

# 1. Spring的bean管理（注解）







## 1-1. 注解介绍

- 1 代码里面特殊标记，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 注解使用在类上面，方法上面 和 属性上面


## 1-2. Spring注解开发准备

### 1 导入jar包

#### (1) 导入基本的jar包

 commons-logging-1.2.jar  
 log4j-1.2.16.jar  
 spring-beans-4.2.4.RELEASE.jar  
 spring-context-4.2.4.RELEASE.jar  
 spring-core-4.2.4.RELEASE.jar  
 spring-expression-4.2.4.RELEASE.jar

#### (2) 导入aop的jar包

 spring-aop-4.2.4.RELEASE.jar

### 2 创建类，创建方法

### 3 创建spring配置文件，引入约束

#### (1) 第一天做ioc基本功能，引入约束beans

#### (2) 做spring的ioc注解开发，引入新的约束

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context" xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">
```

### 4 开启注解扫描

```
<!-- 开启注解扫描
  (1)到包里面扫描类、方法、属性上面是否有注解
-->
<context:component-scan base-package="cn.itcast"></context:component-scan>

<!--
  扫描属性上面的注解
-->
<!-- <context:annotation-config></context:annotation-config> -->
```

## 1-3. 注解创建对象

1 在创建对象的类上面使用注解实现

```
@Component(value="user")    // <bean id="user" class=""/>
public class User {

ApplicationContext context =
    new ClassPathXmlApplicationContext("bean1.xml");
User user = (User) context.getBean("user");
System.out.println(user);
user.add();
```

2 创建对象有四个注解

Spring 中提供@Component 的三个衍生注解：(功能目前来讲是一致的)

```
* @Controller    :WEB 层
* @Service        :业务层
* @Repository     :持久层
```

这三个注解是为了让标注类本身的用途清晰，Spring 在后续版本会对其增强

(1) @Component

(2) @Controller

(3) @Service

(4) @Repository

目前这四个注解功能是一样的，都创建对象

3 创建对象单实例还是多实例

```
@Service(value="user")    //
@Scope(value="prototype")
public class User {
|
```

## 1-4. 注解注入属性

1 创建service类，创建dao类，在service得到dao对象

注入属性第一个注解 @Autowired

(1) 创建dao和service对象

```
@Component(value="userDao")
public class UserDao {
```

```
@Service(value="userService")
public class UserService {
```

(2) 在service类里面定义dao类型属性

```
//得到dao对象
//1 定义dao类型属性
//在dao属性上面使用注解 完成对象注入
@Autowired
private UserDao userDao;
// 使用注解方式时候不需要set方法
```

注入属性第二个注解 @Resource

```
// name属性值 写 注解创建dao对象 value值
@Resource(name="userDao")
private UserDao userDao;
```

## 1-5. 配置文件和注解混合使用

1 创建对象操作使用配置文件方式实现

```
<!-- 配置对象 -->
<bean id="bookService" class="cn.itcast.xmlanno.BookService"></bean>
<bean id="bookDao" class="cn.itcast.xmlanno.BookDao"></bean>
<bean id="ordersDao" class="cn.itcast.xmlanno.OrdersDao"></bean>
```

2 注入属性的操作使用注解方式实现

```
//得到bookdao和ordersdao对象
@Resource(name="bookDao")
private BookDao bookDao;

@Resource(name="ordersDao")
private OrdersDao ordersDao;
```

## 2. AOP

### 2-1. AOP概念

- 1 aop: 面向切面（方面）编程，扩展功能不修改源代码实现
- 2 AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码
- 3 aop底层使用动态代理实现
  - (1) 第一种情况，有接口情况，使用动态代理创建接口实现类代理对象
  - (2) 第二种情况，没有接口情况，使用动态代理创建类的子类代理对象

### 2-2. AOP原理

画图分析原理



aop：横向抽取机制

底层使用 动态代理方式实现

第一种情况

\* 使用jdk动态代理，针对有接口情况



## 第二种情况 没有接口情况

```
public class User {  
    public void add() {  
  
    }  
}
```

使用cglib动态代理，没有接口情况

//动态代理实现

\* 创建User类的子类的代理对象

\* 在子类里面调用父类的方法完成增强

## 2-3.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

\* 切入点：在类里面可以有很多的方法被增强，比如实际操作中，只是增强了类里面add方法和update方法，实际增强的方法称为 切入点

\* 通知/增强：增强的逻辑，称为增强，比如扩展日志功能，这个日志功能称为增强

前置通知：在方法之前执行

后置通知：在方法之后执行

异常通知：方法出现异常

最终通知：在后置之后执行

环绕通知：在方法之前和之后执行

\* 切面：把增强应用到具体方法上面，过程称为切面  
把增强用到切入点过程

## 3. Spring的AOP操作

1 在spring里面进行aop操作，使用aspectj实现

(1) aspectj不是spring一部分，和spring一起使用进行aop操作

(2) Spring2.0以后新增了对AspectJ支持


2 使用aspectj实现aop有两种方式


(1) 基于aspectj的xml配置


(2) 基于aspectj的注解方式


### 3-1. AOP操作准备

1 除了导入基本的jar包之外，还需要导入aop相关的jar包

 aopalliance-1.0.jar

 aspectjweaver-1.8.7.jar

 spring-aop-4.2.4.RELEASE.jar

 spring-aspects-4.2.4.RELEASE.jar

2 创建spring核心配置文件，导入aop的约束

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop" xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd">!--
```



## 3-2. 使用表达式配置切入点

1 切入点：实际增强的方法

2 常用的表达式

execution(<访问修饰符>?<返回类型><方法名>(<参数>)<异常>)

(1) execution(\* cn.itcast.aop.Book.add(..))

(2) execution(\* cn.itcast.aop.Book.\*(..))

(3) execution(\* \*.\*(..))

(4) 匹配所有save开头的方法 execution(\* save\*(..))

## 3-3. Aspectj的aop操作 (xml)

```
<!-- 1 配置对象 -->
<bean id="book" class="cn.itcast.aop.Book"></bean>
<bean id="myBook" class="cn.itcast.aop.MyBook"></bean>

<!-- 2配置aop操作 -->
<aop:config>
  <!-- 2.1 配置切入点 -->
  <aop:pointcut expression="execution(* cn.itcast.aop.Book.*(..))" id="pointcut1"/>
  <!-- 2.2 配置切面
  把增强用到方法上面
  -->
  <aop:aspect ref="myBook">
    <!-- 配置增强类型
    method: 增强类里面使用哪个方法作为前置
    -->
    <aop:before method="before1" pointcut-ref="pointcut1"/>
  </aop:aspect>
</aop:config>
```

```
//环绕通知
public void around1(ProceedingJoinPoint proceedingJoinPoint)
    //方法之前
    System.out.println("方法之前....");

    //执行被增强的方法
    proceedingJoinPoint.proceed();

    //方法之后
    System.out.println("方法之后....");
}
```

```

<!-- 配置增强类型
    method: 增强类里面使用哪个方法作为前置
-->
<aop:before method="before1" pointcut-ref="pointcut1"/>

<aop:after-returning method="after1" pointcut-ref="pointcut1"/>

<aop:around method="around1" pointcut-ref="pointcut1"/>

```

## 4. log4j介绍



1 通过log4j可以看到程序运行过程中更详细的信息

(1) 经常使用log4j查看日志

2 使用

(1) 导入log4j的jar包

(2) 复制log4j的配置文件，复制到src下面

 src/main/resources
   
 log4j.properties

3 设置日志级别

```
log4j.rootLogger=info, stdout
```

(1) info: 看到基本信息

(2) debug: 看到更详细信息

## 5. Spring整合web项目演示

1 演示问题

(1) action调用service, service调用dao

```

action.....
12:11:26,026 INFO ClassPathXmlApplicationContext:578 - Refreshing
12:11:26,119 INFO XmlBeanDefinitionReader:317 - Loading XML bean d
service.....
dao.....
action.....
12:11:48,393 INFO ClassPathXmlApplicationContext:578 - Refreshing
12:11:48,395 INFO XmlBeanDefinitionReader:317 - Loading XML bean d
service.....
dao

```

每次访问action时候，都会加载spring配置文件




## 2 解决方案:

- (1) 在服务器启动时候, 创建对象加载配置文件
- (2) 底层使用Listener、ServletContext对象

## 3 在spring里面不需要我们自己写代码实现, 帮封装

- (1) 封装了一个监听器, 只需要配置监听器就可以了
- (2) 配置监听器之前做事情: 导入spring整合web项目jar包

 spring-web-4.2.4.RELEASE.jar

```
<!-- 配置监听器 -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

- (3) 指定加载spring配置文件位置

java.io.FileNotFoundException: Could not find resource

[/WEB-INF/applicationContext.xml]

```
<!-- 指定spring配置文件位置 -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath:bean1.xml</param-value>
</context-param>
```

Loading XML bean definitions from class path resource [bean1.xml]

