使Maven 2在package、install等阶段跳过运行Test的配置

方法1:

To skip running the tests for a particular project, set the skipTests property to true.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>3.0.0</version>
    <configuration>
        <configLocation>config/checkstyle_checks.xml</configLocation>
        <encoding>${project.build.sourceEncoding}</encoding>
        <consoleOutput>true</consoleOutput>
        <failsOnError>true</failsOnError>
        <includeTestSourceDirectory>true</includeTestSourceDirectory>
        <excludes>generated/*,${checkstyle.excludes}</excludes>
        <skip>true</skip>
        <skipExec>false</skipExec>
    </configuration>
    <executions>
        <execution>
```

方法2:

You can also skip the tests via command line by executing the following command:

mvn install -DskipTests

方法3:

If you absolutely must, you can also use the maven.test.skip property to skip compiling the tests.

maven.test.skip is honored by Surefire and the Compiler Plugin.

mvn install -Dmaven.test.skip=true

定义属性

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.encoding>UTF-8</maven.compiler.encoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <checkstyle.excludes></checkstyle.excludes>
    <pmd.minimumPriority>-1</pmd.minimumPriority>
    <maven.build.timestamp.format>yyMMddHHmmss</maven.build.timestamp.format>
    <build.timestamp>${maven.build.timestamp}</build.timestamp>
    <deployVersion>202201.02.000</deployVersion>
    <jarVersion>1.20220102.0</jarVersion>
    <check.skip>false</check.skip>
    <pmd.skip>false</pmd.skip>
</properties>

<dependencies...>
```

引用配置

```xml
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>3.0.0</version>
    <configuration>
        <configLocation>config/checkstyle_checks.xml</configLocation>
        <encoding>${project.build.sourceEncoding}</encoding>
        <consoleOutput>true</consoleOutput>
        <failsOnError>true</failsOnError>
        <includeTestSourceDirectory>true</includeTestSourceDirectory>
        <excludes>generated/*,${checkstyle.excludes}</excludes>
        <skip>${check.skip}</skip>
    </configuration>
    <executions>
        <execution>
            <id>validate</id>
            <phase>validate</phase>
```

命令行运行，会替换pom中已经存在的属性的值

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02:11 min
[INFO] Finished at: 2022-04-01T14:07:59+08:00
[INFO] ------------------------------------------------------------------------

D:\workspace\gitworkspace\PIBS\server\hsfund-ins3-pibs>mvn install -Dcheck.skip=true -Dpmd.skip=true
```

maven各参数的含义

- mvn -D

mvn -D, --define <arg>

`mvn -DpropertyName=propertyValue clean package` 可以用来临时定义属性和值。如果 pom.xml 中已经有该属性，那么会替换掉 pom.xml 中的值。

如果需要定义多个变量，可以用空格分隔

`mvn -DpropA=valueA -DpropB=valueB -DpropC=valueC clean package`

当然这个属性也可以直接在 pom.xml 文件下配置

```xml
<project>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <android.sdk.path>C:\software\android\sdk</android.sdk.path>
        <maven.test.skip>true</maven.test.skip>
        <maven.javadoc.skip>true</maven.javadoc.skip>
    </properties>
</project>
```

- mvn -P

mvn -P, --activate-profiles <arg>

P 表示 Profiles 配置文件，需要在 <profile> 标签中指定 <id> 才能用 -P 使之生效。

假如 pom.xml 如下：

```
<project>
    ...
    <profiles>
        <profile>
            <id>test</id>
            ...
        </profile>
        <profile>
            <id>dev</id>
            ...
        </profile>
        <profile>
            <id>prod</id>
            ...
        </profile>
    </profiles>
    ...
</project>
```

那么打包时执行 `mvn clean package -P prod` 将只激活 prod 环境的 profile 配置（也就是说此时 test 和 dev 的配置不会生效）

- profiles:

对不同的配置赋予不同的属性

```
<profiles>
    <profile>
        <id>dev</id>
        <properties>
            <mysql.url>http://localhost:3306</mysql.url>
        </properties>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
    </profile>
    <profile>
        <id>prod</id>
        <properties>
            <mysql.url>http://192.168.1.10:3306</mysql.url>
        </properties>
    </profile>
</profiles>
```

activation 元素用来指定激活条件。当没有指定条件，然后指定 activeByDefault 为 true 的时候，就表示没有指明 profile 的情况下默认激活。也就是说 `mvn package` 将使用 dev 的配置，而 `mvn package -P prod` 的时候使用的才是 prod 的配置。

- denpendencyManagement 与 pluginManagement

denpendencyManagement是用于管理项目jar包依赖，pluginManagement是用于管理plugin。

它们的作用是列出依赖的jar包或者plugin包，让子pom来决定是否引用。

- 根据不同配置使用不同的源文件

根据不同配置使用不同的源文件

```
<profiles>
    <profile>
        <id>dev</id>
        <properties>
    <debug.enable>true</debug.enable>
    </properties>
        <plugin>
            <artifactId>maven-resources-plugin</artifactId>
            <version>2.6</version>
            <executions>
                <execution>
                    <id>copy-id</id>
                    <goals>
                        <goal>copy-goal</goal>
                    </goals>
                    <phase>validate</phase>
                    <configuration>
                        <outputDirectory>${basedir}/src/com/companyName/global/</out
                        <resources>
                            <directory>${basedir}/profiles/dev/</directory>
                            <filtering>true</filtering>
                            <includes>
                                <include>**/*.java</include>
                            </includes>
                        </resources>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </profile>
</profiles>
```

然后在 profiles/dev 目录下新建 java 文件

```
(略)
    private boolean isDebugEnable = ${debug.enable}
    (略)
```

执行打包命令 `mvn clean package -DskipTests -Pdev` ，就可以看到 /profiles/dev 目录下的 java 文件全部被复制到了 /src/com/companyName/global 目录下了，并且 isDebugEnable 的值也被替换了。

配置项说明：

parse 用来指明具体在 maven 的哪个生命周期执行该任务

- maven 的生命周期表

生命周期阶段	描述
validate	验证项目是否正确，并且所有必要的信息可用于完成构建过程
initialize	建立初始化状态，例如设置属性
generate-sources	产生任何的源代码包含在编译阶段
process-sources	处理源代码，例如，过滤器值
generate-resources	包含在包中产生的资源

process-resources	复制和处理资源到目标目录，准备打包阶段
compile	编译该项目的源代码
process-classes	从编译生成的文件提交处理，例如：Java类的字节码增强/优化
generate-test-sources	生成任何测试的源代码包含在编译阶段
process-test-sources	处理测试源代码，例如，过滤器任何值
test-compile	编译测试源代码到测试目标目录
process-test-classes	处理测试代码文件编译生成的文件
test	运行测试使用合适的单元测试框架（JUnit）
prepare-package	执行必要的任何操作的实际打包之前准备一个包
package	提取编译后的代码，并在其分发格式打包，如JAR，WAR或EAR文件
pre-integration-test	完成执行集成测试之前所需操作。例如，设置所需的环境
integration-test	处理并在必要时部署软件包到集成测试可以运行的环境
pre-integration-test	完成集成测试已全部执行后所需操作。例如，清理环境
verify	运行任何检查，验证包是有效的，符合质量审核规定
install	将包安装到本地存储库，它可以用作当地其他项目的依赖
deploy	复制最终的包到远程仓库与其他开发者和项目共享

inherited 用来指明 execution 是否传递到子 pom.xml 里

filtering属性用来表示资源文件中的 “EL表达式占位符” 是否需要被替换，true为需要替换