不同场景下的aop

maven引入本地jar包的方法 – 腾讯云开发者社区-腾讯云

打包时需要将依赖一起打包，可以引入maven-assembly-plugin插件

1. Junit

1.1 注解的形式

扩展机制 · junit5学习笔记

1. 实现需要的注解
2. 针对junit5利用 @ExtendWith 注解可以声明在测试方法和类的执行中启用相应的扩展，来实现在测试类方法执行前后插入自定义的逻辑。需要将@ExtendWith 注解和自定义的注解联合使用。提供了一系列类似于 `BeforeEachCallback` 的接口，实现它们并重写对应的方法即可在测试方法前后插入自己的逻辑。
3. 在junit4中也可以使用扩展机制，使用的方式不一样。需要用到@Rule注解，实现一个 `implements TestRule` 的自定义类，在类里面实现自定义逻辑。

使用时需要在测试类中添加如下代码：

```java
@Rule
public AnnotationRule annotationRule = new AnnotationRule();
```

4. 将项目打成jar包，idea的mvn package即可

1.2 配置文件的方式

1.2.1 针对junit4

JUnit 测试监听器– JUnit RunListener示例 · HowToDoInJava 中文系列教程 · 看云

1. 读取配置文件并解析得到需对应需要的参数
2. 利用junit提供的测试监听器(JUnit RunListener)来实现对每个测试方法执行时的监听，需要 `extends RunListener` 。可以重写其中的testStarted和testFinished方法，来实现在方法前后插入自己的逻辑
3. 将项目打成jar包，idea的mvn package即可
4. 在项目中引入jar包和如下插件，其中 `<value>org.example.ExecutionListener</value>` 对应的是jar包中继承了JUnit RunListener 的类。

```xml
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.22.0</version>
  <configuration>
    <properties>
      <property>
        <name>listener</name>
        <value>org.example.ExecutionListener</value>
      </property>
    </properties>
  </configuration>
</plugin>
```

5. 运行 `mvn test` 命令来进行测试，可以选择跑所有的测试，也可以执行单个测试或者方法。（直接使用idea运行测试不会生效，idea不会从pom 文件中解析上述的插件配置）

6. 若运行时出现找不到org.example.ExecutionListener的报错，则说明jar包未正确引入。可以添加如下配置：

```xml
<dependency>
    <groupId>xxxx</groupId>
    <artifactId>xxxx</artifactId>
    <version>1.0-SNAPSHOT</version>
    <scope>system</scope>
    <systemPath>jar包的本地路径</systemPath>
</dependency>
```

1.2.2 针对junit5

1. 读取配置文件并解析得到对应需要的参数
2. 实现一个 `CustomTestExecutionListener` 监听器类（该类需要 `implements TestExecutionListener` ），重写 `TestExecutionListener` 的 `executionStarted和executionFinished` 方法，监听测试方法的执行和结束并插入自己的逻辑。

```java
public class CustomTestExecutionListener implements TestExecutionListener
{

    @Override
    public void testPlanExecutionStarted(TestPlan testPlan) {
    }

    @Override
    public void testPlanExecutionFinished(TestPlan testPlan) {

    }

    @Override
    public void dynamicTestRegistered(TestIdentifier testIdentifier) {
    }

    @Override
    public void executionSkipped(TestIdentifier testIdentifier, String rea
son) {
    }

    @Override
    public void executionStarted(TestIdentifier testIdentifier) {
    }

    @Override
    public void executionFinished(TestIdentifier testIdentifier, TestExecu
tionResult testExecutionResult) {
    }

    @Override
    public void reportingEntryPublished(TestIdentifier testIdentifier, Rep
ortEntry entry) {
    }
}
```

3. Junit5可以利用Java提供的ServiceLoader机制添加监听器。在项目的resources目录下建立 `META-INF/services` 目录， 添加一个文件 `org.junit.platform.launcher.TestExecutionListener` ，文件中的内容是完全限定的监听器类名。
4. 将项目打成jar包对外提供服务，打包时需要引入 `maven-assembly-plugin` 插件将依赖一并打包。

```xml
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-assembly-plugin</artifactId>
  <configuration>
    <!-- get all project dependencies -->
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
  </configuration>
  <executions>
    <!-- bind to the packaging phase -->
    <execution>
      <id>make-assembly</id>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

5. 在项目中引入jar包，Juint在运行时会自动找到 `CustomTestExecutionListener` 中的逻辑，在方法执行前后执行对应的方法。

2. TestNG

2.1 注解的形式

1. 实现需要的自定义注解
2. 通过testng提供的扩展机制（通过@Listeners注解配合实现IInvokedMethodListener接口中的afterInvocation和beforeInvocation方法可以声明在测试方法和类的执行中启用相应的扩展）来实现在测试类方法执行前后插入自定义的逻辑。
3. 将项目打成jar包对外提供服务
4. 使用时需要在测试类前添加@Listeners(xxx.class)注解，在测试方法前添加自定义的注解，配合在一起使用

也可以使用类似于2.2的方式实现

2.2 配置文件的形式

TestNG

TestNG Listeners Example

1. 读取配置文件并解析得到需要的参数
2. 实现一个ExecutionListener监听器类（该类需要implements ITestListener），重写ITestListener的onTestStart和onTestSuccess方法，监听测试方法的执行和结束，并且可以在这两个方法中实现自己的逻辑，在测试方法执行前后执行。

```java
public class ExecutionListener extends Listener implements ITestListener {


    @Override
    public void onTestStart(ITestResult iTestResult) {
    }

    @Override
    public void onTestSuccess(ITestResult iTestResult) {
    }

    @Override
    public void onTestFailure(ITestResult iTestResult) {

    }

    @Override
    public void onTestSkipped(ITestResult iTestResult) {

    }

    @Override
    public void onTestFailedButWithinSuccessPercentage(ITestResult iTestResult) {

    }

    @Override
    public void onStart(ITestContext iTestContext) {

    }

    @Override
    public void onFinish(ITestContext iTestContext) {

    }
}
```

3. TestNG可以利用Java提供的ServiceLoader机制添加监听器。在项目的resources目录下建立META-INF/services目录， 添加一个文件org.testng.ITestNGListener，文件中的内容是完全限定的监听器类名，比如 `org.example.ExecutionListener` 的形式。 `ExecutionListener` 这个类不需要编译成.class文件。
4. 将项目打成jar包对外提供服务，打包时需要引入maven-assembly-plugin插件将依赖一并打

包。

5. 在使用TestNG进行测试的项目中引入该jar包，TestNG在运行时会自动找到ExecutionListener中的逻辑，在方法执行前后执行对应的方法。

3. 通用的方案

3.1 java-agent

使用java-agent配合javassist对类的字节码进行操作，可以指定在某一行对代码进行修改或者增加自己的逻辑。

可以参考另一篇笔记。

3.2 sandbox

3.3 原生的AspectJ

原生AspectJ用法分析以及Spring-AOP原理分析

使用插件 aspectj-maven-plugin 织入 AspectJ AOP | 隔叶黄莺 Yanbin Blog – 软件编程实践

AspectJ 使用介绍_Javadoop

从一个Aspectj织入失效问题的解决说起_互联网全栈架构的博客-CSDN博客

探究使用原生AspectJ时，@Aspect注解不生效和@Around环绕注解执行两次原因 – 编程猎人

1. 实现注解和切面。切面的连接点定义如下：

```java
@Pointcut("execution(* *(..)) && @annotation(org.example.annotation.xxx)")
```

连接点中只使用@annotation(xxxx)定义的话，会导致在调用加了该注解的方法的方法中再次重复织入代码，这属于Ajc编译器的一个Bug，需要通过上述方法来解决。

2. 将实现的项目打成jar包对外部项目提供服务
3. 使用该jar包的项目成功引入jar包后需要在pom.xml文件中引入如下dependency和plugin。

```xml
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjrt</artifactId>
  <version>1.8.9</version>
</dependency>
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.9.3</version>
</dependency>


<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>aspectj-maven-plugin</artifactId>
  <version>1.10</version>
  <configuration>
    <complianceLevel>1.8</complianceLevel>
    <source>1.8</source>
    <!---weaveDependencies属性用来解决aspectj-maven-plugin插件在第三方项目中编译时没有编织切面的问题。groupId和artifactId对应的是jar包项目对应的groupId和artifactId。-->
    <weaveDependencies>
      <weaveDependency>
        <groupId>org.example</groupId>
        <artifactId>xxxxxxx</artifactId>
      </weaveDependency>
    </weaveDependencies>
  </configuration>
  <executions>
    <execution>
      <goals>
        <!---在编译期织入切面，其中<goal>compile</goal>会编织所有的主类，<goal>test-compile</goal>会编织所有的测试类，可以根据需要进行选择。-->
        <goal>compile</goal>
        <goal>test-compile</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

4. 在每次新增一个注解或者删除一个注解时，开发人员需要运行对应的mvn命令来使得注解生效。
    a. 若要在项目主类上生效，需要运行mvn compile –f pom.xml命令

b. 若要在项目测试类上生效，需要运行mvn test –f pom.xml命令