーションとは独立したアプリケーションのビジネス・ロジック部を含むものです。その独立レイヤーは、EJBテクノロジーによって実装される共通サービス・レイヤーです。ここでは、Antビルド・プロセスを使用して、`business` というパッケージを作成することにします。独立したビジネス・レイヤーを採用する理由は、次のとおりです。

- **責任範囲の分離**
  パッケージを別々にすることにより、マネージャーは開発グループ内でさまざまな責任を他の人にゆだねることができます。これは、開発者の責任処理能力の向上に役立ちます。
- **既製品感覚**
  商用ソフトウェアの一部としてパッケージを扱う開発者のことを考えてみてください。それを別個のパッケージにすることにより、既製品感覚が促進されます。パッケージは既製品だったり、組織内の別のグループの開発したものだったりします。

- **不必要なビルドや単体テストをしなくてよい**
  ビルド・プロセスを分離することにより、不必要なビルドや単体テストの作業を避けることができます。
- **インターフェースを使用した開発**
  開発時にインターフェースの点から考えるのに役立ち、混乱を避けることができます。これは特に重要な点です。独自のビジネス・パッケージを作成する場合、ビジネス・クラスでは、Webアプリケーションが呼び出すのか、それとも独立したJavaアプリケーションが呼び出すのかを考慮してはいられません。したがって、ビジネス・レイヤーでは、サーブレットAPI呼び出しやStruts API呼び出しへの参照は避けてください。
- **安定性**
  毎日、毎週、あるいは毎月であっても、リリースするのは容易なことではありません。それで、開発においては、インターフェース・ポイントが安定的であることは重要な点です。ビジネス・パッケージのほうが変動しやすいからといって、Webプロジェクトも同様に変動するということにはしたくないわけです。

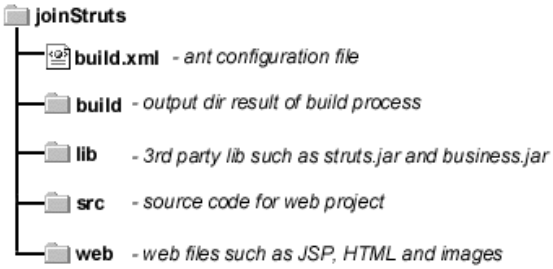
ビジネス・ビルドに関する注釈

私はプロジェクトのビルドにはAntを使い、単体テストにはJUnitを使います。business.zipには、ビジネス・プロジェクトのビルドに必要なあらゆるもの (AntとJUnitを除く) が含まれています。パッケージ・スクリプトは、クラスをビルドし、単体テストを実行し、Javaのdocsファイルとjarファイルを作成し、それらをzipファイルに圧縮して顧客に配布します。`build.xml` を修正すれば、他のプラットフォームにも配布できます。`Business.jar` はWebダウンロード部にあり、このビジネス・パッケージをダウンロードしてビルドする必要はありません。

Webプロジェクト

第2のプロジェクトは、Strutsで開発したWebアプリケーションです。JSP 1.1とServlet 2.2に準拠したコンテナーが必要です。(参考文献を参照)。私にはこれはこれまで大問題で、実際にあなたが使用しているStrutsダウンロードで、私のこのサンプルWebプロジェクトをビルドできるかどうかを保証することはできません。Strutsは絶えず変化しており、私はそれに合わせてプロジェクトを更新し続けてきました。このプロジェクトで使用したのは、jakarta-struts-20010105.zipです。図8に、このWebプロジェクトのレイアウトを示します。Antをインストールしたなら、ビルドを実行することにより、`joinStruts.war` という配布用のwarファイルが作成されます。

図8. Webプロジェクトのレイアウト



リスト4は、変換後のJSPファイル`joinMVC.jsp` です。このファイルには、最初50行のJavaコードが含まれていましたが、その後19行になり、今ではまったく含まれていません。ページ・デザイナーからすると、これは大きな改善点です。

リスト4.`joinMVC.jsp` -- 簡単なJSP改訂版

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts.tld" prefix="struts" %>
<%@ taglib uri="/WEB-INF/struts-form.tld" prefix="form" %>
<html>
<head>
<title><struts:message key="join.title"/></title>
</head>
<body bgcolor="white">
<form:errors/>
<h3>Enter your email to join the group</h3>
<form:form action="join.do" focus="email" >
    <form:text   property="email" size="30" maxlength="30"/>
    <form:submit property="submit" value="Submit"/>
</form:form>
</body>
</html>
```

ページの変更内容

Strutsのタグ・ライブラリーを使用したことによる変更は、下記のとおりです。

- インポート
  `<%@ taglib uri="/WEB-INF/struts.tld" prefix="struts"%>`
  Javaの`<%@page import_>`は、Strutsタグ・ライブラリーの`<%@ taglib uri_>`で置き換えられました。
- 本文
  `<struts:message key="join.title"/>`
  リソース・プロパティー・ファイルには、`join.title`のためのテキストが含まれています。この例では、ApplicationResourcesプロパティー・ファイルに、名前と値のペアが含まれています。これにより、文字列のレビューや国際化のための変更が容易になります。
- エラー
  `<form:errors/>`
  `ActionServlet`または`ActionForm`により、表示のためのエラー・メッセージが作成されます。それらのエラー・メッセージは、プロパティー・ファイルにも含めることができます。また、ApplicationResourcesには、`error.header`および`error.footer`を設定することによってエラーの書式設定をする手段も用意されています。
- HTMLフォーム
  `<form:form action="join.do" focus="email" >`
  - HTMLの`<form>`タグおよび属性は、JSPの`<form>`タグおよび属性で置き換えられています。`<form action="join.jsp" name="join">`は、`<form:form action="join.do" focus="email" >`に変更されました。
  - HTMLの`<input>`タグは、`<form:text/>`で置き換えられました。
  - HTMLの`<submit>`タグは、`<form:submit/>`で置き換えられました。

モデル -- セッションの状態

`JoinForm`は、`ActionForm`をサブクラス化し、フォーム・データを含んでいます。この例のフォーム・データは、単なる電子メール・アドレスです。フレームワークが電子メール・アドレスにアクセスできるようにするため、その設定ルーチンと取得ルーチンを追加しました。この例で

は、validate() メソッドを書き換えて、Strutsのエラー・トラッキング機能を使用するようにしました。StrutsによりJoinForm が作成され、状態情報が設定されます。

モデル -- ビジネス・ロジック

前述のようにAction は、コントローラーと実際のビジネス・オブジェクトの間のインターフェースです。JoinAction は、元々join.jsp に含まれていたbusiness.jar の呼び出しのラッパーです。リスト5に、JoinAction のperform() メソッドを示します。

リスト5. - JoinAction.perform()

```
 public ActionForward perform(ActionMapping mapping,
                              ActionForm form,
                              HttpServletRequest request,
                              HttpServletResponse response)
                              throws IOException, ServletException {
    // Extract attributes and parameters we will need
    JoinForm joinForm = (JoinForm) form;
    String email = joinForm.getEmail();
    ActionErrors errors = new ActionErrors();
    // store input....
    try {
        business.db.MailingList.AddEmail(email);
    } catch (Exception e) {
        // log, print stack
        // display error back to user
        errors.add("email",new ActionError("error.mailing.db.add"));
    }
    // If any messages is required, save the specified error messages keys
    // into the HTTP request for use by the <struts:errors> tag.
    if (!errors.empty()) {
        saveErrors(request, errors);
        // return to the original form
        return (new ActionForward(mapping.getInput()));
    }
    // Forward control to the specified 'success' URI that is in the Action.xml
    return (mapping.findForward("success"));
}
```

注:perform() は、コントローラーが次にどうするべきかを指示するActionForward クラスを戻します。この例では、コントローラーから渡されるマッピングを使用して、次の処理を決定しています。

コントローラー

JSPファイルを修正し、2つの新しいクラスを作成しました。1つはフォーム・データを入れるためのもの、もう1つはビジネス・パッケージを呼び出すためのものです。最後に、それを含むすべての変更を構成ファイルstruts-config.xml に反映させました。リスト6に、joinMVC.jsp のフロー制御のために追加したaction要素を示します。

リスト6. actionの構成

```
<action  path="/join" name="joinForm"
         type="web.mailinglist.JoinAction"
        scope="request"
        input="/joinMVC.jsp"
     validate="true">
    <forward  name="success"  path="/welcome.html"/>
</action>
```

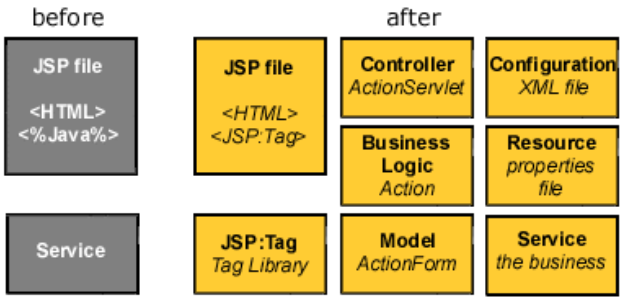
action要素は、要求パスから、要求を処理するために使う対応する`Action`クラスへのマッピングを記述します。各要求タイプには、要求の処理方法を記述するaction要素が対応していなければなりません。join要求において、

1. フォーム・データを入れておくのに`joinForm`を使用します。
2. 検証 (validate) が真 (true) になっているため、`joinForm`はそれ自体を検証します。
3. `web.mailinglist.JoinAction`は、このマッピングのための要求を処理するのに使用するactionクラスです。
4. すべて正常なら、要求が`welcome.jsp`に送られます。
5. ビジネス・ロジックの障害が検出されたなら、フローは、要求元のページ`joinMVC.jsp`に戻ります。これはどうしてでしょうか? リスト6のaction要素は、値が`"/joinMVC.jsp"`の属性`input`です。リスト5に示した私の`JoinAction.perform()`では、ビジネス・ロジックがエラーになると、`perform()`から、`mapping.getInput()`をパラメーターとする`ActionForward`が戻されます。このインスタンスの`getInput()`は`"/joinMVC.jsp"`です。ビジネス・ロジックがエラーになると、要求元のページである`joinMVC.jsp`が戻されます。

Strutsの前と後

図9からわかるように、かなり複雑になり、多くのレイヤーが追加されました。JSPファイルからサービス・レイヤーへの直接呼び出しは、もうありません。

図9. Strutsの前と後



Strutsのメリット

- **JSPタグ・メカニズムの使用**
  タグ機能は、再使用可能コードを促進し、JSPファイルからJavaコードを抽出します。この機能により、タグによるオーサリングを可能にするJSPベースの開発ツールにうまく統合できます。
- **タグ・ライブラリー**
  あるものは使いましょう。ライブラリーの中に必要なものが見つからないなら、あなたが貢献者になりませんか。さらに、StrutsはJSPタグ・テクノロジーを学習するための出発点を提供します。
- **オープン・ソース**
  オープン・ソースであることには、いろいろなメリットがあります。たとえば、コードを見ることができるので、ライブラリーを使用するあらゆる人にコードをレビューしてもらう機会があります。コードを見る目が多ければ、それだけいいレビューができます。

- MVC実装のサンプル
  Strutsは、独自のMVC実装を作成するための参考になります。
- 問題スペースの管理
  分割してからやっつけるという方法は、問題解決のための優れた手法であり、問題を管理しやすくするのに役立ちます。もちろん、もろ刃の剣でもあります。問題の複雑さが大きくなり、管理が余分に必要になります。

Strutsのデメリット

- 日が浅い
  Strutsの開発は、まだ準備段階です。バージョン1.0のリリースに向けて作業が進行中です。しかし、どんな製品のバージョン1.0でもそうですが、機能がすべて提供されるわけではありません。
- 変更
  フレームワークは、急速に変化しています。Strutsの0.5から1.0の間に、かなりの変更がありました。使用すべきでないメソッドを使用することがないようにするためには、Strutsのnightly最新版を常にダウンロードするのがよいでしょう。この6か月間にStrutsのライブラリーのサイズは90Kから270Kになりました。Strutsが変更されるため、私のサンプルは何度も変更しなければならず、またStrutsのバージョンによっては、それらが動作するかどうか保証できません。
- 抽象化の正しいレベル
  Strutsの抽象化のレベルは正しいでしょうか? ページ・デザイナーにとって、抽象化のレベルとしてどの程度が適切でしょうか? これは $64Kの問題です。ページ・デザイナーがページを開発する際に、Javaコードにアクセスできるようにするべきでしょうか? Velocityなど、いくつかのフレームワークではそうなっておらず、Web開発のためにさらに別の言語が提供されています。UI開発においてJavaコードへのアクセスを制限することには、それなりの根拠があります。最も重要な点として、ページ・デザイナーにJavaを少しだけ提供すると、ページ・デザイナーはJavaをむやみに使うようになることでしょう。私は、Microsoft ASP開発に携わっている時に、いつもこのことを経験していました。ASP開発では、COMオブジェクトをいくつか作成してから、それをまとめるためのASPスクリプトを少しだけ書くことが想定されています。しかし、実際のASP開発者は、ASPスクリプトをむやみやたらと使ってしまいがちです。「VBScriptで直接プログラミングできるのに、COM開発者がCOMで開発するのをどうして待つ必要があるの?」という声をよく聞きました。Strutsを採用すれば、タグ・ライブラリーによって、JSPファイルに必要なJavaコードの量を制限できます。そのようなライブラリーの1つはLogic Tagであり、これは、出力の条件付き生成を管理します。しかし、これはUI開発者がやたらにJavaコードを書きまくるのを防ぐものではありません。どんなフレームワークを使用するとしても、フレームワークの配布先となる環境、またフレームワークを管理することになる環境をよく理解しておく必要があります。もちろん、この作業は「言うは易く行うは難し」です。
- 限定された範囲
  Strutsは、HTML、JSPファイル、およびサーブレットによって実装することが意図されたWebベースのMVCソリューションです。
- J2EEアプリケーション・サポート
  Strutsには、JSP 1.1とServlet 2.2の仕様をサポートするサーブレット・コンテナーが必要です。これだけでは、Tomcat 3.2を使用するのでない限り、インストールに関する問題すべてが解決されるわけではありません。私の場合は、Netscape iPlanet 6.0 (J2EE準拠のアプリケー

ション・サーバーとされる最初のもの) でライブラリーをインストールしようとして、いろいろと問題がありました。問題がある場合は、Strutsのメーリング・リスト・アーカイブ (参考文献を参照) をご覧になることをお勧めします。

- **複雑さ**
  問題をいくつかの部分に分離することは、それだけ複雑さを増し加えることでもあります。Strutsを理解するためには、ある程度の教育が必要であることは明らかです。絶えず変更が加えられているため、このことが頭痛の原因となることがよくあります。そのような場合、ぜひWebにアクセスしてみてください。
- **どこに ...**
  これ以外にも、いくつかの点を指摘できるかもしれません (クライアント側の検証、適応可能なワークフロー、およびコントローラーの動的ストラテジー・パターンはどこにあるのか、など)。しかし、現時点で批判すべき点があるのは当然であり、問題点のいくつかはささいなものです。リリース1.0としては妥当なところと言えるでしょう。Strutsのチームの作業状況からすれば、この記事をお読みになるころ、あるいはそれ以降の早いうちに、Strutsのそれらの機能は整っていることでしょう。

Strutsの将来

ソフトウェア開発の新しい時代を迎えて、事態は急展開しつつあります。この5年未満のうちに、cgi/perlからISAPI/NSAPIへ、またVBによるASPへ、そして今やJavaとJ2EEへと発展してきました。SunはJSP/サーブレット・アーキテクチャーの変化に合わせるために懸命であり、それはかつてJava言語とAPIでしたのと同様です。新しいJSP 1.2およびServlet 2.3の仕様のドラフトがSunのWebサイトから入手できます。さらに、JSPファイルの標準タグ・ライブラリーも登場しています。それらの仕様やタグ・ライブラリーについては、参考文献をご覧ください。

最後に

Strutsは、タグとMVCを使用することによって、いくつかの大きな問題を解決しました。このアプローチは、コードの再利用可能性と柔軟性を高めるのに役立ってきました。問題をいくつかの小さな部分に分割することによって、テクノロジーや問題スペースに変化が発生した場合に再利用の可能性が高くなります。さらに、Strutsのおかげで、ページ・デザイナーとJava開発者は、それぞれのベストを発揮することに注意を集中できようになりました。しかし、頑丈な構造にすればするほど複雑になるというトレードオフがあります。Strutsは、簡単な単一JSPページに比べればずっと複雑ですが、大規模なシステムの場合、Strutsは複雑さに対応するために大いに役立ちます。また、私は自分でMVC実装を作成しようとは思いません。ただ、それを知りたいと思っているだけです。Strutsを使うかどうかに関係なく、Strutsのフレームワーク (ライブラリーのことですが) について調べるなら、次のWebプロジェクトのためのJSPファイルやサーブレットの機能、そしてそれらをどう組み合わせるかをよりよく理解できることでしょう。struts (支柱) は翼の構造の不可欠な部分であるのと同じように、Strutsは、次のWebプロジェクトにとって欠くことのできない重要な部分となる可能性を秘めています。

関連トピック

- Strutsのドキュメンテーション、インストールについての注意事項、そしてダウンロードについては、Strutsのホーム・ページをご覧ください。
- 最新のStruts実装は、Jakartaプロジェクトのページからダウンロードできます。
- 初めての方は、参照用JSPサーブレット実装 (最新リリースは、ここを参照) をダウンロードすることをお勧めします。Tomcat以外のコンテナーを使う場合に、特定の環境でStrutsをインストールする方法については、Strutsユーザー・メーリング・リストをご覧ください。
- ドキュメンテーションやインストール・ガイドで説明されていない付加的な情報については、Strutsのユーザー・メーリング・リストのアーカイブをご覧ください。
- ビルド・プロセスとテスト・プロセスを組み合わせることについては、「 AntとJUnitを用いた漸進的開発」をご覧ください。
- Java Servlet 2.3 and JavaServer Pages 1.2 Specifications (Proposed Final Draft 2) もご覧ください。
- JSR-52 A Standard Tag Library for JavaServer Pages もご覧ください。