# 一、Spring的IOC及bean管理

## 1. Spring课程安排

1. spring概念和ioc入门（ioc操作xml配置文件）
2. spring的ioc操作（注解）和aop概念
3. jdbcTemplate和spring事务管理
4. ssh框架整合开发

## 2. Spring概念

1. Spring是开源轻量级框架
2. Spring核心主要两部分：
3. AOP：面向切面编程
4. IOC：控制反转 对象的创建不是通过new方式实现，而是交给spring配置创建类对象
5. Spring是一站式框架
6. spring在javaee三层结构中，每一层都提供不同的解决技术

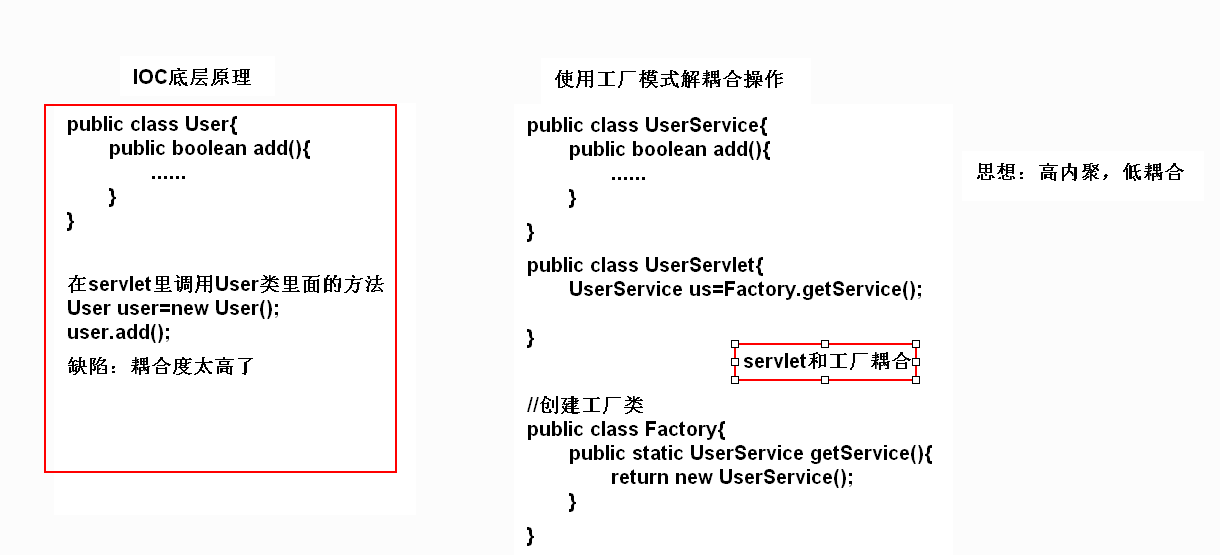
Web层：springMVC

Service层：spring的IOC

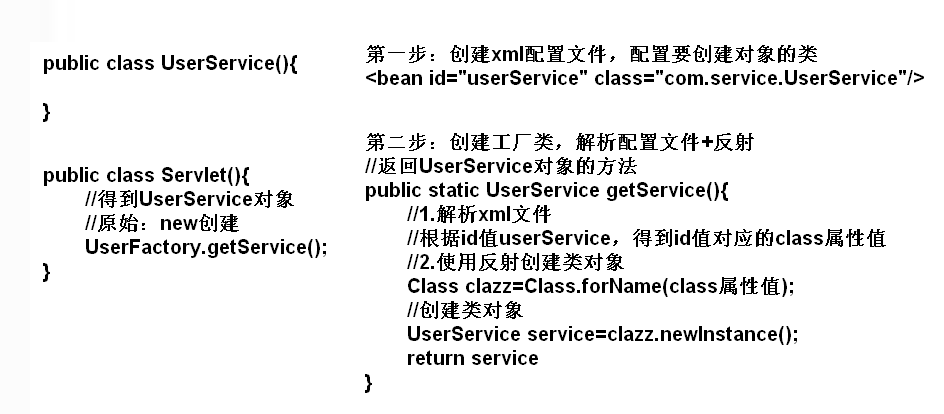
Dao层：Spring的jdbcTemplate

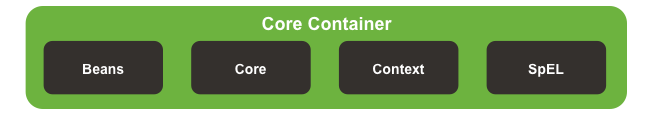
## Spring入门（IOC）

IOC底层原理

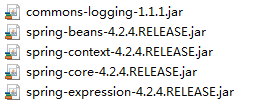


### 第一步：导入jar包

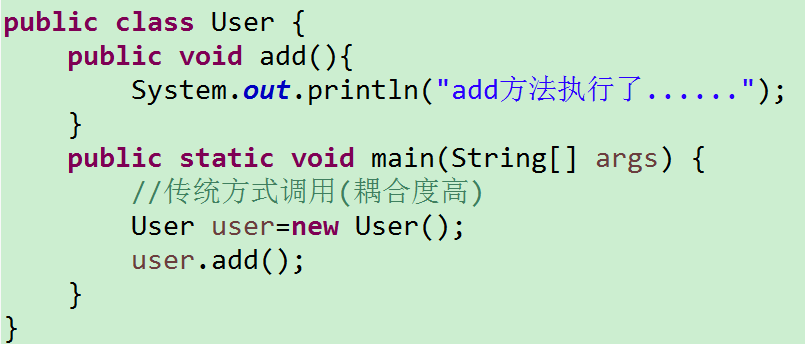




4个核心包+1个依赖



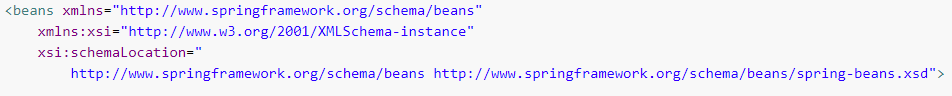
### 第二步：创建类，在类里面写方法



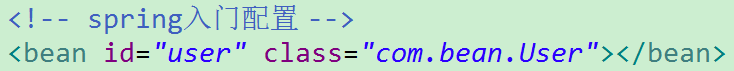
### 第三步：创建spring配置文件，配置创建类

1. spring核心配置文件的名称和位置不是固定的，一般建议放在src下，名称applicationContext.xml
2. 引入schema约束

位置**spring-framework-4.2.4.RELEASE\docs\spring-framework-reference\html**



1. 配置对象创建



### 第四步：编写代码测试对象创建

**public** **class** SpringTest {

@Test

**public** **void** testUser(){

//1.加载spring配置文件，根据配置创建对象

ApplicationContext context=**new** ClassPathXmlApplicationContext("applicationContext.xml");

//2.得到配置创建对象

User user = (User) context.getBean("user");

System.***out***.println(user);

user.add();

}

}

## Spring的bean管理（XML方式）

**Bean实例化的方式**

1. 在spring里面通过配置文件创建对象
2. Bean实例化三种方式实现

### 第一种：使用类的无参构造（重点）

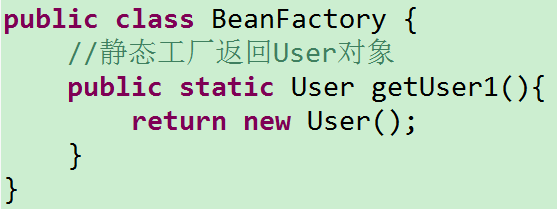


类里面没有无参构造出现异常



### 第二种：使用静态工厂创建

1. 创建静态的方法，返回类对象

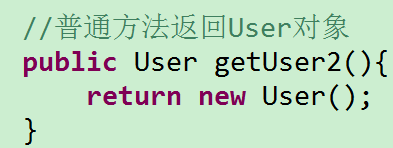


<!-- 使用静态工厂创建对象 -->

<bean id=*"user1"* class=*"com.factory.BeanFactory"* factory-method=*"getUser1"*></bean>

### 第三种：使用实例工厂创建

1. 创建不是静态的方法，返回类对象



<!-- 使用实例工厂创建对象 -->

<bean id=*"beanFactory"* class=*"com.factory.BeanFactory"*></bean>

<bean id=*"user2"* factory-bean=*"beanFactory"* factory-method=*"getUser2"*></bean>

### Bean标签常用属性

1. id属性：起名称，不能包含特殊符号，根据id获得配置对象
2. class属性：创建对象所在类的全路径
3. name属性：功能和id属性相同的，id属性值中不能包含特殊符号，但是name可以
4. scope属性：设置bean的作用范围

singleton：单例（默认值）,只能创建一个对象

prototype：多例的

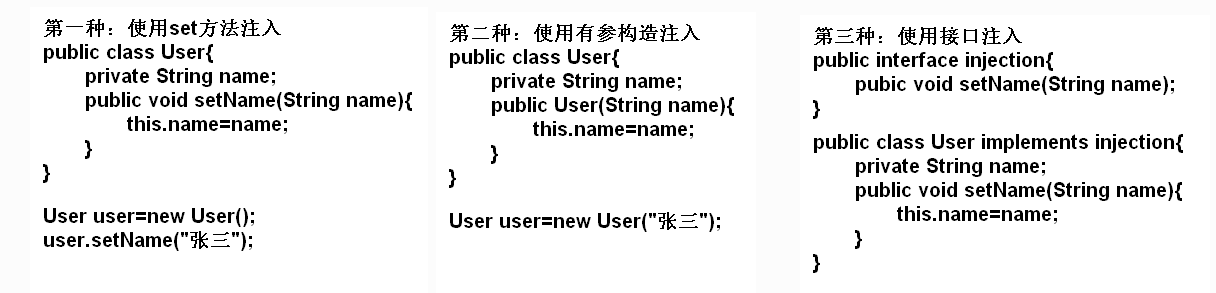
request：将spring创建的对象存放到request域中

session：将spring创建的对象存放到session域中

globalSession：将spring创建的对象存放到globalSession域中（单点登录）

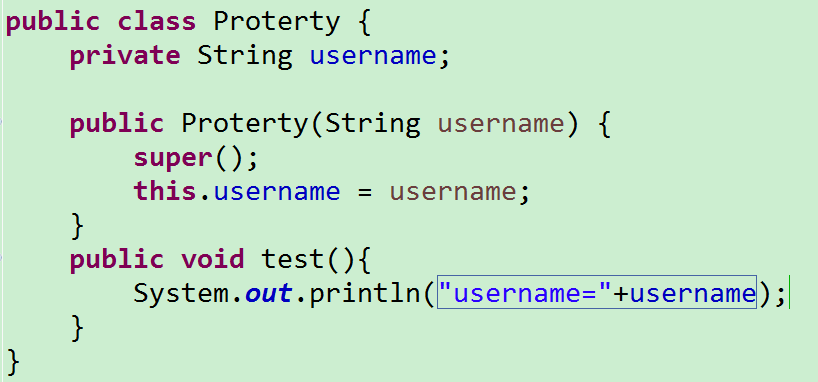
### 属性注入（依赖注入DI）

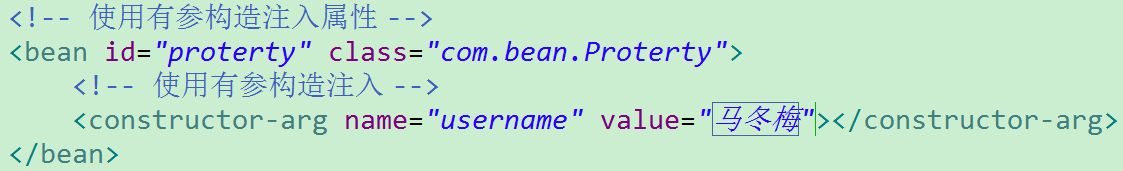
1. 创建对象的时候，给bean中的属性设置值
2. 属性注入的方式介绍（三种）
3. 使用set方法注入
4. 有参构造
5. 使用接口注入



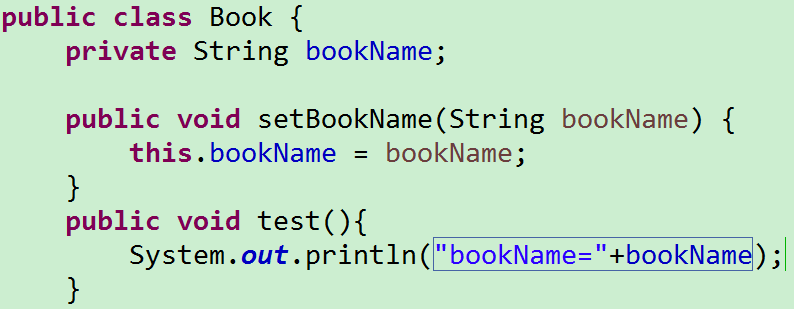
1. 在spring框架里面，支持前两种方式
2. set方法注入（重要）
3. 有参构造注入

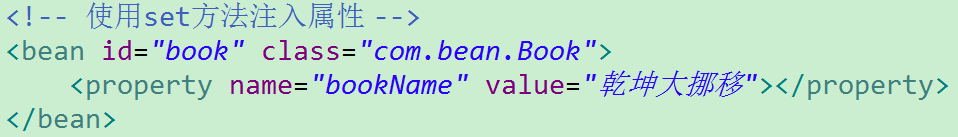
#### 使用有参构造注入属性





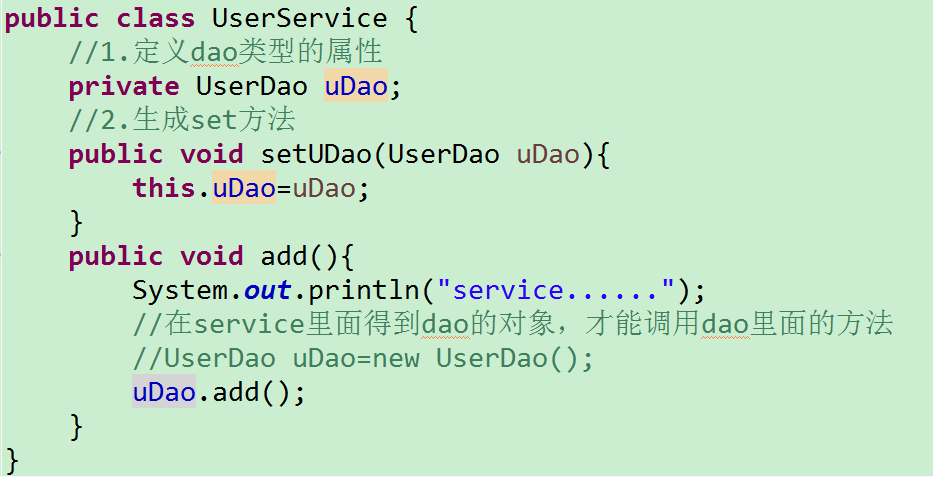
#### 使用set方法注入属性（重点）



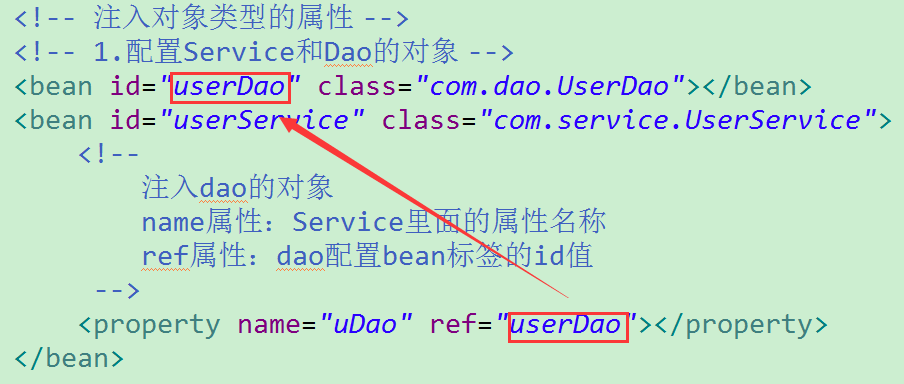


#### 注入对象类型的属性（重点）

1. **创建Service类，创建Dao类，在Service得到Dao的对象**
2. **具体实现过程**
3. **在service里面把dao作为类型属性**
4. **生成dao类型属性的set方法**

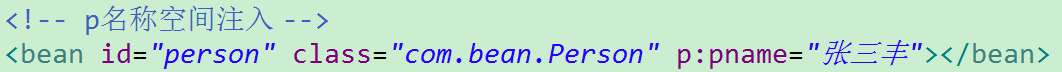


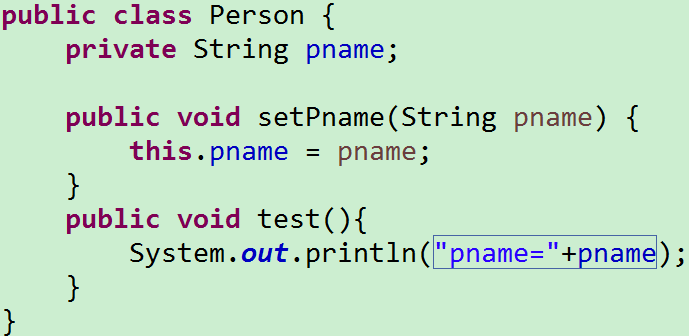
1. **配置文件注入关系**



**P名称空间注入，引入p名称空间**







#### 注入复杂类型的属性

1. **数组**
2. **List集合**
3. **Map集合**
4. **Properties类型**

<!-- 注入复杂类型的属性值 -->

<bean id=*"person"* class=*"com.bean.Person"*>

<!-- 数组 -->

<property name=*"array"*>

<list>

<value>张三</value>

<value>李四</value>

<value>王五</value>

</list>

</property>

<!-- List集合 -->

<property name=*"list"*>

<list>

<value>科比</value>

<value>詹姆斯</value>

<value>奥尼尔</value>

</list>

</property>

<!-- Map集合 -->

<property name=*"map"*>

<map>

<entry key=*"呼保义"* value=*"宋江"*></entry>

<entry key=*"玉麒麟"* value=*"卢俊义"*></entry>

<entry key=*"智多星"* value=*"吴用"*></entry>

</map>

</property>

<!-- Properties类型 -->

<property name=*"properties"*>

<props>

<prop key=*"driverClass"*>com.mysql.jdbc.Driver</prop>

<prop key=*"username"*>root</prop>

</props>

</property>

</bean>

#### IOC和DI的关系

1. **IOC：控制反转 把对象的创建交给spring进行管理**
2. **DI：依赖注入 向类里面的属性中设置值**
3. **关系：依赖注入不能单独存在，需要在IOC基础之上完成操作**

## Spring整合web项目原理

### 加载spring核心配置文件

//1.加载spring配置文件，根据配置创建对象

ApplicationContext context=**new** ClassPathXmlApplicationContext("applicationContext.xml");

（1）new对象，功能可以实现，效率低

1. 实现思想：把加载配置文件和创建对象的过程，在服务器启动时候完成

### 实现原理：

1. ServletContext对象
2. 监听器
3. 具体使用：

在服务器启动的时候，为每个项目创建一个ServletContext对象，在ServletContext对象创建的时候，使用监听器可以监听到ServletContext对象什么时候创建，使用监听器监听到ServletContext对象创建的时候，加载spring配置文件，把配置文件中配置的对象创建，把创建出来的对象放到ServletContext域对象里面（setAttribute方法）；获取对象的时候，到ServletContext域得到（getAttribute方法）

# 二、注解及AOP操作

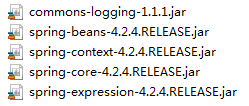
## 1. Spring的Bean管理（注解）

**注解介绍**

1. 代码里面的特殊标记，使用注解可以完成功能
2. 注解写法：@注解名称(属性名称=属性值)
3. 注解可以用在类，方法和属性上面

### Spring注解开发准备

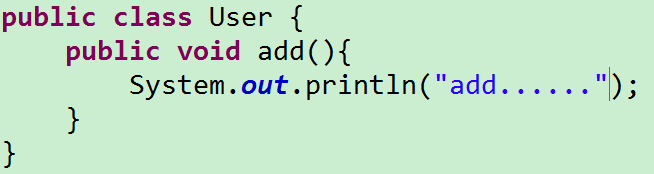
1. **导入jar包**
2. 导入基本的jar包



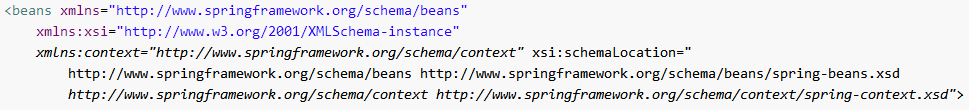
1. 导入aop的jar包



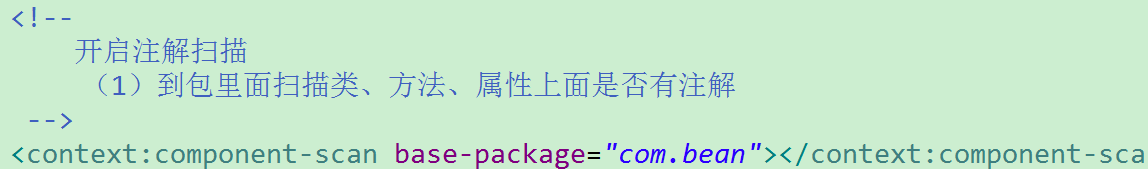
1. **创建类，创建方法**



1. **创建spring配置文件，引入约束**

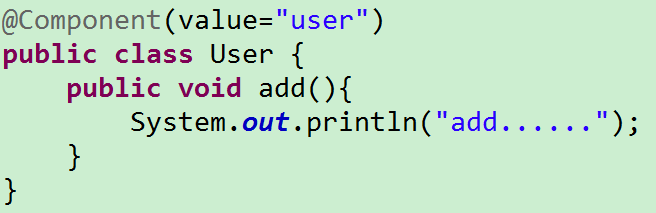


1. **开启注解扫描**



### 注解创建对象

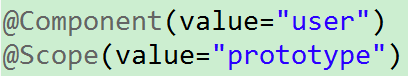
1. **在创建对象的类上面使用注解实现**



1. **创建对象有4个注解**
2. @Component
3. @Controller WEB层
4. @Service 业务层
5. @Repository 持久层

目前这四个注解功能是一样的，都能创建对象

1. 创建对象是单实例还是多实例



单例设计模式的特点

1、单例类只能有一个实例。

2、单例类必须自己创建自己的唯一实例。

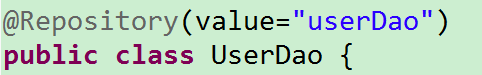
3、单例类必须给所有其他对象提供这一实例。

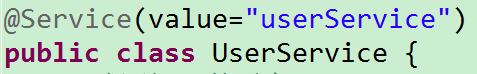
4. 注解注入属性

1. 创建Service类，创建Dao类，在Service得到Dao的对象

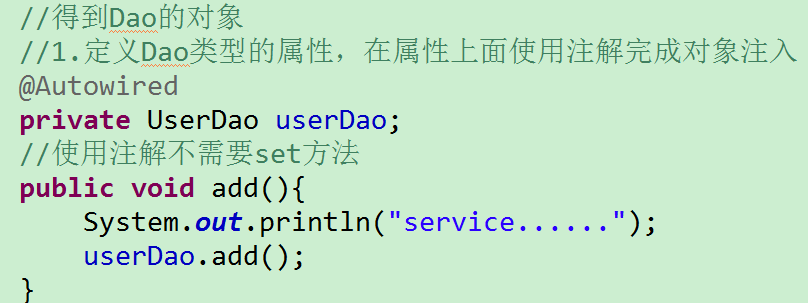
注入属性第一个注解@Autowired（自动装配）

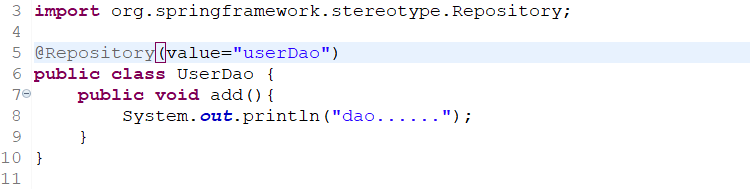
1. 创建Dao和Service对象



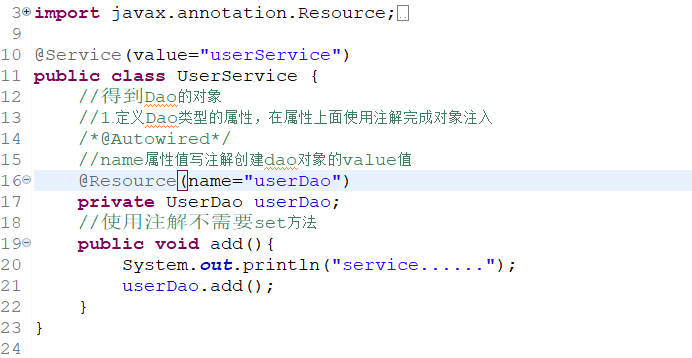


1. 在Service里面定义Dao类型的属性



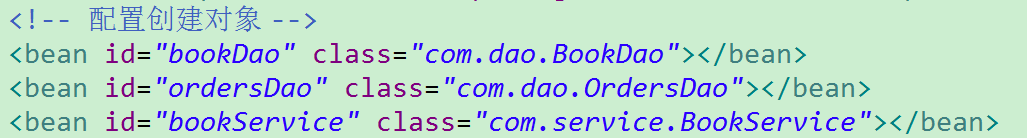


注入属性第二个注解@Resource



### 3.配置文件和注解混合使用创建对象

1. 创建对象操作使用配置文件方式实现



1. 注入属性的操作使用注解方式实现

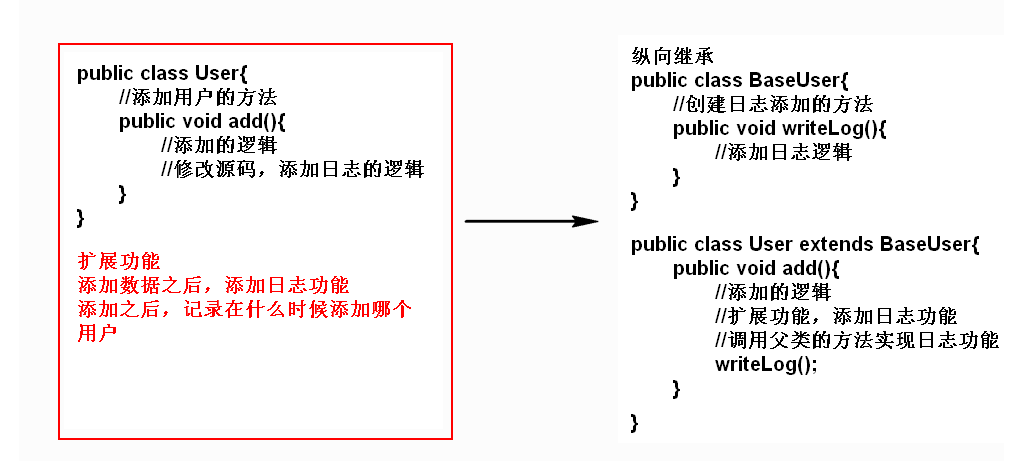


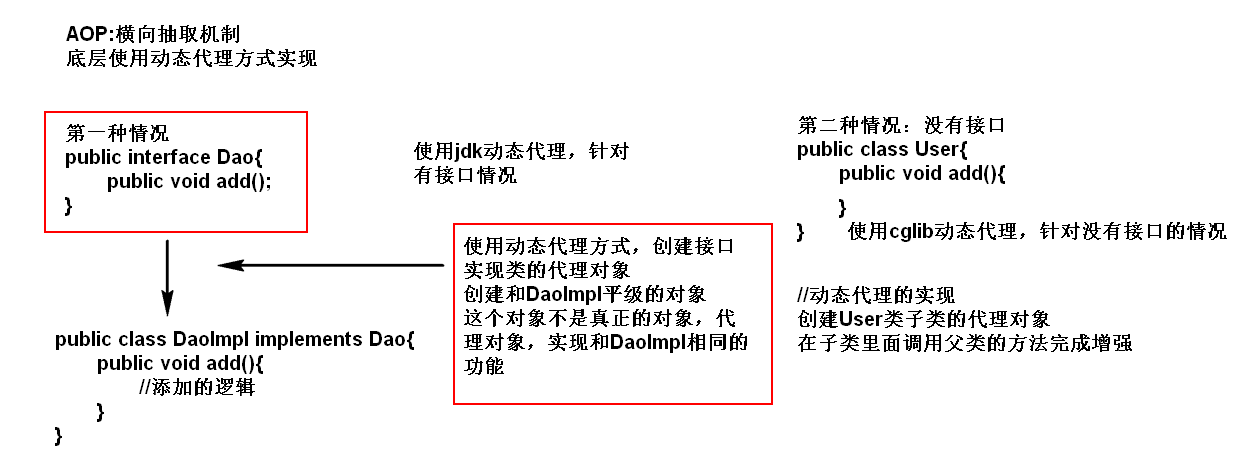
## AOP

### AOP概念

1. AOP：面向切面编程，扩展功能不修改源代码实现
2. AOP采用横向抽取，取代了传统纵向继承体系的重复性代码

### AOP底层原理





### AOP操作术语

Joinpoint(连接点):类里面可以被增强的方法，这些方法称为连接点

Pointcut(切入点):所谓切入点是指我们要对哪些Joinpoint进行拦截的定义.

Advice(通知/增强):所谓通知是指拦截到Joinpoint之后所要做的事情就是通知.通知分为前置通知,后置通知,异常通知,最终通知,环绕通知(切面要完成的功能)

Aspect(切面):是切入点和通知（引介）的结合

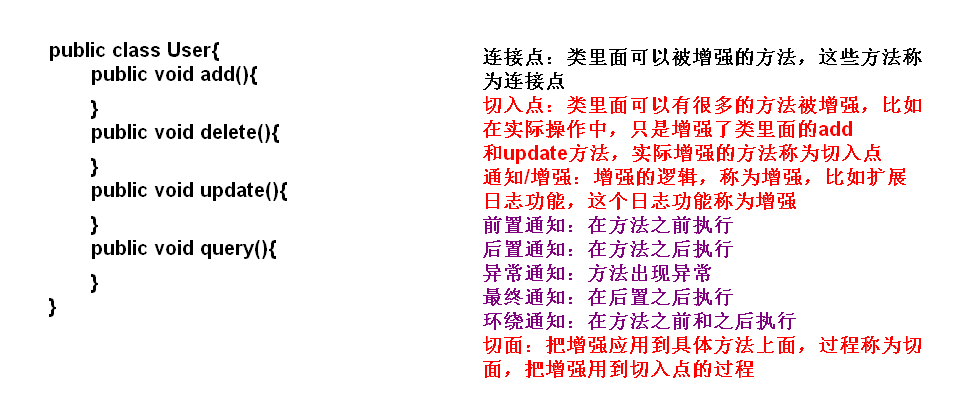
Introduction(引介):引介是一种特殊的通知在不修改类代码的前提下,Introduction可以在运行期为类动态地添加一些方法或Field.

Target(目标对象):代理的目标对象(要增强的类)

Weaving(织入):是把增强应用到目标的过程.

把advice应用到target的过程

Proxy（代理）:一个类被AOP织入增强后，就产生一个结果代理类

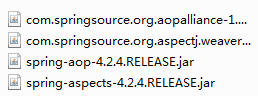


## Spring的AOP操作

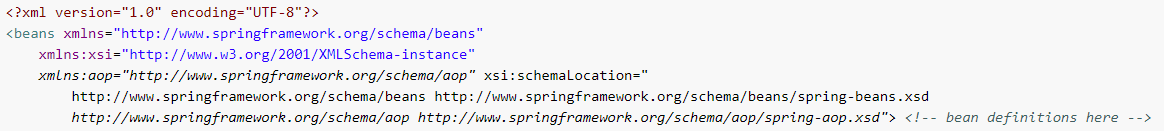
1. 在spring里面进行AOP操作，使用aspectj实现
2. Aspectj是一个AOP的框架，Spring引入aspectj作为自身的AOP开发
3. 使用aspectj实现aop有两种方式
4. 基于aspectj的xml配置
5. 基于aspectj的注解方法

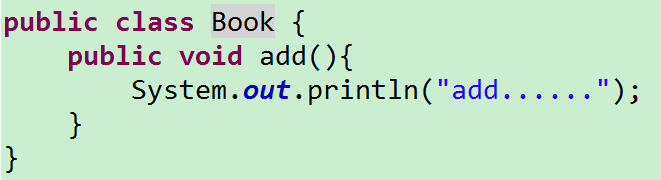
**AOP操作准备**

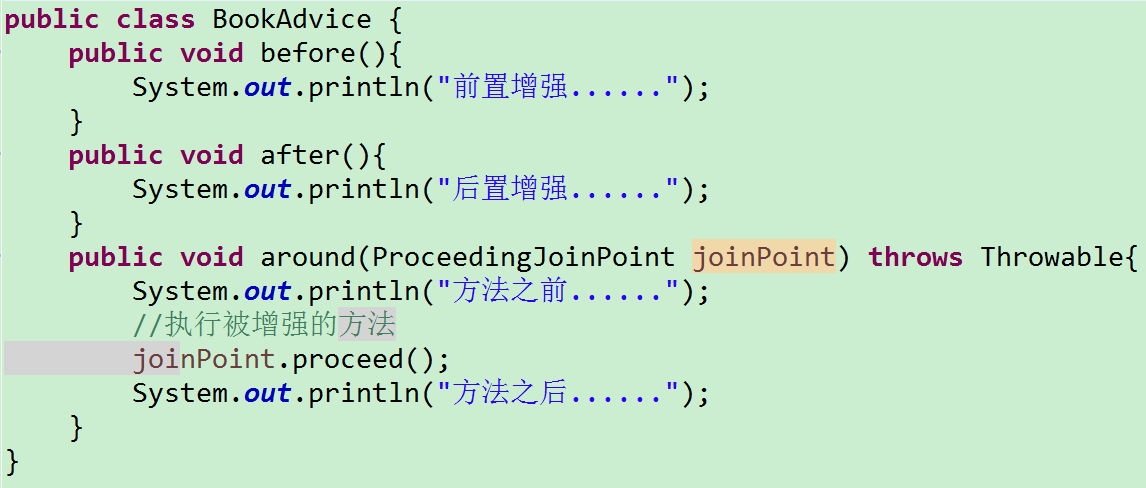
1. 除了导入基本的jar包外，还需要导入AOP相关的jar包



### 创建Spring核心配置文件，导入aop的约束







### 使用表达式配置切入点

1. 切入点：实际增强的方法
2. 语法：execution(访问权限修饰符 方法的返回值 包名.类名.方法名(参数))

execution(public void com.aop.Book.add())

1. 常用的表达式：
2. execution(\* com.aop.Book.add(..))
3. execution(\* com.aop.Book.\*(..))
4. execution(\* \*.\*(..))

<!-- 1.配置目标对象 -->

<bean id=*"book"* class=*"com.aop.Book"*></bean>

<!-- 2.配置通知对象 -->

<bean id=*"bookAdvice"* class=*"com.aop.BookAdvice"*></bean>

<!-- 3.配置AOP操作 -->

<aop:config>

<!-- 3.1配置切入点 -->

<aop:pointcut expression=*"execution(\* com.aop.Book.\*(..))"* id=*"pointcut1"*/>

<!-- 3.2配置切面 -->

<aop:aspect ref=*"bookAdvice"*>

<!--

配置增强的类型

method属性：增强类里面使用哪个方法作为增强

-->

<aop:before method=*"before"* pointcut-ref=*"pointcut1"*/>

<aop:after-returning method=*"after"* pointcut-ref=*"pointcut1"*/>

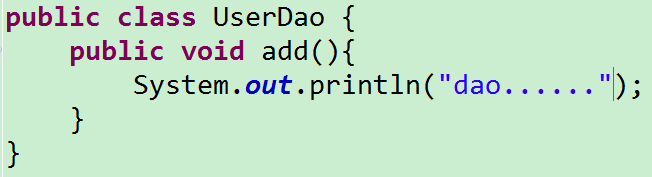
<aop:around method=*"around"* pointcut-ref=*"pointcut1"*/>

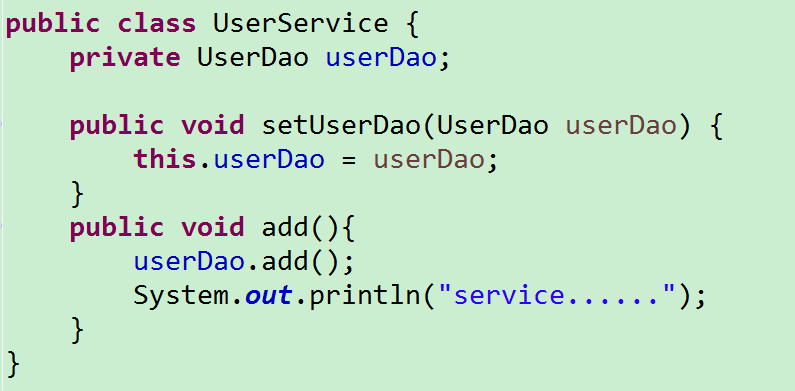
</aop:aspect>

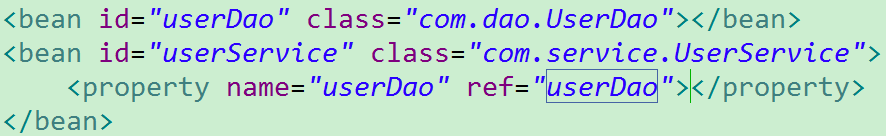
</aop:config>

## Spring整合Web项目演示

1. 演示问题
2. action调用service，service调用dao







**public** **class** UserAction **extends** ActionSupport{

@Override

**public** String execute() **throws** Exception {

System.***out***.println("action......");

ApplicationContext context=**new** ClassPathXmlApplicationContext("applicationContext.xml");

UserService userService=(UserService)context.getBean("userService");

userService.add();

**return** ***NONE***;

}

}

每次访问action的时候，都会加载Spring配置文件

1. 解决方案
2. 在服务器启动的时候，创建对象加载配置文件
3. 底层使用ServletContext对象、监听器

### 配置监听器(web.xml)

1. 导入spring整合web项目的jar包



<!-- 配置监听器 -->

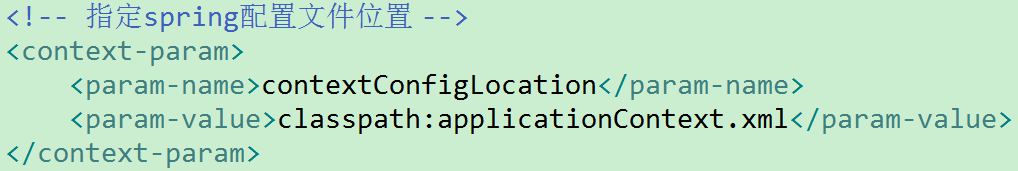
<listener>

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

</listener>

1. 指定加载spring配置文件位置





1. 从ServletContext域中获得配置对象

ServletContext sc = ServletActionContext.*getServletContext*();

WebApplicationContext context = WebApplicationContextUtils.*getWebApplicationContext*(sc);

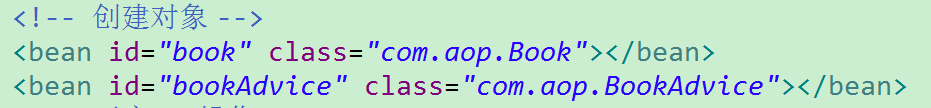
UserService userService=(UserService)context.getBean("userService");

# 三、AOP的注解操作/jdbcTemplate/事务

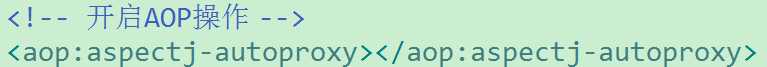
## 1. 基于Aspectj的注解方式实现aop

1. 使用注解方式实现aop操作

第一步：创建对象



第二步：在spring核心配置文件中开启AOP操作



第三步：在增强类上面使用注解完成AOP操作

@Aspect

**public** **class** BookAdvice {

@Before(value="execution(\* com.aop.Book.\*(..))")

**public** **void** before(){

System.***out***.println("前置增强......");

}

@AfterReturning(value="execution(\* com.aop.Book.\*(..))")

**public** **void** after(){

System.***out***.println("后置增强......");

}

@Around(value="execution(\* com.aop.Book.\*(..))")

**public** **void** around(ProceedingJoinPoint joinPoint) **throws** Throwable{

System.***out***.println("方法之前......");

//执行被增强的方法

joinPoint.proceed();

System.***out***.println("方法之后......");

}

}

## Spring的jdbcTemplate操作

1. Spring是一站式框架
2. spring在javaEE三层结构中，每一层都提供不同的解决技术
3. 在dao层，使用jdbcTemplate
4. Spring对不同的持久层技术都进行了封装
5. jdbcTemplate对jdbc进行了封装

**增加**

1. 导入jdbcTemplate使用的jar包



1. 创建数据源，设置数据库信息
2. 创建jdbcTemplate，设置数据源
3. 调用jdbcTemplate对象里面的方法实现操作

//添加

@Test

**public** **void** testAdd(){

//创建数据源，设置数据库信息

DriverManagerDataSource dataSource=**new** DriverManagerDataSource();

dataSource.setDriverClassName("com.mysql.jdbc.Driver");

dataSource.setUrl("jdbc:mysql://localhost:3306/userdb");

dataSource.setUsername("root");

dataSource.setPassword("root");

//创建jdbcTemplate，设置数据源

JdbcTemplate jdbcTemplate=**new** JdbcTemplate();

jdbcTemplate.setDataSource(dataSource);

String sql="insert into t\_user(name,password,email) values(?,?,?)";

//调用jdbcTemplate对象里面的方法实现操作

**int** rows = jdbcTemplate.update(sql, "张三丰","123456","zsf@qq.com");

System.***out***.println(rows)

}

### Spring配置数据源

<!-- 配置数据源 -->

<bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*></property>

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/userdb"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"root"*></property>

</bean>

<!-- 配置jdbcTemplate -->

<bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

ApplicationContext context=**new** ClassPathXmlApplicationContext("applicationContext.xml");

JdbcTemplate jdbcTemplate=(JdbcTemplate)context.getBean("jdbcTemplate");

String sql="insert into t\_user(name,password,email) values(?,?,?)";

//调用jdbcTemplate对象里面的方法实现操作

**int** rows = jdbcTemplate.update(sql, "张无忌","123456","zwj@qq.com");

#### 删除

//删除

@Test

**public** **void** testDelete(){

ApplicationContext context=**new** ClassPathXmlApplicationContext("applicationContext.xml");

JdbcTemplate jdbcTemplate=(JdbcTemplate)context.getBean("jdbcTemplate");

String sql="delete from t\_user where name=?";

//调用jdbcTemplate对象里面的方法实现操作

**int** rows = jdbcTemplate.update(sql, "张无忌");

System.***out***.println(rows);

}

#### 修改

//修改

@Test

**public** **void** testUpdate(){

ApplicationContext context=**new** ClassPathXmlApplicationContext("applicationContext.xml");

JdbcTemplate jdbcTemplate=(JdbcTemplate)context.getBean("jdbcTemplate");

String sql="update t\_user set password=? where name=?";

//调用jdbcTemplate对象里面的方法实现操作

**int** rows = jdbcTemplate.update(sql, "888", "张三丰");

System.***out***.println(rows);

}

查询

#### 查询1 查询返回某一个值

//查询表中有多少条记录

@Test

**public** **void** testCount(){

ApplicationContext context=**new** ClassPathXmlApplicationContext("applicationContext.xml");

JdbcTemplate jdbcTemplate=(JdbcTemplate)context.getBean("jdbcTemplate");

String sql="select count(\*) from t\_user";

//调用jdbcTemplate对象里面的方法实现操作

Integer nums = jdbcTemplate.queryForObject(sql, Integer.**class**);

System.***out***.println(nums);

}

#### 查询2 查询返回对象

//查询返回对象

@Test

**public** **void** testObject(){

ApplicationContext context=**new** ClassPathXmlApplicationContext("applicationContext.xml");

JdbcTemplate jdbcTemplate=(JdbcTemplate)context.getBean("jdbcTemplate");

String sql="select \* from t\_user where name=?";

//调用jdbcTemplate对象里面的方法实现操作

RowMapper<User> rowMapper=**new** BeanPropertyRowMapper<User>(User.**class**);

User user = jdbcTemplate.queryForObject(sql, rowMapper, "张三丰");

System.***out***.println(user);

}

#### 查询2 查询返回list集合

//查询返回集合

@Test

**public** **void** testList(){

ApplicationContext context=**new** ClassPathXmlApplicationContext("applicationContext.xml");

JdbcTemplate jdbcTemplate=(JdbcTemplate)context.getBean("jdbcTemplate");

String sql="select \* from t\_user";

//调用jdbcTemplate对象里面的方法实现操作

RowMapper<User> rowMapper=**new** BeanPropertyRowMapper<User>(User.**class**);

List<User> list= jdbcTemplate.query(sql, rowMapper);

System.***out***.println(list);

}

### Spring配置连接池和dao使用jdbcTemplate

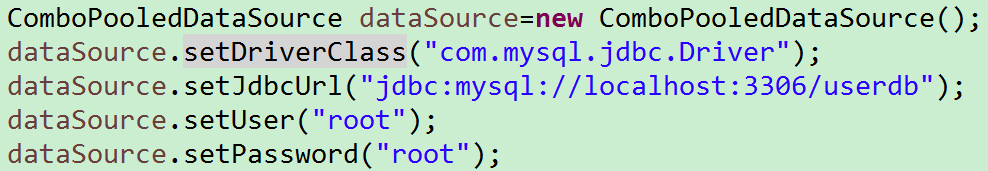
**数据库连接池的概念：**

用池来管理Connection，可以重复使用Connection，使用连接池，我们就不用自己来创建Connection，而是通过池来获取Connection对象，当使用完Connection后，调用Connection的close()方法也不会真的关闭Connection，而是把Connection“归还”给池，池就可以再利用这个Connection对象了

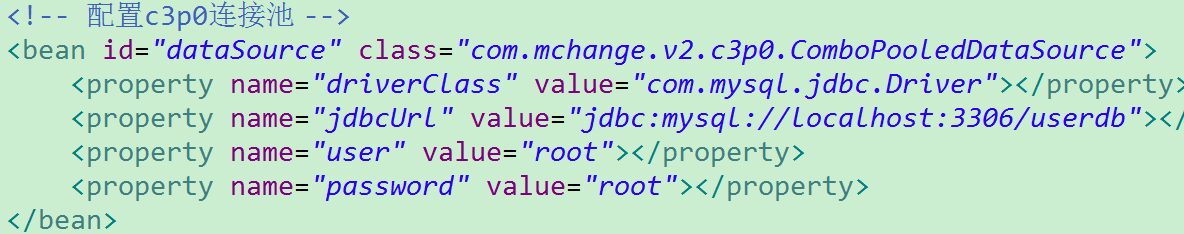
#### Spring配置c3p0连接池

第一步：导入jar包



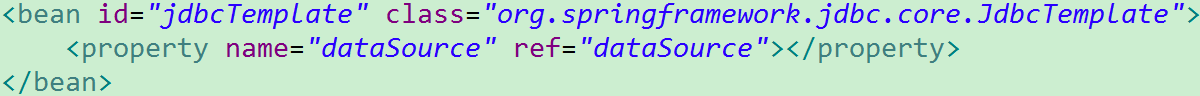


第二步：创建spring配置文件，配置连接池

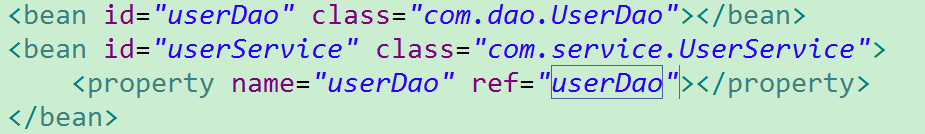


#### jdbcTemplate在dao中的使用方式

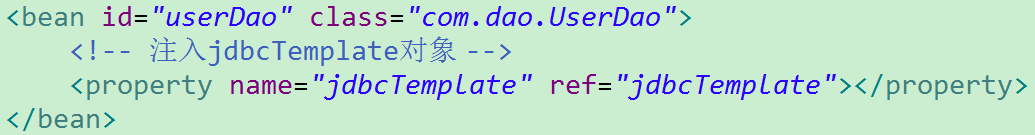
1. 在jdbcTemplate对象里面注入dataSource

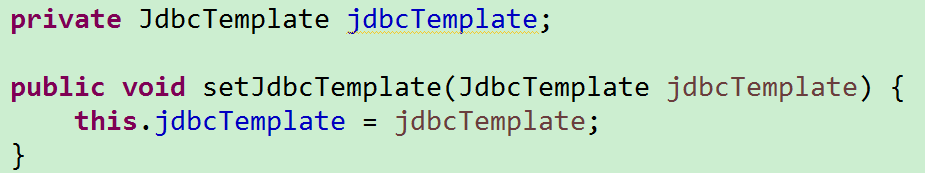


1. 创建service和dao，配置service和dao对象，在service注入dao对象



1. 创建jdbcTemplate对象，把模板对象注入到dao里面





## Spring的事务管理

**事务概念**

1. **什么是事务**
2. **事务的特性**

**原子性、持久性、隔离性、一致性**

1. **不考虑隔离性产生问题**
2. 脏读
3. 不可重复读
4. 虚读（幻读）
5. **设置事务的隔离级别**

**Spring事务管理两种方式**

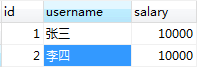
第一种：编程式事务管理（不用）

第二种：声明式事务管理

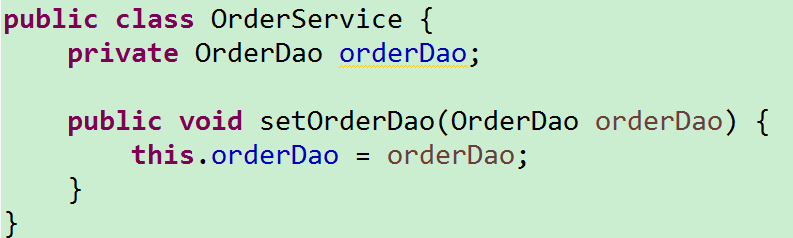
1. 基于xml配置文件实现
2. 基于注解实现

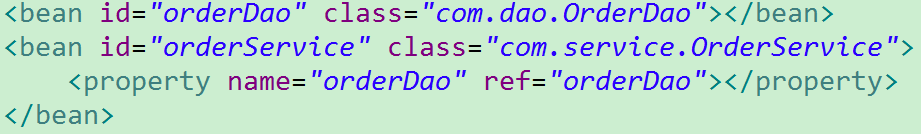
**搭建转账环境**

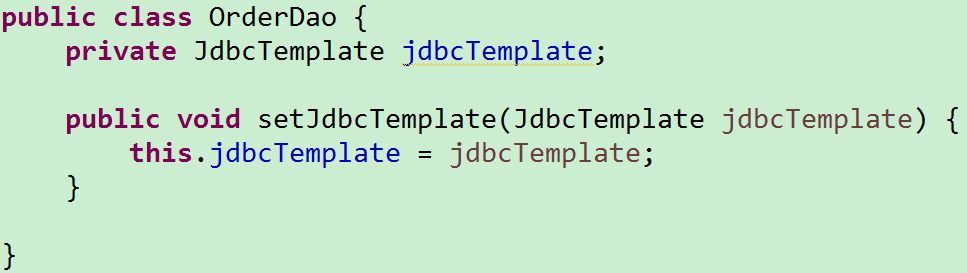
1. **创建数据库，添加数据**

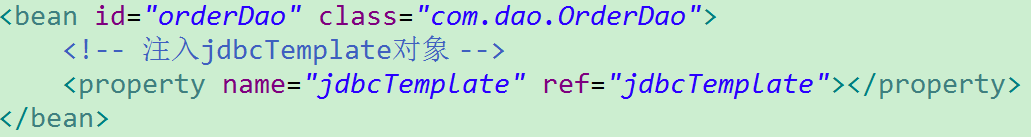


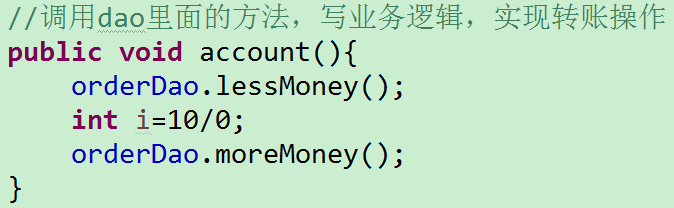
1. 创建service和dao，完成注入关系







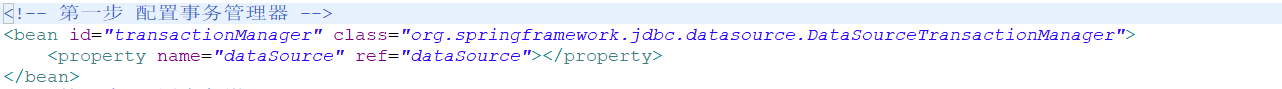


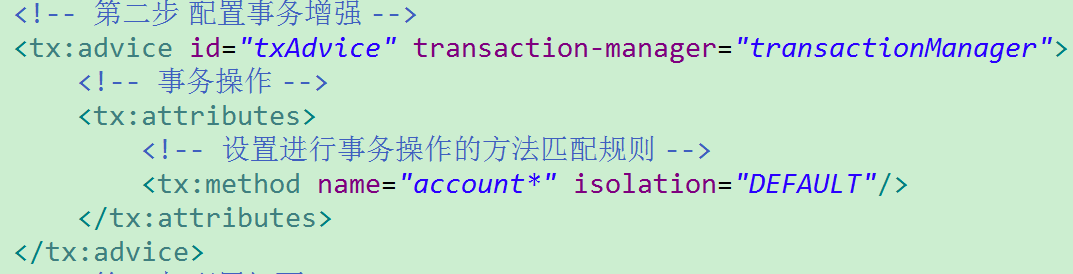


1. 产生问题：
2. 如果张三少了1000之后，出现异常，李四不会多钱，钱丢了
3. 解决方法
4. 添加事务解决，出现异常进行回滚操作

### 声明式事务管理（XML配置）

**第一步 配置事务管理器**

**第二步 配置事务增强**



**第三步 配置切面**

<!-- 第三步 配置切面 -->

<aop:config>

<!-- 切入点 -->

<aop:pointcut expression=*"execution(\* com.service.OrderService.\*(..))"* id=*"pointcut"*/>

<!-- 切面 -->

<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"pointcut"*/>

</aop:config>

### 声明式事务管理（注解）

**第一步 配置事务管理器**

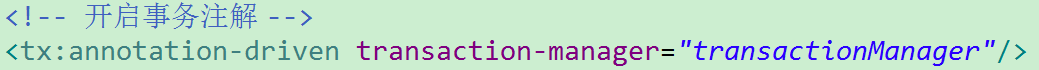
<!-- 第一步 配置事务管理器 -->

<bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

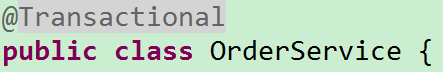
<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

**第二步 配置事务注解**



**第三步 在使用事务的方法所在类上面添加注解**



# 四、SSH框架整合

## 1. 回顾SSH框架知识点

### 1. Hibernate框架

1. Hibernate核心配置文件
2. 数据库信息
3. Hibernate信息
4. 映射配置

如果单纯使用hibernate框架，核心配置文件名称和位置是固定的

Hibernate和Spring整合的时候，hibernate核心配置文件名称和位置没有固定要求

1. Hibernate映射配置文件
2. 实体类和数据库表的映射关系 orm思想
3. Hibernate操作步骤
4. spring框架对hibernate进行封装，使用hibernateTemplate

### Struts2框架

1. Action操作
2. action创建三种方式 继承类ActionSupport
3. 配置action访问路径

创建struts.xml配置文件，这个文件的名称和位置是固定的

1. 配置访问action的多个方法

使用通配符方式配置

1. 在action获取表单提交数据

获取request对象

属性封装

模型驱动

1. 在action操作域对象

使用ServletActionContext获取域对象

1. 配置struts2的过滤器StrutsPrepareAndExecuteFilter
2. 值栈
3. 向值栈放数据

set方法

push方法

定义变量，生成get方法

1. 从值栈获取数据

在jsp中使用struts2标签+ognl表达式

<s:property>

<s:iterator>

1. 拦截器
2. aop和责任链模式
3. 自定义拦截器

继承MethodFilterInterceptor

重写类里面的方法

配置拦截器和action关联

<package name=*"default"* extends=*"struts-default"* namespace=*"/"*>

<!-- 声明拦截器 -->

<interceptors>

<interceptor name=*"loginintercept"* class=*"com.interceptor.LoginInterceptor"*></interceptor>

</interceptors>

<action name=*"user\_\*"* class=*"com.action.UserAction"* method=*"{1}"*>

<!-- 使用自定义拦截器 -->

<interceptor-ref name=*"loginintercept"*>

<param name=*"excludeMethods"*>login</param>

</interceptor-ref>

<!-- 把默认拦截器手动执行一次 -->

<interceptor-ref name=*"defaultStack"*></interceptor-ref>

<result name=*"success"*>/main.jsp</result>

<result name=*"login"*>/login.jsp</result>

<result name=*"input"*>/login.jsp</result>

</action>

</package>

### Spring框架

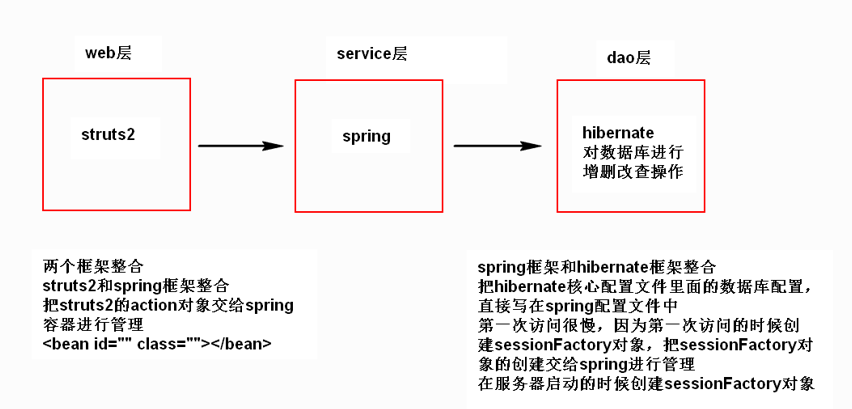
1. Spring核心配置文件
2. 名称和位置没有固定要求
3. 在spring核心配置文件中引入schema约束
4. 创建对象
5. xml配置方式<bean id=”” class=””></bean>
6. 注解方式：四个注解
7. 注入属性（对象类型的属性）
8. xml配置方式
9. 注解方式：两个注解

@AutoWired

@Resource

1. 使用ServeltContext对象和监听器
2. 在服务器启动的时候，加载spring配置文件，创建对象
3. 配置Spring的监听器
4. 指定Spring配置文件位置
5. jdbcTemplate
6. Spring事务配置
7. xml方式
8. 注解方式

## SSH框架整合思想



### 整合Struts2和Spring框架

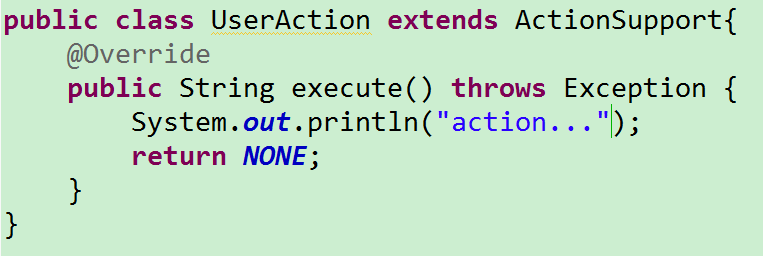
1. 把struts2的action交给Spring进行管理
2. 实现过程：

**第一步：导入struts2的jar包**

1. 导入用于整合的jar包



**第二步：创建action**



**第三步：创建struts2核心配置文件，配置action**

1. 位置在src下面，名称是struts.xml

**第四步：配置struts2过滤器**

**第五步：导入spring的jar包**

**第六步：创建spring配置文件**

1. 引入约束
2. 配置spring监听器

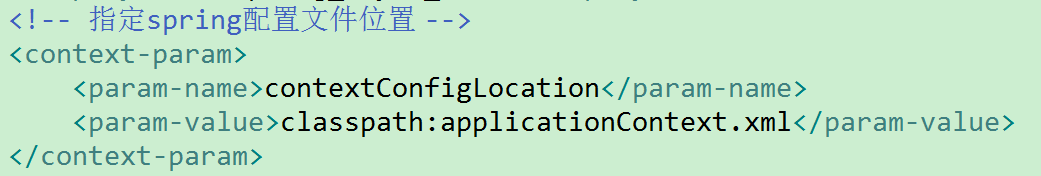
<!-- 配置监听器 -->

<listener>

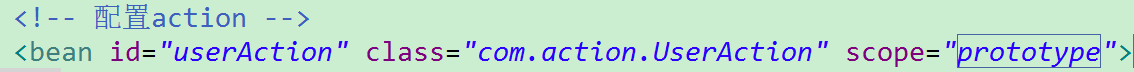
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

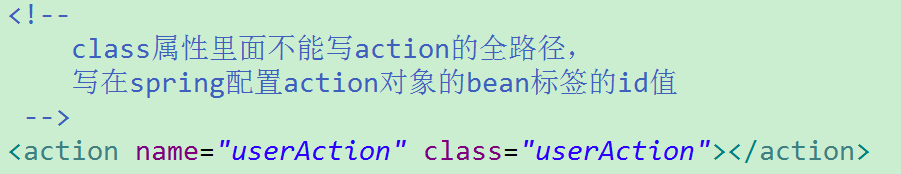
1. 指定spring配置文件的位置



**第七步：把action交给spring进行配置**



（1）只需要在spring里面配置action对象，不要在struts.xml中配置



### Spring框架整合hibernate框架

1. 把hibernate核心配置文件中配置数据库的信息，在spring中进行配置
2. 把hibernate里面的sessionFactory创建交给spring进行管理

**具体实现：**

**第一步：导入hibernate的jar包**

1. **struts2的javassist-3.11.0.GA.jar与hibernate的javassist-3.18.1-GA.jar重复，删除低版本的jar包**
2. **导入spring整合持久层框架的jar包**



**第二步 搭建hibernate的环境**

1. 创建实体类
2. 配置实体类映射关系
3. 创建核心配置文件

第三步 把hibernate核心配置文件中的数据库配置，在spring进行配置

<!-- 配置c3p0连接池 -->

<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>

<property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"*></property>

<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/test"*></property>

<property name=*"user"* value=*"root"*></property>

<property name=*"password"* value=*"root"*></property>

</bean>

第四步 把hibernate的sessionFactory交给spring配置

<!-- sessionFactory创建交给spring进行管理 -->

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<!-- 因为hibernate核心配置文件中没有数据配置，数据库在spring里面配置，注入dataSource -->

<property name=*"dataSource"* ref=*"dataSource"*></property>

<!-- 指定hibernate使用核心配置文件 -->

<property name=*"configLocations"* value=*"classpath:hibernate.cfg.xml"*></property>

</bean>

第五步：在dao里面使用hibernateTemplate

<!-- 创建hibernateTemplate对象 -->

<bean id=*"hibernateTemplate"* class=*"org.springframework.orm.hibernate5.HibernateTemplate"*>

<!-- 注入sessionFactory -->

<property name=*"sessionFactory"* ref=*"sessionFactory"*></property>

</bean>

**public** **void** add(){

//hibernateTemplate对hibernate进行封装

/\*HibernateTemplate hibernateTemplate=new HibernateTemplate(sessionFactory);

hibernateTemplate.save(entity);\*/

System.***out***.println("dao......");

User user=**new** User();

user.setName("马冬梅");

user.setPassword("888");

user.setEmail("mdm@qq.com");

//使用set方法实现添加操作

hibernateTemplate.save(user);

}

没有配置事务，做操作的时候出现异常

Turn your Session into FlushMode.COMMIT/AUTO or remove 'readOnly' marker from transaction definition.

第六步：配置事务

<!-- 第一步：配置事务管理器 -->

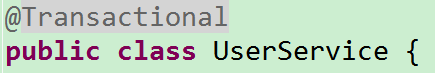
<bean id=*"transactionManager"* class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*>

<property name=*"sessionFactory"* ref=*"sessionFactory"*></property>

</bean>

<!-- 第二步：开启事务注解 -->

<tx:annotation-driven transaction-manager=*"transactionManager"*/>



## HibernateTemplate介绍

1. HibernateTemplate对hibernate框架进行封装，直接调用HibernateTemplate里面的方法实现功能

### HibernateTemplate常用的方法

#### （1）save 添加操作



#### （2）update 修改操作



#### （3）delete 删除操作



#### （4）get 根据id查询





#### （5）扩大session作用范围

为了避免使用懒加载时出现no-session的问题，需要扩大session的作用范围

<filter>

<filter-name>openSessionInView</filter-name>

<filter-class>org.springframework.orm.hibernate5.support.OpenSessionInViewFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>openSessionInView</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

#### （6）find 查询全部



#### （7）条件查询



1. 第一个参数是hql语句
2. 可变参数

DetachedCriteria criteria = DetachedCriteria.*forClass*(User.**class**);

criteria.add(Restrictions.*eq*("name", "张三"));

List<User> list = (List<User>) hibernateTemplate.findByCriteria(criteria);

#### （8）模糊查询

//模糊查询

**public** List<User> selUserByName(String name){

//第一种方式

/\*SessionFactory sessionFactory = this.getSessionFactory();

Session session = sessionFactory.getCurrentSession();

Query query = session.createQuery("from User where name like ?");

query.setParameter(0, "%"+name+"%");

List<User> list = query.list();\*/

//第二种方式

/\*List<User> list=(List<User>) this.getHibernateTemplate().find("from "

+ "User where name like ?", "%"+name+"%");\*/

//第三种方式

//1.创建离线查询对象，设置对哪个实体类进行操作

DetachedCriteria criteria=DetachedCriteria.*forClass*(User.**class**);

//2.设置对实体类哪个属性进行查询

criteria.add(Restrictions.*like*("name", "%"+name+"%"));

List<User> list = (List<User>) **this**.getHibernateTemplate().findByCriteria(criteria);

**return** list;

}

#### （9）分页查询

//分页查询

**public** List<User> selUserByPage(Integer pageIndex,Integer pageSize){

//第一种 使用hibernate底层代码实现

/\*SessionFactory sessionFactory = this.getSessionFactory();

Session session = sessionFactory.getCurrentSession();

Query query = session.createQuery("from User");

//设置分页信息

query.setFirstResult((pageIndex-1)\*pageSize);

query.setMaxResults(pageSize);

List<User> list = query.list();\*/

//第二种方式

DetachedCriteria criteria=DetachedCriteria.*forClass*(User.**class**);

List<User> list = (List<User>) **this**.getHibernateTemplate().findByCriteria(criteria,

(pageIndex-1)\*pageSize, pageSize);

**return** list;

}

## SSH框架整合过程

**第一步：导入jar包**

**第二步：搭建struts2环境**

1. **创建action，创建struts.xml配置文件，配置action**
2. **配置struts2过滤器**

**第三步：搭建hibernate环境**

1. **创建实体类**
2. **配置实体类和数据库表映射关系**
3. **创建hibernate核心配置文件，引入映射配置文件**

**第四步：搭建spring环境**

1. **创建spring的核心配置文件**
2. **让spring配置文件在服务器启动的时候加载，配置监听器，指定spring配置文件的位 置**

**第五步：struts2和spring整合**

1. **把action对象的创建交给spring（action是多实例的）**
2. **在struts.xml中action标签的class属性里面写bean的id值**

**第六步：spring和hibernate整合**

1. **把hibernate核心配置文件中的数据库配置，在spring里面配置**
2. **把hibernate的sessionFactory在spring配置**

**第七步：在dao里面使用hibernateTemplate**

1. **在dao注入hibernateTemplate对象**
2. **在hibernateTemplate对象中注入sessionFactory**

**第八步：配置事务**

## 整合其它方式

1. Spring整合hibernate的时候，可以不写hibernate核心配置文件
2. 把hibernate核心配置文件中，基本信息配置和映射引入都放到spring配置

<!-- 如果不写hibernate核心配置文件 -->

<!-- 配置hibernate基本信息 -->

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.show\_sql"*>true</prop>

<prop key=*"hibernate.format\_sql"*>true</prop>

<prop key=*"hibernate.hbm2ddl.auto"*>update</prop>

<prop key=*"dialect"*>org.hibernate.dialect.MySQLDialect</prop>

</props>

</property>

<!-- 配置映射文件引入 -->

<property name=*"mappingResources"*>

<list>

<value>com/bean/User.hbm.xml</value>

</list>

</property>

## Spring分模块开发

1. 在spring里面配置多个内容，造成配置混乱，不利于维护
2. 把spring核心配置文件中，一部分配置放到单独的配置文件中，在spring核心配置文件中引入单独配置文件

