# jacoco+maven+sonar+springboot单元测试代码覆盖率统计

如题，你能看到本文，说明你应该对本文涉及的框架和组件已经初步了解。
总的来说，本文所述集成方案是对代码的单元测试用例执行覆盖率进行检测统计，主要是用来对开发人员单元测试用例编写程度的一种检测。
其整体过程大概可以分为：

> 1、使用 jacoco 代理，在 maven 执行之前植入 jacoco 代理。
> 2、maven 对项目进行编译打包，并自动执行单元测试用例（如 junit）。
> 3、测试用例执行过程中，jacoco 代理会记录代码执行的所有行信息，生成记录文件 （target/jacoco.exec）。
> 4、使用 jacoco 命令将记录的代码执行情况文件转换生成为 jacoco 报表 （target/site/jacoco/index.html）。
> 5、使用 sonar 命令，将代码覆盖率报表推送到 sonarqube 平台中。
> 6、登录 sonarqube 查看代码覆盖率统计报告（也可以本地访问 jacoco/index.html 直接查看报告）。

本文所涉组件和框架版本信息如下：

> 1、springboot（2.1.8.RELEASE）
> 2、sonarqube（docker 版 sonarqube:7.9.1-community）
> 3、maven（3.6.1）
> 4、jacoco（springboot 框架内已自带，无需增加配置）

随着各种版本不断升级，未来版本使用方法可能有异，如上版本实践正常。
因为撰写本文时，springboot 已为主流技术栈，所以本文不陈述 springmvc 等其他框架的操作方法，理论上来说 springmvc 可能需要在 pom.xml 文件中新增 jacoco 插件配置，其他方面应该和本文没有区别。

---

下面陈述一下整体操作步骤
1、 一个标准的 springboot 工程，代码和 pom 文件都是标准的，没有任何特殊配置和依赖引用。
2、 在 springboot 工程中，随便写几个 Service Controller 测试方法。

> 随便写几个 Service Controller 测试方法，没有任何特殊要求，随便写，此处省去代码。

3、 在 springboot 工程中，编写测试用例，本文使用 springboot 自带的 junit ，如下代码示例：

```
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import java.io.UnsupportedEncodingException;

import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultHandlers;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.transaction.annotation.Transactional;
```

```
import org.springframework.web.context.WebApplicationContext;

@RunWith(SpringRunner.class)
@SpringBootTest(classes = StartApplication.class)
//测试环境使用，用来表示测试环境使用的ApplicationContext将是WebApplicationContext类型的
@WebAppConfiguration
public class ExampleTests {

 private static final Logger LOG = LoggerFactory.getLogger(ExampleTests.class);

 @Autowired
 private WebApplicationContext context;

 private MockMvc mockMvc;

 @Autowired
 private ExampleValidatorService exampleValidatorService;

 @Before
 public void setUp() {
  mockMvc = MockMvcBuilders.webAppContextSetup(context).build();// 建议使用这种
 }

 /**
  * 测试 Controller 接口
  * @throws UnsupportedEncodingException
  * @throws Exception
  */
 @Test
 public void testController() throws UnsupportedEncodingException, Exception {
  String responseString = mockMvc.perform(MockMvcRequestBuilders.get("/example/test1") // 请求的url,请求的方法是get
    .contentType(MediaType.APPLICATION_JSON_UTF8) // 数据的格式
    .accept(MediaType.APPLICATION_JSON_UTF8)).andExpect(MockMvcResultMatchers.status().isOk()) // 返回的状态是200
    .andDo(MockMvcResultHandlers.print()) // 打印出请求和相应的内容
    .andExpect(jsonPath("$.code").value("1"))
    .andExpect(jsonPath("$.data").value("单红宇 CSDN catoop"))
    .andReturn().getResponse().getContentAsString(); // 将相应的数据转换为字符

  LOG.info(responseString);
 }

 @Test
 @Transactional // 会自动回滚，不污染数据库
 public void testService() {
  String age = "22";
  String result = exampleValidatorService.show("22");
  Assert.assertEquals(age, result);
 }
}
```

4、 使用命令进行 maven 编译构建（注意不是使用 IDE 自带功能），确保执行结果为 BUILD SUCCESS

```
# 构建命令如下（执行该命令会自动启动服务执行测试用例）
mvn clean package
```

5、 配置本地 maven 配置文件 conf/settings.xml，新增如下片段内容

```
 <pluginGroups>
<!-- SonarQube 插件 -->
<pluginGroup>org.sonarsource.scanner.maven</pluginGroup>
 </pluginGroups>

 <profiles>
<!-- SonarQube 插件 -->
<profile>
 <id>sonar</id>
 <activation>
  <activeByDefault>true</activeByDefault>
 </activation>
 <properties>
  <!-- Optional URL to server. Default value is http://localhost:9000 -->
  <sonar.host.url>
    http://192.168.10.88:9000
  </sonar.host.url>
 </properties>
</profile>
 </profiles>
```

6、 执行如下编译构建生成报告推送 sonar 命令

```
# 该命令所经历阶段顺序：清理、设置jacoco代理、打包、执行测试用例、生成jacoco报告、推送报告到sonarqube
mvn clean org.jacoco:jacoco-maven-plugin:prepare-agent package org.jacoco:jacoco-maven-plugin:report sonar:sonar
```

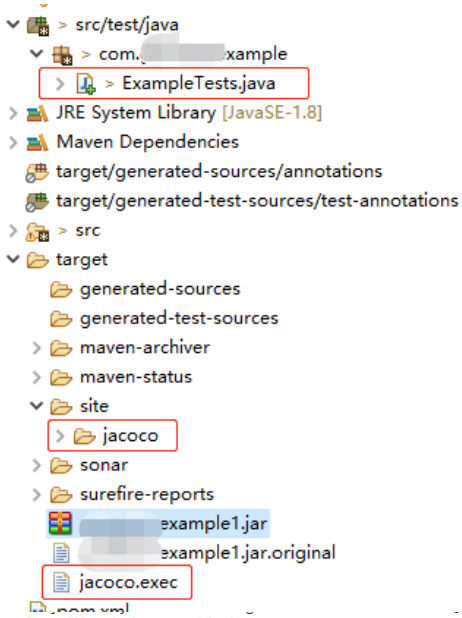
待该命令执行完毕并且 BUILD SUCCESS 后，可以在项目的 target 目录下正常看到文件 jacoco.exec 和目录 site\jacoco

7、 你可以用浏览器打开 jacoco\index.html 文件直接查看报告，也可以打开 sonarqube 去查看报告（前提条件是你 sonarqube 中 jacoco 的相关插件都已经安装）。

PS：如果你不需要将结果推送到 sonar 查看，去掉 mvn 命令最后的 sonar:sonar 即可，本地报告查看也是很清晰方便的，我之所以这么做，是因为 sonar 上面还有其他很多检测的东西，给开发者查看比较方便，毕竟本地报告只是在自己电脑上。

---

至此完成，如下是一些相关截图：




```
[INFO] 17 files had no CPD blocks
[INFO] Calculating CPD for 38 files
[INFO] CPD calculation finished
[INFO] Analysis report generated in 82ms, dir size=290 KB
[INFO] Analysis report compressed in 168ms, zip size=169 KB
[INFO] Analysis report uploaded in 76ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://sonar.
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://sonar.
[INFO] Analysis total time: 12.289 s
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

```
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  54.239 s
[INFO] Finished at: 2020-02-19T16:56:17+08:00
[INFO] ------------------------------------------------------------------------

Administrator@SHANHY MINGW64 /e/code/workspace-[redacted]xample1 (develop)
```

文件 | E:/code/workspa[redacted]/target/site/jacoco/index.html

shanhy-example1 Sessions

# shanhy-example1

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| com.shanhy.example.common.utils | | 0% | | 0% | 139 | 139 | 456 | 456 | 84 | 84 | 19 | 19 |
| com.shanhy.example.module.user.entity | | 0% | | 0% | 80 | 80 | 13 | 13 | 29 | 29 | 1 | 1 |
| com.shanhy.example.module.sys.entity | | 0% | | 0% | 70 | 70 | 11 | 11 | 28 | 28 | 2 | 2 |
| com.shanhy.example.module.user.service | | 2% | | 0% | 35 | 38 | 87 | 91 | 23 | 26 | 0 | 3 |
| com.shanhy.example.common.spring | | 46% | | 28% | 24 | 41 | 51 | 103 | 8 | 25 | 0 | 3 |
| com.shanhy.example.module.sys.service | | 5% | | n/a | 9 | 12 | 38 | 41 | 9 | 12 | 0 | 2 |
| com.shanhy.example.module.example.model | | 0% | | 0% | 20 | 20 | 3 | 3 | 9 | 9 | 1 | 1 |
| com.shanhy.example.module.example.controller | | 17% | | 0% | 20 | 27 | 38 | 46 | 18 | 25 | 0 | 5 |
| com.shanhy.example.common.redis | | 72% | | 0% | 11 | 23 | 26 | 99 | 6 | 18 | 1 | 4 |
| com.shanhy.example.module.user.controller | | 8% | | 0% | 12 | 14 | 27 | 30 | 5 | 7 | 0 | 1 |
| com.shanhy.example.common.config | | 79% | | 0% | 4 | 17 | 9 | 52 | 1 | 14 | 0 | 7 |
| com.shanhy.example.common.exception | | 0% | | n/a | 10 | 10 | 20 | 20 | 10 | 10 | 2 | 2 |
| com.shanhy.example.common.filter | | 0% | | 0% | 5 | 5 | 11 | 11 | 2 | 2 | 1 | 1 |
| com.shanhy.example.module.example.service | | 34% | | n/a | 2 | 7 | 7 | 13 | 2 | 7 | 0 | 2 |
| com.shanhy.example.module.job | | 18% | | 0% | 2 | 4 | 6 | 8 | 1 | 3 | 0 | 1 |
| com.shanhy.example.common.persistent | | 0% | | n/a | 3 | 3 | 5 | 5 | 3 | 3 | 1 | 1 |
| com.shanhy.example | | 70% | | n/a | 1 | 5 | 3 | 10 | 1 | 5 | 0 | 1 |
| com.shanhy.example.module.sys.controller | | 27% | | n/a | 2 | 4 | 2 | 4 | 2 | 4 | 0 | 2 |
| com.shanhy.example.common.interceptor | | 100% | | n/a | 0 | 2 | 0 | 2 | 0 | 2 | 0 | 1 |
| Total | 4,507 of 5,350 | 15% | 407 of 416 | 2% | 449 | 521 | 813 | 1,018 | 241 | 313 | 28 | 59 |


(END)