基于XML的AOP配置，搭建SpringAOP的开发环境

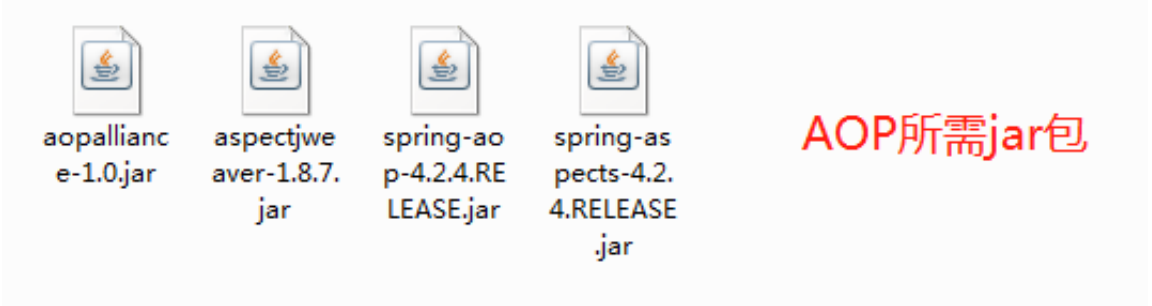
步骤：

a、导入必备的jar包

spring的所有组件运行都需要IOC的支持。

必须先导入IOC的jar包。

再导入aop的jar包。aop一共有4个jar包。



b、创建spring的核心配置文件

在类的根路径下创建一个bean.xml文件

c、导入schema约束

注意：不仅要导入ioc的约束，还要导入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">
</beans>
```

```
<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">
</beans>
```

d、把资源都交给spring来管理

IOC的配置

配置业务实现类、工具类、业务方法增强类、持久层类等交由Spring来管理

```
<!-- 把业务层的创建交给spring来管理 -->
<bean id="customerService" class="com.itheima.service.impl.CustomerServiceImpl"></bean>

<!-- 把通知(增强)类交给spring来管理 -->
<bean id="logger" class="com.itheima.utils.Logger"></bean>
```

e、aop配置

aop:config标签是用于开始aop的配置

aop:aspect标签指定使用的bean。

属性：

id：指定一个唯一标识符

ref：指定通知类bean的id

aop:before标签：用于指定通知的类型是前置通知。

```
<!-- 用于增强的方法,配置通知的类型：前置通知 -->
<!-- 配置通知的类型：
        前置通知:aop:before 永远在业务核心方法之前执行
        后置通知：aop:after-returning
        异常通知：aop:after-throwing
        最终通知：aop:after
<aop:before method="beforePrintLog" pointcut-ref="pt1"/>
<aop:after-returning method="afterReturningLog" pointcut-ref="pt1"/>
<aop:after-throwing method="afterThrowingLog" pointcut-ref="pt1"/>
<aop:after method="afterLog" pointcut-ref="pt1"/>
-->
<aop:before method="printLog" pointcut-ref="pt1"/>
```

属性：

method：指定通知类中哪个方法用于增强

pointcut：指定切入点表达式

```
<aop:config>
    <!-- aop:pointcut 定义通用切入点表达式（指明是给哪个类哪个方法增强），所有切面可用-->
    <!-- pointcut属性：用于指定切入点表达式
                切入点表达式：指定在业务层实现类中哪个方法上织入增强的代码
                切入点表达式写法：关键字：execution(表达式)
                    表达式写法：访问修饰符 返回值 包名.包名....类名.方法名(参数列表)
                    全匹配：public void com.itheima.service.impl.CustomerServiceImpl.saveCustomer()
                    访问修饰符可以省略    void com.itheima.service.impl.CustomerServiceImpl.saveCustomer()
                    返回值可以使用*代替，表示任意返回值
                        * com.itheima.service.impl.CustomerServiceImpl.saveCustomer()
                    包名可以使用*来代替，表示任意包名称。但是不能表示曾经，也就是说有几级包，就需要写几个*
                        * *.*.*.*.CustomerServiceImpl.saveCustomer()
                    包名可以使用..来代替，表示当前包及其子包
                        * com..CustomerServiceImpl.saveCustomer()
                    类名可以使用*号代替，表示任意类名称
                        * com..*.saveCustomer()
                    方法名称可以使用*号代替，表示任意方法
                        * com..*.*()
                    参数列表可以使用*号代替，表示任意数据类型，但是有一个*号就对应一个参数      * com..*.*(*)
                    参数类别可以使用..代替，表示有无参数均可。有参数的话，参数可以是任意数据类型      * com..*.*(..)
                    全通配       * *..*.*(..)
                    实际开发中：一般都都切到业务层实现类，使用部分匹配的方式
                        例如:      * com.itheima.service.impl.*.*(..)
        -->
    <aop:pointcut expression="execution(* com.itheima.service.impl.*.*(..))" id="pt1"/>
```

pointcut-ref：引用通用切入点表达式

aop:pointcut标签：用于定义通用的切入点表达式

属性：

expression：指定表达式

id：指定表达式的唯一标识符

通知的类型：

前置通知：aop:berfore

后置通知：aop:after-returning

异常通知：aop:after-throwing

最终通知：aop:after

环绕通知：aop:around

```
<!-- 把通知(增强)类交给spring来管理 -->                    1
<bean id="logger" class="com.itheima.utils.Logger"></bean>
```

这个类是用来给业务类方法增强用的IOC配置

```
<!-- aop配置 -->
<aop:config>                    3 可以配置到通知类型里
    <!-- aop:pointcut 定义通用切入点表达式，所有切面可用，即是给哪个类哪个方法增强-->
    <aop:pointcut expression="execution(* com.itheima.service.impl.*.*(..))" id="pt1"/>
    <!-- 配置（用于增强业务类方法的类）切面 -->2
    <aop:aspect id="logAdvice" ref="logger">
    <!-- 配置通知的类型：这个增强的方法是在业务类方法的什么时间执行-->
        <aop:before method="printLog" pointcut-ref="pt1"/>  4 配置通知类型
    </aop:aspect>
</aop:config>
```

AOP引用IOC配置来告诉Spring它是用于给业务增强的类

spring框架为我们提供的一种可以在代码中手动控制通知执行时间点的方式。

配置环绕通知：作用是在代码中手动控制通知执行时间点

可以在通知类中的通知方法中指定：

1、首先要在配置文件中配置通知类型为：环绕通知

```
<aop:around method="aroundLog" pointcut-ref="pt1"/>
```

2、在通知类（方法增强类）中对环绕通知的方法手动指明：

方法参数使用：ProceedingJoinPoint接口

调用pjp.proceed()： 接口方法，就相当于执行了业务类的核心方法

写在调用接口方法前的操作就是前置加强

写在调用接口方法后的操作就是后置加强

如果没有配置文件中配置了环绕通知，而增强方法中没有进行手动配置那么，运行程序就不会执行业务核心方法，只会执行增强方法

```
/**
 * 配置环绕通知
 * 解决办法：
 *  在执行通知代码时，明确的调用业务核心方法一下。
 *  spring框架为我们提供了一个接口ProceedingJoinPoint，该接口可以作为环绕通知的参数
 *  当程序执行时，spring框架会为我们提供该接口的实现类，供我们使用。
 *  ProceedingJoinPoint接口中有一个方法：proceed()，该方法就相当于invoke方法
 *  环绕通知：
 *      spring框架为我们提供的一种可以在代码中手动控制通知执行时间点的方式。
 */
public void aroundLog(ProceedingJoinPoint pjp){
    try {
        System.out.println("Logger类中的aroundLog方法开始记录日志了。。。。。前置");
        pjp.proceed();//就相当于执行了业务类的核心方法
        System.out.println("Logger类中的aroundLog方法开始记录日志了。。。。。后置");
    } catch (Throwable e) {
        System.out.println("Logger类中的aroundLog方法开始记录日志了。。。。。异常");
        e.printStackTrace();
    }finally{
        System.out.println("Logger类中的aroundLog方法开始记录日志了。。。。。最终");
    }
```