JAVAJIGI.net 오픈 소스 스터디

jpetstore 예제로 살펴보는 Spring MVC와 iBatis 연동

Added by Sang Hyup Lee, last edited by Sang Hyup Lee on 1월 16, 2007 (view change)
Labels: (None)

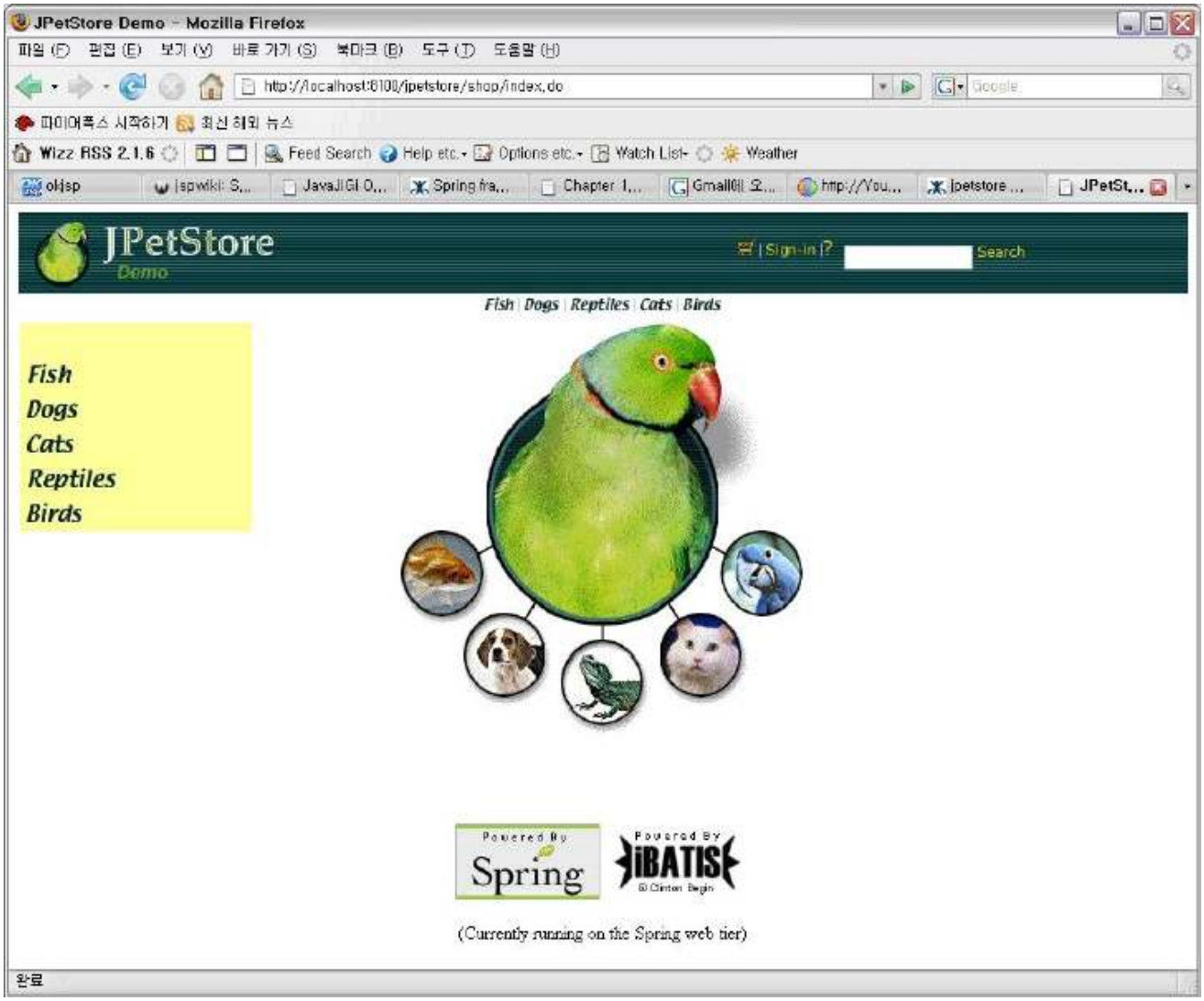
지금까지 Spring MVC를 셋팅하는 과정에서부터 하나의 HTTP Request를 처리하는 과정, 그리고 Spring MVC를 처리하는 과정 중에 발생하는 중요한 HandlerMapping과 View를 결정하는 것까지 살펴보았다. 이제 전체적인 Spring MVC 처리 과정을 정리하는 입장에서 하나의 예제를 소스 코드와 함께 살펴보면서 알아보도록 하자.

Table of Contents

jpetstore의 실제적인 초기화면(index.do)

"jpetstore 설치하기" 세션에서 jpetstore를 Tomcat에 배치하고 웹 브라우저에서 정상적으로 실행해 보는 것까지 했었다. 물론 정상적으로 설치하였다면 내부로 들어가 여러가지 기능을 해 보았으리라 생각된다. 가능하면 순서대로 모든 것을 살펴보면 좋겠지만, 문서의 성격상 Spring framework의 많은 기능 중 jpetstore에 해당하는 부분을 중심으로 살펴보다보니 전부 다 기술하진 못한다. 이해해 주길 바라면서 설치과정 후, 다음 페이지를 통해서 이번 세션에서 설명하고자 하는 것을 살펴보자.

jpetstore를 정상적으로 설치하고 첫 페이지에서 "Enter the Store"" 클릭하여 들어가보면 다음 그림과 같은 화면을 볼 수 있다.



실제 주소를 살펴보면 "jpetstore/shop/index.do" 인것을 볼 수 있다. URL이 ".do" 로 끝나는 것은 초기 WEB-INF/web.xml 에서 HTTP Request를 처음으로 받아서 처리하고 중개하는 DispatcherServlet을 "petstore""" 서블릿 명으로 "*.do"" URL Pattern으로 매핑하는 것으로 알 수 있다.

```
<servlet>
  <servlet-name>petstore</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>petstore</servlet-name>
  <!--
  <servlet-name>action</servlet-name>
  -->
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

따라서 실제로 index.do 라는 URL로 HTTP를 요청하게 되면 DispatcherServlet 이 처리하는 것을 알 수 있다. 하지만, index.do에 해당하는 페이지는 별다른 설명을 할 필요없는 경우임으로 여기서는 생략하도록 한다.

모든 URL에 해당하는 petstore-servlet.xml 에서의 셋팅

이제 jpetstore에서 Spring MVC의 전체적인 과정을 살펴보기 위해서 카테고리를 살펴보는 페이지(HTTP Request)를 분석해 보자.
jpetstore 실제 초기 화면(index.do)에서 각 카테고리(동물)에 해당하는 것을 아무거나 클릭하면 "shop/viewCategory.do" 라는 URL로 요청되는 것을 볼 수 있다. 여기에 해당하는 URL을 통해서 Spring MVC의 전체적인 처리 과정을 살펴볼 것이다.

먼저, 모든 URL에 해당하는 Controller를 셋팅하고 있는 petstore-servlet.xml 파일을 살펴보자.

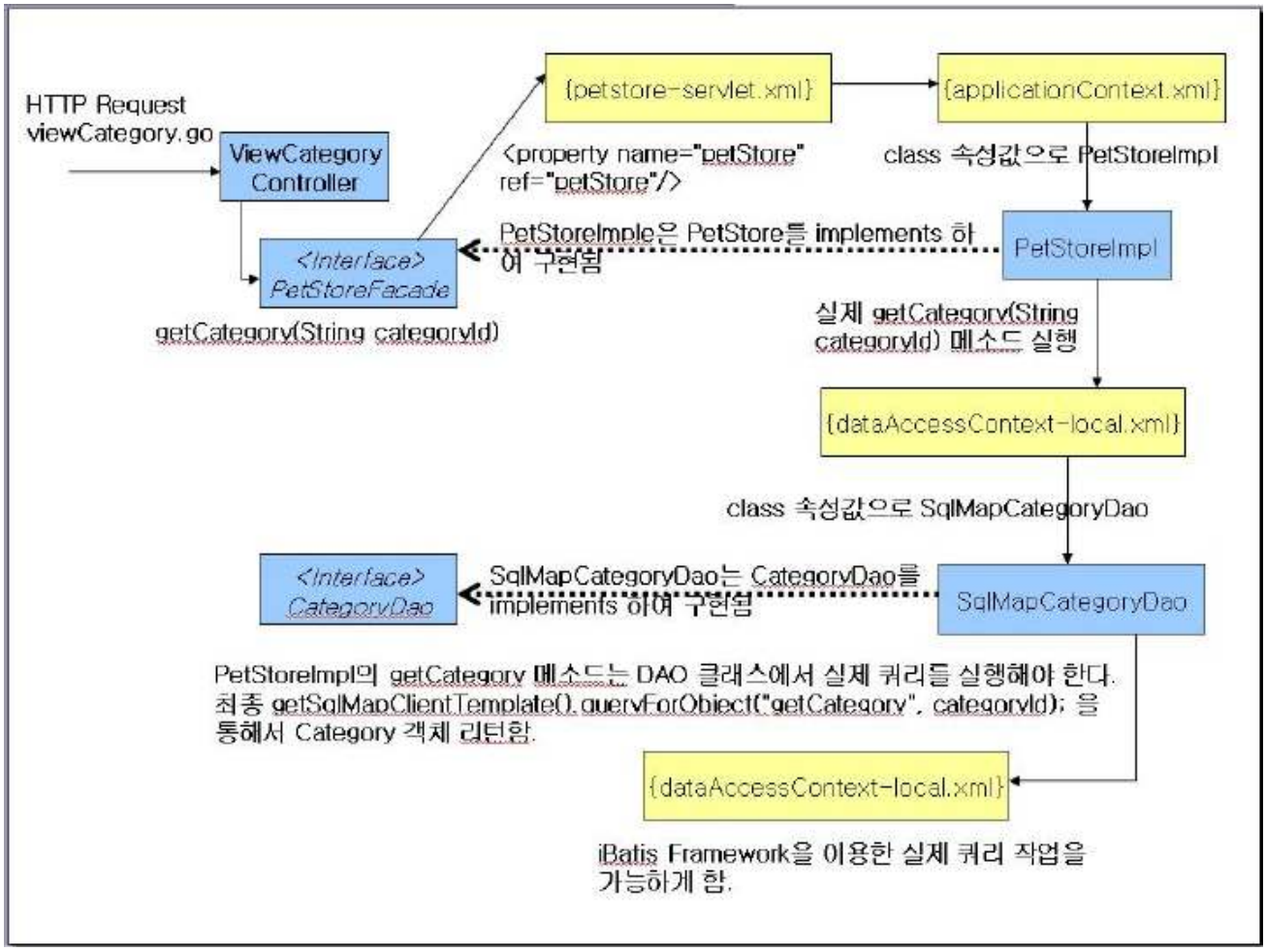
```
<!-- ========================= DEFINITIONS OF PUBLIC CONTROLLERS ========================= -->
<bean id="defaultHandlerMapping" class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />
<bean name="/shop/addItemToCart.do" class="org.springframework.samples.jpetstore.web.spring.AddItemToCartController" >
      <property name="petStore" ref="petStore"/>
</bean>

<bean name="/shop/checkout.do" class="org.springframework.samples.jpetstore.web.spring.ViewCartController" >
      <property name="successView" value="Checkout"/>
</bean>

<bean name="/shop/index.do" class="org.springframework.web.servlet.mvc.ParameterizableViewController">
      <property name="viewName" value="index"/>
</bean>

<bean name="/shop/newAccount.do" class="org.springframework.samples.jpetstore.web.spring.AccountFormController" >
      <property name="petStore" ref="petStore"/>
      <property name="validator" ref="accountValidator"/>
      <property name="successView" value="index"/>
</bean>

<bean name="/shop/removeItemFromCart.do" class="org.springframework.samples.jpetstore.web.spring.RemoveItemFromCartContr

<bean name="/shop/signoff.do" class="org.springframework.samples.jpetstore.web.spring.SignoffController" />

<bean name="/shop/searchProducts.do" class="org.springframework.samples.jpetstore.web.spring.SearchProductsController" >
      <property name="petStore" ref="petStore"/>
</bean>

<bean name="/shop/signon.do" class="org.springframework.samples.jpetstore.web.spring.SignonController" >
      <property name="petStore" ref="petStore"/>
</bean>

<bean name="/shop/signonForm.do" class="org.springframework.web.servlet.mvc.ParameterizableViewController">
      <property name="viewName" value="SignonForm"/>
</bean>

<bean name="/shop/updateCartQuantities.do" class="org.springframework.samples.jpetstore.web.spring.UpdateCartQuantitiesContr

<bean name="/shop/viewCart.do" class="org.springframework.samples.jpetstore.web.spring.ViewCartController" >
      <property name="successView" value="Cart"/>
</bean>

<bean name="/shop/viewCategory.do" class="org.springframework.samples.jpetstore.web.spring.ViewCategoryController" >
      <property name="petStore" ref="petStore"/>
</bean>

<bean name="/shop/viewItem.do" class="org.springframework.samples.jpetstore.web.spring.ViewItemController" >
      <property name="petStore" ref="petStore"/>
</bean>

<bean name="/shop/viewProduct.do" class="org.springframework.samples.jpetstore.web.spring.ViewProductController" >
      <property name="petStore" ref="petStore"/>
</bean>
```

"shop/viewCategory.do" 요청을 처리하는 과정

우리가 살펴볼 URL은 "shop/viewCategory.do"". 그런데 여기서 모든 URL에 해당하는Controller를 바인딩 하는 부분을 추가한 것은 앞으로 설명한 처리 과정을 동일하게 따르고 있음으로 보여주기 위함이다. 그러면 "shop/viewCategory.do" 요청에 해당하는 Controller를 바인딩하기 위한 WebApplicationContext의 설정을 살펴보면 다음과 같다.

```
<bean name="/shop/viewCategory.do" class="org.springframework.samples.jpetstore.web.spring.ViewCategoryController" >
  <property name="petStore" ref="petStore"/>
</bean>
```

URL의 동일한 name으로 bean 태그를 셋팅하고 있다. 그리고 사용하는 클래스가 org.springframework.samples.jpetstore.web.spring.ViewCategoryController가 된다. 즉, "shop/viewCategory.do"" 해당하는 HTTP 요청을 처리하는 컨트롤러가 ViewCategoryController가 된다는 것이다. 그러면, 여기서 "shop/viewCategory.do" 요청을 처리하는 Spring MVC Controller의 전체적인 처리 과정을 그림으로 살펴보고 계속 설명하기로 한다.



위의 그림은 Spring MVC 처리 과정 중에 Controller가 ModelAndView 객체를 리턴하기 까지의 과정을 그림으로 도식화 한 것이다. 위의 그림에서 노란색 박스로 처리된 것은 WebApplicationContext 설정을 포함하고 있는 XML 파일이고, 파란색 박스로 처리된 것은 처리 과정 중에 사용되는 인터페이스 또는 클래스이다. 처리 과정에서 XML 파일을 표시해 둔 것은 실제 처리 과정에서 XML 파일을 직접적으로 사용하는 것은 아니나(컨테이너에 이미 적재-메모리상에- 되어있다), 이해를 돕기 위해 XML 설정을 포함한 설명을 하기 위함이다.

Spring MVC 처리 과정 중에서 "shop/viewCategory.do"" 해당하는 요청을 처리하기 위해서 DispatcherServlet은 HandlerMapping(petstore-servlet.xml)을 통해서 ViewCategoryController가 해당 요청을 처리하는 Controller 임을 알아온다. 그리고 ViewCategoryController는 DispatcherServlet에게 ModelAndView를 반환하기 위해서 위의 그림과 같은 처리 과정을 거치게 된다.
DispatcherServlet이 HTTP 요청을 최초로 받아 최종 View 객체를 얻어 응답하기까지 중요한 역할을 담당하지만, 개발자는 Spring MVC의 처리 과정을 이해하고 사용하기만 할 뿐 실제 소스 코드로 작성해야 할 부분은 비즈니스 로직에 해당하는 Controller 클래스를 작성하는 데 있다. 따라서 실제 중요한 부분은 Controller가 처리하는 과정을 해당 요청에 맞는 형태로 개발할 수 있는 것이 중요하다. 그런 면에서 jpetstore의 예로 "shop/viewCategory.do"" 해당하는 처리 과정을 자세하게 살펴보는 것은 중요하다라고 할 수 있다.

ViewCategoryController

이제 실제 비지니스 레이어를 담당하는 Controller 클래스에 대해서 알아보자. 여기서는 ViewCategoryController 클래스를 살펴보게 되는데, 모든 Controller 클래스들은 org.springframework.web.servlet.mvc.Controller 를 implements 하여 구현한 클래스임으로 구현해야 할 부분이 많치 않다. 실제로 org.springframework.web.servlet.mvc.Controller 를 implements하여 구현하고자 할 경우, handleRequest(HttpServletRequest request, HttpServletResponse response) 메소드를 구현해 주기만 하면 된다.

이제 ViewCategoryController 클래스를 살펴보자.

ViewCategoryController.java

```
package org.springframework.samples.jpetstore.web.spring;

import java.util.HashMap;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.support.PagedListHolder;
import org.springframework.samples.jpetstore.domain.Category;
import org.springframework.samples.jpetstore.domain.logic.PetStoreFacade;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

/**
 * @author Juergen Hoeller
 * @since 30.11.2003
 */
public class ViewCategoryController implements Controller {

	private PetStoreFacade petStore;

	public void setPetStore(PetStoreFacade petStore) {
		this.petStore = petStore;
	}

	public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
		Map model = new HashMap();
		String categoryId = request.getParameter("categoryId");
		if (categoryId != null) {
			Category category = this.petStore.getCategory(categoryId);
			PagedListHolder productList = new PagedListHolder(this.petStore.getProductListByCategory(categoryId));
			productList.setPageSize(4);
			request.getSession().setAttribute("ViewProductAction_category", category);
			request.getSession().setAttribute("ViewProductAction_productList", productList);
			model.put("category", category);
			model.put("productList", productList);
		}
		else {
			Category category = (Category) request.getSession().getAttribute( "ViewProductAction_category");
			PagedListHolder productList = (PagedListHolder) request.getSession().getAttribute( "ViewProductAction_productList'
			if (category == null || productList == null) {
				throw new IllegalStateException("Cannot find pre-loaded category and product list");
			}
			String page = request.getParameter("page");
			if ("next".equals(page)) {
				productList.nextPage();
			}
			else if ("previous".equals(page)) {
				productList.previousPage();
			}
			model.put("category", category);
			model.put("productList", productList);
		}
		return new ModelAndView("Category", model);
	}

}
```

사실상, 특별히 설명이 필요한 부분은 거의 없다. Request의 파라미터로 넘어오는 categoryId를 받아와서 categoryId가 null이 아닌 경우, 해당 카테고리에 해당하는 리스트를 ModelAndView로 "Category"라는 이름으로 Model을 반환해 준다. categoryId가 null인 경우, 해당 session attribute에 "ViewProductAction_category"로 Category 클래스를 가져와 ModelAndView로 "Category"라는 이름으로 Model을 반환해 준다.

오히려 여기서 필요한 설명은 IoC를 지원하기 위해서 setter injection을 지원하는 다음의 코드 부분이다. 위에서 보는 바와 같이 PetStoreFacade 인터페이스를 이용하여 실제 구현하고자 하는 클래스들은 지원하고 있다 이를 setter injection으로 지원하고 있다.

그리고, 각 카테고리에 해당하는 리스트를 얻기 위해서 this.petStore.getProductListByCategory(categoryId) 에 대한 이해가 필요하다. 이것은 iBatis framework를 사용하기 때문에 다음 세션에서 살펴보도록 하겠다. getProductListByCategory 메소드는 PetStoreFacade 인터페이스에서 정의해 두고, PetStoreImpl에서 구현해 둔 메소드가 된다. 그리고 실제 쿼리를 실행하는 부분에서 iBatis를 사용하기 때문에 별도로 살펴보아야 한다.

jpetstore에서의 iBatis 프레임워크 셋팅

iBatis를 이용하여 어떻게 웹 애플리케이션을 개발하는지에 대해서는 여기서 언급하지 않겠다. 현재 다양하게 나와 있는 레퍼런스들이 있기 때문이

다. 참고문헌을 참조하여 iBatis에 대한 study를 개인적으로 진행해야 한다. 여기서 다룰 내용은 jpetstore에서 Spring 프레임워크와 iBatis 프레임워크를 어떻게 셋팅하고 두 개의 프레임워크가 어떻게 연동되어져 개발되는지에 대한 내용이다.
먼저, jpetstore에서 iBatis 프레임워크를 셋팅하는 방법에 대해서 알아보자.

applicationContext.xml 에서 property 파일 로드

이전에 jpetstore 에서의 Spring 초기 설정과 IoC에서 언급한 바와 같이 web.xml의 context-param 엘리먼트의 param-value의 값으로 정의되어진 dataAccessCont ext-local.xml 과 applicationContext.xml 파일이 POJO 빈의 생명주기를 관리하는 정보를 담고 있다고 하였다. 여기서 근간이 되는 파일이 applicationContext.xml 인데 해당하는 모든 내용을 살펴보진 않았다.
이 중에서 우리가 여기서 살펴볼 부분은 property 파일을 WAS에 로드해 두고 사용하게 하는 부분이다. value 엘리먼트로 주어진 WEB-INF/jdbc.properties 파일을 iBatis DataSource 설정 파일로 사용하게 되는데 내용은 다음과 같다.

```
<!-- Configurer that replaces ${...} placeholders with values from properties files -->
<!-- (in this case, mail and JDBC related properties) -->
<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
	<property name="locations">
		<list>
			<value>WEB-INF/mail.properties</value>
			<value>WEB-INF/jdbc.properties</value>
		</list>
	</property>
</bean>
```

위에서 잠시 언급한 바와 같이 value 엘리먼트의 값으로 주어진 두 개의 properties 파일을 WAS 로드하여 웹 애플리케이션에서 사용할 수 있게 된다. 여기서는 특별한 설명이 필요한 것은 아니다. WEB-INF 디렉토리에 properties 파일이 존재하면 되고, 일반적으로 작성되는 properties 규칙을 따르고 있기 때문에 이에 대한 설명을 할 필요도 없을 듯 하다. 다음은 properties 파일을 WAS에 로드하고 Spring 프레임워크의 컨텍스트 파일에서 Data Source를 어떻게 셋팅하는지 알아보자.

dataAccessContext-local.xml 에서 Data Source 셋팅하기

web.xml의 contextConfigLocation 이라는 param-name의 value로 주어진 두 개의 파일 중, dataAccessContext-local.xml 파일이 Data Source 셋팅을 담당하고 있으며 여기서 기본적인 데이터베이스와 관련된 셋팅을 하고 있다. 먼저, dataAccessContext-local.xml 파일의 내용중 위에서 로드하였던 properties 파일의 내용을 셋팅하는 부분을 살펴보자.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<!--
  - Application context definition for JPetStore's data access layer.
  - Accessed by business layer objects defined in "applicationContext.xml"
  - (see web.xml's "contextConfigLocation").
  -
  - This version of the data access layer works on a combined database,
  - using a local DataSource with DataSourceTransactionManager. It does not
  - need any JTA support in the container: It will run as-is in plain Tomcat.
  -->
<beans>

	<!-- ========================== RESOURCE DEFINITIONS ========================== -->

	<!-- Local Apache Commons DBCP DataSource that refers to a combined database -->
	<!-- (see dataAccessContext-jta.xml for an alternative) -->
	<!-- The placeholders are resolved from jdbc.properties through -->
	<!-- the PropertyPlaceholderConfigurer in applicationContext.xml -->
	<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
		<property name="driverClassName" value="${jdbc.driverClassName}"/>
		<property name="url" value="${jdbc.url}"/>
		<property name="username" value="${jdbc.username}"/>
		<property name="password" value="${jdbc.password}"/>
	</bean>

	<!-- 중간 생략 -->
</beans>
```

"WEB-INF/jdbc.properties" 파일을 살펴보면 jdbc.driverClassName 부터해서 데이터 베이스 커넥션과 관련된 정보를 담고 있다. 위의 <bean id="dataSource"> 에서 property의 value로 주어지는 ${}에 해당하는 값들이 jdbc.properties에서 셋팅되어진 값들이 들어가게 된다. 이 부분은 ANT와 같이 XML을 로드하는 방법을 따름으로 별도의 설명은 필요없을 듯 하다.

다음은 Spring 프레임워크에서 트랜잭션 매니져를 담당하는 클래스를 셋팅하는 부분이다.

```
<!-- Transaction manager for a single JDBC DataSource -->
<!-- (see dataAccessContext-jta.xml for an alternative) -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
</bean>
```

위에서 셋팅한 dataSource는 org.springframework.jdbc.datasource.DataSourceTransactionManager 클래스를 이용하여 트랜잭션 관리는 하는 것을 알 수 있다. org.springframework.jdbc.datasource.DataSourceTransactionManager 클래스는 Spring 프레임워크에서 기본적으로 제공하는 TransactionManager 클래스가 된다. 소스 코드를 따라가보면 기본적으로 제공하는 commit이나 rollback 등의 기본을 메소드로 제공하고 있다.

다음은 실제 iBatis 프레임워크를 사용하여 Database layer를 담당하겠다는 셋팅을 해 주는 부분이다.

```
<!-- SqlMap setup for iBATIS Database Layer -->
<bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
        <property name="configLocation" value="WEB-INF/sql-map-config.xml"/>
        <property name="dataSource" ref="dataSource"/>
</bean>
```

<bean> 엘리먼트 태그 아이디로 "sqlMapCliet"가 주어정 있고, org.springframework.orm.ibatis.SqlMapClientFactoryBean 클래스를 class 값을 하고 있다. org.springframework.orm.ibatis.SqlMapClientFactoryBean 클래스는 Spring 프레임워크에서 iBatis를 지원하기 위한 FactoryBean 클래스이다. 두 개의 property 엘리먼트를 가지고 있는데, 이미 정의된 name과 value 값이 중요하다.
여기서는 value로 "WEB-INF/sql-map-config.xml" 파일을 설정하고 있다. sql-map-config.xml 파일은 iBatis에서 쿼리를 관리하고 있는 XML 파일을 하나의 config 파일로 관리하는 파일이다. 실제 내용은 다음 부분에서 살펴보도록 한다.

다음은 DAO 인터페이스를 implements하고 있는 각각의 iBatis 클래스를 sqlMapClient와 연결시켜주는 설정이다. 여러 개의 설정 중에 하나만 살펴보면 된다.

```
<bean id="categoryDao" class="org.springframework.samples.jpetstore.dao.ibatis.SqlMapCategoryDao">
        <property name="sqlMapClient" ref="sqlMapClient"/>
</bean>
```

위의 <bean> 엘리먼트의 id로 categoryDao로 설정되어져 있고, 이를 구현한 클래스가 org.springframework.samples.jpetstore.dao.ibatis.SqlMapCategoryDao 이다. SqlMapCategoryDao는 CategoryDao 인터페이스를 iBatis 프레임워크와 연동하여 구현한 클래스이다.
이러한 개념을 잠시 확장해 보면 iBatis와 마찬가지로 Spring 프레임워크는 persistence layer를 담당하고 있는 유명한 프레임워크(예를들면, Hibernate 같은)를 지원하고 있다. org.springframework.orm 패키지가 이를 담당하고 있다.

여기까지 살펴본 후 실제 각각의 DAO 인터페이스를 구현한(implements) 클래스의 소스 코드를 살펴볼 수 있다. 각각의 구현 클래스들은 org.springframework.orm.ibatis.support.SqlMapClientDaoSupport 클래스를 extends하여 Spring 프레임워크에서 제공하는 iBatis support 메소드를 사용하여 최종 구현하고 있다. 이렇게 하면 소스코드가 아주 깔끔하면서도 읽기 쉽도록 구현할 수 있다. 예제 소스코드는 sql-map-config.xml 파일을 살펴본 후 살펴보도록 하겠다.

sql-map-config.xml 파일 살펴보기

sql-map-config.xml 파일의 내용은 실제로 간단하다. 각 DAO(Data Access Object) 클래스에 해당하는 쿼리를 담당하고 있는 XML파일의 경로를 가지고 있기 때문이다.
특별한 설명은 필요없을 듯 하다. iBatis에 대한 보다 자세한 내용은 참고문헌을 참조하면 된다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"
    "http://www.ibatis.com/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
        <sqlMap resource="org/springframework/samples/jpetstore/dao/ibatis/maps/Account.xml" />
        <sqlMap resource="org/springframework/samples/jpetstore/dao/ibatis/maps/Category.xml" />
        <sqlMap resource="org/springframework/samples/jpetstore/dao/ibatis/maps/Product.xml" />
        <sqlMap resource="org/springframework/samples/jpetstore/dao/ibatis/maps/Item.xml" />
        <sqlMap resource="org/springframework/samples/jpetstore/dao/ibatis/maps/Order.xml" />
        <sqlMap resource="org/springframework/samples/jpetstore/dao/ibatis/maps/LineItem.xml" />
        <sqlMap resource="org/springframework/samples/jpetstore/dao/ibatis/maps/Sequence.xml" />
</sqlMapConfig>
```

참고문헌

SQL Maps2.0 개발자 가이드(한글판) : openframework.or.kr에서 번역된 iBatis 개발자 가이드
SQL Maps2.0 tutorial(한글판) : openframework.or.kr에서 번역된 iBatis Tutorial

문서에 대하여

최초작성자 : OSS:이상협
최초작성일 : 2006년 11월 02일
버전 : 0.5
문서이력 :

- 2006년 11월 02일 이상협 문서 최초 생성 : jpetstore 예제로 살펴보는 Spring MVC와
- 2007년 1월 16일 이상협 문서 수정 : jpetstore에서의 iBatis 프레임워크 셋팅 부분 추가중

내가 하려는 걸 먼저 해 뒀군요. ^^
제가 2기 스터디때 했어야 하는 부분인데..
그냥 고마울 따름이죠..~~~

Posted by 장회수 at 12월 15, 2006 16:06