1 AOP思想

1.1 原理思想

横切逻辑场合

- 日志记录、事务控制、访问控制

AOP：面向切面编程

- 是OOP的扩展与补充，可以对业务逻辑的各部分进行隔离，
- 降低各部分之间的耦合度，提高程序的可重用性，提高开发效率。
- 在不修改源码的情况下，对业务功能进行增强

连接点（Joinpoint）：连接点是在应用执行过程中能够插入切面的一个点。

切入点(Pointcut)：定义了在何处应用切面，AOP 通过切入点定位到特定的连接点。

切面（Aspect）：切面是切入点和通知的组合，通知和切点共同定义了切面的内容

通知（Advice）：切面的工作称为通知，描述了切面的工作内容和执行工作的时间。

代理（Proxy）：即一个类被AOP织入通知后产生的新类

- 前置通知（Before）在目标方法被调用之前，调用前置通知；
- 后置通知（After） 在目标方法被调用完成之后，调用后置通知；
- 异常通知（After-throwing）在目标方法抛出异常之后，调用异常通知；
- 返回通知（After-returning）在目标方法成功执行之后，调用返回通知；
- 环绕通知（Around advice）在目标方法调用之前和调用之后，分别执行自定义的行为；

1.2 代理机制

代理机制原理

代理类和实际业务类应实现相同的功能，这是代理机制实现的根本

实现方式

- 接口方式：java 的 Proxy 类
- 继承方式：使用第三方的 cglib，要求被代理类不能 final

1) 静态代理

为每个实际业务类创建一个代理类
这种方式实际上代理类和实际业务类本质上相同,代理类只是起到一个转发的作用,这种代理会使系统中类规模增大，并且不易维护。

2) 基于接口的动态代理

通过接口的方式, 让代理类和实际业务类实现统一接口,并且借助代理类统一管理接
涉及的类：`Proxy`
创建代理对象的方法：`Proxy.newProxyInstance`

- `ClassLoader`，类加载器，和被代理对象使用相同的【类加载器】，固定写法
- `Class[]`，字节码数组，和被代理对象具有相同的【行为】、实现相同的【接口】，固定写法
- `InvocationHandler`，接口，代理的具体内容即【增强代码】，使用匿名内部类实现
    - 依赖于接口,如果一个类不实现接口则无法实现代理。

要点：实现 `InvocationHandler` 接口

3) 基于子类的动态代理

`Cglib` 采用了非常底层的字节码技术

- 其通过字节码技术为一个类创建子类,
- 在子类中采用方法拦截的技术拦截所有父类方法的调用,顺势织入横切逻辑。
- 因为是使用继承, 所以不能对 final 修饰的类进行代理。

相关包：`cglib-2.2.2.jar`、`asm-3.3.1.jar`
涉及的类：`Enhancer`
要点：实现 `MethodInterceptor` 接口

JDK 动态代理与 Cglib 的比较

- JDK 动态代理是利用反射机制生成一个实现代理接口的匿名类，再调用具体方法前调用 `InvokeHandler` 处理
- `Cglib` 动态代理是利用 `ASM` 开源包，对代理对象类的 class 文件加载，通过修改字节码生成子类处理

2 Sping AOP

- 对事务处理、数据库连接等高重复性工作分离出来单独作为一个模块
- 在程序编译或运行阶段，再将这些抽取出来的代码应用到需要执行的地方
- OOP 实现的是父子关系的纵向重用，不能实现横向抽取

即没有专门的编译器，也没有特殊的类装载器，在【运行期】通过代理方式向目标对象织入通知，又称为动态代理

动态代理：在不改变源码的基础上对已有的方法增强
动态代理与静态代理（装饰者模式）的区别：

- 动态代理的字节码随用随创建，随用随加载
- 静态代理的字节码一上来就建好，并完成加载

2.1 ProxyFactoryBean

- 全匹配方式：`public void com.qst.bean.User.add()`
- 省略访问修饰符：`void com.qst.bean.User.add()`
- `*`表示任意返回类型，不能省：`* com.qst.bean.User.add()`
- *表示任意包，有几级包写几个*：`*.*.*.*.User.add()`
- `*` 表示任意类，不能省：`* *.*.*.*.add()`
- `*` 表示任意方法，不能省：`* *.*.*.*.*()`
- `..`表示参数列表中任意参数：`* *.*.*.*.*(..)`

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
       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/context
                           http://www.springframework.org/schema/context/spring-
context.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop.xsd">

  <!-- 1、定义业务bean -->
  <bean id="mathInterImpl" class="com.qst.pkg5.MathInterImpl"></bean>
  <!-- 2、定义切面bean -->
  <bean id="timeTool" class="com.qst.pkg5.TimeTool"></bean>
  <aop:config>
    <!-- 3、定义切入点 -->
    <aop:pointcut id="subPointCut"
                  expression="execution (public int
com.qst.*.MathInterImpl.sub(..))"/>
    <!-- 4、定义切面 -->
    <aop:aspect ref="timeTool">
      <!-- 4.1、配置切面的before方法为切入点的前置通知 -->
      <aop:before method="before" pointcut-ref="subPointCut"/>
      <!-- 4.2、配置切面的after方法为切入点的后置通知 -->
      <aop:after  method="after"  pointcut-ref="subPointCut"/>
    </aop:aspect>
  </aop:config>
</beans>
```