The Spring Petclinic Application (rough draft)

Introduction

Spring is a collection of small, well-focused, loosely coupled Java frameworks that can be used independently or collectively to build industrial strength applications of many different types. It is particularly useful for building enterprise applications.

Enterprise applications tend to have the following attributes:

- Lots of Persistent Data
- Lots of User Interface displays and forms
- Complex Business Logic
- Concurrent access by potentially many simultaneous users

The Petclinic sample application is designed to show how the Spring application frameworks can be used to build simple, but powerful database-oriented enterprise applications.

It will demonstrate the use of Spring's core functionality:

- JavaBeans based application configuration
- Model-View-Controller web presentation layer
- Practical database access through JDBC
- Data Validation that is not dependent on the presentation layer

In addition to showing developers how to get started with Spring, the Petclinic sample will endeavor to demonstrate how Spring can used as an effective tool for implementing applications that are concise, flexible, testable, and maintainable.

The Spring frameworks provide a great deal of much needed infrastructure to simplify the tasks faced by application developers. It provides infrastructure that helps developers to create applications that are :

- **concise** by handling a lot of the complex control flow that is needed to use the Java API's, such as JDBC, JNDI, JTA, RMI, and EJB.
- **flexible** by simplifying the process of external application configuration through the use of Reflection and JavaBeans. This allows the developer to achieve a clean separation of configuration data from application code. All application and web application objects, including validators, workflow controllers, and views, should be JavaBeans that can be configured externally. The validation and web application frameworks provide good support for Internationalization.
- **testable** by supplying an interface based design to maximize pluggability. This facilitates the testing of Business Logic without requiring the presence of application or live database servers.
- **maintainable** by facilitating a clean separation of the application layers. It most importantly helps maintain the independence of the Business Logic layer from the Presentation layer through the use of a Model-View-Controller based web presentation framework that can work seamlessly with many different types of view technologies. The web Spring web application framework helps developers to implement their Presentation as a clean and thin layer focused on its main missions of translating user actions into application events and rendering model data.

It is assumed that users of this tutorial will have a basic knowledge of object-oriented design, Java, Servlets, JSP, and relational databases. It also assumes a basic knowledge of the use of a J2EE web application container, Tomcat in particular.

Since the purpose of the sample application is tutorial in nature, the implementation presented here will of course provide only a small subset of the functionality that would be needed by a real world version of the Petclinic.

Petclinic Sample Application Requirements

The application requirement is for an information system that is accessible through a web browser. The users of the application are employees of the clinic who in the course of their work need to view and manage information regarding the veterinarians, the clients, and their pets.

The sample application supports the following Use Cases :

- View a list of veterinarians and their specialties
- View information pertaining to a pet owner
- Update the information of a pet owner
- Add a new pet owner to the system
- View information pertaining to a pet
- Update the information of a pet
- Add a new pet to the system
- View information pertaining to a pet's visitation history
- Add information pertaining to a visit to the pet's visitation history

The view technologies that are used for rendering the application are Java Server Pages (JSP) along with the Java Standard Tag Library (JSTL).

Petclinic Sample Application Implementation

Server Technology

The sample application should be usable with any J2EE web application container that is compatible with the Servet 2.3 and JSP 1.2 specifications. Some of the deployment files provided are designed specifically for Apache Tomcat. These files specify container-supplied connection pooled data sources. It is not necessary to use these files. The application has been configured by default to use a data source without connection pooling to simplify usage. Configuration details are provided in the Developer Instructions section.

Database Technology

The sample application uses a relational database for data storage. Support has been provided for a choice of 1 of 2 database selections, Mysql or HypersonicSQL. HypersonicSQL version 1.7.1 is the default choice and a copy is provided with the application. It is possible to easily configure the application to use either database. Configuration details are provided in the Developer Instructions section.

Development Environment

A copy of the Spring runtime library jar file is provided with the sample application along with some of the other required jar files. The developer will need to obtain the following tools externally, all of which are freely available:

- Java SDK 1.4.1
- Ant 1.5.1
- Tomcat 4.1.18, or some other web application container
- JUnit 3.8.1 - needed to run the unit tests
- Mysql 3.23.53 with mysql-connector 2.0.14 (optional)

NOTE: The version numbers listed were used in the development of the Petclinic application. Other versions of the same tools may or may not work.

Download links for the various tools needed are provided in the Developer Instructions section.

Petclinic Database

The following is an overview of the database schema used in Petclinic. Detailed field descriptions can be found in the "initDB.txt" SQL script files in the databse specific "db" sub-directories. All "id" key fields are type of Java type *int*. For each table with a PRIMARY KEY there is a corresponding sequencer table that is used to aid in the generation of new primary keys. The name of the sequencer table is a conjunction of the original table's name with "_seq".

TABLE: ***owners***
PRIMARY KEY id
TABLE: ***owners_seq*** - sequencer for ***owners***

TABLE: ***types***
PRIMARY KEY id
TABLE: ***types_seq*** - sequencer for ***types***

TABLE: ***pets***
PRIMARY KEY id
FOREIGN KEY type_id references the ***types*** table id field
FOREIGN KEY owner_id references the ***owners*** table id field
TABLE: ***pets_seq*** - sequencer for ***pets***

TABLE: ***vets***
PRIMARY KEY id, int
TABLE: ***vets_seq*** - sequencer for ***vets***

TABLE: ***specialties***
PRIMARY KEY id, int
TABLE: ***specialties_seq*** - sequencer for ***specialties***

TABLE: ***vet_specialties*** - a link table for ***vets*** and their ***specialties***
FOREIGN KEY vet_id references the ***vets*** table id field
FOREIGN KEY specialty_id references the ***specialties*** table id field

TABLE: ***visits***
PRIMARY KEY id
FOREIGN KEY pet_id references the ***pets*** table id field
TABLE: ***visits_seq*** - sequencer for ***visits***

Petclinic Object Model

NOTE: The business layer is COMPLETELY independent of the Presentation layer.

Business Layer classes

- ***petclinic.Clinic*** (interface) - The high-level business API
- ***petclinic.ClinicImpl*** - default implementation of Clinic. Provides caches of all specialties, all vets, all pet types, as well as of owners and their pets and visits that have been found.
- ***petclinic.Entity*** - simple JavaBean superclass of all database related Javabeans
- ***petclinic.NamedEntity*** - adds name property to Entity, used to implement specialties and pet types
- ***petclinic.Person*** - superclass of Vet and Owner
- ***petclinic.Vet*** - holds List of specialties
- ***petclinic.Owner*** - holds List of pets
- ***petclinic.Pet*** - holds List of visits
- ***petclinic.Visit***
- ***petclinic.NoSuchEntityException*** (unchecked)
- ***petclinic.validation.FindOwnerValidator***
- ***petclinic.validation.OwnerValidator***
- ***petclinic.validation.PetValidator***
- ***petclinic.validation.VisitValidator***

Persistence Layer classes

- ***petclinic.ClinicDAO*** (interface) - The high-level persistence interface API
- ***petclinic.ClinicJdbcDAO*** - The default implementation of ClinicDAO. Provides inner class Data Access Objects that implement 8 types of Queries, 3 types of Inserts, and 2 types of Updates.

Presentation Layer (web application) classes

- ***petclinic.web.ClinicController*** - subclass of MultiactionController to handle simple display oriented URL's
- ***petclinic.web.AbstractSearchFormController*** - subclass of SimpleFormController adds the notion of a

search form via an abstract search method

- ***petclinic.web.FindOwnerForm*** - subclass of petclinic.web.AbstractSearchFormController which implements the search method
- ***petclinic.web.AbstractClinicForm*** - subclass of SimpleFormController that is the superclass of the Add and Edit forms
- ***petclinic.web.AddOwnerForm*** - used to add a new Owner
- ***petclinic.web.EditOwnerForm*** - used to edit an existing Owner
- ***petclinic.web.AddPetForm*** - used to add a new Pet
- ***petclinic.web.EditPetForm*** - used to edit an existing Pet
- ***petclinic.web.AddVisitForm*** - adds a new Visit
- ***petclinic.WebUtils*** - provides static utility methods

<u>Test Layer Classes</u>

- ***petclinic.ClinicImplTest*** - a simple and incomplete JUnit test class for ClinicImpl

<u>Views & Implemented Use Cases</u>

- ***welcomeView*** - home screen provides links to display vets list, find an owner, or view documentation
- ***vetsView*** - displays all vets and their specialties
- ***findOwnersForm*** - allows user to search for owners by last name
- ***findOwnersRedirectView*** - redirects to findOwnerForm
- ***selectOwnerView*** - allows user to select from multiple owners with the same last name
- ***ownerView*** - displays a user's data and a list of the owner's pets and their data
- ***ownerRedirect*** - redirects to ownerView
- ***ownerForm*** - allows adding/editing an owner
- ***petForm*** - allows adding/editing a pet
- ***visitForm*** - allows adding a visit

JSP Pages

- ***index.jsp*** - redirects to the welcome page
- ***includes.jsp*** - statically included in all JSP's, sets session="false" and specifies the taglibs in use
- ***uncaughtException.jsp*** - the configured "error-page" displays a stack trace
- ***welcome.jsp*** - displays links to search for an owner or display the vets page
- ***vets.jsp*** - displays a list of vets and their specialties
- ***findOwners.jsp*** - displays a form allowing the user to search for an owner by last name
- ***owners.jsp*** - displays a list of owners meeting the search criteria with a selection button for each owner
- ***owner.jsp*** - displays an owner and the pets owned along with the visit history
- ***ownerForm.jsp*** - displays a form that allows adding or editing an owner
- ***petForm.jsp*** - displays a form that allows adding or editing a pet
- ***visitForm.jsp*** - displays a form that allows adding a visit

The following JSP's each display a form field and the bound error data for that field:

- ***address.jsp***
- ***city.jsp***
- ***telephone.jsp***
- ***lastName.jsp***
- ***firstName.jsp***

The following JSTL tags are used:

- ***c:out***
- ***c:redirect***
- ***c:url***
- ***c:forEach***
- ***c:if***
- ***fmt:message***
- ***fmt:formatDate***

DESIGN NOTES:

- all JSP's are stored under /WEB-INF/jsp except for index.jsp which is the configured "welcome-file"
- The use of JSP technology in the appplication is not exposed to the user
- The end user never sees a URL ending in ".jsp".
- By convention, all URL's in the application ending in ".htm" are handled by web application controllers. Static html pages ending in ".html", such as Javadocs, will be directly served to the end user.
- The results of all form entries are handled using browser round trip redirection to minimize possible end user confusion.
- All pages are extremely simple implementations focusing only on functionality

Directory Structure

f-- indicates a file
d-- indicates a directory
D-- indicates a directory that is created by the Ant build script

d-- ***petclinic*** : the root directory of the project
  f-- ***build.xml*** : the Ant build script
  f-- ***build.properties*** : a properties file for the Ant build script
  d-- ***src*** : Java source file directory tree
  d-- ***war*** : the web application source directory tree
  d-- ***test*** : a parallel testing directory tree
    d-- ***src*** : Java testing source file directory tree
    D-- ***.classes*** : compiled .class files
    D-- ***reports/html*** : generated html-formatted test reports
  d-- ***db*** : database sql scripts and related files
    d-- ***hsqldb*** : directory for files related to HSQL, contains scripts and a context definition
    d-- ***mysql*** : directory for files related to MYSQL, contains scripts and a context definition
    f-- ***petclinic_tomcat_all.xml*** : context definition for all db's

f-- ***build.xml*** : Ant script for populating/emptying the database
D-- ***.classes*** : compiled .class files
D-- ***dist*** : Java archive files
D-- ***docs*** : Javadoc files

IMPORTANT NOTE: Examine/edit the ***build.properties*** file in the project root directory. Database configuration information is stored there.

Petclinic Application Configuration

The Petclinic web application is configured via the following files:

- ***war/WEB-INF/web.xml*** - the standard web application configuration file
- ***war/WEB-INF/applicationContext.xml*** - configures application wide objects
- ***war/WEB-INF/petclinic-servlet.xml*** - configures the petclinic dispatcher servlet and the other controllers and forms that it uses
- ***war/WEB-INF/classes/views*.properties*** - configures the definition of internationalizable Spring views
- ***war/WEB-INF/classes/messages*.properties*** - configures the definition of internationalizable message resources
- ***war/WEB-INF/classes/log4j.properties*** - configures the definition of ***log4j*** loggers

Examine the contents of each of these files for a more in-depth understanding of the details of how the application is configured.

General

- In ***web.xml***, a context-param, "webAppRootkey", provides the key for a system property that specifies the root directory for the web application. This parameter "petclinic.root" can be used to aid in configuring the application.

- In ***web.xml***, a ***ContextLoaderListener*** is defined that loads the root application context of this web app at startup, - by default from ***/WEB-INF/applicationContext.xml***. The root context is the parent of all servlet-specific contexts. This means that its beans are automatically available in these child contexts, both for ***Spring*** getBean(name) calls and (external) bean references. Spring ***ApplicationContext*** objects provide access to a map of user-defined JavaBeans that specify singleton or prototype instances that are used in any and all layers of the application.
- In ***web.xml***, a Servlet named "petclinic" is specified to act as a dispatcher for the entire application. This Servlet will be used to handle all URL's matching the pattern "*.htm". As with any Servlet, multiple URL mappings may be defined. It is also possible to define multiple instances of ***DispatcherServlet***. Each ***DispatcherServlet*** dispatches requests to registered handlers (Controller implementations). Each ***DispatcherServlet*** has its own ***ApplicationContext*** , by default defined in "{servlet-name}-servlet.xml". In this case, it is "petclinic-servlet.xml". This ***ApplicationContext*** is used to specify the various web application user interface beans and URL mappings that are used by the ***DispatcherServlet*** .
- In ***web.xml***, the Spring tag library uri and location are specified for use by the JSP pages.

<u>ApplicationContext</u>

Five globally accessible beans are defined in the Petclinic ***applicationContext.xml*** file.

- ***messageSource*** - a singleton bean that defines a message source for this context, loaded from localized "messages_xx" files in the classpath, such as "/WEB-INF/classes/messages.properties" or "/WEB-INF/classes/messages_de.properties". "getMessage" calls to this context will use this source. Child contexts can have their own message sources, which will inherit all messages from this source and are able to define new messages and override ones defined in this source.

- ***incrementer*** - a prototype bean (i.e. singleton="false") that defines for the application how to intially construct key generators for the database tables. The protoypical instances created are then further configured by the application for table that contains a primary key.
- ***dataSource*** - a singleton bean that defines the implementation of the source of database connections used by the application.
- clincDAO - a singleton bean that defines the implementation of the clinicDAO interface that provides the primary Persistence layer object of the application.
- clinic - - a singleton bean that defines the implementation of the clinic interface that provides the primary Business layer object of the application.

Logging

Petclinic uses ***Apache Log4J*** to provide sophisticated and configurable logging capabilities. The files ***web.xml*** and ***log4j.properties*** specify the configuration of logging in the system.

- In ***web.xml***, a "***log4jConfigLocation***" context-param is specified that sets the location of the ***Log4j*** configuration file. This default location for this file is ***/WEB-INF/classes/log4j.properties***. Specifying this parameter explicitly allows the location to be changed from the default and it also is used to cause periodic ***Log4j*** configuration refresh checks.
- In ***web.xml***, a ***Log4jConfigListener*** is specified that will initialize ***Log4j*** using the specified configuration file when the web app starts.
- In ***log4j.properties***, 2 loggers are specified. A logger called "stdout" provides logging messages to the container's log file. A logger called "logfile" provides logging messages to a rolling file that is specifed using the previously defined "petclinic.root".

DispatcherServlet

Fourteen beans accessible to the "petclinic" DispatcherServlet are defined in the Petclinic ***petclinic-servlet.xml*** file. The dispatcher uses these defintions to delegate actual display and form processing tasks to implementations of the Spring interface ***Controller***.

- ***viewResolver*** - a singleton bean that defines the implementation of the source of view mappings used by the dispatcher. This definition specifies explicit view mappings in a resource bundle (porperties file) instead of the default ***InternalResourceViewResolver***. It fetches the view mappings from localized "views_xx" classpath files, i.e. "/WEB-INF/classes/views.properties" or "/WEB-INF/classes/views_de.properties". Symbolic view names returned by controllers will be resolved in the respective properties file, allowing for arbitrary mappings between names and resources.
- ***urlMapping*** - a singleton bean that defines the URL mapping handler that is to be used by the dispatcher. The definition specifies the use of a ***SimpleUrlHandlerMapping*** instead of the default ***BeanNameUrlHandlerMapping***. Specific URL's are mapped directly to Controllers.
- ***clinicController*** - a singleton bean that defines the Controller used by the dispatcher to handle non-form based display tasks. This bean is a ***MultiActionController*** that can handle multiple requests.
- ***clinicControllerMethodnameResolver***- a singleton bean that defines a mapping of URL's to method names for the ***clinicController*** bean.
- ***findOwnersForm*** - a singleton form bean that is used to search for pet owners by last name. It is an extension of the Petclinic class ***AbstractSearchForm*** which is an extension of the Spring class ***SimpleFormController***.
- 5 singleton form beans are defined that are extensions of the Petclinic class ***AbstractClinicForm*** which is an extension of the Spring class ***SimpleFormController***. These beans are used to handle the various Add and Edit form processing tasks for the dispatcher.
- 4 singleton beans are defined that are implementations of the ***Validator*** interface. These beans are used by the form beans to validate their input parameters and to bind

localizable error messages for display should a processing error occur on submission.

Views

The Controllers used by the dispatcher handle the control flow of the application. The actual display tasks are delegated by the Controllers to implementations of the Spring ***View*** interface. These ***View*** objects are themselves beans that can render a particular different type of view. The handling Controller supplies the View with a data model to render. Views are only responsible for rendering a display of the data model and performing any display logic that is particular to the type of View being rendered. Spring provides support for rendering many different types of views: JSP, XSLT, PDF, Velocity templates, Excel files, and others. By using a ***View*** mapping strategy, Spring supplies the developer with a great deal of flexibility in supporting easily configurable view substitution. Petclinic defines 10 ***View*** beans in the file, ***views.properties***. 2 of these beans are instances of ***RedirectView*** which simply redirects to another URL. The other 8 ***View*** beans are instances of ***JstlView*** which provides some handy support for supporting internationalization/localization in JSP pages that use JSTL.

Messages

The *messages*.properties* files provide localized messages for the supported languages. Petclinic supplies only a single localized message, "welcome" in the default, English, and German properties files respectively.

Petclinic Application Internal Workings

(not yet available)

Developer Instructions

Tools

- Download and install the Spring Framework (examples, including Petclinic are provided)

- Download and install a Java Software Developer Kit, preferably version 1.4 or later
- Download and install Apache Ant, preferably version 1.5.1 or later
- Download and install JUnit, preferably version 3.8.1 or later
- Download and install Apache Tomcat, preferably version 4.1.18 or later
- Download and install Mysql, (optional)
- Hypersonic SQL version 1.7.1 is provided with the application.
- Petclinic and Spring use the Apache Commons Logging and Log4J packages

Ant Setup

Make sure that the Ant executable is in your command shell PATH. Ant will need to reference classes from JUnit and the database(s) of interest. Place a copy of any needed jar files in Ant's ***/lib*** directory, i.e.:

- JUnit - junit.jar
- HSQL - hsqldb.jar
- MYSQL - mysql-connector-java-2.0.14-bin.jar or other

HSQL Setup

Create a new directory containing the a copy of the entire contents of the directory ***petclinic/db/hsqldb***. The file ***petclinic.script*** is the data file that will be used by the server. It has been initialized with some sample data.
Start a server on the standard port 9001 by running the Java command in the file ***server.bat*** or double-clicking on it (Windows). A useful database manager can be started by running the Java command in the file ***manager.bat*** or double-clicking on it (Windows). When the application opens, connect to the "HSQL Database Engine Server" using the default parameters. This tool can also be used to manage other databases. To use a different port, it will be necessary to change the Petclinic Database Setup. It may also be necessary to consult the HSQL documentation for instructions on to change the port the server uses.

MYSQL Setup (optional)

Add the database to an running server by running the SQL script ***db/mysql/initDB.txt***. The Petclinic expects, by default to be able to access the server via the standard port 3306. To use a different port, it will be necessary to change the Petclinic Database Setup.

Petclinic Database Setup

To use a connection pooled data source with Tomcat, it will be necessary to use and possibly edit the appropriate context definition file for the petclinic webapp. To use it, deploy a copy in Tomcat's webapps directory and restart the server. Consult the Tomcat log file if something goes wrong when starting either Tomcat or the Petclinic application. The file is named ***petclinic_tomcat_*.xml***, where ***** is a codename for the database.

An Ant script (***build.xml*** target:setupDB) has been provided that can be used to re-initialize either database.

To select or configue the data source used, you will need to edit the files:

- ***build.properties***
- ***war/WEB-INF/applicationContext.xml***
- ***test/src/applicationContext.xml***

Building the Petclinic Application

Open a command line shell and navigate to the directory containing Petclinic and execute "ant". This will display a list of the Ant targets that are available. Make sure the database is running and execute "ant all". This will run the Ant "all" target which will clean and compile everything, produce Javadocs, and execute a live test using the database. The other Ant targets provide subsets of this functionality.

Deploying the Petclinic Application

Deploy the web application to the server in the usual way. If you need instructions for this, see the Tomcat documentation for details. The Web ARchive file is ***petclinic.war*** and can be found in the ***dist*** directory.

Using the Petclinic Application

Make sure the petclinic web application is running and browse to http://localhost:8080/petclinic.