SonarQube 工具使用

一、（后端）maven 方式集成 Sonarqube

1、配置好 jdk 和 maven 环境变量

（1）此步骤省略（注意 HOME 变量要配置到根目录下）

（2）检验是否配置完成。

① 检验 jdk

```
java -version
```

```
C:\Users\name>java -version
java version "1.8.0_311"
Java(TM) SE Runtime Environment (build 1.8.0_311-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.311-b11, mixed mode)
```

② 检验 maven

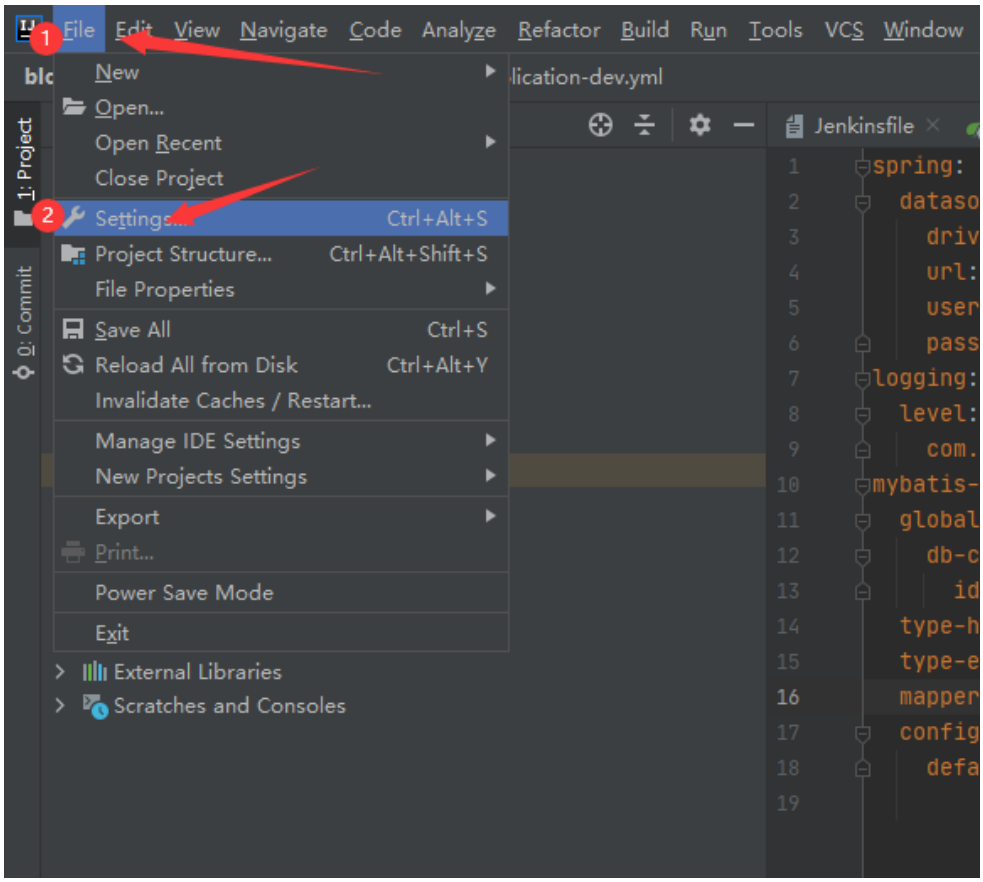
```
mvn -v
```

```
C:\Users\name>mvn -v
Apache Maven 3.6.3 (c[redacted]2883f)
Maven home: D:\Application\IntelliJ IDEA 2020.2.2\plugins\maven\lib\maven3\bin\..
Java version: 1.8.0_131, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_131\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

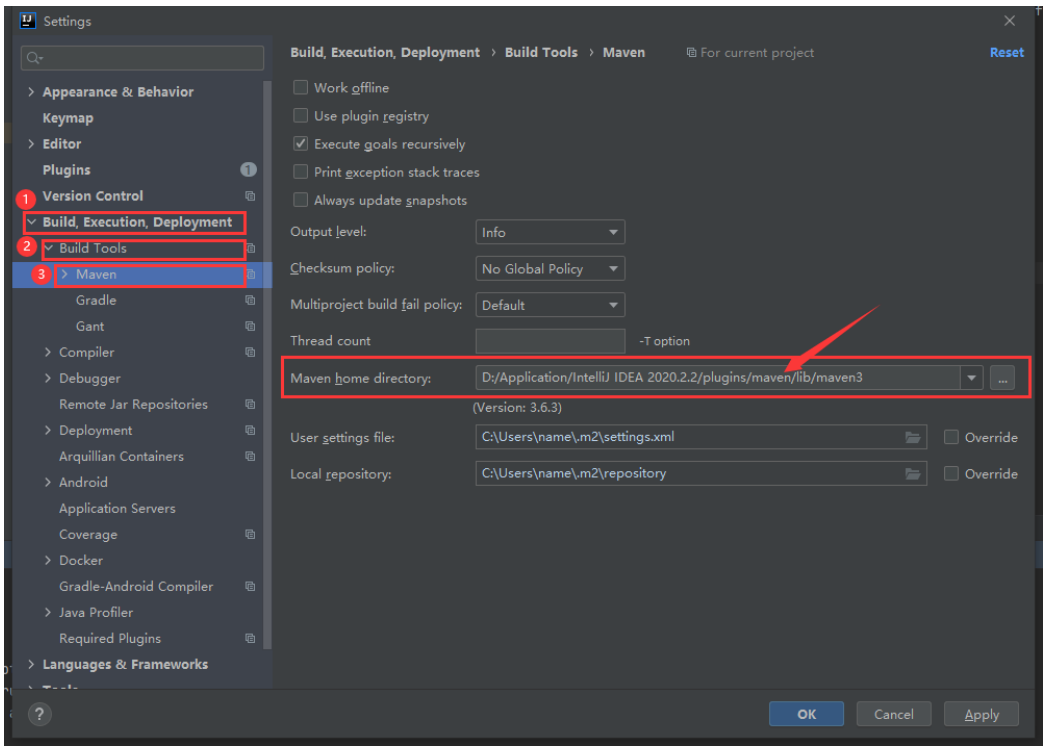
2、更改本地 maven 依赖

（1）找到本地 maven 配置路径（以 IDEA 为例）

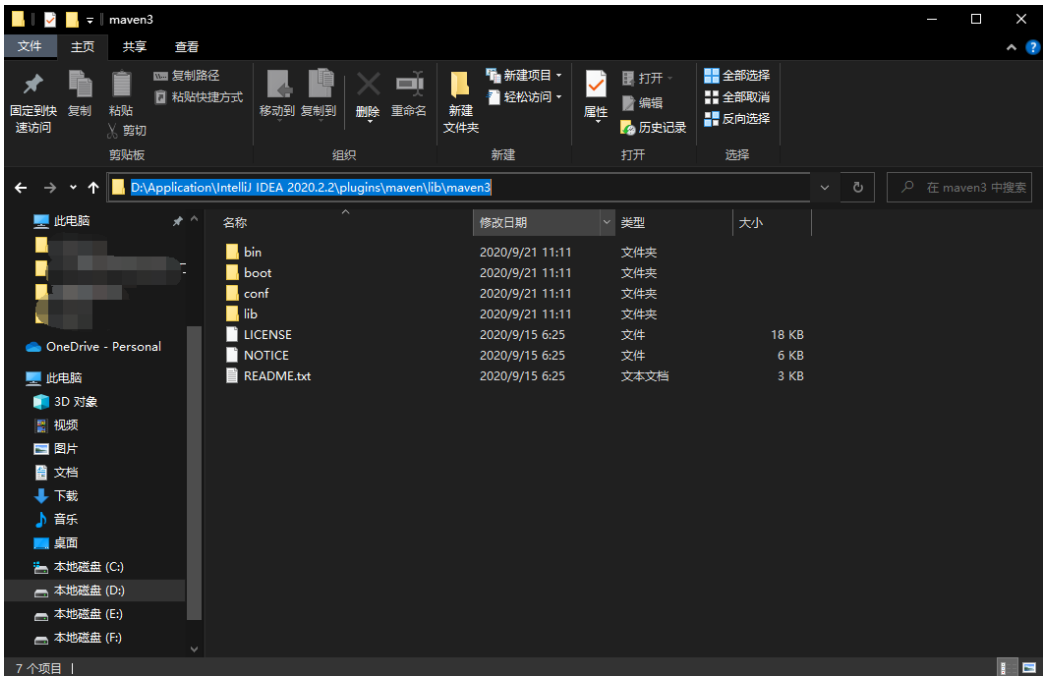
① File --> Settings



② 左侧选择 Maven 项，将右侧“Maven hone directory”路径记下

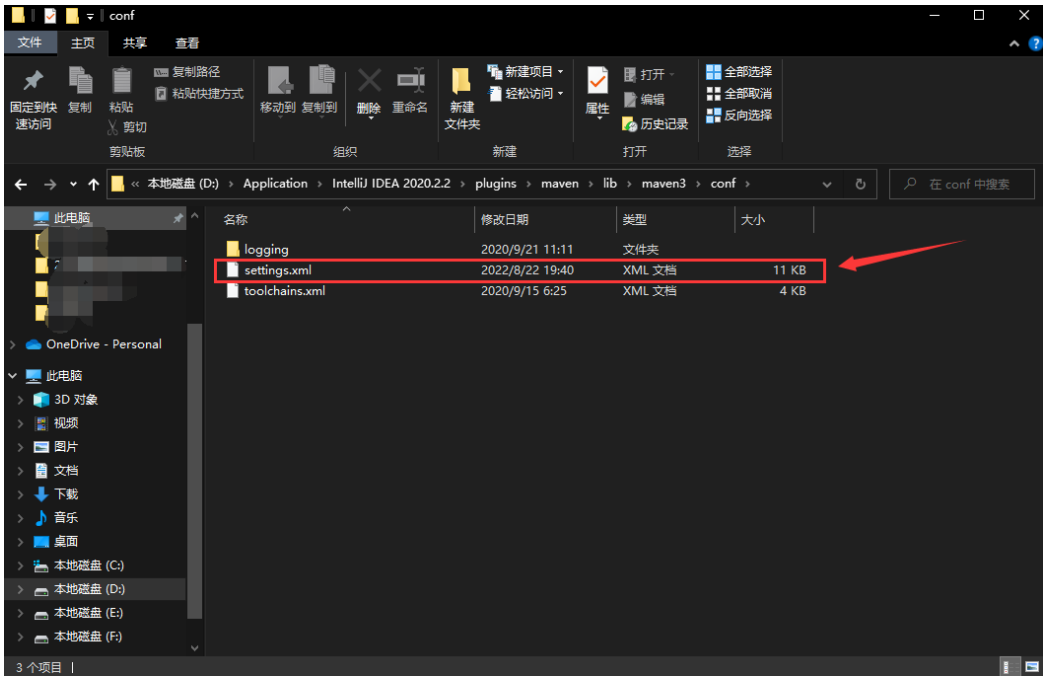


③ 进入该目录下

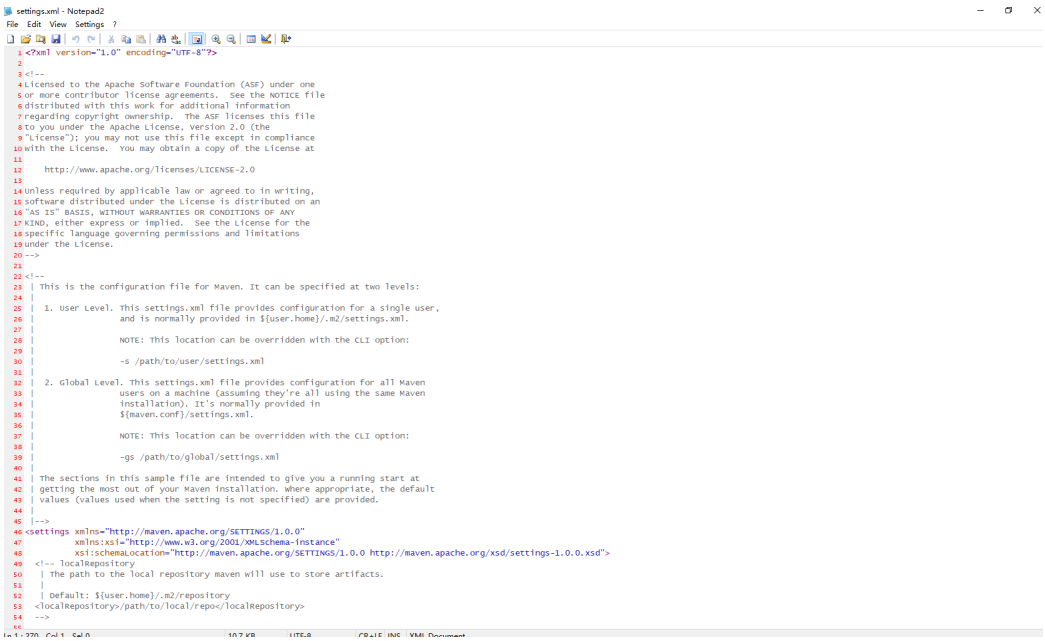


（2）修改 settings.xml 配置文件

④ 进入 conf 目录，找到“settings.xml”



⑤ 打开 “settings.xml”可以看到 xml 格式的依赖项



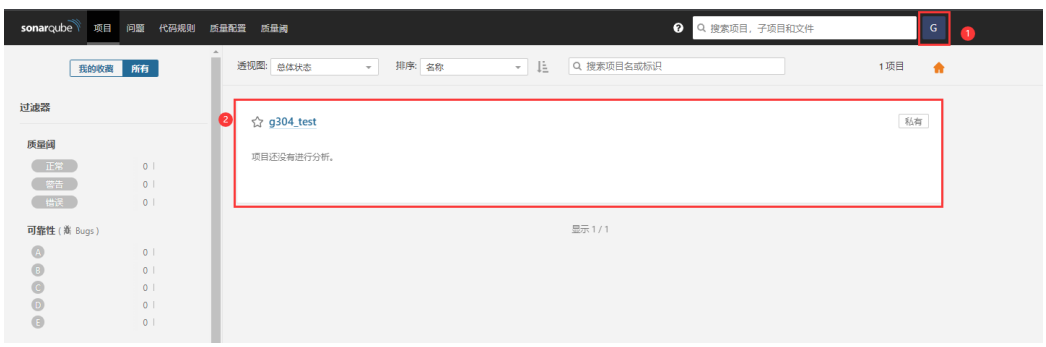
⑥ 在“<profiles></profiles>”标签块内加入以下配置

```
<pluginGroups>
    <pluginGroup>org.sonarsource.scanner.maven</pluginGroup>
</pluginGroups>
<profiles>
    <profile>
      <id>sonar</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        <sonar.host.url>
          http://10.134.136.70:9000
        </sonar.host.url>
      </properties>
    </profile>
</profiles>
```

【注意】xml 标签都是成对出现的，注意不要写错。

3、登录 Sonarqube 平台（http://10.134.136.70:9000）

（1）输入分配好的用户名和密码，登录 Sonarqube。登录成功后，可以看到下面的界面。



（2）点击右上角红色 ①，选择“我的账号”，进入账号管理页面。进入“安全”tab页，可以看到如下的信息。

（3）在“生成令牌”一栏填写任意的标识名称，可以生成你自己的专属 token。

（4）这一步一定要记录这个 token，很重要很重要，再次进入页面将不会再次显示！！

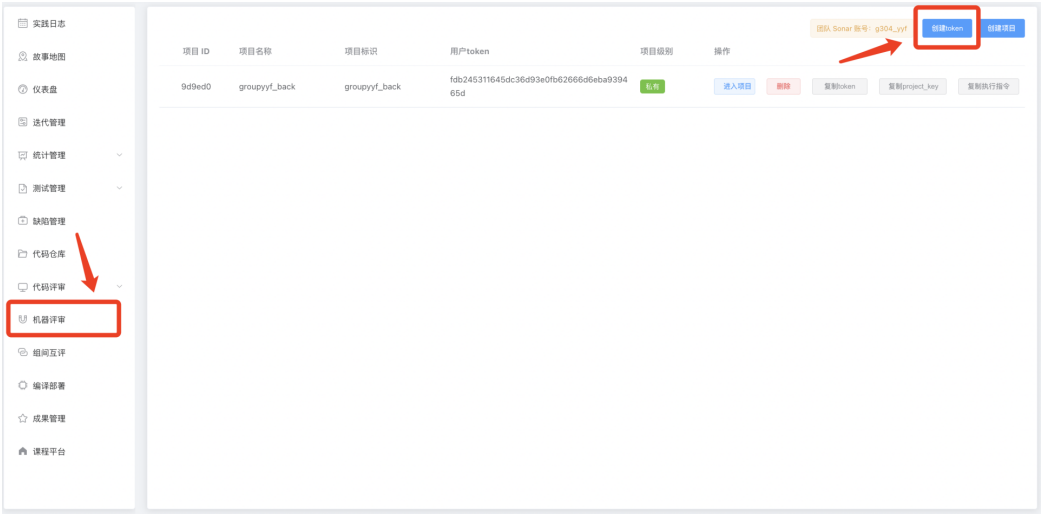
```
token = aa7ca2b8693fb77b24772c4daf937ed88b56d79a
```

（5）点击步骤（1）中的红色 ②，进入已经创建好的项目。

（6）可有看到项目标识。请记住这个标识。

```
key = g304_test
```

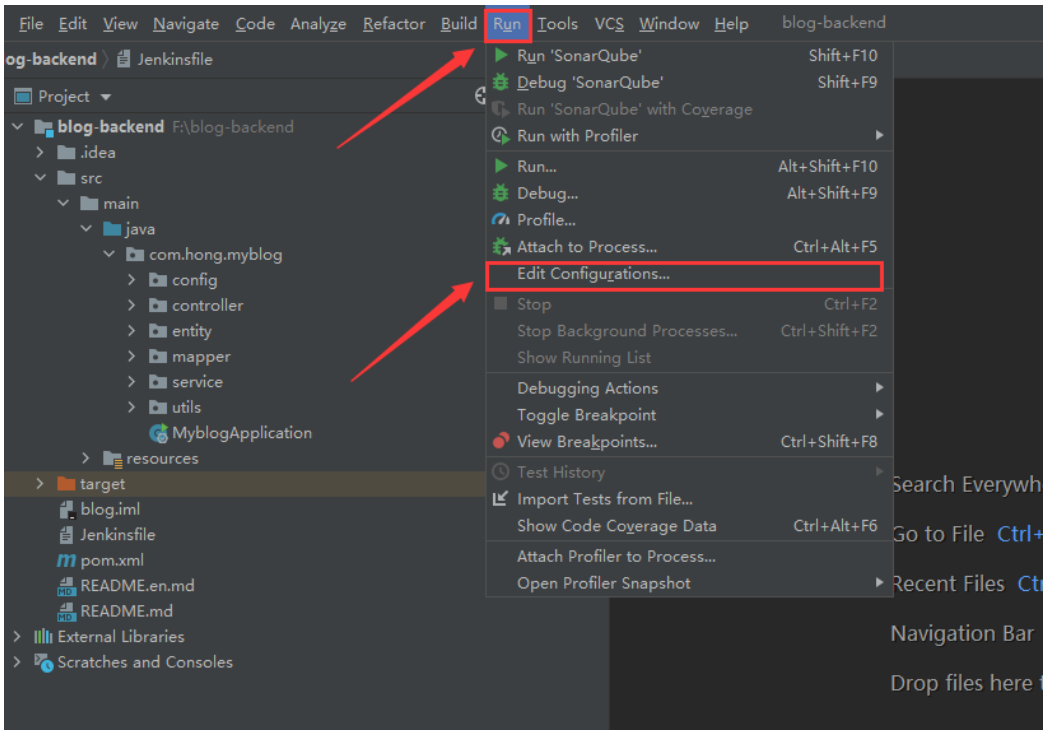
（7）生成 Sonar 用户 Token 还可以通过协同系统完成。



- 点击机器评审，选择“创建 token”
- 输入一个唯一的 token 名称即可生成。

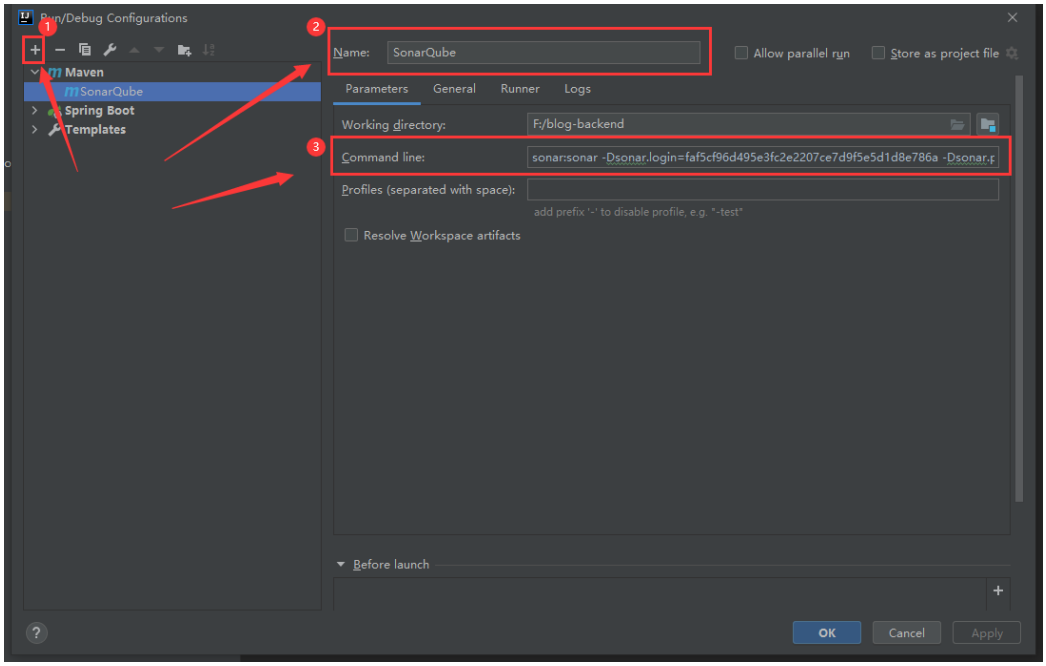
4、创建 maven 指令，并绑定 token 和 key。

（1）打开项目 IDE，点击上方菜单栏“Run”，选择“Edit Configurations...”



（2）点击左上角“+”号，添加 maven 运行配置（①），填写配置名称（②），并输入运行指令（③）。创建完毕点击“OK”。

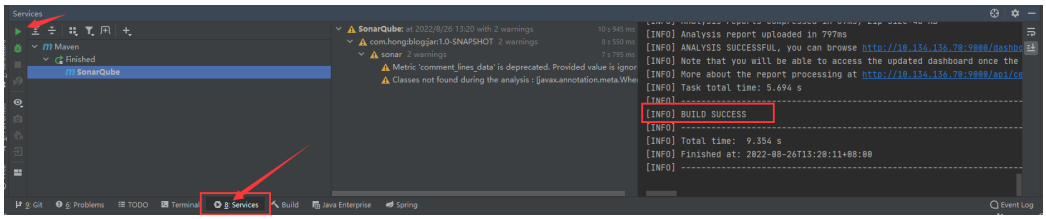
```
sonar:sonar -Dsonar.login=aa7ca2b8693fb77b24772c4daf937ed88b56d79a -Dsonar.projectKey=g304_test
```



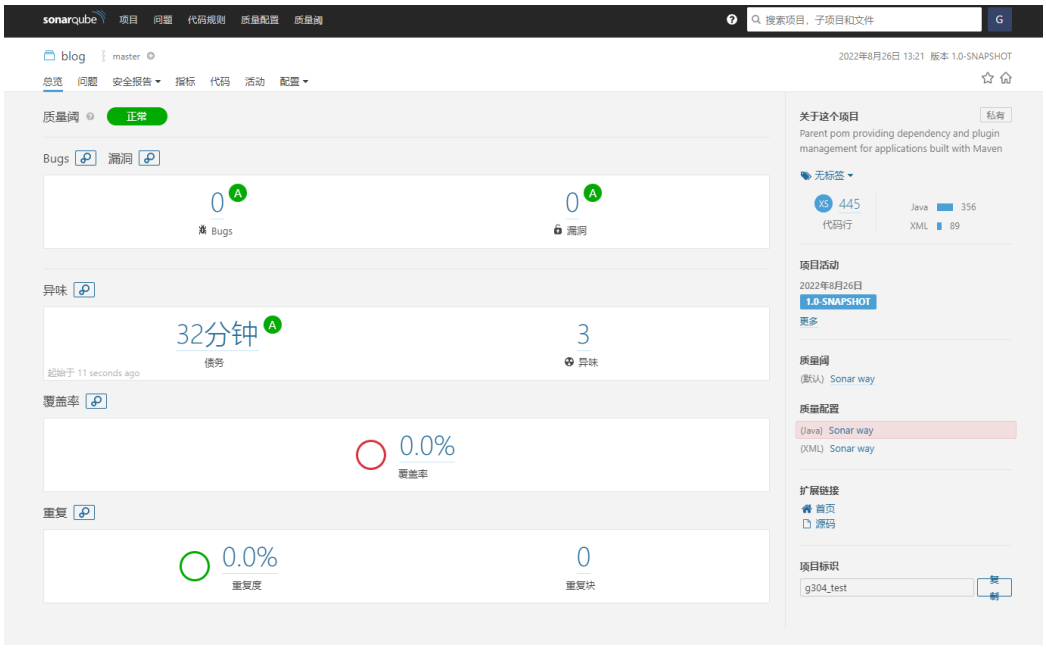
【注意】：

- 此处的“–Dsonar.login”值是刚刚记下的 token；
- 此处的“–Dsonar.projectKey”值是刚刚记下的 key；
- “–Dsonar.login”和“–Dsonar.projectKey”前均有一个空格，注意不要写错。

（3）点击“Run”，就可以看到，分析成功了。

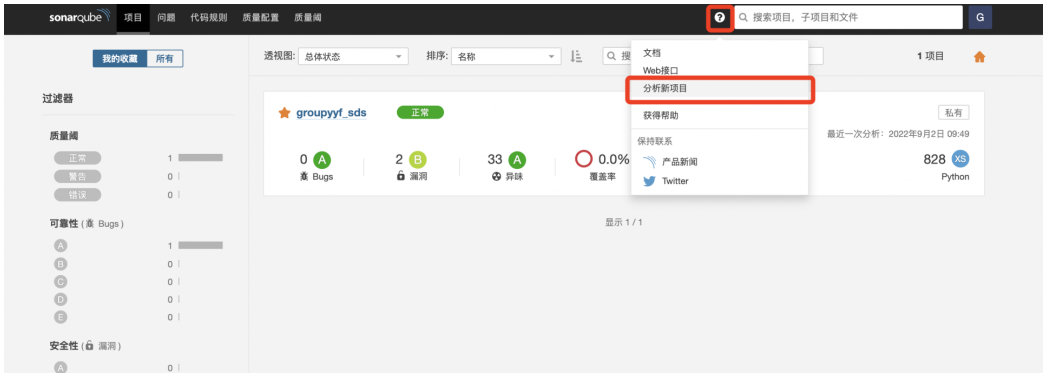


（4）登录 http://10.134.136.70:9000 ，就可以看到项目分析的结果了。

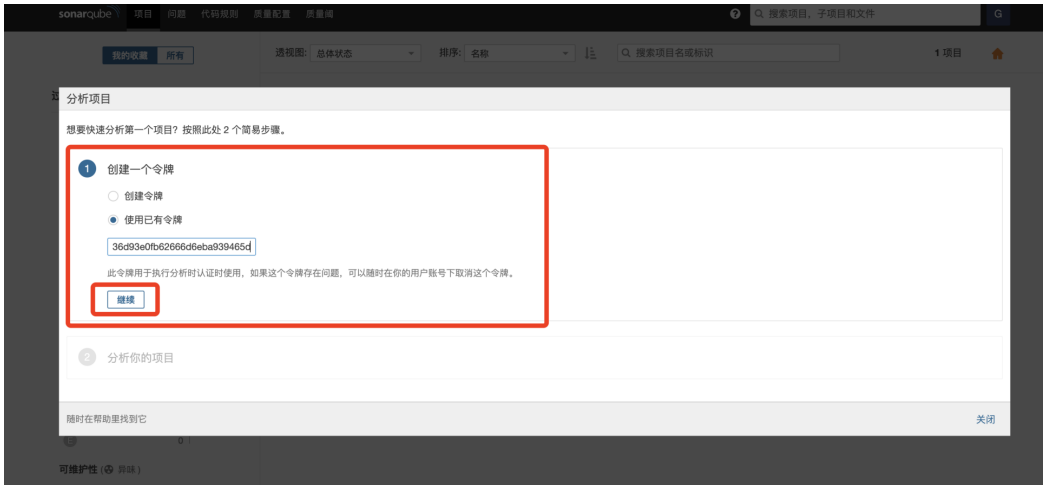


5、其他后端项目

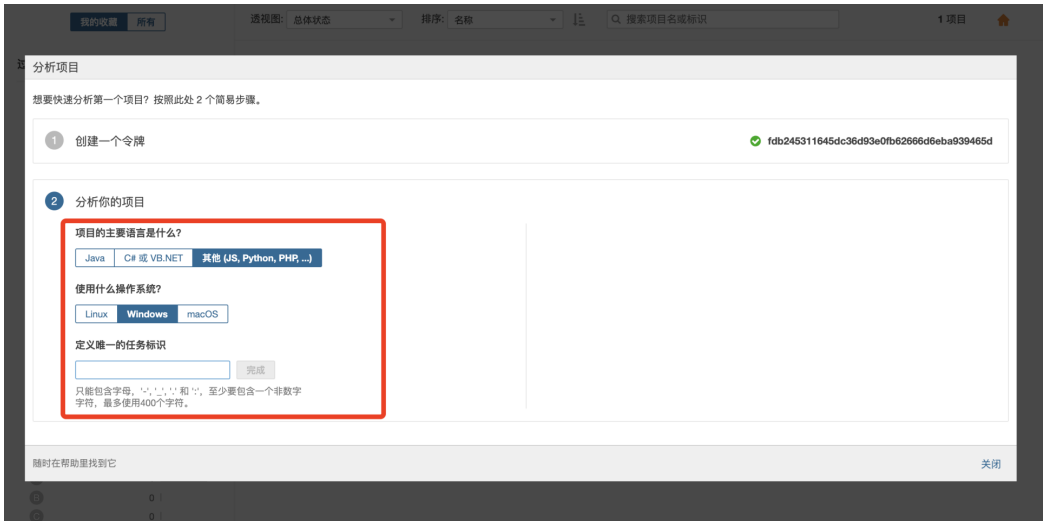
（1）Sonarqube登录成功后，可以看到下面的界面，点击？，选择“分析新项目”



（2）选择创建一个令牌或者使用已有令牌，并点击“继续”



（3）选择和输入项目所使用的语言、任务标识等



（4）按照给出的步骤进行操作即可

下载并解压macOS平台的扫描器

将 `bin` 目录添加至 `PATH` 环境变量

下载

在你的电脑上e执行SonarQube扫描

使用SonarQube分析是非常简单的。只需要在你的项目目录下执行如下命令。

复制

```
sonar-scanner \
  -Dsonar.projectKey=hhhh \
  -Dsonar.sources=. \
  -Dsonar.host.url=http://10.134.136.70:9000 \
  -Dsonar.login=fdb245311645dc36d93e0fb62666d6eba939465d
```

请访问 SonarQube 扫描器官方文档 了解更多信息。

6、常见问题及排查方案

（1）JAVA_HOME 报错

```
The JAVA_HOME environment variable is not defined correctly This environme
nt variable is needed to run this program NB: JAVA_HOME should point to a
 JDK not a JRE
```

【分析】Java 环境变量配置有问题，建议排查 JAVA_HOME 环境变量。

（2）Process terminated 错误

① 配置文件“setting.xml”错误，请检查一下是否有笔误（格式化问题、缺少标签、）。

② 修改完记得重新加载一下项目文件。

（3）Not authorized 错误

```
Failed to execute goal org.sonarsource.scanner.maven:sonar-maven-plugin:3.
9.1.2184:
```

```
sonar (default-cli) on project framework-export-plus:
Not authorized. Please check the properties sonar.login and sonar.passwor
d.
```

遇到权限问题，请及时联系助教。

（4）Maven 打包错误

```
idea:No compiler is provided in this environment. Perhaps you are running
 on a JRE
```

【分析】项目使用的 jdk 和运行环境（jre）要统一。在安装 jdk 的时候可能会安装一个 jre，但是这个 jre 与所安装的 jdk 在 maven 这里也不一定相同。所以要使用与 bin 所在同一目录下的 jre。

二、（前端）Vue 项目集成 Sonarqube

1、安装 Sonar Scanner

```
npm i -g sonar-scanner
```

2、进入 Vue 项目的根目录下，添加配置文件 sonar-project.properties

```
# sonarqube服务地址
sonar.host.url=http://10.134.136.70:9000/
sonar.login=91e0ef163c46536e4163ae5f380934788c407ef9
sonar.projectKey=front_yyz
# 中文的字段需要用Unicode转码，展示到sonarqube的web中才不会乱码
sonar.projectName=\u6d4b\u8bd5\u9879\u76ee

# 公共可用的配置项
sonar.projectVersion=1.0
sonar.sourceEncoding=UTF-8
sonar.sources=./src
sonar.exclusions=/node_modules/
sonar.tests=./src
sonar.test.inclusions=**/*.spec.ts
```

【注意】

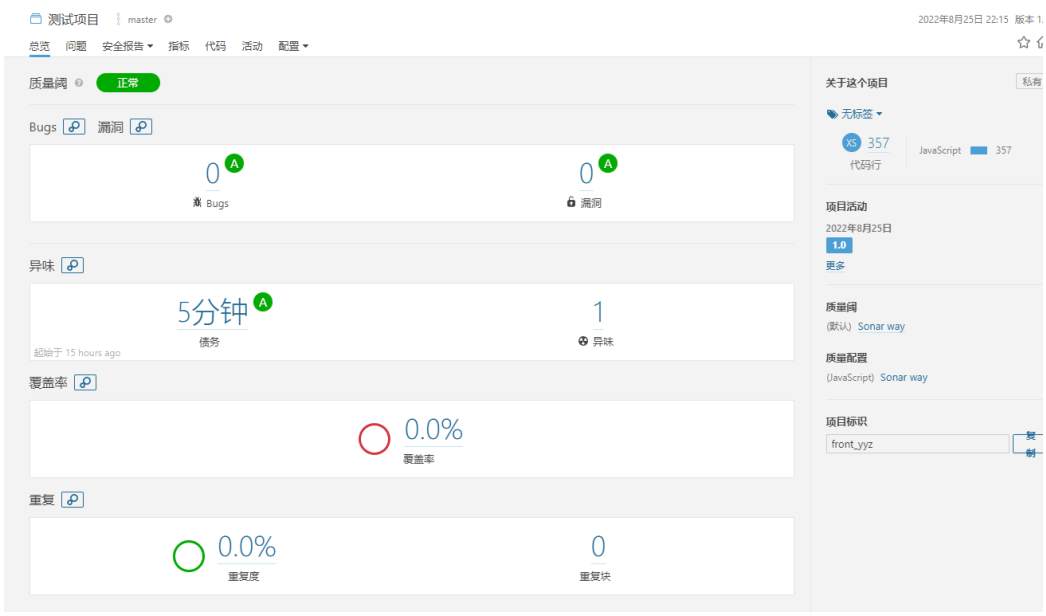
- sonar.login 需要填写之前记下的 token；
- sonar.projectKey 需要填写前端项目的标识 key；
- 其余默认就好。

3、运行扫描指令

```
sonar-scanner
```

可以看到，执行成功。

4、登录 http://10.134.136.70:9000 ，就可以看到项目分析的结果了。



三、Sonarqube 简介

1、何为 Sonarqube ？

SonarQube 是一种自动代码审查工具，用于检测代码中的错误、漏洞和代码异味。它可以与您现有的工作流程集成，以实现跨项目分支和拉取请求的持续代码检查。

SonarQube 是一个用于代码质量管理的开源平台，用于管理源代码的质量。 通过插件形式，可以支持包括 java, C#, C/C++, PL/SQL, Cobol, JavaScrip, Groovy 等二十几种编程语言的代码质量管理与检测。

2、Sonarqube 的主要作用。

sonar通过配置的代码分析规则，从可靠性、安全性、可维护性、覆盖率、重复率等方面分析项目，风险等级从A~E划分为5个等级；同时，sonar可以集成pmd、findbugs、checkstyle等插件来扩展使用其他规则来检验代码质量；sonar设置了质量门，通过设置的质量门评定此次提交分析的项目代码是否达到了规定的要求。

3、Sonarqube 的各个指标

（1）可靠性 —— Bugs

评估	级别	描述
A	无 Bug	表示代码无bug，最高级别。
B	次要	界面、性能缺陷，建议类问题，不影响操作功能的执行，可以优化性能的方案等。如：错别字、界面格式不规范，页面显示重叠、不该显示的要隐藏，描述不清楚，提示语丢失，文字排列不整齐，光标位置不正确，用户体验感受不好，可以优化性能的方案等。
C	重要	功能没有完全实现但是不影响使用，功能菜单存在缺陷但不会影响系统稳定性。如：操作时间长、查询时间长、格式错误、边界条件错误，删除没有确认框、数据库表中字段过多等。
D	严重	系统主要功能部分丧失、数据库保存调用错误、用户数据丢失，一级功能菜单不能使用但是不影响其他功能的测试。功能设计与需求严重不符，模块无法启动或调用，程序重启、自动退出，关联程序间调用冲突，安全问题、稳定性等。如：软件中数据保存后数据库中显示错误，用户所要求的功能缺失，程序接口错误，数值计算统计错误等。
E	阻断	阻碍开发或测试工作的问题；造成系统崩溃、死机、死循环，导致数据库数据丢失，与数据库连接错误，主要功能丧失，基本模块缺失等问题。如：代码错误、死循环、数据库发生死锁、重要的一级菜单功能不能使用等。

（2）安全性 —— 漏洞

评估	级别	描述
A	无 Bug	表示代码无漏洞，最高级别。
B	次要	能够获取一些数据，但不属于核心数据的操作； 在条件严苛的环境下能够获取核心数据或者控制核心业务的操作； 需要用户交互才可以触发的漏洞。包括但不限于XSS漏洞、CSRF漏洞、点击劫持。
C	重要	需要在一定条件限制下，能获取服务器权限、网站权限与核心数据库数据的操作。包括但不限于交互性代码执行、一定条件下的注入、特定系统版本下的getshell等； 任意文件操作漏洞。包括但不限于任意文件写、删除、下载，敏感文件读取等操作； 水平权限绕过。包括但不限于绕过限制修改用户资料、执行用户操作。
D	严重	直接获取普通系统权限的漏洞。包括但不限于远程命令执行、代码执行、上传webshell、缓冲区溢出等； 严重的逻辑设计缺陷和流程缺陷。包括但不限于任意账号密码修改、重要业务配置修改、泄露； 可直接批量盗取用户身份权限的漏洞。包括但不限于普通系统的SQL注入、用户订单遍历； 严重的权限绕过类漏洞。包括但不限于绕过认证直接访问管理后台、cookie欺骗。 运维相关的未授权访问漏洞。包括但不限于后台管理员弱口令、服务未授权访问。
E	阻断	阻断 阻碍开发或测试工作的问题；造成系统崩溃、死机、死循环，导致数据库数据丢失，与数据库连接错误，主要功能丧失，基本模块缺失等问题。如：代码错误、死循环、数据库发生死锁、重要的一级菜单功能不能使用等。

（3）可维护性 —— 债务/异味

【债务】

“技术债务”这个概念，最早是在 1992 年由 Ward Cunningham 在他的论文“The WyCash Portfolio Management System”中提出的，之后被软件工程界投受井挂广。《重构》的作者 Martin Fowler 也在其网站上对技术债务有所介绍。

“技术债务”的原理可以理解为“出来混早晚要还的”，即，当前不规范的代码，会对以后产品修改的成本造成影响。

可维护性等级范围从 A（非常好）到 E（非常差）。 评级由技术债务比率的值决定，技术债务比率是将项目的技术债务与从零开始重写代码所需的成本进行比较。 A 到 D 的默认值为：0.05、0.1、0.2、0.5，任何超过 0.5 评级就为 E。

【例：假设开发成本是30分钟，2,500 LOC（开发一行代码的成本）的技术债务为24,000分钟的项目将有技术债务比率为24000 /（30 * 2,500）= 0.32。 因此项目的可维护性评级就是D。】

【异味】

“异味”是指，如果一段代码是不稳定或者有一些潜在问题的，那么代码往往会包含一些明显的痕迹。正如食物要腐坏之前，经常会发出一些异味一样。

以下条件均有可能导致代码“异味”。

- Duplicated Code（重复的代码）
- Long Method（过长方法）
- Large Class（过大的类）
- Long Parameter List（过长参数列）
- Divergent Change（发散式变化）
  - 牵一发而动全身。耦合度太高，一处变化导致其他地方也跟着变化。
- ……

篇幅有限不列举过多，感兴趣的可以参考这篇博客（http://www.noobyard.com/article/p-bfpcpebu-hc.html）

(4) 覆盖率

指单元测试覆盖率，包括分支覆盖率和代码覆盖率。

(5) 重复

重复度 = 重复行数 / 总行数 * 100%
重复块：重复代码块的行数