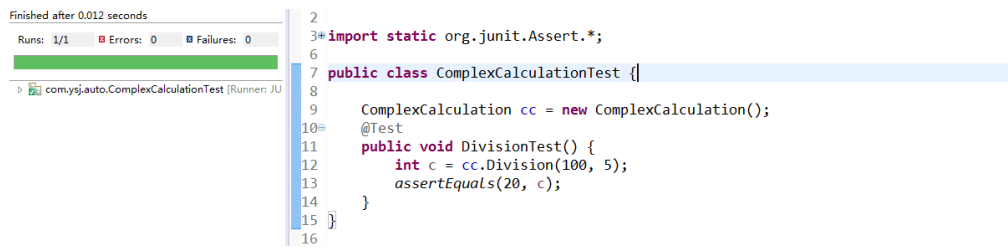


单元测试报告自动生成及代码覆盖率报告

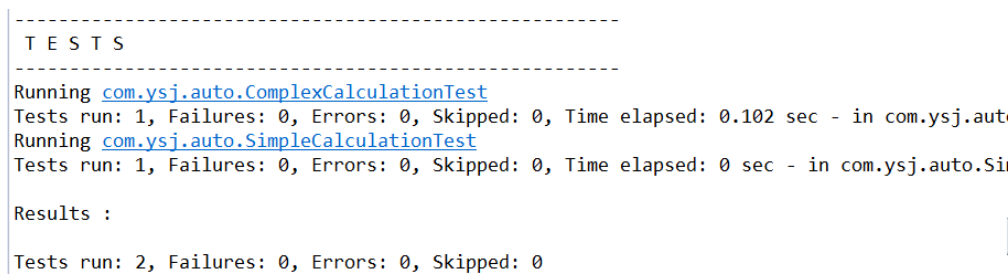
1.单元测试报告自动生成

1.1 单元测试报告是什么？

我们在完成一个功能点时，需要写一些单元测试来检测程序功能是否实现及其健壮性。我们在Eclipse中运行JUnit之后，会看到该测试用例的一些详细信息。



当测试代码很多时，我们可以使用一些方式集中地运行这些测试用例，下图即直观的展示了单元测试报告。

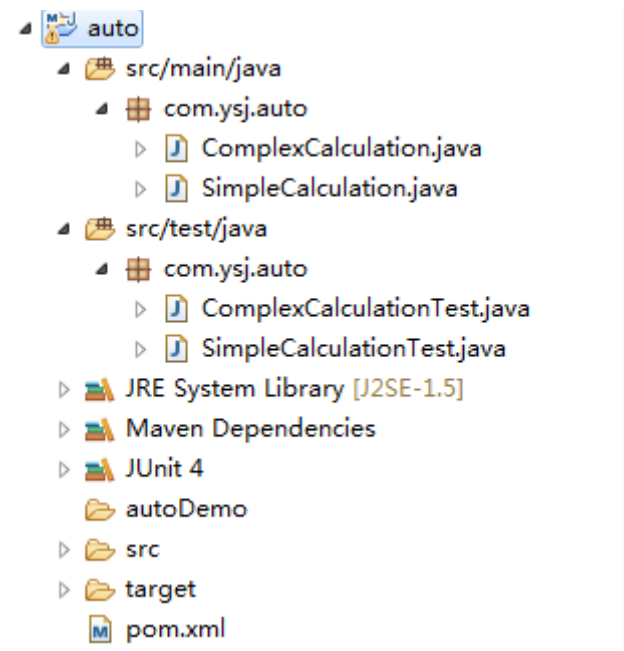


1.2 单元测试报告的意义

相当于测试人员的黑盒测试结果展示。在该阶段，测试人员关心的是程序功能是否完成。通过这些直观的展示，测试人员可以了解到程序的功能情况。

1.3 环境搭建（以 Maven + JUnit 为例）

新建一个Maven项目auto



```

1 package com.ysj.auto;
2
3 public class ComplexCalculation {
4     public int Division(int a, int b){
5         return (a/b);
6     }
7     public int Multiply(int a, int b){
8         return (a*b);
9     }
10 }
1 package com.ysj.auto;
2
3 public class SimpleCalculation {
4     public int Add(int a,int b){
5         return (a+b);
6     }
7     public int Subtration(int a,int b){
8         return(a-b);
9     }
10 }
1 package com.ysj.auto;
2
3 import static org.junit.Assert.*;
4
5
6
7 public class ComplexCalculationTest {
8
9     ComplexCalculation cc = new ComplexCalculation();
10     @Test
11     public void DivisionTest() {
12         int c = cc.Division(100, 5);
13         assertEquals(20, c);
14     }
15 }

```

```

1 package com.ysj.auto;
2
3 import static org.junit.Assert.*;
4
5
6
7 public class SimpleCalculationTest {
8
9     SimpleCalculation sc = new SimpleCalculation();
10    @Test
11    public void AddTest() {
12        int c = sc.Add(3, 5);
13        assertEquals(8, c);
14    }
15 }

```

1.4 单元测试报告自动生成

在pom.xml文件中使用jacoco，surefire插件即可自动生成单元测试报告。

```

<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.7.6.201602180812</version>
  <executions>
    <execution>
      <id>pre-unit-test</id>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
      <configuration>
        <destFile>${project.build.directory}/coverage-reports/${project.artifactId}-jacoco-ut.exec</destFile>
        <propertyName>surefireArgLine</propertyName>
      </configuration>
    </execution>
    <execution>
      <id>post-unit-test</id>
      <phase>test</phase>
      <goals>
        <goal>report</goal>
      </goals>
      <configuration>
        <dataFile>${project.build.directory}/coverage-reports/${project.artifactId}-jacoco-ut.exec</dataFile>
        <outputDirectory>${project.reporting.outputDirectory}/jacoco-ut</outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.15</version>
  <configuration>
    <testFailureIgnore>true</testFailureIgnore>
    <argLine>${surefireArgLine}</argLine>
    <skipTests>false</skipTests>
    <includes>
      <exclude>/**/*.Test.java</exclude>
    </includes>
  </configuration>
</plugin>

```

2. 代码覆盖率报告

2.1 代码覆盖率是什么？

代码覆盖率是一个抽象的概念，它是一种度量代码覆盖程度的方式。Jacoco是一个开源的覆盖率工具，它可以嵌入到Ant、Maven中，很多第三方的工具提供了对Jacoco的集成，如

sonar，Jenkins等。

2.2 代码覆盖率的意义

如果说单元测试用例对测试人员来说是黑盒测试，那代码覆盖率就像是白盒测试。使用它运行单元测试后，可以给出代码中哪些部分被单元测试测到，哪些部分没有测到，并且给出

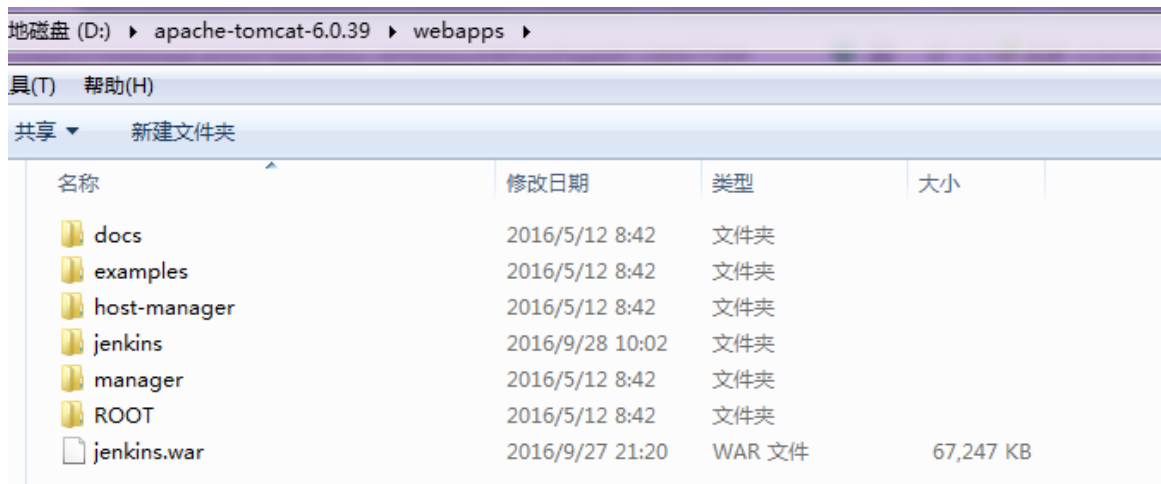
整个项目的单元测试覆盖情况百分比，看上去一目了然。jacoco代码覆盖率的几个重要指标包括：指令覆盖率，分支覆盖率，圈复杂度覆盖率，行覆盖率，方法覆盖率，类覆盖率等。

2.3 环境搭建(jenkins + maven + svn)

2.3.1 下载

jenkins.war文件，复制到tomcat的webapps目录。启动tomcat，输入localhost:8080/jenkins即可访问。第一次的登录密码会在tomcat启动时提示用户，同时也能在jenkins目录下

的initialAdminPassword中找到，然后可以自行设置jenkins的用户名和密码。



2.3.2 下载SVN服务端 (Visual Server) 和客户端 (TortoiseSVN)

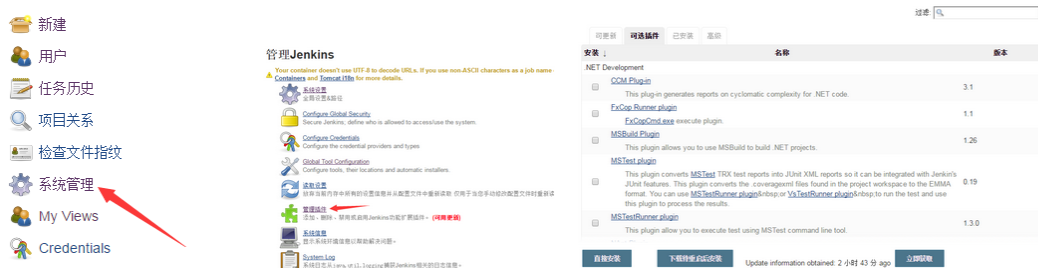
2.3.2.1 配置svn服务器的用户名和密码 (后面会使用到)

2.3.3 为jenkins安装一些插件

jenkins内置了一些插件，然而在这里我们还需要其他的一些插件 (SonarQube Plugins , Jacoco Plugins , Subversion Plug_in)

安装方式基本包括两种：

第一种方式：系统管理 -> 管理插件



第二种方式：下载插件安装文件，然后在系统管理 -> 管理插件 -> 高级 -> 上传插件，安装完成后重启tomcat即可在“已安装”列表中看到



上传插件

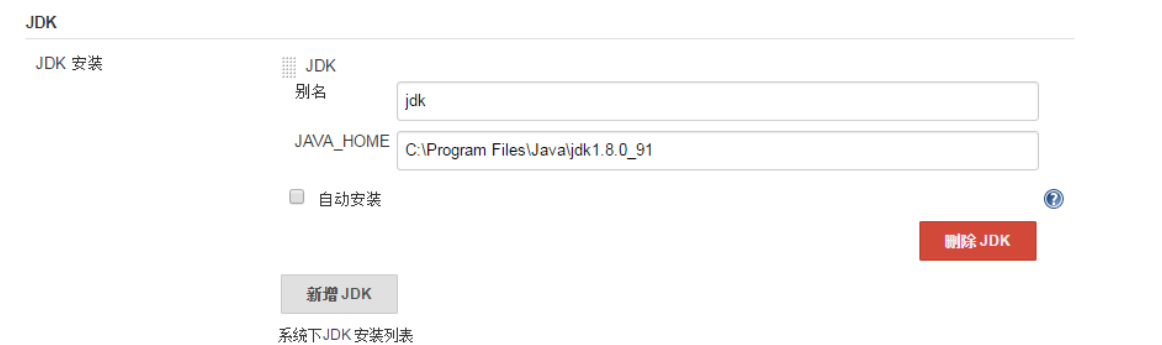
您可以通过上传一个 .hpi 文件来安装插件。

文件: 未选择任何文件

2.3.4 进行全局的设置

系统管理 -> Global Tool COnfiguration, 然后像配置环境变量一样配置 JDK, ANT, MAVEN, SonarQube Runner (在此之前, 要在系统中下载和配置SonarQube和SonarQube

Runner)




2.4 如何在jenkins构建项目中自动生成代码覆盖率报告


2.4.1 新建一个Job


Enter an item name


Auto 1

» Required field

 **构建一个自由风格的软件项目** 2
这是Jenkins的主要功能,Jenkins将会结合任何SCM和任何构建系统来构建你的项目,甚至可以构建软件以外的系统。

 **构建一个maven项目**
构建一个maven项目,Jenkins利用你的POM文件,这样可以大大减轻构建配置。

 **External Job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

 **构建一个多配置项目**
适用于多配置项目,例如多环境测试,平台指定构建,等等。

if OK 3 to create a new item from other existing, you can use this option:

2.4.2 然后对该Job进行配置 (主要配置SVN和构建步骤)

General **源码管理** 构建触发器 构建环境 构建 构建后操作

☐ None
☒ Subversion

Modules

Repository URL 项目svn地址

Credentials svn用户名和密码

Local module directory jenkin工作空间的文件夹名(可以不写,采取默认的 ./)

Repository depth

Ignore externals ☒

Add module...

Additional Credentials

Check-out Strategy

☒ Invoke top-level Maven targets

Maven Version

Goals MAVEN 命令

高级...

Invoke Standalone SonarQube Analysis

Task to run:

JDK:

JDK to be used for this sonar analysis

Path to project properties:

Analysis properties:

```

sonar.projectKey=$JOB_NAME
sonar.projectName=$JOB_NAME
sonar.projectVersion=$BUILD_NUMBER
sonar.projectBaseDir=.
sonar.sources=src/main/java
sonar.binaries=target/classes
sonar.tests=src/test/java
sonar.language=java
sonar.java.coveragePlugin=jacoco
sonar.jacoco.reportPath=target/coverage-merged/auto-jacoco.exec
sonar.junit.reportsPath=target/junit
sonar.sourceEncoding=UTF-8

```

Additional arguments:

构建后操作

Record JaCoCo coverage report

Path to exec files (e.g.: `**/target/**/*.exec, **/jacoco.exec`):

Path to class directories (e.g.: `**/target/classDir, **/classes`):

Path to source directories (e.g.: `**/mySourceFiles`):

Inclusions (e.g.: `**/*.class`):

Exclusions (e.g.: `**/*Test*.class`):

Instruction: % Branch: % Complexity: % Line: % Method: % Class:

Instruction: % Branch: % Complexity: % Line: % Method: % Class:

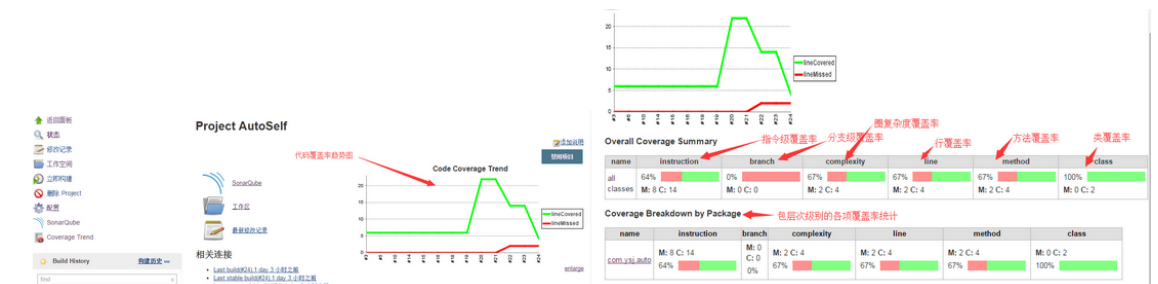
Change build status according to the thresholds

2.4.3 点击，应用和保存后，即可构建项目。Jenkins从SVN服务器上代码拉到工作空间，然后执行构建步骤，即执行clean package命令。生成exec文件。然后由

SonarQube

Analysis根据提供的参数（主要提供源码，.class文件，.exec文件）进行分析。最后又jacoco插件进行代码覆盖率的统计。构建成功时，可看到如

下的效果。



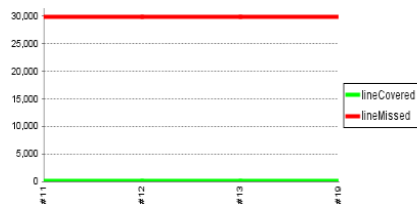
2.5 APM 项目的应用情况

2.5.1

特殊情况说明：由于现在的apm项目的测试代码很多都不能成功跑起来，而且测试中有Thrift连接，导致构建起来需要很久（12个小时的时间内测试代码都没有跑完），因此我在自己搭

建的svn中将一些测试代码删除从而进行校验。

2.5.2 最终APM项目的覆盖率报告效果如图



Overall Coverage Summary

name	instruction	branch	complexity	line	method	class
all	0% <div></div>	0% <div></div>	0% <div></div>	0% <div></div>	0% <div></div>	0% <div></div>
classes	M: 137713 C: 3	M: 12013 C: 0	M: 10779 C: 1	M: 29947 C: 1	M: 4679 C: 1	M: 508 C: 1

Coverage Breakdown by Package

name	instruction	branch	complexity	line	method
com.broadapm.ajax	M: 909 C: 0 0% <div></div>	M: 32 C: 0 0% <div></div>	M: 113 C: 0 0% <div></div>	M: 251 C: 0 0% <div></div>	M: 97 C: 0 0% <div></div>
com.broadapm.base	M: 212 C: 0 0% <div></div>	M: 13 C: 0 0% <div></div>	M: 13 C: 0 0% <div></div>	M: 52 C: 0 0% <div></div>	M: 5 C: 0 0% <div></div>
com.broadapm.cache	M: 1951 C: 0 0% <div></div>	M: 140 C: 0 0% <div></div>	M: 166 C: 0 0% <div></div>	M: 459 C: 0 0% <div></div>	M: 96 C: 0 0% <div></div>
com.broadapm.common.util	M: 998 C: 0 0% <div></div>	M: 128 C: 0 0% <div></div>	M: 84 C: 0 0% <div></div>	M: 226 C: 0 0% <div></div>	M: 20 C: 0 0% <div></div>

注意事项：1. 插件的安装可能会产生一些意想不到的情况，例如jacoco 2.0.1 产生的报告页面显示的不是表格显示，而是 html 文本，卸载重新安装 jacoco 2.0.0插件即可正常显

示。SunarQube的最新版本的插件，在构建时，下拉框里没有 Invoke Standalone SunarQube Analysis 选项，降级到低版本即可。

2. 关于路径的问题。SonarQube Analysis 那里的路径是基于当前路径 (pom.xml所在目录)。

3. 不提供 sonar.tests 这个配置参数也能产生覆盖率报告。

最后，感谢姜林峰，郭庆这两位同事的热心帮助~