# 1、AOP

## 1.1 AOP概述

1. 在软件业，**AOP为Aspect Oriented Programming**的缩写，意为：**面向切面编程**，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。
2. AOP是OOP（面向对象编程）的延续，是软件开发中的一个热点，也是Spring框架中的一个重要内容，是函数式编程的一种衍生范型。
3. 利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。
4. 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
5. **经典应用：事务管理、性能监视、安全检查、缓存 、日志等【画图】**
6. Spring AOP使用纯Java实现，不需要专门的编译过程和类加载器，在运行期通过代理方式向目标类织入增强代码
7. **AspectJ是一个基于Java语言的AOP框架**，Spring2.0开始，Spring AOP引入对Aspect的支持，AspectJ扩展了Java语言，提供了一个专门的编译器，在编译时提供横向代码的织入

## 1.2 AOP实现原理

1. aop底层将采用代理机制进行实现。
2. 接口 + 实现类 ：spring采用 jdk 的**动态代理Proxy**。
3. 实现类：spring 采用 **cglib字节码**增强。

## 1.3 AOP术语

**1.target：**目标类，需要被代理的类。例如：UserService

**2.Joinpoint**(连接点):所谓连接点是指那些可能被拦截到的方法。例如：所有的方法

**3.PointCut** 切入点：已经被增强的连接点。例如：addUser()

4.**advice 通知/增强**，**增强代码**。例如：after、before

**5. Weaving**(织入):是指把增强advice应用到目标对象target来创建新的代理对象proxy的过程.

**6.proxy** 代理类

**7. Aspect**(切面): 是切入点pointcut和通知advice的结合

一个线是一个特殊的面。

一个切入点和一个通知，组成成一个特殊的面。

|  |
| --- |
|  |

## 1.4 手动代理

### 1.4.1 JDK动态代理

#### 目标类

|  |  |
| --- | --- |
|  |  |

#### 切面类

|  |
| --- |
|  |

#### 工厂类

|  |
| --- |
|  |

#### 测试类

|  |
| --- |
|  |

### 1.4.2 cglib 增强字节码

* 没有接口，只有实现类。
* 采用字节码增强框架 **cglib，在运行时 创建目标类的子类**，从而对目标类进行增强。
* 导入jar包：

自己导包（了解）：

核心：hibernate-distribution-3.6.10.Final\lib\bytecode\cglib\cglib-2.2.jar

依赖：struts-2.3.15.3\apps\struts2-blank\WEB-INF\lib\asm-3.3.jar

**spring-core..jar 已经整合以上两个内容**

|  |
| --- |
|  |

#### 工厂类

|  |
| --- |
|  |

#### 测试结果

|  |
| --- |
|  |

## 1.5 AOP联盟通知类型

|  |
| --- |
| AOP联盟为通知Advice定义了org.aopalliance.aop.Advice  Spring按照通知Advice在目标类方法的连接点位置，可以分为5类  •**前置通知** org.springframework.aop.MethodBeforeAdvice  •在目标方法执行前实施增强  •**后置通知** org.springframework.aop.AfterReturningAdvice  •在目标方法执行后实施增强  •**环绕通知** org.aopalliance.intercept.MethodInterceptor  •在目标方法执行前后实施增强  •**异常抛出通知** org.springframework.aop.ThrowsAdvice  •在方法抛出异常后实施增强  •**引介通知** org.springframework.aop.IntroductionInterceptor  在目标类中添加一些新的方法和属性 |

|  |
| --- |
| **环绕通知**，**必须手动执行目标方法**  try{  //前置通知  //执行目标方法  //后置通知  } catch(){  //抛出异常通知  } |

## 1.6 Spring编写代理半自动

目标：掌握让spring 创建代理对象，从spring容器中手动的获取代理对象。

### 第一步：导Jar包

【核心4+1 、AOP联盟（规范）、spring-aop （实现）】

|  |
| --- |
|  |

### 第二步：目标类

|  |  |
| --- | --- |
|  |  |

### 第三步：切面类

|  |
| --- |
|  |

### 第四步：spring配置

|  |
| --- |
|  |

Name=”optimize” 可以设置 代理方式，默认是利用Proxy 代理，加上ture后改用cglib代理

### 第五步：测试

|  |
| --- |
|  |

## 1.7 Spring AOP全自动编程

目录：明白什么是全自动织入

### 第一步：导入jar包

spring-framework-3.0.2.RELEASE-dependencies\org.aspectj\com.springsource.org.aspectj.weaver\1.6.8.RELEASE

dependencies包下

|  |
| --- |
|  |

### 第二步：Spring 的AOP配置

|  |
| --- |
|  |

### 第三步：测试

|  |
| --- |
|  |

# AspectJ

## 2.1 AspectJ简介

* AspectJ是一个基于Java语言的AOP框架
* Spring2.0以后新增了对AspectJ切点表达式支持
* @AspectJ 是AspectJ1.5新增功能，通过JDK5注解技术，允许直接在Bean类中定义切面
* **新版本Spring框架，建议使用AspectJ方式来开发AOP**
* **主要用途：自定义开发**

## 2.2 切入点表达式【掌握】

### execution()

用于描述方法 【掌握】

|  |
| --- |
| 语法：execution(修饰符 返回值 包.类.方法名(参数) throws异常) |

**修饰符，一般省略**

public 公共方法

\* 任意

**返回值，不能省略**

void 返回没有值

String 返回值字符串

\* 任意

**包，[省略]**

com.gyf.crm 固定包

com.gyf.crm.\*.service crm包下面子包任意 （例如：com.gyf.crm.staff.service）

com.gyf.crm.. crm包下面的所有子包（含自己）

com.gyf.crm.\*.service.. crm包下面任意子包，固定目录service，service目录任意包

**类，[省略]**

UserServiceImpl 指定类

\*Impl 以Impl结尾

User\* 以User开头

\* 任意

**方法名，不能省略**

addUser 固定方法

add\* 以add开头

\*Do 以Do结尾

\* 任意

**(参数)**

() 无参

(int) 一个整型

(int ,int) 两个

(..) 参数任意

**throws ,可省略，一般不写**。

**案例1：**

**execution(\* com.gyf.crm.\*.service..\*.\*(..))**

**案例2：或**

**<aop:pointcut expression="execution(\* com.gyf.crm.service.\*.\*(..)) ||**

**execution(\* com.gyf.\*Do.\*(..))" id="myPointCut"/>**

**第一个\* 是返回值**

### within:

匹配包或子包中的方法(了解)

|  |
| --- |
| within(com.gyf.aop..\*) |

### this:

匹配实现接口的代理对象中的方法(了解)

|  |
| --- |
| this(com.gyf.aop.user.UserDAO) |

### target:

匹配实现接口的目标对象中的方法(了解)

|  |
| --- |
| target(com.gyf.aop.user.UserDAO) |

### args:

匹配参数格式符合标准的方法(了解)

|  |
| --- |
| args(int,int) |

### bean(id)

对指定的bean所有的方法(了解)

|  |
| --- |
| bean('userServiceId') |

## 2.3 AspectJ 通知类型

|  |
| --- |
| aop联盟定义通知类型，具有特性接口，必须实现，从而确定方法名称。  aspectj 通知类型，只定义类型名称，以及方法格式。  个数：6种，**知道5种，掌握1**中。  before:前置通知(应用：各种校验)  在方法执行前执行，如果通知抛出异常，阻止方法运行  afterReturning:后置通知(应用：常规数据处理)  方法正常返回后执行，如果方法中抛出异常，通知无法执行  必须在方法执行后才执行，所以可以获得方法的返回值。  around:环绕通知(应用：十分强大，可以做任何事情)  方法执行前后分别执行，可以阻止方法的执行  **必须手动执行目标方法**  afterThrowing:抛出异常通知(应用：包装异常信息)  方法抛出异常后执行，如果方法没有抛出异常，无法执行  after:最终通知(应用：清理现场)  方法执行完毕后执行，无论方法中是否出现异常 |

|  |  |
| --- | --- |
|  |  |

查看源码

|  |
| --- |
|  |
|  |

## 2.4 Aspect案例讲解【其于xml】

### 第一步：导包

|  |  |
| --- | --- |
|  | AOP联盟规范 |
|  | AOP实现 |
|  | 规范 |
|  | aspectj实现 |

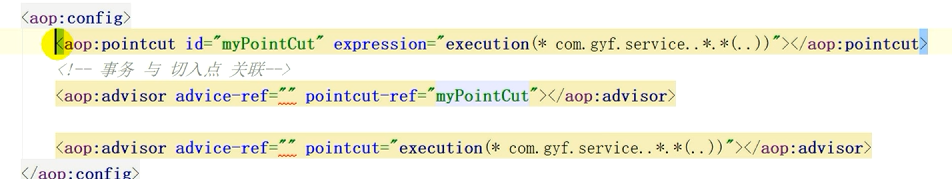
|  |
| --- |
|  |

### 第二步：实现类和切面类

|  |
| --- |
|  |
|  |

### 第三步：spring的xml配置

|  |
| --- |
|  |



这两种效果相同！

**看不懂了~ 好像看懂了…**

### 第四步：测试

|  |
| --- |
|  |

## 2.5 Aspect案例讲解【其于xml】

### 第1步：声明使用注解

|  |
| --- |
|  |

### 第2步：替换service和 切面 bean

|  |
| --- |
|  |
|  |
|  |

### 第3步：声明切面

|  |
| --- |
|  |
|  |

### 第4步：声明前置通知

|  |
| --- |
|  |

### 第5步：声明公共切入点

|  |
| --- |
|  |

随便写一个方法，声明公共切入点，此切入点其他方法就不用声明切入点了

### 第6步：声明后置通知

|  |
| --- |
|  |

**Value 这要指向 公共切入点**

### 第7步：声明环绕通知

|  |
| --- |
|  |

### 第8步：声明异常通知

|  |
| --- |
|  |

### 第9步：声明最终通知

|  |
| --- |
|  |

### 注解总结

@Aspect 声明切面，修饰切面类，从而获得 通知。

通知

@Before 前置

@AfterReturning 后置

@Around 环绕

@AfterThrowing 抛出异常

@After 最终

切入点

@PointCut ，修饰方法 private void xxx(){} 之后通过“方法名”获得切入点引用

可以通过这个注解，设置统一的 生效位置

# 3 JdbcTemplate

## 3.1 简介

* jdbcTemplate类似人DBUtils,用于操作Jdbc的工具类，它需要依赖于连接池DataSource(数据源)
* JDBC（Java DataBase Connectivity,java数据库连接）是一种用于执行SQL语句的Java API
* ODBC（Open Database Connectivity，ODBC）开放数据库连接,是微软公司开提供了一组对数据库访问的标准API（应用程序编程接口）
* DBCP（DataBase Connection Pool）数据库连接池，是java数据库连接池的一种，由Apache开发
* C3P0是一个开源的JDBC连接池，它实现了数据源和JNDI绑定，支持JDBC3规范和JDBC2的标准扩展。目前使用它的开源项目有Hibernate，Spring等。

**c3p0与dbcp区别**

dbcp没有自动回收空闲连接的功能

c3p0有自动回收空闲连接功能

## 3.2 环境搭建

### 创建数据库和表

|  |
| --- |
| create database spring\_day02;  use spring\_day02;  create table t\_user(  id int primary key auto\_increment,  username varchar(50),  password varchar(32)  );  insert into t\_user(username,password) values('jack','520');  insert into t\_user(username,password) values('rose','521'); |

### 创建工程导入Jar包

|  |
| --- |
|  |

### 创建JavaBean,数据模型

|  |
| --- |
|  |

## 3.3 API使用【了解】

|  |
| --- |
|  |

## 3.4 配置DBCP

|  |
| --- |
| beans.xml |
|  |
| 测试 |
|  |

## 3.5 配置c3p0

|  |
| --- |
|  |

## 3.6 使用JdbcDaoSupport

## 继承一个含有dts的类，不用添加dts属性了

### dao层

|  |
| --- |
|  |

继承 JdbcDaoSupport 其中有个setDataSource 方法 （就不用声明一个DataSource）

### Beans.xml

|  |
| --- |
|  |

### 源码分析

**通过数据源创建模板**

|  |
| --- |
|  |

## 3.7 配置properties

目数据库的连接信息配置到一个独立的文件中

### 在src写个db.properties文件

|  |
| --- |
|  |

### beans.xml

<context:property-placehoalder location=”classpath:db.properties”/>

|  |
| --- |
|  |

对于bean 这个东西，只要配置了class spring就会自动去创建 对象；

加注解 autoW…. 只事在开始时扫描整个包。