Struts2-Hibernate-Spring整合

大结局

Struts2 + Spring

启动Spring容器

- 对于使用Spring的Web应用，无需手动创建Spring容器，而是通过配置文件，声明式地创建Spring容器。因此在Web应用中创建Spring容器有如下两种方式：
  - 直接在web.xml文件中配置创建Spring容器
  - 利用第三方MVC框架的扩展点，创建Spring容器

  第一种创建Spring容器的方法更加常见。

启动Spring容器

- 为了让Spring容器随Web应用的启动而自动启动，借助于ServletContextListener监听器即可完成，该监听器可以在Web应用启动时回调自定义方法---该方法就可以启动Spring容器。
- Spring提供了一个ContextLoaderListener，该监听器类实现了ServletContextListener接口。该类可以作为Listener使用，它会在创建时自动查找WEB-INF/下的applicationContext.xml文件，因此，如果只有一个配置文件，并且文件名为applicationContext.xml，则只需在web.xml文件中增加如下配置片段即可：

```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

启动Spring容器

- Spring根据指定配置文件创建WebApplicationContext对象，并将其保存在Web应用的ServletContext中。

MVC框架与Spring整合的思考

- 对于一个基于B/S架构的Java EE应用而言，用户请求总是向MVC框架的控制器请求，而当控制器拦截到用户请求后，必须调用业务逻辑组件来处理用户请求，而当控制器拦截到用户请求后，必须调用业务逻辑组件来处理用户请求。此时有一个问题：控制器应该如何获得业务逻辑组件？

MVC框架与Spring整合的思考

- 最容器想到的策略是，直接通过new关键字创建业务逻辑组件，然后调用业务逻辑组件的方法，根据业务逻辑方法的返回值确定结果。
- 但在实际应用中，不会这么做，其主要原因有三：
  - 控制器直接创建业务逻辑组件，导致控制器和业务逻辑组件的耦合降低到代码层次，不利于高层解耦。
  - 控制器不应该负责业务逻辑组件的创建，控制器只是业务逻辑组件的使用，无需关心业务逻辑组件的实现。
  - 每次创建新的业务逻辑组件导致性能下降。

MVC框架与Spring整合的思考

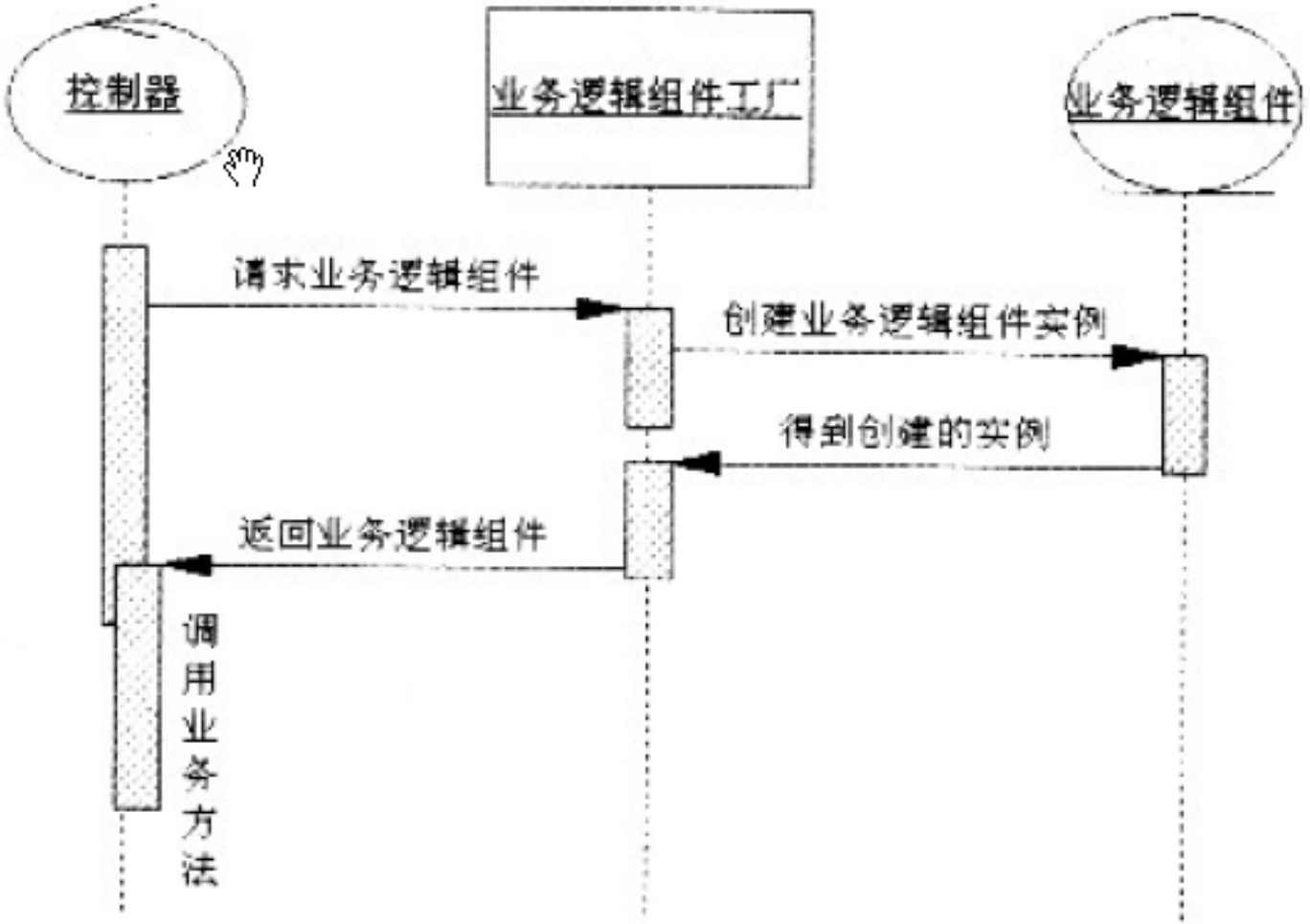

图 13.3　工厂模式顺序图

MVC框架与Spring整合的思考

- 如果系统采用Spring框架，则Spring成为最大的工厂。Spring负责业务逻辑组件的创建和生成，并可管理业务逻辑组件的生命周期。可以如此理解：Spring是个性能非常优秀的工厂，可以生产出所有的实例，从业务逻辑组件，到持久层组件，甚至控制器组件。
- 那么问题是：控制器如何访问到Spring容器中的业务逻辑组件呢？为了让Action访问Spring的业务逻辑组件，有两种策略：
  - Spring容器负责管理控制器Action，并利用依赖注入为控制器注入业务逻辑组件。
  - 利用Spring的自动装配，Action将会自动从Spring容器中获取所需的业务逻辑组件。

让Spring管理控制器

- Struts2的核心控制器首先拦截到用户请求，然后将请求转发给对应的Action处理，在此过程中，Struts2将负责创建Action实例，并调用其execute()方法，这个过程是固定的。
- 但现在是由Spring容器来管理Action实例，而不是Struts2。那么，核心控制器如何知道调用Spring容器中的Action，而不是自行创建Action实例呢？这个工作由Struts2提供的Spring插件完成。

让Spring管理控制器

- Struts2提供的Spring插件提供了一种伪Action，当我们在struts.xml文件中配置Action时，通常需要指定class属性，该属性就是用于创建Action实例的实现类。但Spring插件允许我们指定class属性时，不再指定Action的实际实现类，而是指定为Spring容器中的Bean ID，这样Struts2不再自己负责创建Action实例，而是直接通过Spring容器去获取Action对象。

让Spring管理控制器

- 如前所述，整合的关键是：当Struts2将请求转发给指定的Action时，Struts2中的该Action只是一个代号，并没有指定实际的实现类，当然也不可能创建Action实例，而隐藏在该Action下的是Spring容器中的Action实例---它才是真正处理用户请求的控制器。

让Spring管理控制器

- 示例：SpringManageAction
- Spring容器为控制器注入业务逻辑组件，这也是Spring和Struts2整合的关键所在。
- 当使用Spring容器管理系统的Action，在struts.xml文件中配置该Action时，class属性并不是指向该Action的实现类，而是指定了Spring容器中Action实例的ID。

Spring整合Hibernate

Spring提供的DAO支持

- DAO模式是一种标准的Java EE设计模式，DAO模式的核心思想是：所有的数据库访问，都通过DAO组件完成，DAO组件封装了数据库的增，删，改等原子操作。业务逻辑组件依赖于DAO组件提供的数据库原子操作，完成系统业务逻辑的实现。

- 轻量级Java EE应用的架构以Spring IoC容器为核心，承上启下：其向上管理来自表现层的Action，向下管理业务逻辑层组件，同时负责管理业务逻辑层所需的DAO对象。

Spring提供的DAO支持

- Spring对实现DAO组件提供了许多工具类，系统的DAO组件可通过继承这些工具类完成，从而可以更加简便地实现DAO组件。
- Spring提供多种数据库访问技术的DAO支持，包括Hibernate，JDO，TopLink，iBatis，OJB，JPA等。就Hibernate的持久层访问技术而言，Spring提供了如下三个工具类(或接口)来支持DAO组件的实现：
  - HibernateDaoSupport
  - HibernateTemplate
  - HibernateCallback

使用HibernateTemplate

- HibernateTemplate提供持久层访问模板化，它只需要提供一个SessionFactory的引用，就可以执行持久化操作。
- SessionFactory对象既可通过构造参数传入，也可通过设值方式传入。HibernateTemplate提供如下三个构造函数：
  - HibernateTemplate()：在创建了HibernateTemplate实例之后，还必须使用方法setSessionFactory(SessionFactory sf)为HibernateTemplate注入SessionFactory对象，然后才可以进行持久化操作。

使用HibernateTemplate

- HibernateTemplate(org.hibernate.SessionFactory sessionFactory)
- HibernateTemplate(org.hibernate.SessionFactory sessionFactory, boolean allowCreate)

对于Web应用，通常应用启动时会自动创建ApplicationContext，SessionFactory和DAO对象都处在Spring容器的管理下，因此无需在代码中显式设值，可采用依赖注入实现SessionFactory和DAO的解耦，依赖关系通过配置文件来设值。

使用HibernateTemplate

- 示例：HibernateTemplate

- HibernateTemplate是Spring众多模板工具类之一，Spring正是通过这种简便的模板工具类，为我们完成了开发中大量需要重复进行的工作。

实现DAO组件

- 为了帮助实现DAO组件，Spring提供了大量的XxxDaoSupport类，这些DAO支持类对于实现DAO组件大有帮助，因为这些DAO支持类已经完成了大量基础性工作。
- Spring为Hibernate的DAO提供工具类：HibernateDaoSupport。该类主要提供如下两个方法来简化DAO的实现：
  - public final HibernateTemplate getHibernateTemplate()
  - public final void setSessionFactory(SessionFactory sessionFactory)

实现DAO组件

- 其中，setSessionFactory()方法可用于接收Spring的依赖注入，允许使用依赖注入Spring管理的SessionFactory实例。
- getHibernateTemplate()则用于返回一个HibernateTemplate对象。一旦获得了HibernateTemplate对象，剩下的DAO实现将由该HibernateTemplate来完成。

使用IoC容器组装各种组件

- 现在，Java EE应用所需的各种组件---从控制器组件，到业务逻辑组件，以及持久层的DAO组件---都已讲述，但这些组件并未直接耦合，组件与组件之间面向接口编程，所以还需要利用Spring的IoC容器将它们组合在一起。

- 从用户角度来看，用户发出HTTP请求，当MVC框架的控制器组件拦截到用户请求时，将调用系统的业务逻辑组件，业务逻辑组件则调用系统的DAO组件，而DAO组件则依赖于SessionFactory和DataSource等底层组件实现数据库访问。

使用IoC容器组装各种组件

- 从系统实现角度，IoC容器先创建SessionFactory和DataSource等底层组件，然后将这些底层组件注入给DAO组件，提供一个完整的DAO组件，并将此DAO组件注入给业务逻辑组件，从而提供一个完整的业务逻辑组件，而业务逻辑组件又被注入给控制器组件，控制器负责拦截用户请求，并将处理结果呈现给用户---这一系列的衔接，都是由Spring的IoC容器提供实现。