# Spring IOC

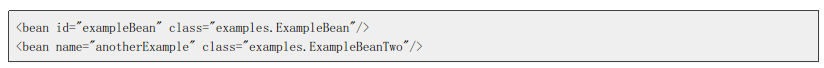
## 何为IOC?

通常我们用到的对象会用到其他对象一起协作工作，所以该对象会存在对其他对象的引用，如果这个获取过程要靠自身实现，那么代码将高度耦合。而IOC控制反转的解决方案就是将这个控制权从类自身转换到平台或者框架中。

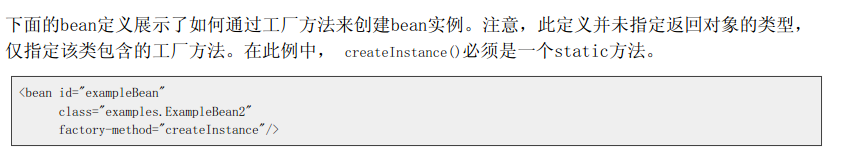
## Bean的定义

## Bean的实例化方式

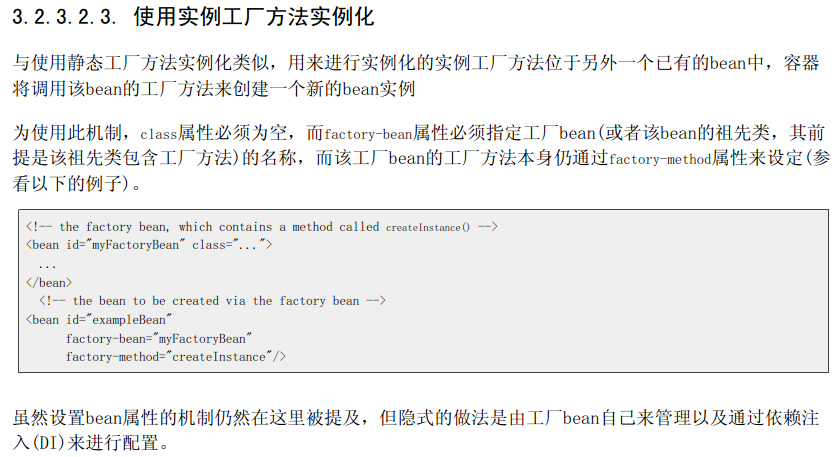
### 构造器（默认）



### 静态工厂方法



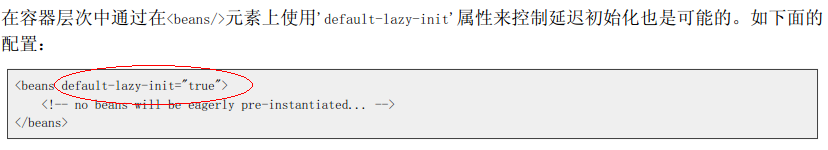
### 实例工厂方法

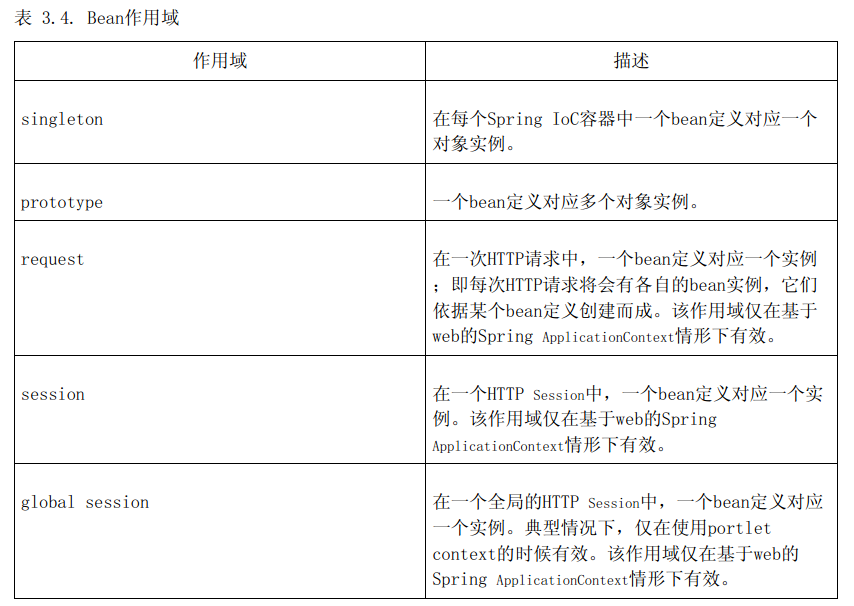


## Bean的作用域

Bean与容器同时被创建（默认）

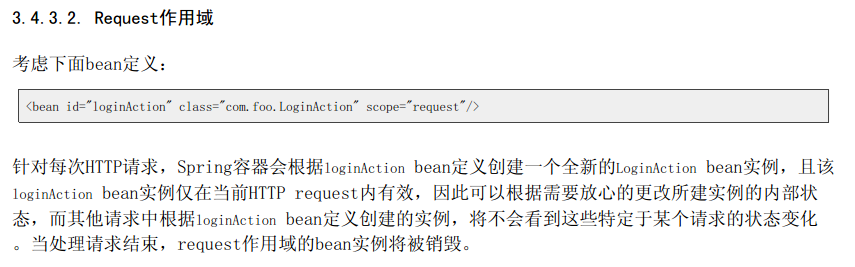
用到时被创建











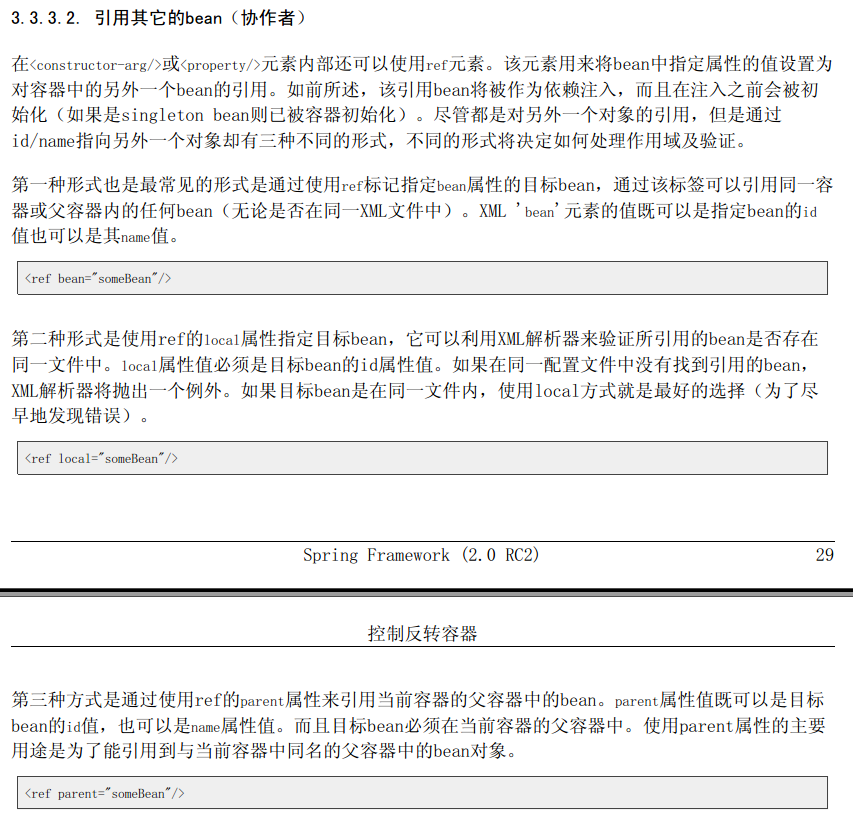


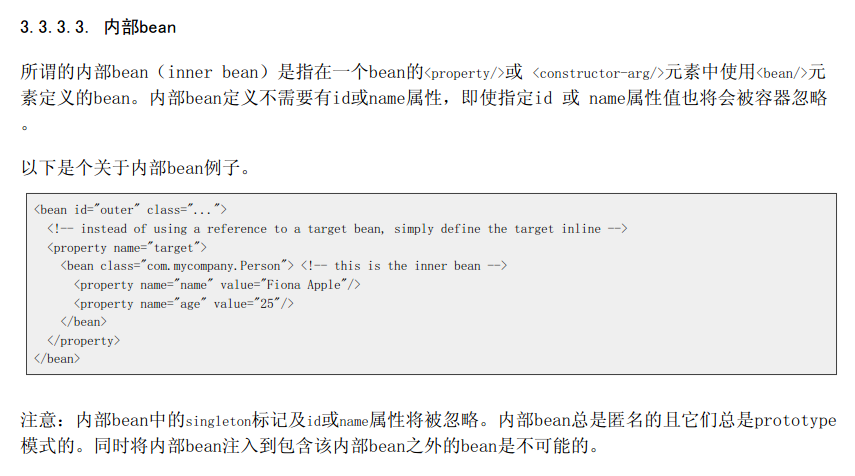


## Bean属性及构造参数的类型

****

****

****

****

****

## 依赖的注入方式

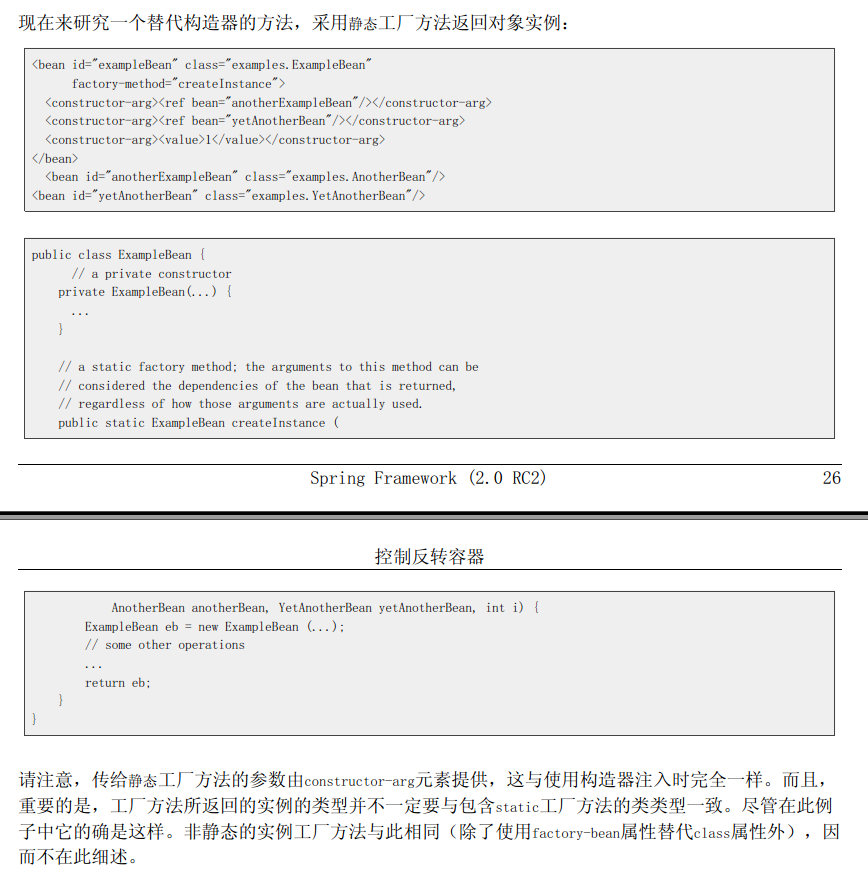
Setter注入

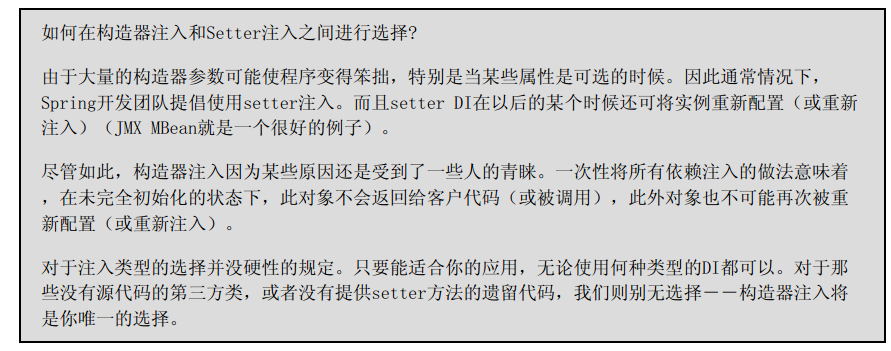


构造器注入

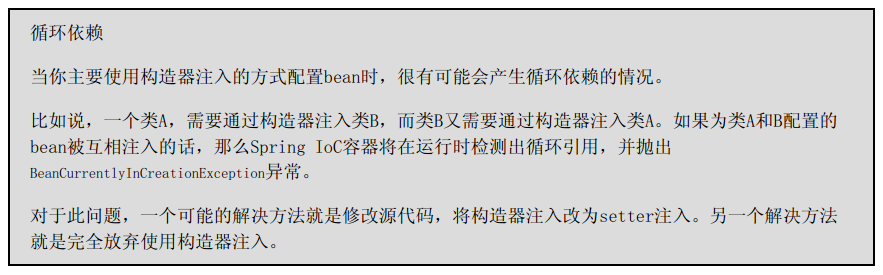


静态工厂注入：



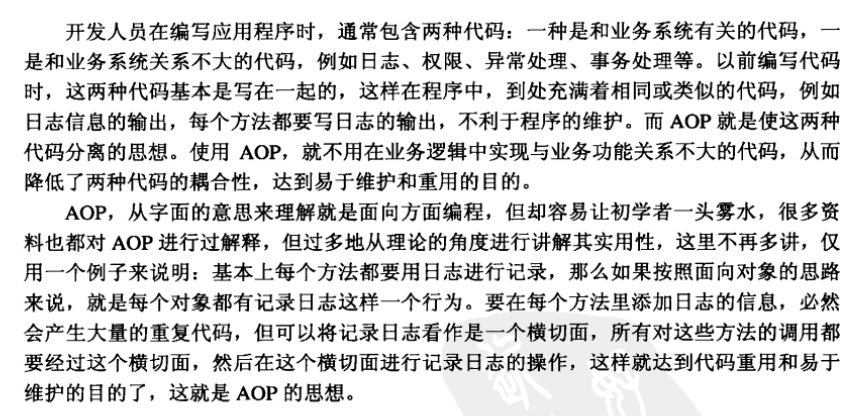


**注意：**



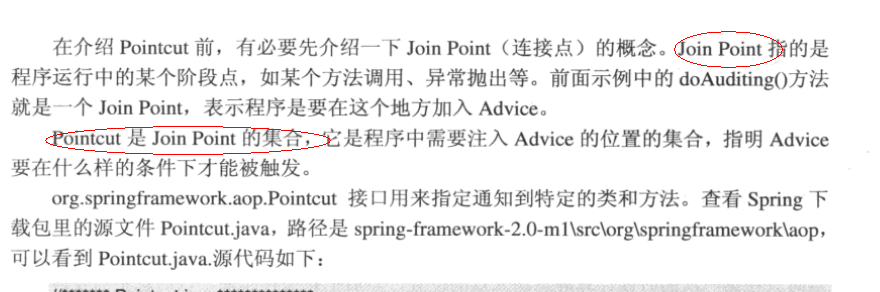
# Spring AOP

## AOP思想

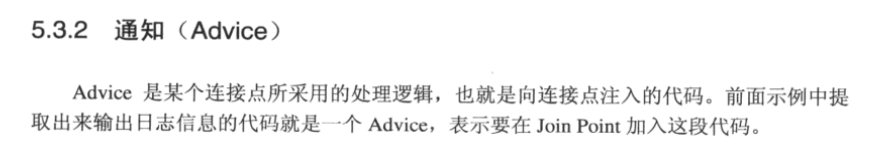


## AOP的三个关键概念

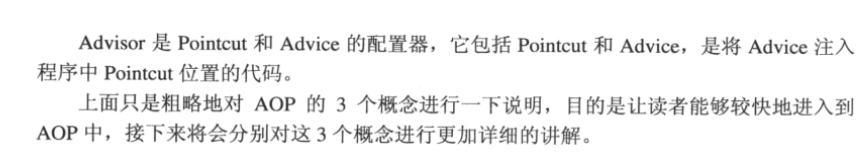
切入点



通知：（Advice）

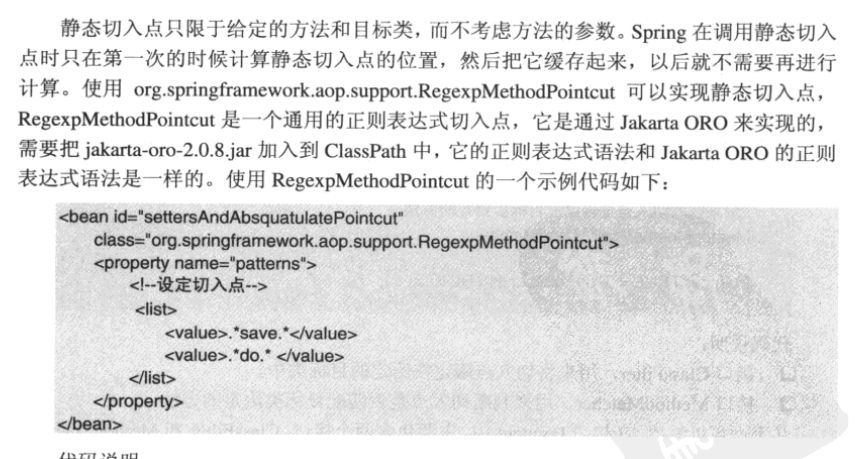


Advisor

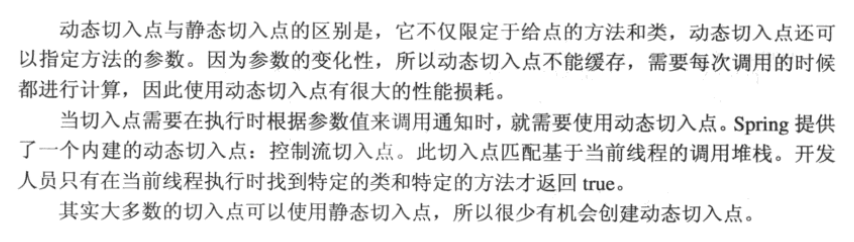


## Spring三种切入点的实现

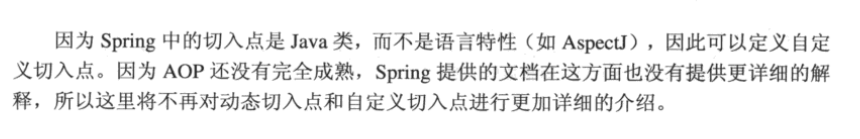
### 静态切入点：



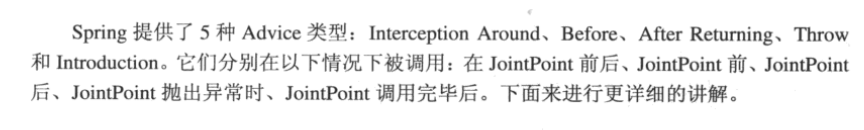
### 动态切入点：



### 自定义切入点：



## Spring通知

****

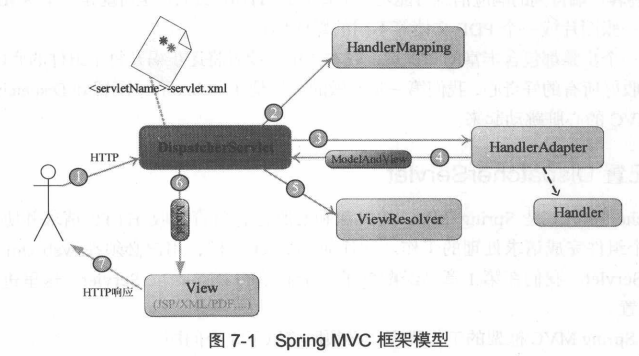
# spring事务管理

|  |
| --- |
| 第一种配置方法：基于XML的事务管理 这种方法不需要对原有的业务做任何修改，通过在XML文件中定义需要拦截方法的匹配即可完成配置，要求是，业务处理中的方法的命名要有规律，比如setXxx，xxxUpdate等等。详细配置如下：   1. <!-- 定义事务处理类，不同的数据访问方式，事务处理类不同 2. 比如：Hibernate操作的HibernateTransactionManager，JDBC操作的使用DataSourceTransactionManager 3. --**>** 4. **<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">** 5. **<property name="dataSource" ref="dataSource"></property>** 6. **</bean>** 8. <!-- 定义事务通知 --> 9. **<tx:advice id="txAdvice" transaction-manager="transactionManager">** 10. <!-- 定义方法的过滤规则 --> 11. **<tx:attributes>** 12. <!-- 所有方法都使用事务 --> 13. **<tx:method name="\*" propagation="REQUIRED"/>** 14. <!-- 定义所有get开头的方法都是只读的 --> 15. **<tx:method name="get\*" read-only="true"/>** 16. **</tx:attributes>** 17. **</tx:advice>** 19. <!-- 定义AOP配置 --> 20. **<aop:config>** 21. <!-- 定义一个切入点 --> 22. **<aop:pointcut expression="execution (\* com.iflysse.school.services.impl.\*.\*(..))" id="services"/>** 23. <!-- 对切入点和事务的通知，进行适配 --> 24. **<aop:advisor advice-ref="txAdvice" pointcut-ref="services"/>** 25. **</aop:config>**  第二中配置方法：基于@Transactional的事务管理 这种方法，只需要在Spring配置文件中定义一个事务管理对象（如DataSourceTransactionManager），然后加入<tx:annotation-driven/>节点，引用该事务管理对象，然后即可在需要进行事务处理的**类**和**方法**使用@Transactional进行标注。示例如下：  Spring配置XML文件   1. <!-- 定义事务处理类，不同的数据访问方式，事务处理类不同 2. 比如：Hibernate操作的HibernateTransactionManager，JDBC操作的使用DataSourceTransactionManager 3. --**>** 4. **<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">** 5. **<property name="dataSource" ref="dataSource"></property>** 6. **</bean>** 7. <!-- 声明使用注解式事务 --> 8. **<tx:annotation-driven transaction-manager="transactionManager"/>**   Java代码中@Transactional示例   1. /\*\* 2. \* @author ZYWANG 2011-3-24 3. \*/ 4. @Service 5. @Transactional(propagation = Propagation.REQUIRED) 6. **public class TeacherServicesImpl implements TeacherServices {** 7. @Autowired 8. TeacherDao dao = **null;** 10. **public TeacherDao getDao() {** 11. **return dao;** 12. } 14. **public void setDao(TeacherDao dao) {** 15. **this.dao = dao;** 16. } 18. @Override 19. @Transactional(readOnly=**true,propagation=Propagation.NEVER)** 20. **public List<Teacher> getAllTeachers() {** 21. **return getDao().getAllTeachers();** 22. } 24. @Override 25. **public void addTeacher(Teacher teacher) {** 26. **if(teacher.getId() > 0){** 27. getDao().updateTeacher(teacher); 28. }**else{** 29. getDao().addTeacher(teacher); 30. } 31. } 32. }      个人认为第一中方式的配置的侵入性低多了，写程序时只需要关注业务和命名规范，不需要考虑事务处理；第二种方式的配置可以更加灵活的使用事务，个性化定制。 |

# Spring MVC 框架

## 什么是Spring MVC

## Spring MVC的框架原理



## 入门程序

### 引入依赖jar包

### 在Web.xml配置前端控制器

/SSM/WebRoot/WEB-INF/web.xml



### 配置处理器适配器、处理器映射器

/SSM/config/springMVC/spring-mvc-conf.xml

非注解的处理器映射器和非注解的处理器适配器：

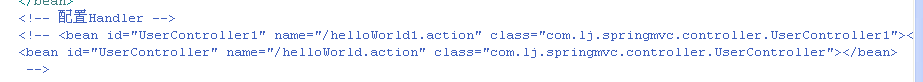


### 编写Handler

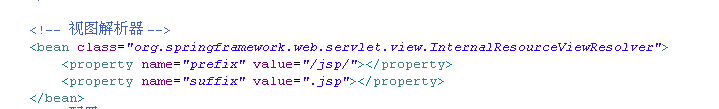
实现方式一：继承Controller接口



### 配置Handler



### 配置视图解析器



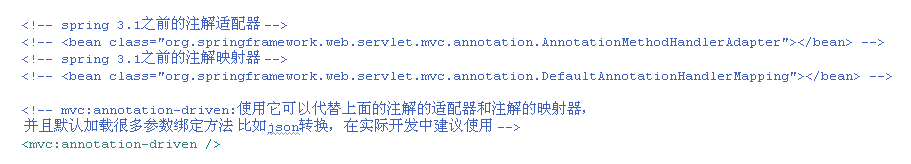
## 处理器适配器和处理器映射器

/org/springframework/web/servlet/DispatcherServlet.properties定义了默认的处理器适配器和处理器映射器

非注解的处理器适配器和非注解的处理器映射器配置：



注解的处理器适配器和注解的处理器映射器：



## 参数绑定

### 默认支持类型

HTTPRequest

HTTPResponse

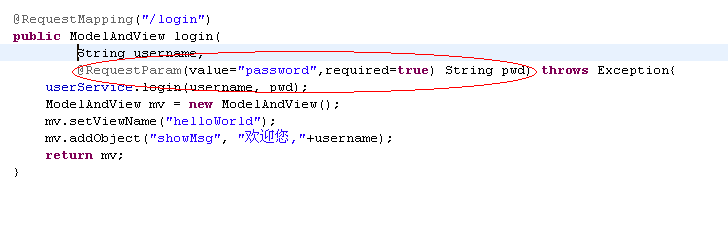
HTTPSession

Model/ModelMap

### 简单类型

简单类型包括整型、字符、单精、双精、布尔

通过@RequestParam对简单参数类型进行绑定，如果不使用则要求request中参数名称和controller方法参数名称一致方可绑定成功。



### POJO

页面中的name与POJO中属性名称一致才可绑定成功。

### 自定义参数绑定

对于controller入参POJO中有Date类型，则需要自定义参数绑定。

方法：

需要向处理器适配器中注入自定义的转换器

|  |
| --- |
| <!-- mvc:annotation-driven:使用它可以代替上面的注解的适配器和注解的映射器，  并且默认加载很多参数绑定方法 比如json转换，在实际开发中建议使用 -->  <mvc:annotation-driven conversion-service=*"conversionService"* />    <!-- 自定义参数转换器 -->  <bean id=*"conversionService"* class=*"org.springframework.format.support.FormattingConversionServiceFactoryBean"*>  <property name=*"converters"*>  <list>  <!-- 日期类型转换器 -->  <bean class=*"com.lj.springmvc.controller.converter.DateConverter"*></bean>  </list>  </property>  </bean> |

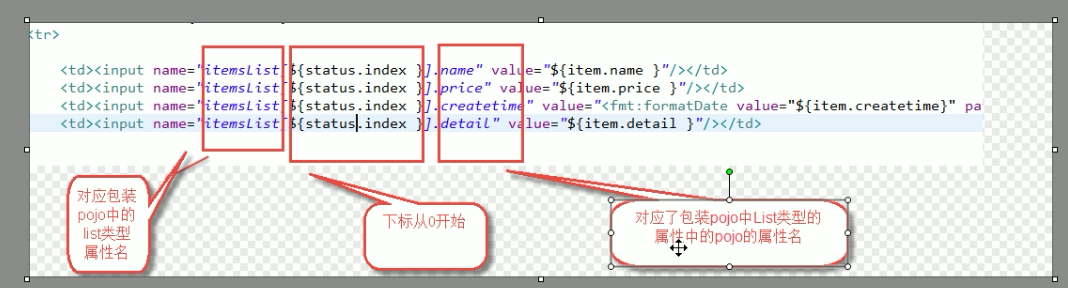
### 包装类型的参数绑定

包装类型：User类中包装了Address类即PoJO中属性是Pojo。

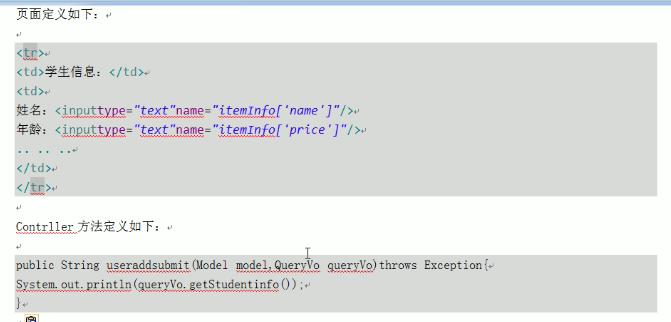
可以使用类型OGNL的形式进行绑定，即在页面中将name定义为address.addressName即可。

### 集合类型

List

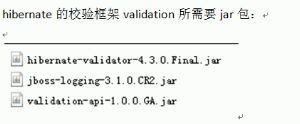


MAP



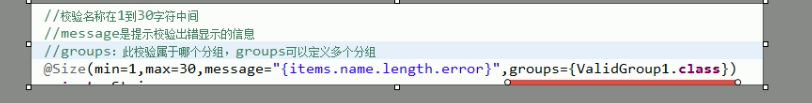
## 数据校验

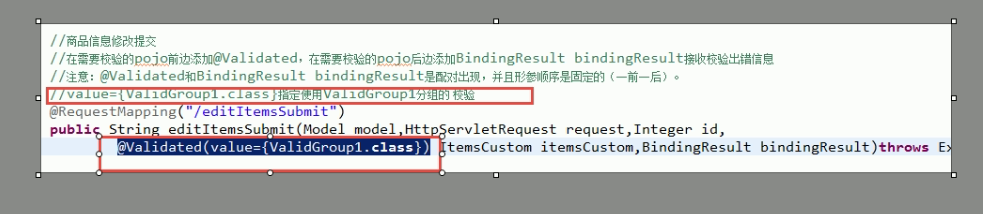
1.引入校验需要的jar包，这里使用hibernate提供的校验框架。



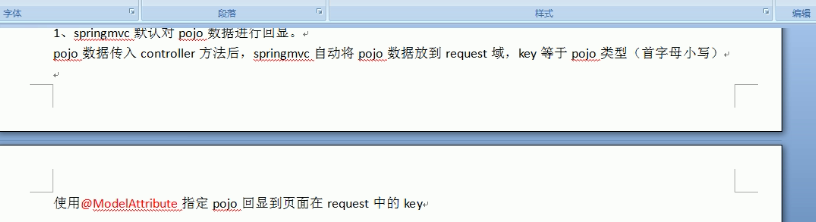
2.

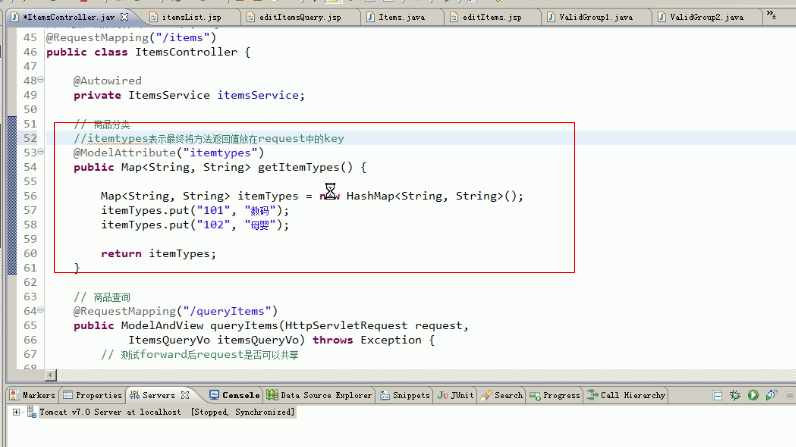
校验分组

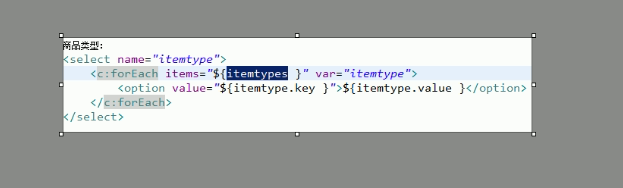




## 数据回显







## 文件上传

## 异常处理

## 拦截器

## Spring mvc和mybatis整合

加入依赖jar包

Sqlmapconfig.xml

applicationContext-dao.xml

# Spring-activiti整合

# Spring-quartz整合

# 附件

Web.xml

Spring-mvc.xml

# 问题

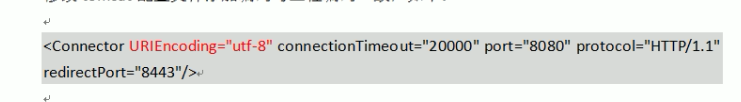
## Post提交乱码

在web.xml中添加过滤器

|  |
| --- |
| <filter>  <filter-name>encodingFilter</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>encodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

## Get提交乱码

解决方案一：



解决方案二：

