Spring和Struts整合

Spring+Struts整合三种方式

- ✓ 方式一：通过Spring的ActionSupport类实现
- ✓ 方式二：通过Spring的DelegatingRequestProcessor类实现
- ✓ 方式三：通过Spring的DelegatingActionProxy类实现

- ✓ 不管使用哪种方式都需要自动装载创建Spring容器对象

初始化Spring容器

- ✓ Spring中提供了对Java Web和Struts的支持，无需显式的获得ApplicationContext容器对象，可以自动装载Spring应用环境
- ✓ 通过两种方式可以自动初始化Spring容器
  - ➢ Struts插件
  - ➢ WEB监听器
- ✓ 这两种加载Spring环境的方式只能用其中一种，否则会加载两个不同的Spring容器！

利用Struts插件初始化Spring容器

```
<plug-in
    className="org.springframework.web.struts.ContextLoaderPlugIn">

    <!-- contextConfigLocation属性名是固定的，value设置spring配置文件的路径，多个文件路径用，隔开 -->
    <set-property property="contextConfigLocation"
                    value="/WEB-INF/applicationContext.xml"/>

</plug-in>
```

利用监听器初始化Spring容器

- ✓ 如果不使用struts，那么就无法使用注册插件来加载Spring容器，还有另外一种方式，在web.xml文件中增加一个监听器，这个监听器类是由Spring提供的
- ✓ 配置初始化参数，contextConfigLocation名称是固定的，value值是classpath*:开头表示按类路径搜索，后面是Spring配置文件名称，多个名称用逗号间隔，可以用*作为通配符匹配多个文件

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:applicationContext*.xml,/WEB-INF/applicationContext*.xml
    </param-value>
</context-param>
<listener>
      <listener-class>
          org.springframework.web.context.ContextLoaderListener
      </listener-class>
</listener>
```

方式一：通过Spring的ActionSuuport类实现

- ✓ 这种方式下Action本身不受Spring容器管理，在Action中显式的获得Spring容器对象，然后显式的获得Bean引用。
- ✓ 可以让Action继承Spring提供的ActionSupport类而不是Action类，这样可以利用继承的getWebApplicationContext()方法直接获得Spring容器对象

第一种方式：示例代码

applicationContext.xml配置文件：

```
<bean id="userServiceImpl" class="com.china.ssh.service.UserServiceImpl"/>
```

Action类：

```
public class LoginAction extends ActionSupport {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
                    HttpServletRequest request, HttpServletResponse response) {
        LoginForm loginForm = (LoginForm) form;
        //获得Spring容器
        ApplicationContext context = this.getWebApplicationContext();
        //获得Bean对象
        UserService service = (UserService) context.getBean("userServiceImpl");
        service.checkLogin(loginForm.getUsername(), loginForm.getPassword());
        return null;
    }
}
```

方式二：通过Spring的DelegatingRequestProcessor类实现

- ✓ 这种方式的原理是用Spring自带的DelegatingRequestProcessor类代替struts默认的RequestProcessor类
- ✓ 使用方式二，需要做以下改变
  - ➢ 在struts-config.xml中重新设置<controller>，注意这个标签的位置是<action-mappings>和<message-resources>标签之间
  - ➢ 在Spring配置文件中配置Action对应的Bean，需要注意的是不要用id属性，而是用name属性，name属性的值，而且其值必须与struts配置文件struts-config.xml中<action>标签里的path属性值一样
  - ➢ struts配置文件struts-config.xml中<action>标签里的type属性就没有用处了
  - ➢ 在Action中声明需要注入的Bean属性

第二种方式：示例代码

struts-config.xml配置文件

```
<action-mappings>
    <action attribute="loginForm" input="/login.jsp" name="loginForm"
            path="/loginAction" scope="request" />
</action-mappings>
<controller
    processorClass="org.springframework.web.struts.DelegatingRequestProcessor"/>
```

applicationContext.xml配置文件

```
<!-- 配置业务Bean -->
<bean id="userServiceImpl" class="com.china.ssh.service.UserServiceImpl"/>
<!-- 配置Action的Bean scope="prototype"表示每次发出请求都会创建一个新的Action对象-
    ->
<bean name="/loginAction" class="com.china.ssh.action.LoginAction"
    scope="prototype">
    <property name="service" ref="userServiceImpl"/>
</bean>
```

第二种方式：示例代码

Action类：

```
package com.china.ssh.action;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import org.springframework.context.ApplicationContext;
import org.springframework.web.struts.ActionSupport;
import com.china.ssh.form.LoginForm;
import com.china.ssh.service.UserService;
public class LoginAction extends ActionSupport {
    private UserService service;
    public void setService(UserService service) {
        this.service = service;
    }
    public ActionForward execute(ActionMapping mapping, ActionForm form,
                    HttpServletRequest request, HttpServletResponse response) {
        LoginForm loginForm = (LoginForm) form;
        service.checkLogin(loginForm.getUsername(), loginForm.getPassword());

        return null;
    }
}
```

方式三：通过Spring的DelegatingActionProxy类实现

- ✓ 使用DelegatingActionProxy不需要占用RequestProcessor扩展点，还是用ActionServlet转发请求，请求转发给Action，而Action的实现类全都是DelegatingActionProxy类型，DelegatingActionProxy再把请求转发给Spring容器中的Action
- ✓ 使用方式三，需要做以下修改：
  - ➢ 去掉struts-config.xml中的<controller />配置
  - ➢ 把struts-config.xml中的<action>的type属性全都改为org.springframework.web.struts.DelegatingActionProxy类型

第三种方式：示例代码

struts-config.xml配置文件：

```
<action-mappings>
    <action attribute="loginForm" input="/login.jsp" name="loginForm"
            path="/loginAction" scope="request"
            type="org.springframework.web.struts.DelegatingActionProxy"
    />
</action-mappings>
```

OpenSessionInView

- ✓ 延迟加载的问题，我们可以把所有的lazy都是设置为false，但是也可以使用Spring提供的OpenSessionInViewFilter这个过滤器，这样的话hibernate的Session就会自动的在请求结束之后关闭，防止出现当在视图层使用数据的时候Session已关闭的情况

OpenSessionInView示例

需要在web.xml中加入以下过滤器：

```
<filter>
   <filter-name>hibernateFilter</filter-name>
   <filter-class>
    org.springframework.orm.hibernate3.support.OpenSessionInViewFilter
    </filter-class>
</filter>
<filter-mapping>
   <filter-name>hibernateFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

Spring设置请求编码

需要在web.xml中加入以下过滤器：

```
<filter>
    <filter-name>Spring character encoding filter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-
     class>
    <init-param>
     <param-name>encoding</param-name>
     <param-value>UTF-8</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>Spring character encoding filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```