SpringWebFlow による名大 ID ユーザ用プログラム作成

○堤守政

共通基盤技術支援室　情報通信技術系

概要

名古屋大学情報連携統括本部は、本学の構成員に対し名大 ID による情報サービス基盤を提供している。本稿は、これらのサービスプログラムの一つを改訂した報告である。開発環境としては Eclipse 上に Maven プラグインを導入し、Java アプリケーションフレームワークとして SpringWebFlow を利用した。また、実行サーバを構築しプログラム配備を行った。

本論文では、これらの開発環境、Spring フレームワークの概要、プログラム開発、サーバ環境構築と実装について紹介する。

1　はじめに

近年、特に Java のプログラミングは、統合開発環境 Eclipse を導入し、ビルドツールとして Ant や Maven を利用することが一般的である。また、Struts や Spring そして Seasar2 フレームワーク等を用いたプログラミングも雑誌等では薦められている。筆者は今回、Spring のサブプロジェクトで開発された SpringWebFlow フレームワークを使ったプログラムを、Eclipse を使って改訂・機能追加した。そして新規構築したサーバ上に配備した。2011 年 11 月 1 日からは、改訂版プログラムを用いて正式にユーザサービスを提供している。この経験から、技術的に参考になると考えられる部分を述べる。

2　開発環境

プログラム開発とテストは、ノート PC（FMV Lifebook Windows Vista Business）上で行った。ソフトウェア開発環境は以下の通りである。

jdk1.6.0_25(http://java.com/ja/download/windows_ie.jsp?locale=ja)

eclipse3.6 Helios Pleiades All in One(http://mergedoc.sourceforge.jp/)

m2eclipse -Maven Integration for Eclipse 0.12.1.20110112-1712 (http://m2eclipse.sonatype.org/sites/m2e)

apache-tomcat-6.0.33（http://tomcat.apache.org/download-60.cgi）

maven2.2.1(http://maven.apache.org/download.html)

2.1　Eclipse

Eclipse はオープンソースのプログラム統合開発環境である。Java 向けの場合は、プログラムコードの入力補助、インポート文の自動挿入、コンパイルエラーチェック、デバッカー、実行、プラグインの検索と導入等々の機能を有する。今回は Web 上でのユーザサービスプログラム開発を目的にしたため、Tomcat 起動制御及び 2.3 で示す M2eclipse をプラグインしたものを使用した。

2.2　Maven

Maven はオープンソースのソフトウェアプロジェクト管理ツールである。プログラムのコンパイル、テスト、ビルド、デプロイなどの作業を自動化することができる。Ant が、あらかじめ XML ファイルに記述され

た処理手順に従って動作するのに対し、Maven はソースプログラムや設定ファイルを既定のディレクトリ構成に配置することで、標準的なビルドが行われるという方式である。さらに Maven はソフトウェアが依存するライブラリをネットワーク経由で再帰的に自動取得する。

2.3 M2eclipse

M2eclipse は Eclipse 環境で Maven を利用するプラグインである。これを導入することによって、Eclipse 上で以下の機能が使用できる。

1. Java アプリケーションに応じた Maven のプロジェクトタイプを選択し、新規プロジェクトの雛型を作成する。
2. ライブラリは、Maven リポジトリから検索でき、一覧画面から選択した jar ファイルの名前とバージョンは、当該プロジェクトの pom.xml ファイルに自動設定される。
3. pom.xml に設定された外部 jar ファイルは、Maven がネット上から自動ダウンロードし、ビルドパスに加える。
4. プログラマは、対象プロジェクトに対し Maven のゴールを適宜選択して実行する。

3 Spring フレームワークの概要

2002 年 10 月に Rod Johnson 氏が出版した「Expert One-on-One J2EE Design and Development」に掲載されたコードを元にした Java のオープンソースアプリケーションフレームワークである。たとえば、あるクラスで必要になるオブジェクトを、そのクラス内ではなく、設定ファイルを介して組み込む「依存性の注入」DI(Dependency Injection)や、ログ出力など複数の場所で共通する処理を切り離して、特定のポイントを指定して実行させる AOP（Aspect Oriented Programming)などの特徴がある。

Spring フレームワークは Java アプリケーション全般の機能を有するが、筆者が取り組んだのは Web に関する SpringMVC フレームワークや SpringWebFlow の一部分である。

3.1 SpringMVC

Web のプログラムにおいて MVC というのは、データベースなどを介して処理の主機能を担う Model と、結果を表示する View、その間を制御する Controller とで作業を分離しプログラムを整理する手法である。

SpringMVC フレームワークの場合は、一般的な MVC パターンによる処理構造があらかじめ準備されていると考えればよい。たとえば SimpleFormController を継承したクラスインスタンスの動きをみると、HTTP/GET で呼ばれた場合と HTTP/POST で呼ばれた場合とで、その振る舞いが異なる。前者は、ユーザに入力を促す画面を表示することを想定し、後者はユーザから受け取った内容で何らかの処理を行い、結果を表示することを想定している。プログラマはその構成を理解して、必要な処理を適切な場所に組み込んでいく作業を行うことになる。

一方 Web アプリケーションについて、ユーザがブラウザから指定する URL については、フレームワークが中間経路を介してその記述内容を変更する。たとえば、URL 記述の末尾が xxx.html となっているとき、実際に呼び出すモジュールは xxx.jsp にできる。このように、プログラムの実装内容をネットワークから隠蔽することでセキュリティを向上させる工夫がある。

3.2 SpringWebFlow

Spring のサブプロジェクトで開発されている、MVC でいうところの Controller 部分を担当するフレームワークである。Web アプリケーションの画面遷移や使用する Java Beans プログラムを XML ファイルに定義することで、様々な制御内容を整理して記述することが可能である。このフロー定義は、状態とイベントの組

によって実行すべき処理を決定する。さらには、イベントを実行する前後に組み込む処理を指定することも可能である。フロー定義 XML ファイルの一例を、図３－１に示す。

```
<start-state idref="startIn"/>
<view-state id="startIn" view="LoginView">
        <render-actions>
                <action bean=”formAction” method=”setupForm”/>
        </render-actions>
        <transition on="login" to="menu">
                <action bean=”formAction” method=”bindAndValidate”/>
        </transition>
</view-state>
</view-state id=”menu” view=”menuForm”>
        :
        :
<end-state id="success" view="successForm"/>
```

図３－１　フロー定義 XML ファイルの一例

Web アプリケーションにおいて、たとえば元の画面に戻るなどの画面遷移を制御するときの状態管理は、一般に煩雑となる。またユーザがアプリケーションのボタンではなく、ブラウザの戻るボタンをクリックした場合も制御が難しい。SpringWebFlow は、このような状況でも Continuation flow execution repository と Flow Execution キーを使うことによって、正しい動作を保証している。前者はユーザ個々の要求と結果をスナップショットとして記録しておくもので、後者はそのうちどれを利用するのかを特定するためのキーとなる。

3.3　DI（Dependency Injection）

DI は AOP と共に Spring の中でも最も中心的な特徴とされている。日本語では「依存性の注入」と訳す。簡単には、プログラムの中で必要な変数値やクラスインスタンスを、外部ファイルから取り込む仕組みであると思ってよい。プログラムからは直接に変数値やクラス名を指定せず、間接的に使用することで、両者の結合度を弱める目的がある。その間の橋渡しの仕事は、Spring フレームワークが受け持つ。

ここでは、変数値を XML ファイルから注入する DI の例を示す。

クラス名 classA の中に、変数の宣言とその setter メソッドを書いておく。一方 beans.xml ファイルの中で、

<bean id=”classA” class=”プログラムのパス”>

<property name=”変数名１” value=”値１”/>

<property name=”変数名２” value=”値２”/>

の行を並べておく。この状態で classA がインスタンス化されると、Spring フレームワークの DI 機能は、setter メソッドによって beans.xml ファイルに書かれた値を classA インスタンスの変数に自動設定する。

DI は、変数のみではなく、クラスを注入することも可能である。クラスのインスタンス化も Spring に依頼できる。本来は、この機能を Java のインタフェースと組み合わせ、クラス間を疎結合にする目的で使用する。これにより、呼び出し元クラスは、インタフェースを実装したクラスのメソッドが使えること以外は関知しない状態にできる。したがって、実装クラスを入れ替えたい場合は、XML 設定ファイルの書き換えのみでよく、呼出し元のソースプログラムは変更する必要がない。

一方、もしフレームワークをさけて、自分でクラス間の疎結合を実現したい場合は、GoF デザインパターンの中で「Factory Method パターン」を使用すれば、同様の機能が実現できるはずである。ポイントは、インタフェースを介して、“実装クラスをインスタンス化する”メソッドを準備しておき、メイン側からこのメソ

ッドを呼んでやると、間接的に実装クラスがインスタンス化される。つまり、メイン側からは実装クラス名を知らなくてよい状態がつくられる。

4　プログラム開発

名大 ID ユーザ用プログラムを改訂したときのプログラミングで特徴的な処理、または工夫したところについて説明する。

4.1　LDAP サーバへのアクセス

名大 ID のエントリ情報は LDAP サーバ上に格納されているため、ユーザアプリケーションは LDAP サーバへのアクセスが必要である。LDAP サーバへのアクセスライブラリは novell 社提供の jldap4.3 を利用した。ここで必要になる値は、Spring の DI を利用して注入している。

ユーザエントリへのアクセス内容は、コマンドを例にすると ldapsearch, ldapadd, ldapmodify の機能を使用する。事前に行った接続テストの結果、エントリ内の single value の属性と属性値の組であれば、それが存在するしないにかかわらず、Replace モードで更新が可能であることが分かった。しかし属性が「objectClass」の場合は、設定したい属性値が既にエントリに存在するときに、同じ属性値を Add または Replace しようとするとエラーになる。結論としては、事前に当該 objectClass 値の存在検査を行い、なかった場合のみ Add モードで挿入するようにプログラミングした。

4.2　オブジェクトの受け渡し

別々のインスタンス間で変数やオブジェクトを受け渡したいことはよくある。2,3 個の変数であれば、メソッドを呼び出すときの引数に指定してもよいが、多くの変数内容を受け渡ししたいときは複雑になり分かりにくい。今回のプログラムの場合は、たとえば、LDAP サーバへのアクセスに必要なパラメータを一つのインスタンスに格納することで対応した。変数内容は XML ファイルに設定しておき、DI 機能を使ってインスタンスに取り込む。それらの内容が必要なメソッドは、当該変数を格納したクラスのインスタンスが持つ getter メソッドを使って、個々の変数を取得することができる。一方、メソッドの戻り値を配列にしたい場合は、LinkedList クラスを使用した。

4.3　エラー制御

多数のユーザが利用するプログラムを作成するときに気を使うことの一つに、エラー処理がある。ユーザの入力文字数制限など、主要なものはあらかじめメッセージとして用意できる。しかし個々のメソッドを呼び出したときの例外を、すべてユーザに表示してしまえば混乱する可能性がある。この場合、呼び出されたメソッドがその内部で起きた例外を catch したときは、もう一度自分の例外を throw してやる。これは、一段上の呼び出し側で catch できるので、ユーザにはあらためて適切なメッセージを表示すればよい。もちろん管理者にとっては、何が起きたかが把握できるように、内部での log 出力も必要であろう。

4.4　log 出力

デバッグや運用時において、プログラムの実行ログがあると便利なことがある。これは Log4j 機能を用いた。プログラムの途中に、デバッグ時にのみ出力したい内容は log.debug の行を使い、通常の実行時に出力したいメッセージは log.info 行を書いておく。info や debug 出力レベルは、適宜 log4j.properties ファイルの中で指定すれば変更可能である。Spring の場合は、ここに AOP を適用することも考えられる。

4.5　国際化対応

Windows の英語 OS パソコン利用者に対する国際化対応については、今回のバージョンでも設定したつも

りではあった。しかしサービス開始後にテストしたところ、日本語だけが表示されることが分かった。現在のところ、手元に英語 OS がないこともあって、国際化対応のデバッグはできていない。今後の課題としたい。

5　サーバ環境構築と実装

作成したプログラムでユーザサービスを行うためには、プログラムを配備するサーバマシンと Web アプリケーションを動作させるためのソフトウェア環境が必要である。ここでは、そのためのハードウェアの準備とプログラム実行環境について紹介する。

5.1　実装環境の構築

サーバは、このアプリケーションの実行だけに特化するために、単体マシンを使うことにした。ハードウェアとして Sun Fire V210 を使用し、OS は Solaris10 をインストールした。ファイルシステムは、72GB のディスクを 2 台使用しソフトウェア RAID1 を構築した。これを前提に、一方のディスクが故障しても、正常なディスク側からブートするように設定した。

今回作成したWebアプリケーションはApache2+Tomcatの環境で動作させる。そのために必要なソフトウェアはhttp://www.sunfreeware.com/とhttp://tomcat.apache.org/からダウンロードしインストールした。ソフトウェアApache2,Tomcatの動作は、SolarisのSMF (Service Management Facility)により管理している。両者のSMF用マニフェストファイルはWeb検索で入手し、システムに合わせて修正したものを使用している。

ユーザサービス可能状態のシステムは ufsdump によって SDLT 装置のテープデバイスにバックアップした。

5.2　サーバ証明書の取得と設定

これまでサーバ証明書は、国立情報学研究所（NII）の「UPKI オープンドメイン証明書自動発行検証プロジェクト」を利用して取得していた。今回はその更新時期であったため、次の手順でサーバ証明書を継続取得し、新サーバに設定した。

1．openssl コマンドを使って秘密鍵を作成
2．同じく証明書署名リクエスト(csr)の生成
3．https://app.icts.nagoya-u.ac.jp/csi_server_cert/upki-server.htmlを参照して、サーバ証明書の発行申請
4．https://app.icts.nagoya-u.ac.jp/csi_server_cert/apache+mod_ssl.htmlを参照して、秘密鍵、受け取ったサーバ証明書、そしてNIIからダウンロードした中間CA証明書の各ファイルパスを使って、Apache2 ディレクトリの中でのsslのコンフィグレーション

5.3　プロジェクト名の変更

今回開発したプログラムは、これまでユーザサービスが行われてきたものと区別するため、当初は別のプロジェクト名にしていた。しかし、既にリンク設定されている URL を変更することの煩雑さを考慮し、eclipse 上でプロジェクト名を変換することにした。実際には、プロジェクト名を右クリック→リファクタリング→名前変更のところで更新後、Tomcat 設定、pom.xml、他のファイルについても検索しながら置換した。そして M2eclipse を使い Maven package clean の後で Maven package を行ったところ、呼び出すプログラム名を元と一致させることができた。

5.4　プログラム配備

M2eclipse では、プログラムの実行時に Maven package を選択すると、target ディレクトリ下に war ファイルが作成される。このファイルは SCP などを使って実装するサーバにバイナリ転送し、Tomcat の webapps

ディレクトリに置けば、自動的にプロジェクト名ディレクトリが作成されてファイル展開される。そこでTomcat,Apache2 の再起動を行えば、当該プログラムが動作する。必要があれば、Maven の機能を使ってサーバへのデプロイも自動化可能である。

6　おわりに

以上、現在名大構成員にサービス中のユーザ用プログラムの作成と実装について紹介した。安全上、プログラム内容の詳細を述べることはできなかったが、Spring フレームワークの DI については、本質的な説明を行ったつもりである。

ところで、DI や SpringWebFlow の考え方は合理的で良くできている。しかし、このフレームワークを使って実際にプログラミングするのは、筆者にとってはかなり難しく時間が掛る作業であった。一つの原因としては、フレームワークがブラックボックスとなっていることである。プログラム処理の流れが、どのファイルのどの設定内容を使ってどのような順で動作するのか、ということを正確に理解していないとプログラムが書けない。実際、複数のファイルを跨いで処理があちこちに飛び回る。また XML ファイルについても、タグの意味や設定内容の種類と約束事が多く、全体を理解するのが困難である。今後、もっと自然で理解しやすいフレームワークなり言語処理系が出てくることを期待したい。

最後に、日頃、何かとお世話になっている情報連携統括本部、情報基盤センター及び情報通信技術系の同僚諸氏に感謝する。

参考文献

[1] “日本語 Eclipse / Pleiades All in One 日本語ディストリビューション”,(http://mergedoc.sourceforge.jp/)

[2] “Java と XML のマニアな解説：Maven2 Eclipse 編”, (http://www.jxpath.com/maven2/index.html)

[3] “JavaとXMLのマニアな解説：Spring Web編”, (http://www.jxpath.com/springWeb/index.html)

[4] “Spring Web Flow解説（１）～（１２）, 番外編”, (http://www.e-depot.co.jp/)

[5] “Spring Web Flow: A Practical Introduction ”, (http://www.ervacon.com/products/swf/intro/index.html)

[6] “spring source community”,(http://www.springsource.org/)

[7] 阪田浩一, “Spring による Web アプリケーションスーパーサンプル第２版”, ソフトバンククリエイティブ,2010 年 11 月

[8] 結城浩,”Java 言語で学ぶデザインパターン入門 増補改訂版”, ソフトバンククリエイティブ,2004 年 6 月

[9] http://developer.novell.com/documentation/samplecode/jldap_sample/index.htm