spring 注解和aop配置文件方式执行

注解方式管理bean

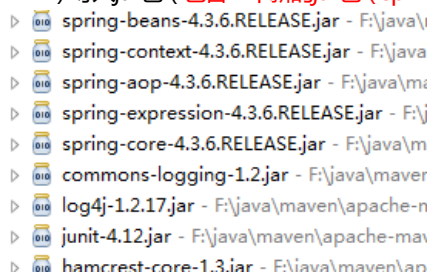
1）代码里面的特殊标记

2）注解写法@注解名称（属性名称=属性值）

3）可以使用在类，方法，属性上。

1注解方式创建对象

1）导入jar包（包含一个新的jar包（spring-aop））



2)创建类，创建方法

```
public class User {
 public void add(){
  System.out.println("add.............spring day02 注解方式！");
 }
}
```

3）创建spring的配置文件，引入约束

引入spring-context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">
```

开启注解扫描

```
<!-- 开启注解扫描
     1.到包里面扫描类，方法，属性上面的注解（多个包可以用逗号隔开，或直接扫描根目录）
     -->
     <context:component-scan base-package="com.wmr.springDay02"></context:component-scan>
     <!-- 只扫描属性上面的注解 -->
     <context:annotation-config></context:annotation-config>
```

4）创建对象

在创建对象的类上加上注解@Component(value="beanName")

```
@Component(value="user")
public class User {
    public void add(){
        System.out.println("add.............spring day02注解方式! ");
    }
}
```

测试：

```
public class SpringTestDay02 {
 @Test
 public void test(){
  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
  User u = (User) context.getBean("user");
  u.add();
 }
```

}

spring中的常用注解

（都可以用来创建对象）

@Component ：

@Controller :web层

@Service ：业务层

@Respository :持久层

创建对象是单实例还是多实例

在类上加@Scope(value="**prototype/singleton**")两个属性任选一个

```
@Component(value="user")
@Scope(value="prototype")
public class User {
    public void add(){
        System.out.println("add.............spring day02注解方式! ");
    }
}
```

2使用注解注入属性

创建两个类，并在userService类中添加UserDao类的属性，在属性字段上添加@Autowired注解

@Autowired自动注入，根据类名去找属性

@Resource(name="属性字段的value值")

（@Autowired不用配置属性字段的值，@Resource可以指定对象）

```
@Component(value="dao")
public class UserDao {
 public void add(){
  System.out.println("add Dao!");
 }
}

public class UserService {
 //得到对象
 //使用注解不需要set方法
 //在属性上添加注解@Autowired
 @Autowired
 private UserDao userDao;

 public void add(){
  System.out.println("add Service!");
  userDao.add();
 }
}
```

配置文件和注解混合使用的方式

1.创建对象一般使用配置文件的方式

2.属性注入一般使用注解的方式

配置文件

```
<bean id="bookService" class="com.wmr.springDay02.BookService"></bean>
<bean id="bookDao" class="com.wmr.springDay02.BookDao"></bean>
<bean id="orderDao" class="com.wmr.springDay02.OrderDao"></bean>
```

代码：

```
public class BookDao {
 public void bookDao(){
  System.out.println("bookDao running!");
```

```
 }
}

public class OrderDao {
 public void buy(){
  System.out.println("orderDao running!");
 }
}

public class BookService {
 //得到bookDao和orderDao的队形
 @Resource(name="bookDao")
 private BookDao bookDao;
 @Resource(name="orderDao")
 private OrderDao orderDao;
 public void add(){
  System.out.println("service running!");
  bookDao.bookDao();
  orderDao.buy();
 }
}
```

Aop

1.aop概念

1）aop:面向切面编程，扩展功能不通过修改源代码实现。

2）aop采用横向抽取机制，取代传统纵向继承机制重复性代码（性能监听，事务管理，安全检查，缓存）

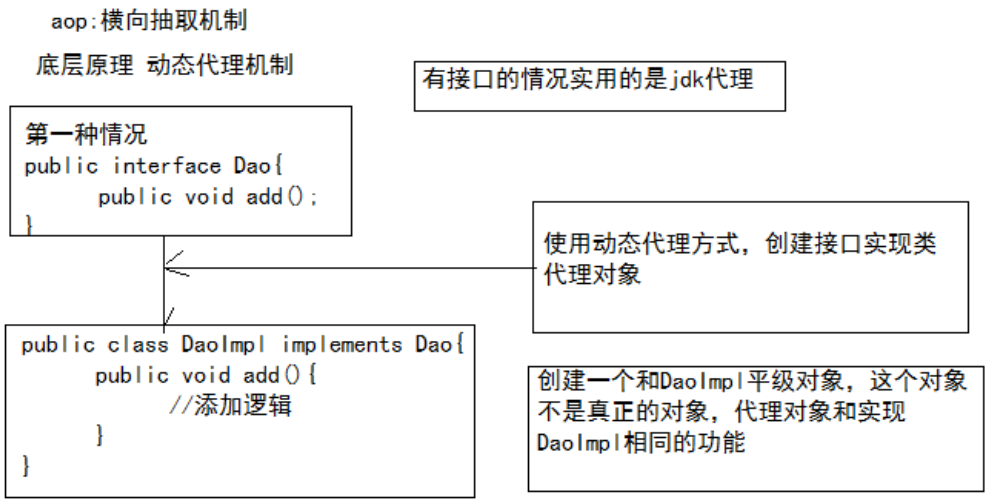
2.aop底层原理

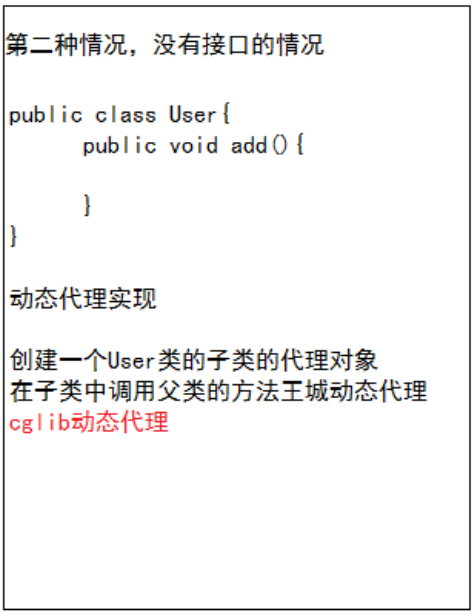
```
public class User{
       //添加用户的方法
       public void add () {
              //添加逻辑
       }
}

*扩展功能
添加日志功能，记录添加的用户及添加时间
```

纵向抽取机制

```
public class BaseUser{
       //创建日志添加方法
       public void writeLog(){
              //添加日志逻辑
       }
}

public class User extends BaseUser{
       public void add(){
              //添加用户逻辑
              //调用父类方法
              super.writeLog();
       }
}
```

缺点：父类方法名发生改变，子类中的代码一样需要许改，不能彻底解决。





3.aop操作相关术语

joinpoint:连接点，指那些被拦截的点。在spring中这些点指的是方法，因为spring只支持方法类型的连接点。

Pointcut：切入点，指那些我们要对哪些Joinpoint进行拦截定义。

Advice：通知、增强，只我们拦截到Joinpoint之后要做的事情，通知分为前置通知和后置通知，异常通知，最终通知，环绕通知（切面要完成的任务）

Aspect：切面，是切入点和通知的结合（引介）。

Introduction：引介，是一种特殊的通知，在不修改代理的前提下，Introduction可以在运行在运行期为类动态地添加一些方法或字段

Target：目标对象，代理的目标（要增强的类）

Weaving：织入，是把增强应用的目标过程

Proxy：代理，一个类被AOP织入增强后，就会产生一个结果代理类

spring的aop操作

spring中要使用Aop操作，使用AspectJ实现

1）AspectJ不是spring的一部分，只是和spring一起使用进行aop操作。

2）spring在2.0之后增加了对Aspectj的支持。

2.使用Aspectj实现aop的操作有两种方式、

1）基于xml配置文件方式

导入jar包：

创建spring核心配置文件导入新的约束：aop约束

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd

使用表达式配置切入点

1）切入点：实际增强的方法

2）常用表达式

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

execution(* X.X.X.X.***(..)) 对***方法进行增强

execution(* XXX.XX*(..)) 队XX开都的方法进行增强

execution(* *.*(..)) 对所有方法进行增强

配置文件：

```
<!-- aop操作 ，创建实例-->
  <bean id="book" class="com.wmr.springDay02.aop.Book"></bean>
  <bean id="myBook" class="com.wmr.springDay02.aop.Mybook"></bean>
  <!-- 配置aop操作 -->
  <aop:config>
   <!-- 配置切入点 -->
   <aop:pointcut expression="execution(* com.wmr.springDay02.aop.Book.add(..))" id="pointcut1"/>
   <!-- 配置切面 -->
   <aop:aspect ref="myBook">
    <aop:before method="before1" pointcut-ref="pointcut1"/>
   </aop:aspect>
  </aop:config>
```

代码：

```
public class Book {
 public void add(){
  System.out.println("add.....Book!");
 }
}

public class Mybook {
 public void before1(){
  System.out.println("前置增强,before！");
 }
}
```

测试：

```
public class AopTest {
 @Test
 public void testAop(){
  ApplicationContext context = new ClassPathXmlApplicationContext("\\applicationContext.xml");
  Book book = (Book)context.getBean("book");
  book.add();
 }

}
```

结果：

```
log4j:WARN No appenders could be found for logger (org.springframework.core.env.StandardEnvironment).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
前置增强,before!
add.....Book!
```

前置执行，后置执行，环绕执行：

```
public class Mybook {
 public void before1(){
  System.out.println("前置增强,before！");
 }

 public void after1(){
  System.out.println("后置执行。。。。。。after!");
 }

 //环绕执行
 public void around1(ProceedingJoinPoint proceedingJoinPoint) throws Throwable{
  //方法前
  System.out.println("方法前执行。。。。。。环绕");

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

  //方法后
  System.out.println("方法后执行。。。。。。环绕");
 }

public class Book {
 public void add(){
  System.out.println("add.....Book!");
 }
}
```

配置文件：

```
  <!-- aop操作 ，创建实例-->
  <bean id="book" class="com.wmr.springDay02.aop.Book"></bean>
  <bean id="myBook" class="com.wmr.springDay02.aop.Mybook"></bean>
  <!-- 配置aop操作 -->
  <aop:config>
   <!-- 配置切入点 -->
   <aop:pointcut expression="execution(* com.wmr.springDay02.aop.Book.add(..))" id="pointcut1"/>
   <!-- 配置切面 -->
   <aop:aspect ref="myBook">
    <aop:before method="before1" pointcut-ref="pointcut1"/>
    <aop:after method="after1" pointcut-ref="pointcut1"/>
    <aop:around method="around1" pointcut-ref="pointcut1"/>
   </aop:aspect>
  </aop:config>
```

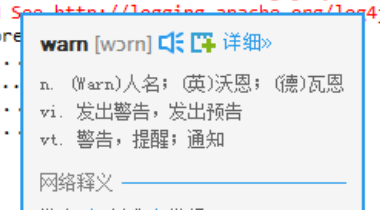
测试：

```
 @Test
 public void testAop1(){
```

```
    ApplicationContext context = new ClassPathXmlApplicationContext("\\applicationContext.xml");
    Book book = (Book)context.getBean("book");
    book.add();
  }
```

执行结果：

```
log4j:WARN No appenders could be found for logger (org.springframework.core.env.StandardEnvironment).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
前置增强,before
方法前执行.....
add.....
方法后执行.....
后置执行.....
```



2）基于注解的方式。

log4j

1.通过log4j中可以看到程序运行中一些更详细的信息。

1）通常使用log4j查看日志

2.导入log4j的jar包，复制log4j的配置文件