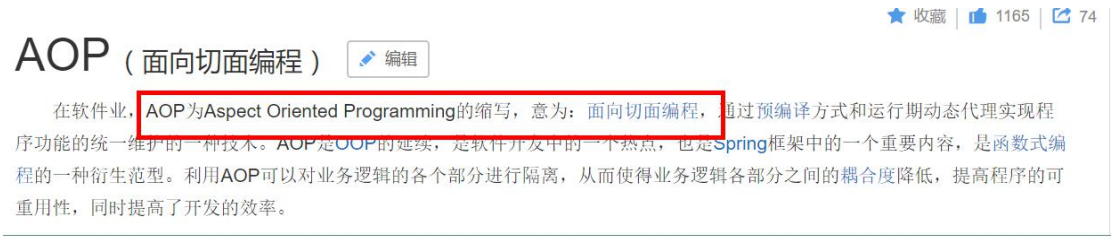
本章学习目标

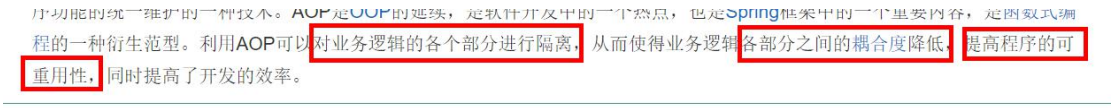
- AOP 思想概述
- AOP 底层技术实现
- AOP 术语介绍
- SpringAOP 的 XML 方式 HelloWorld
- SpringAOP 的 XML 方式配置细节
- SpringAOP 的注解方式
- SpringAOP 的零配置方式

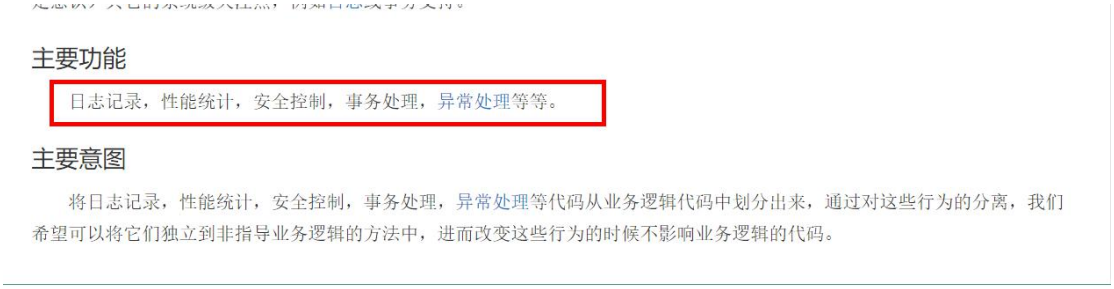
1. AOP 思想概述

1.1. AOP 思想简介



1.2. AOP 的作用





2. AOP 底层技术实现

关键词：代理模式

代理模型分为两种：

1）接口代理（JDK 动态代理）

2）子类代理（Cglib 子类代理）

需求：CustomerService 业务类，有 save，update 方法，希望在 save，update 方法执行之前记录日志。

2.1. 接口代理（JDK 动态代理）

```
public class JDKProxyUtils {


    /**
     * 使用 JDK 动态代理获取代理对象
     * target: 目标对象
     * @return
     */
    public static Object getProxy(final Object target){
        return Proxy.newProxyInstance(
                target.getClass().getClassLoader(),    // 和目标对象一样的
类加载器
                target.getClass().getInterfaces(), // 目标对象的接口列表
                new InvocationHandler() {


                    //invoke: 这个方法在每次调用代理类对象的时候被执行啦！！！
                    @Override
```

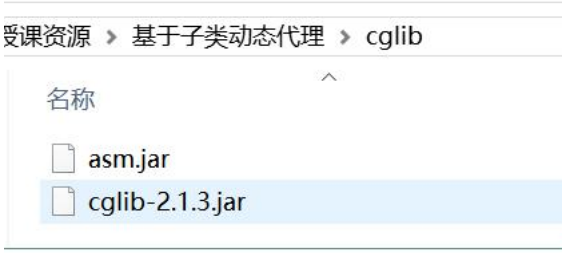
```
                public Object invoke(Object proxy, Method method,
Object[] args)
                        throws Throwable {
                    System.out.println("记录日志");

                    //调用目标对象的方法
                    return method.invoke(target, args);
                }
            });
    }
}
```

注意：JDK 动态代理必须是基于接口的代理！

如果目标对象没有接口，那么只能子类代理！

2.2. 子类代理（Cglib 方式）

2.2.1. 导入 cglib 的 jar 包



2.2.2. 编写 Cglib 子类代理代码

```
public class CglibProxyUtils {


    /**
```

```
     * 使用 Cglib 工具创建目标对象的子类对象
     * @param target
     * @return
     */
    public static Object getProxy(final Object target){
        return Enhancer.create(CustomerServiceImpl2.class, new
MethodInterceptor() {
            //intercept：每次代理类对象执行方法的时候执行该方法
            @Override
            public Object intercept(Object arg0, Method method, Object[]
arg2,
                    MethodProxy arg3) throws Throwable {
                System.out.println("记录日志");
                //调用目标对象的方法
                return method.invoke(target, arg2);
            }
        });
    }
}
```

3. AOP 术语介绍

1）目标对象（Target）

2）代理对象（Proxy）

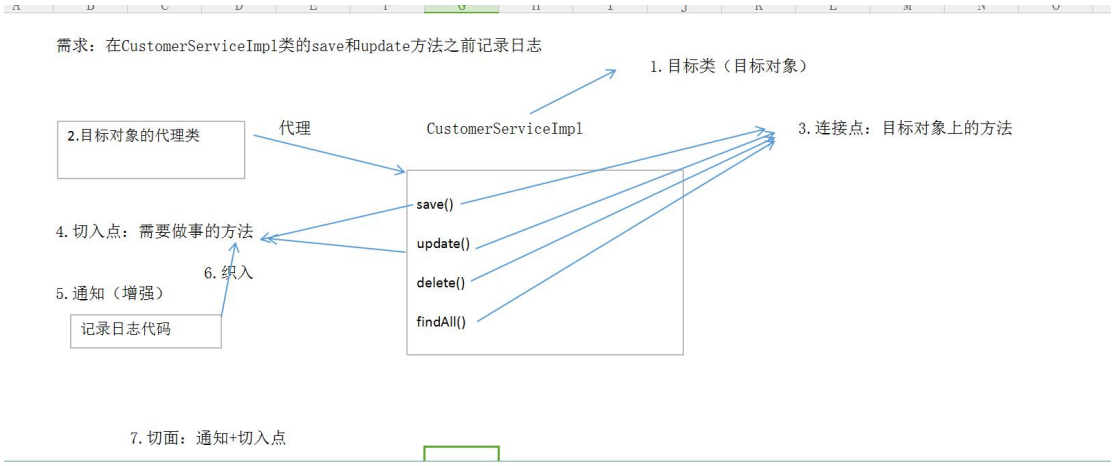
3）连接点（Joinpoint）

4）切入点（Pointcut）

5）通知（增强）（Advice）

6）切面（Aspect、Advisor）

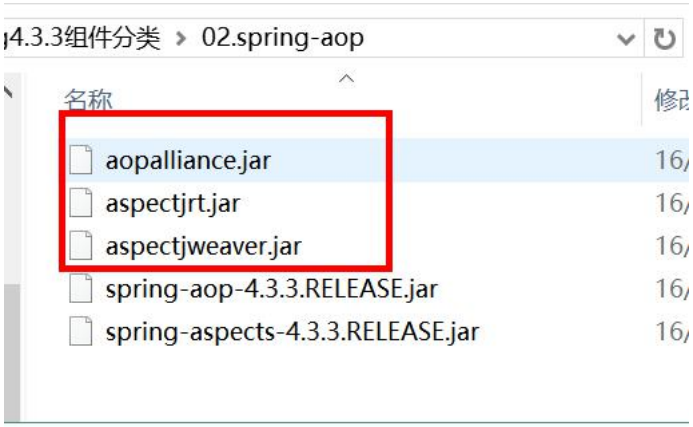
7）织入（切入）（weaving）



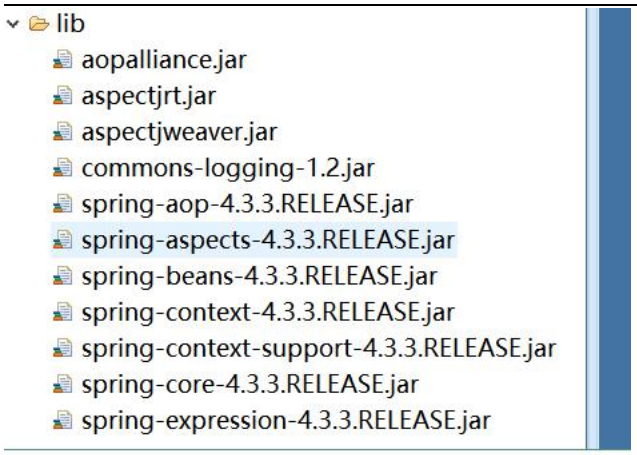
4. SpringAOP 的 XML 方式

4.1. HelloWolrd 入门程序

4.1.1. 导入 spring 的 aop 相关 jar 包



注意：使用了 aspectJ 插件，作用是简化 Spring 的 XML 方式 AOP 编程



4.1.2. 编写目标类（有接口的情况）

```
/**
 * 目标对象
 * @author lenovo
 *
 */
public class CustomerServiceImpl implements CustomerService {


    @Override
    public void save() {
        System.out.println("执行 save 方法");
    }


    @Override
    public void update() {
        System.out.println("执行 update 方法");
    }


}
```

4.1.3. 编写切面类

```
/**
 * Spring 的 AOP 的切面类
 * @author lenovo
 *
 */
public class MyAspect1 {


    /**
     * 通知方法
     */
    public void log(){
        System.out.println("使用 spring 的 AOP 切入日志...");
    }
}
```

4.1.4. 编写 applicationContext.xml（重点）

引入 aop 新的名称空间：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    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">
```

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

    xmlns:p="http://www.springframework.org/schema/p"
```

```
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    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">

    <!-- spring 的 AOP 编写-- XML 方式 -->

    <!-- 1.创建目标对象 -->
    <bean id="customerService"
class="cn.sm1234.service.impl.CustomerServiceImpl"/>

    <!-- 2.创建切面类对象 -->
    <bean id="myAspect1" class="cn.sm1234.apsect.MyAspect1"/>

    <!-- 3.配置 AOP 切面 -->
    <aop:config>
        <!-- 切面 = 通知+切入点 -->
        <aop:aspect ref="myAspect1">
            <aop:before method="Log" pointcut-ref="pt"/>
            <aop:pointcut expression="execution(public void
cn.sm1234.service.impl.CustomerServiceImpl.*())" id="pt"/>
        </aop:aspect>
    </aop:config>

</beans>
```

4.1.5. 编写测试代码

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class Demo1 {


    @Resource
    private CustomerService customerService;


    @Test
    public void test1(){
        customerService.save();
        customerService.update();
    }
}
```

4.2. SpringAOP 的 XML 配置细节

4.2.1. 切入点表达式

```
<aop:config>
        <aop:aspect ref="myAspect1">
            <aop:before method="log" pointcut-ref="pt"/>
            <!--
                切入点表达式的语法
                    1. execution(): 代表切入方式，固定语法
                    2. public: 方法的修饰符 ，通常为 public 方法
                    3. void: 方法的返回值。可以使用通配符： *
                    4. cn.sm1234.service.impl: 类所在的包
```

```
                    4.1 可以使用通配符:   *  (* 只能匹配一级目录)
                    4.2 可以使用 *..* 匹配任意级目录
                5.CustomerServiceImpl: 类名称
                    5.1 可以使用通配符:   *  （匹配任意字符）
                6. save() : 代表方法
                    6.1 可以使用通配符:   *  （匹配任意字符）
                7.  方法的参数
                    7.1 可以使用通配符:  ..  (匹配任何参数类型的方法)


            -->
           <aop:pointcut expression="execution(public *
cn.sm1234.*..*.*ServiceImpl.*(..))" id="pt"/>
        </aop:aspect>
    </aop:config>
```

4.2.2. 通知类型

4.2.2.1. 前置通知

在方法的前面执行

4.2.2.2. 最终通知

在方法的最后执行，无论方法是否出现异常，通知都会被执行

4.2.2.3. 后置通知

在方法的最后执行，只有在方法成功执行之后才被执行

4.2.2.4. 异常通知

在方法出现异常的时候执行

4.2.2.5. 环绕通知

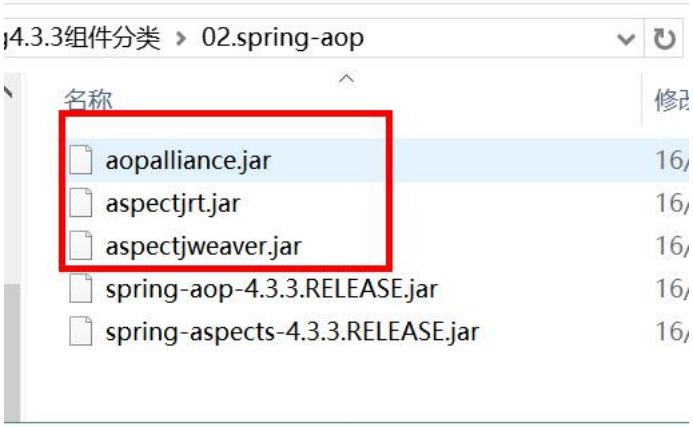
在方法的前后执行

```
<aop:config>
        <aop:aspect ref="myAspect1">
            <!-- <aop:before method="before" pointcut-ref="pt"/> -->
            <!-- <aop:after method="after" pointcut-ref="pt"/> -->
            <!-- <aop:after-returning method="afterReturning"
pointcut-ref="pt"/>    -->
            <!-- <aop:after-throwing method="afterThrowing"
pointcut-ref="pt"/> -->
            <aop:around method="around" pointcut-ref="pt"/>
            <aop:pointcut expression="execution(public *
cn.sm1234.service.impl.CustomerServiceImpl.*(..))" id="pt"/>
        </aop:aspect>
    </aop:config>
```

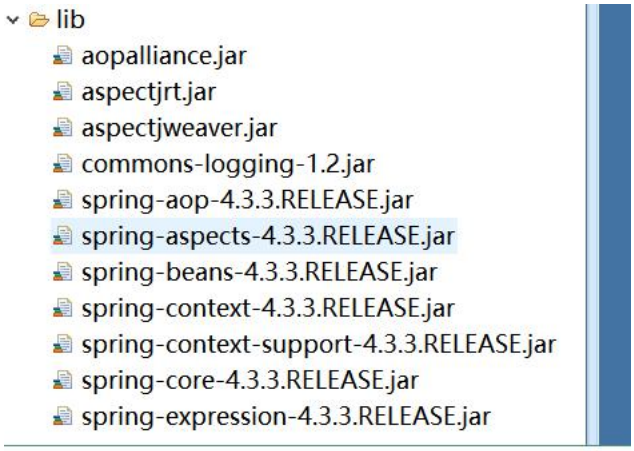
5. SpringAOP 的注解方式

5.1. HelloWorld 的入门程序

5.1.1. 导入 spring 的 aop 相关 jar 包



注意：使用了 aspectJ 插件，作用是简化 Spring 的 XML 方式 AOP 编程



5.1.2. 编写目标类（有接口的情况）

```
/**
 * 目标对象
 * @author lenovo
 *
 */
```

```
public class CustomerServiceImpl implements CustomerService {


    @Override
    public void save() {
        System.out.println("执行 save 方法");
    }


    @Override
    public void update() {
        System.out.println("执行 update 方法");
    }


}
```

5.1.3. 编写切面类，添加切面相关的注解

```
/**
 * 注解版本的切面类
 * @author lenovo
 *
 */
@Aspect
public class MyAspect {


    @Before(value="execution(public *
cn.sm1234.service.impl.CustomerServiceImpl.*.(..))")
    public void log(){
        System.out.println("记录日志");
    }
}
```

5.1.4. 编写 applicationContext.xml（重点）

引入 aop 新的名称空间：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    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">
```

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

    xmlns:p="http://www.springframework.org/schema/p"

    xmlns:aop="http://www.springframework.org/schema/aop"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    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">


    <!-- spring 的 AOP 编写之注解方式 -->


    <!-- 1.创建目标对象 -->

    <bean id="customerService"

class="cn.sm1234.service.impl.CustomerServiceImpl"/>


    <!-- 2.创建切面类对象 -->
```

```
    <bean id="myAspect" class="cn.sm1234.apsect.MyAspect"/>


    <!-- 3.开启 AOP 切面注解 -->
    <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
</beans>
```

5.1.5. 编写测试

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class Demo1 {


    @Resource
    private CustomerService customerService;


    @Test
    public void test1(){
        customerService.save(10);
        customerService.update();
    }
}
```

```
@Before(value = "execution(public *
cn.sm1234.service.impl.CustomerServiceImpl.*(..))")
    public void before() {
        System.out.println("执行前置通知....");
    }


    // 最终通知
```

```
    @After(value = "execution(public *
cn.sm1234.service.impl.CustomerServiceImpl.*(..))")
    public void after() {
        System.out.println("执行最终通知...");
    }



    // 后置通知
    @AfterReturning(value = "execution(public *
cn.sm1234.service.impl.CustomerServiceImpl.*(..))")
    public void afterReturning() {
        System.out.println("执行后置通知...");
    }



    // 异常通知
    @AfterThrowing(value = "execution(public *
cn.sm1234.service.impl.CustomerServiceImpl.*(..))")
    public void afterThrowing() {
        System.out.println("执行异常通知...");
    }



    // 环绕通知
    @Around(value = "execution(public *
cn.sm1234.service.impl.CustomerServiceImpl.*(..))")
    public void around(ProceedingJoinPoint jp) {
        System.out.println("前面执行的代码....");
        // 执行目标的方法
```

```
        try {
            jp.proceed();
        } catch (Throwable e) {
            e.printStackTrace();
        }
        System.out.println("后面执行的代码....");
    }
```

6. SpringAOP 的零配置方式

```
/**
 * Spring 配置类
 * @author lenovo
 *
 */
@Configurable
@ComponentScan(basePackages="cn.sm1234")
@EnableAspectJAutoProxy  // 开启 AOP 注解功能
public class SpringConfig {


}
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=SpringConfig.class)
public class Demo1 {


    @Resource
    private CustomerService customerService;
```

```
    @Test
    public void test1(){
        customerService.save(10);
        customerService.update();
    }
}
```