Hudson

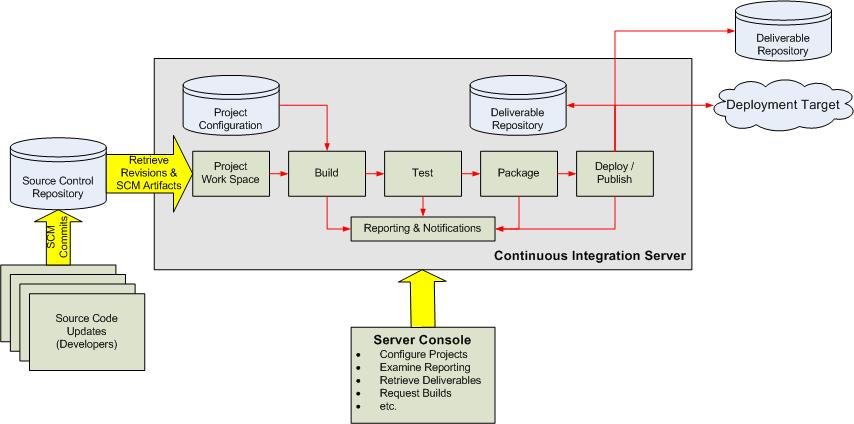


# 概述

## 何为持续集成

* *持续反馈*
* *持续编译*
* *持续测试*
* *持续整合*
* *项目透明*
* *团队共享*
* *团队共享*
* 持续集成的基本原则很简单：尽早集成，经常集成。
* **持续自动构建** ：使用CI，您只要按一下按钮，它会依照预先制定的时间表，或者响应某一特定事件，就开始进行一次构建过程。
* **持续自动检查** ：CI系统能够设定成持续地对新增或修改后签入的源代码执行构建，也就是说，当软件开发团队需要周期性的检查新增或修改后的代码时，CI系统会不断要求确认这些新代码是否破坏了原有软件的成功构建。这减少了开发者们在手动检查彼此相互依存的代码中变化情况需要花费的时间和精力
* **持续自动测试** ：这个是构建检查的扩展部分，这个过程将确保当新增或修改代码时不会导致预先制定的一套测试方案在构建构件后失败。构建测试和构建检查一样，失败都会触发通知(Email,RSS等等)给相关的当事人，告知对方一次构建或者一些测试失败了。
* **持续部署**:一旦自动化检查和测试的构建已经完成，一个软件构件的构建周期中可能也需要一些额外的任务，诸如生成文档、打包软件、部署构件到一个运行环境或者软件仓库。只有这样，构件才能更迅速地提供给用户使用。

## CI系统的基本结构



1. 开发者检查新增和修改到源代码仓库后的代码。
2. CI服务器会为每一个项目创建了一个单独的工作区。当预设或请求一次新的构建时，它将把源代码仓库的源码存放到对应的工作区，在哪里构建就执行到哪个目录下。
3. CI服务器会在新近创建或者更新的工作区内执行构建过程。
4. 一旦构建完成，CI服务器就会在一个新的构件中选择性地执行原先定义的一套测试。如果构建失败，相关责任人将会通过电子邮件、即时短信或者其他的方式获取到(失败)通知。
5. 如果构建成功，这个构件会被打包并转移到一个部署目标(如应用服务器) 和/或存储为软件仓库中的一个新版本。这个如软件仓库可以是CI服务器的一部分，也可以是一个外部的仓库，诸如一个文件服务器或者像Java.net、SourceForge分发的一个有效网址。源代码仓库和构件仓库是可以分开的，实际上它可以利用一些根本没有包含任何源代码控制系统(CVS、SVN、CSS等等)的CI服务器。
6. CI服务器通常会通过某种控制台来进行项目的配置和调试，并且根据请求响应相应的操作，诸如即时构建、生成报告，或者检索构件。

## Hudson的优势

* 安装和配置简单
* 基于Java
* *纯Web UI操作*
* *配置时帮助即时反馈*
* *插件非常丰富*
* 这是迄今为止我使用过的所有CI产品在安装和配置上中最简单的(CI产品)。
* 基于Web的用户界面非常友好、直观和灵活，在许多情况下，还对需单独配置的部分提供了基于AJAX的即时反馈。
* Hudson是基于java开发的(如果你是一个Java开发人员，这是非常有用的)，但它不仅限于构建基于Java的软件，还能构建.net、Python、Ruby等。
* Hudson本身是一个很简洁的组件，但它提供了一组很明确和可扩展API的Hudson组件。这批组成一个大的类库的Hudson组件反过来又丰富了Hudson的功能；它们都是开源的，而且它们可以直接通过Hudson的控制台来进行安装。

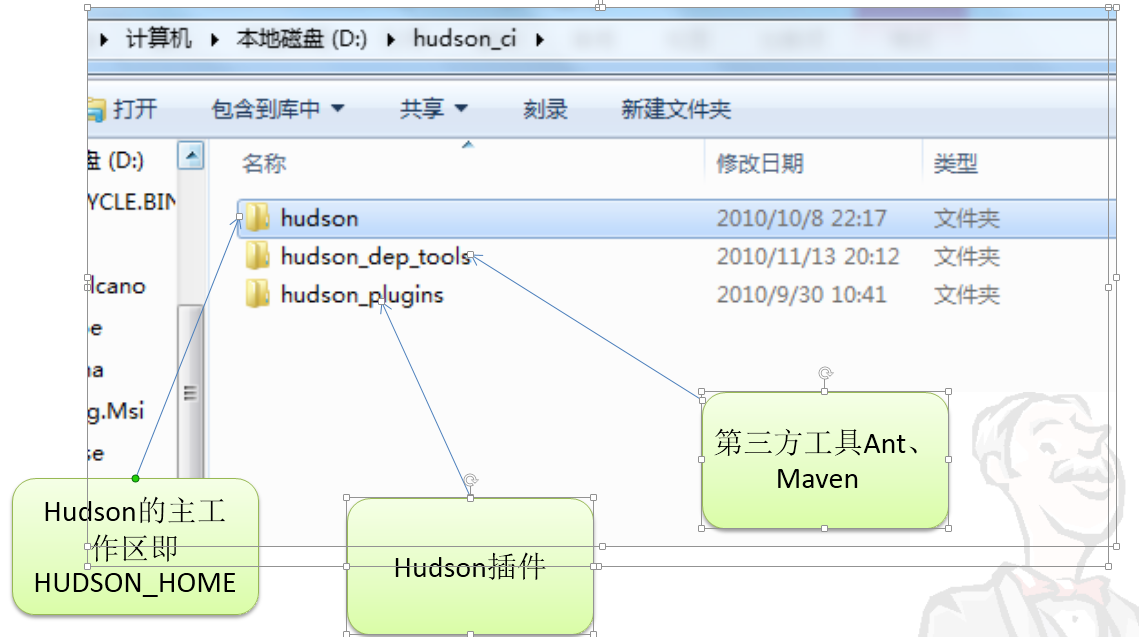
## Hudson的安装

### 命令行启动

下载Hudson.war

java -jar hudson.war  
 指定端口和安全端口：  
 java -jar hudson.war –httpPort=8084 --ajp13Port=9009

### 基于Web容器(Tomcat)[推荐]



1、在启动Hudon之前最好首先配置环境变量。

默认情况下，它被设置在~/.hudson，你知道的 ，C盘总是很紧张。所以，您可以通过下列方式改变它的位置：

首先您会把hudson.war存放在servlet容器的发布目录，那么在启动servlet容器前先设置“HUDSON\_HOME”环境变量并添加一个新的地址做为Hudson的主目录。

## hudson页面

### Hudson首页



* 导航菜单：位于页面左上方，方便用户执行各类Hudson 操作，如新建任务、系统管理等等。
* 生成队列：页面左边中间的部分，表示等待执行构建的任务。
* 生成状态：页面左边下面的部分，表示正在执行构建的任务。
* 任务状态：页面右边的部分，显示了所有任务的状态。

### Hudson系统管理



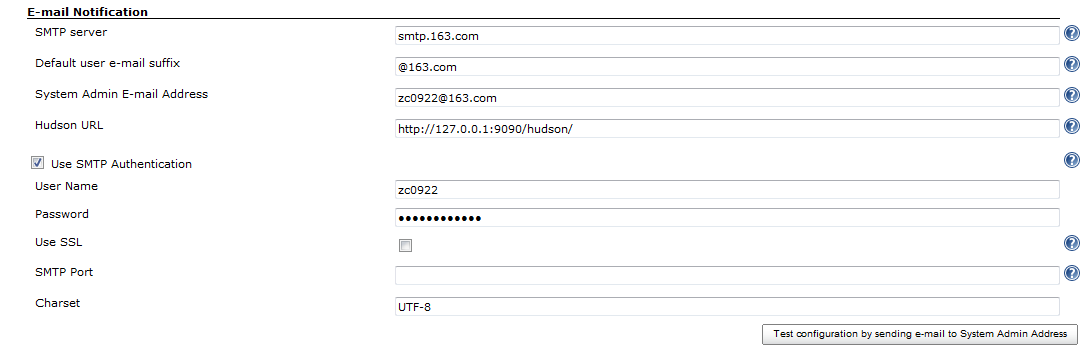
### Hudson系统设置



### 配置JDK

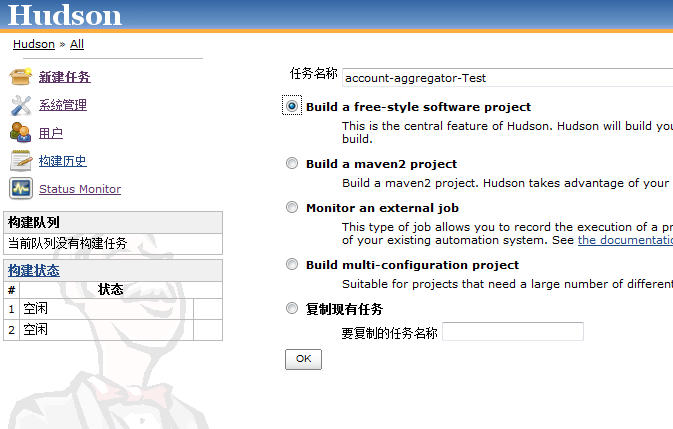


### 配置邮件



### 创建任务。

对于一般的Maven 项目来说，可选择的类型有**Build a free-style software project** 和**Build a maven2 project**，前者不仅支持Maven 项目，还支持其它类型的构建工具如Ant、Shell。对于Maven 用户来说，两者最大的不同在于前者需要用户进行多一点的配置，而后者会使用Hudson 自带的Maven，且从项目的 POM中获取足够的信息以免去一些配置。除非你非常熟悉Hudson，否则我推你使用前者，至少出问题了能多些检查点。



### 创建项目

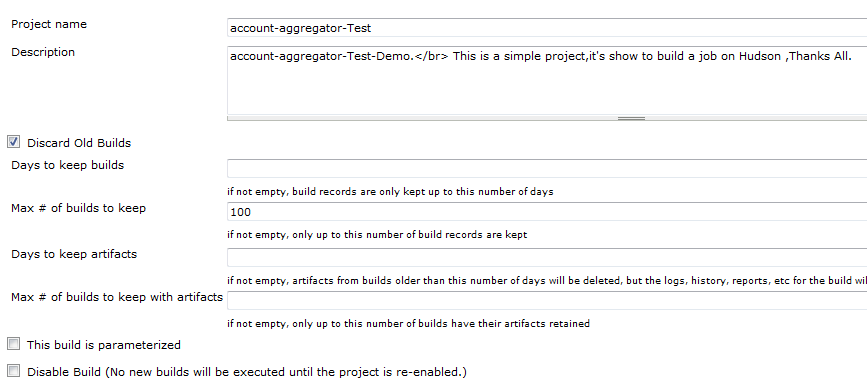
**Project name** :我已经把这个项目命名为account-aggregator-Test，但你也可以在这里修改它。

**Description** : 这是一个自由项，主要用来说明你关于这次构建工作的描述。可不填。(帮助:这说明放在项目的首页，以便访问者可以知道这个工作的内容。您也可以在这里使用任何HTML标记。)

**Discard old builds** : Hudson默认将保留过去的构建，除非你事先选中此框。(帮助:这里控制着您想要在Hudson所在的磁盘把构建记录存储的有效期(诸如控制台输出、编译构件等等)。Hudson为此提供了两个标准：1。时间驱动。在Hudson中您可以判断如果达到一定时限来删除一条记录(例如，七天前)。2。数量驱动。在Hudson中您可以确保它拥有N份构建。如果又有新的构建开始，最早的那份(记录)就将被删除。Hudson也可以让您建立的个别构建定义为'永远保持这个记录'，以便防止某些重要的构建被自动丢弃。)

**This build is parameterized** : 如果选择此选项，Hudson将允许您提供一套任意的键值对参数，它们会被传递到构建过程里。配置的参数往往是构建运行环境中的一些环境变量。(帮助：当您使用了Hudson的各种自动化，有时要求在构建过程中提供一组用户的输入，使用“parameterize”就能够更方便构建。例如，您可能会设立一个按需测试，在那里用户可以提交一个二进制文件的压缩文件来进行测试。   
本节参数可以完全按照您构建的需要配置。参数是以名字区分的，所以您可以有多个参数，只要它们名称不同。   
关于此功能的更多资料请查看[维基专题讨论](http://hudson.gotdns.com/wiki/display/HUDSON/Parameterized+Build) 。)

**Disable build** : 如果这里被勾选，这项工作将不会执行构建，直到选项禁用为止。(帮助：有时候，你会想暂停某个构建中的项目。例如，也许您正准备一次大的迁移，而且你知道新版本会失败。或者您想每一个小时构建一次，但您却发现CVS服务器将在未来24小时内down机。当这个选项被设置后，关于这个项目就不会再有新的构建。这样一来，您就可以在不想改变外部依赖或者提交错误通知的情况下禁用构建过程。)



### 配置SVN

1、如果访问Subversion 仓库需要认证，Hudson 会自动探测并提示用户输入认证信息

2、点击 **enter credential** 后，Hudson 会弹出一个页面让用户选择认证方式并输入认证信息，

输入正确凭证之后，Hudson 就能读取仓库源代码。我采取的是用户名和密码的方式。



### 配置构建

**Build after other projects are built** : 此选项支持一条装配(流水作业)线——作业依赖: 该作业依赖于另一个作业的输出的情况 —— 或者如以下情形：你只是想简单的把一些有关的工程构建编入一个组以便一起构建。当您一选择它，你将得到一个字段，输入其他工程的名字[多个项目名间用逗号分隔]后，这个构建应该就可以运行。

**Poll SCM** : 这是CI 系统中常见的选项。当您选择此选项，您可以指定一个定时作业表达式来定义Hudson每隔多久检查一下您源代码仓库的变化。如果发现变化，就执行一次构建。例如，表达式中填写0,15,30,45 \* \* \* \*将使Hudson每隔15分钟就检查一次您源码仓库的变化。更多信息请查阅[Quartz CronTrigger](http://quartz.sourceforge.net/javadoc/org/quartz/CronTrigger.html)中关于这个定时作业语法的详细描述(新版Quartz支持从秒开始触发，所以请查阅下面的cron帮助，对此引起的误导表示歉意)。

**Build periodically** : 此选项 (也是使用定时作业表达式)仅仅通知Hudson按指定的频率对项目进行构建，而不管SCM是否有变化。我这个作业就属于目标测试环境是按某种方式定期修订的而SCM却是静态的情况。如果您想在这个作业中运行一些测试用例的话，它可能就很有帮助。

**Add build step** : 按一下这个按钮，添加了一项指令以执行构建脚本。您的指令可以是下列之一：

执行 Shell

执行 Windows 批处理

使用Ant

使用Maven

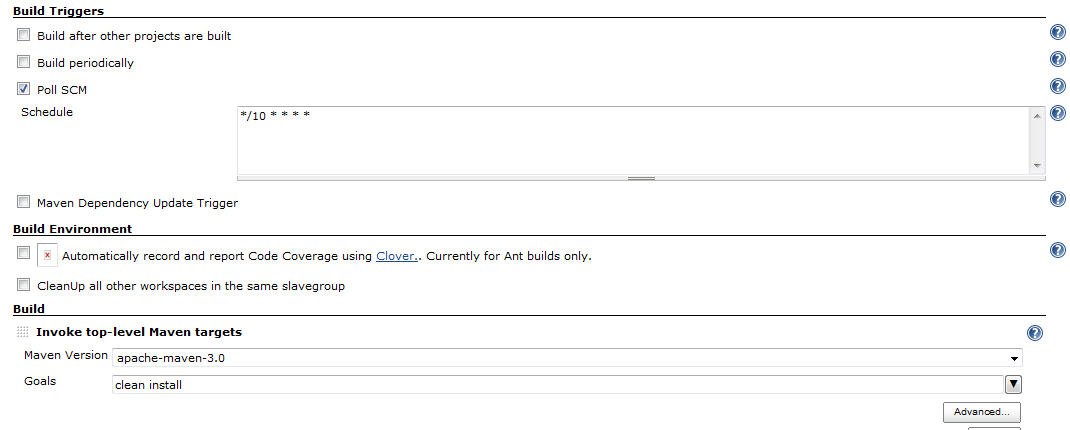
**\* \* \* \***：每分钟

q **5 \* \* \* \***：每小时中的第5 分钟

q **\*/10 \* \* \* \***：每隔10 分钟

q **45 10 \* \* 1-5**：每周一到周五的上午10:45

q **0,30 \* 13 \* 5**：每月13 号的每半小时，或者每周五的每半小时



### 配置测试

*\*\*/target/surefire-reports/TEST-\*.xml*

该表达式表示匹配任意目录下target/surefire-reports/子目录中以*TEST-*开头的XML 文

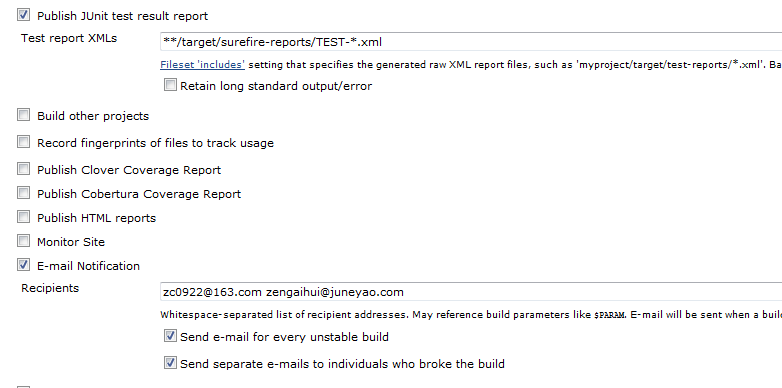
件，这也就是匹配所有maven-surefire-plugin 生成的XML 格式报告文件。

**Publish JUnit test result report** : 如果您的构建脚本执行了JUnit测试，此选项将指示Hudson处理XML测试文档并为每次连续构建产生一份可持续的报告，依据正在进行的测试汇总处理 结果。其结果是当前工作主页的一份报告，作业中的单元测试会随着时间的推移按由老至新进行陈列。

**E-mail notification** : 定义一个邮件列表( 多项用空格间隔),以便当构建失败时发送一份通知。当一次作业总是不稳定或者被终止，"**Send email for every unstable build** "就可以不选，以阻止Hudson不断发送一份已获知条件的邮件通知。

发送邮件的场景：

1. 失败的构建会触发邮件反馈。
2. 成功构建后的一次不稳定构建会触发邮件反馈。不稳定往往是由于失败的测试，因此成功后的一次不稳定往往表示有回归性测试失败。
3. 失败或不稳定构建后的一次成功构建会触发邮件反馈，以通知用户集成恢复到了健康状态。
4. 用户可以配置是否每次不稳定构建都触发邮件反馈。



### 执行构建

蓝色(绿色)：任务最近一次的构建是成功的。

q 红色：任务最近一次的构建是失败的。

q 黄色：任务最近一次的构建表成功了但不稳定（主要是因为有失败的测试）、

q 灰色：任务从未被执行过或者被禁用了。

如果图标在闪烁，表示任务正在执行一次构建。

图中的第二列天气（W）也需要稍作解释，Hudson 使用一组天气的图标表示任务长期

的一个状态，它们分别为：

万里晴空，任务80%以上的集成都是成功的。

稍有乌云，任务有60%-80%的集成是成功的。

乌云密布，任务只有40%-60%的集成是成功的。

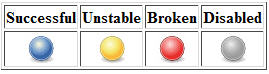
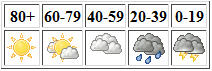
阴雨绵绵，任务的集成陈功率只有20%-40%。

电闪雷鸣，任务的集成成功率不到20%。

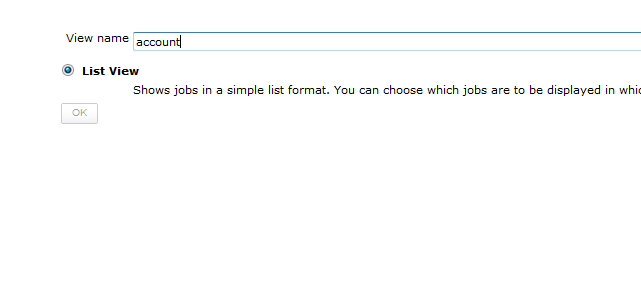
关于全局状态需要再次强调的是，当团队看到任务的集成状态不够健康时，应该尽快采

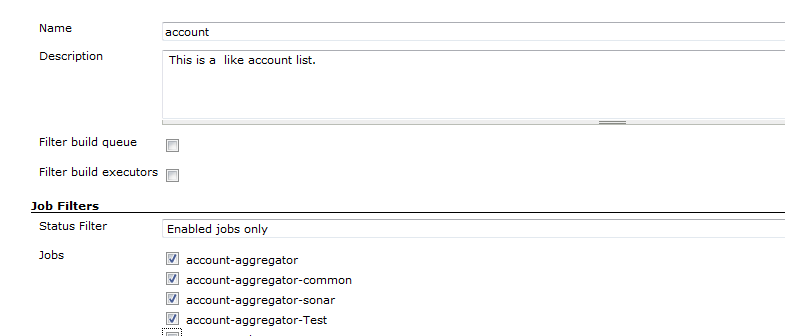
取措施修复问题





### 配置用户视图



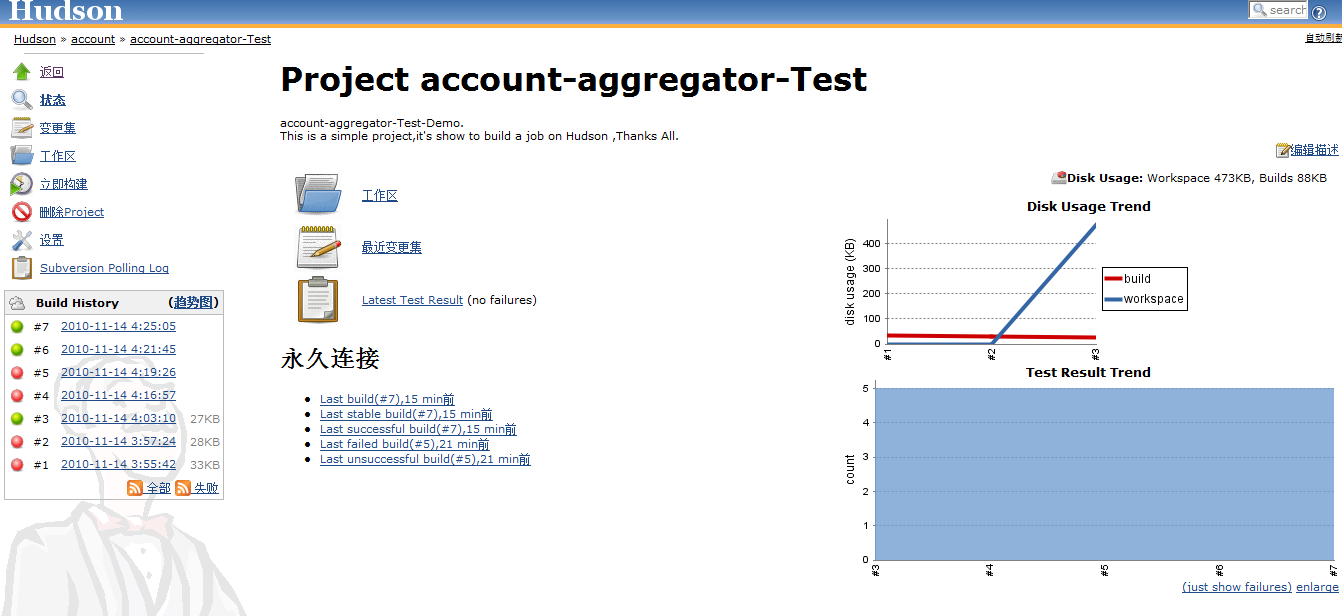




## 6.Hudson详解

首先左下角是是构建历史，该例中显示了最近7次构建，包括每次构建的时间。

图中间下方还有几种个永久连接，分别指向了最近一次构建、最近一次失败的构建、以及最近一次成功的构建。



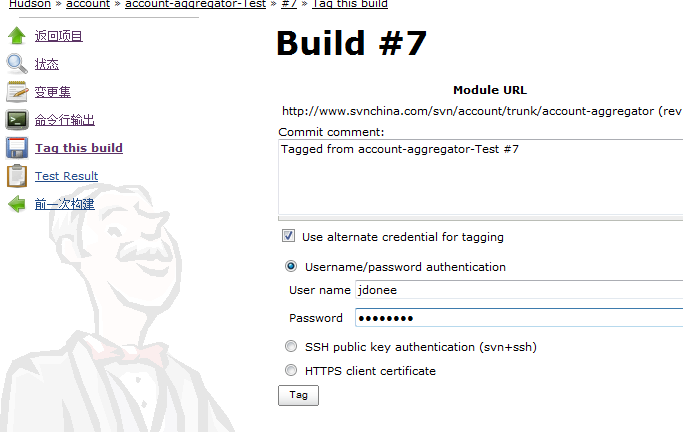
[Hudson](http://127.0.0.1:9090/hudson/) » [account](http://127.0.0.1:9090/hudson/view/account/) » [account-aggregator-Test](http://127.0.0.1:9090/hudson/view/account/job/account-aggregator-Test/) » [#7](http://127.0.0.1:9090/hudson/view/account/job/account-aggregator-Test/7/) 表示表示目前的位置是Hudson服务器下account 任务的第7次构建。

我们可以了解到这次构建所发生的时间、相关的代码变更等信息。

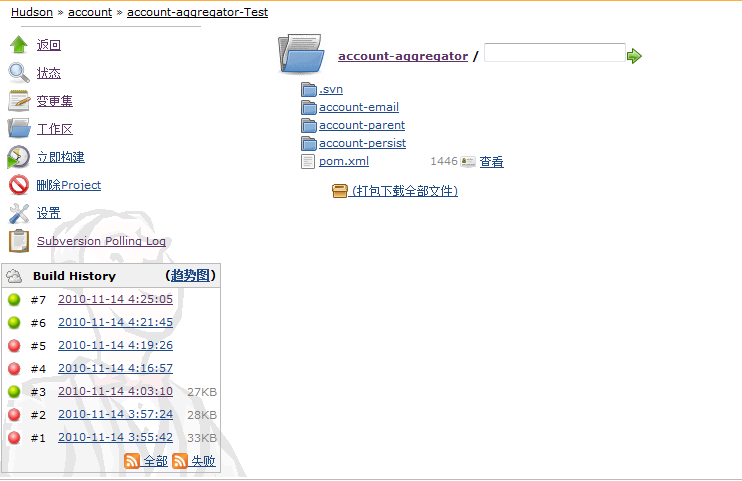
### 命令行输出



### 打Tag标记



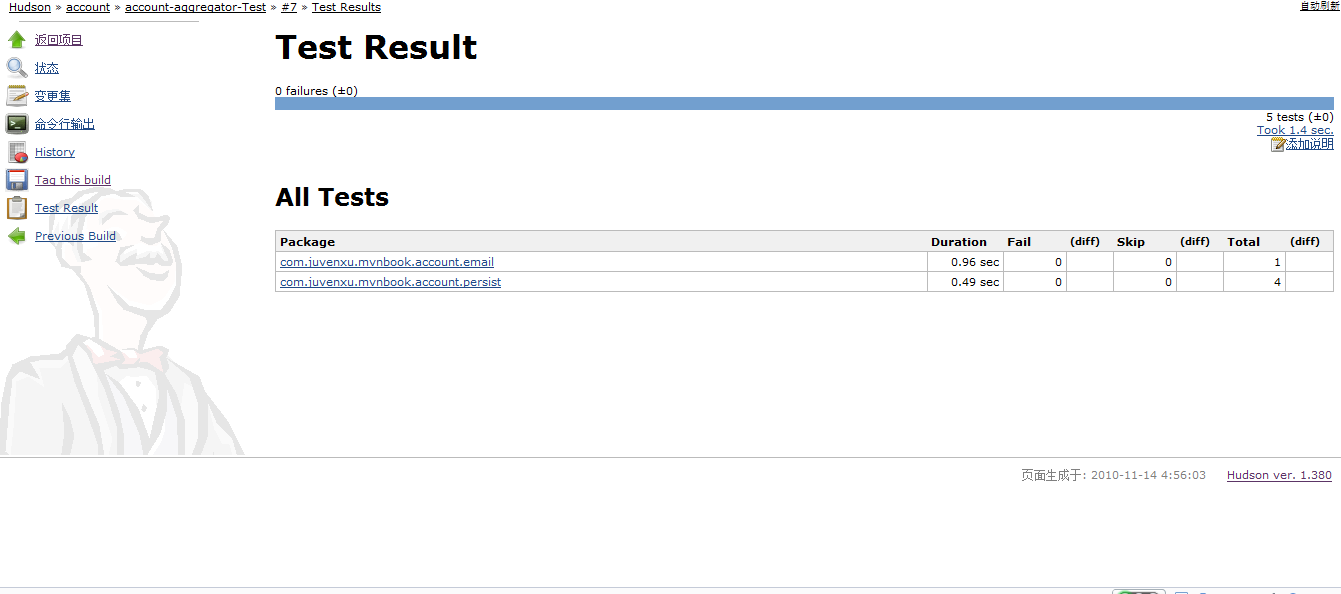
### 工作区



### 测试结果

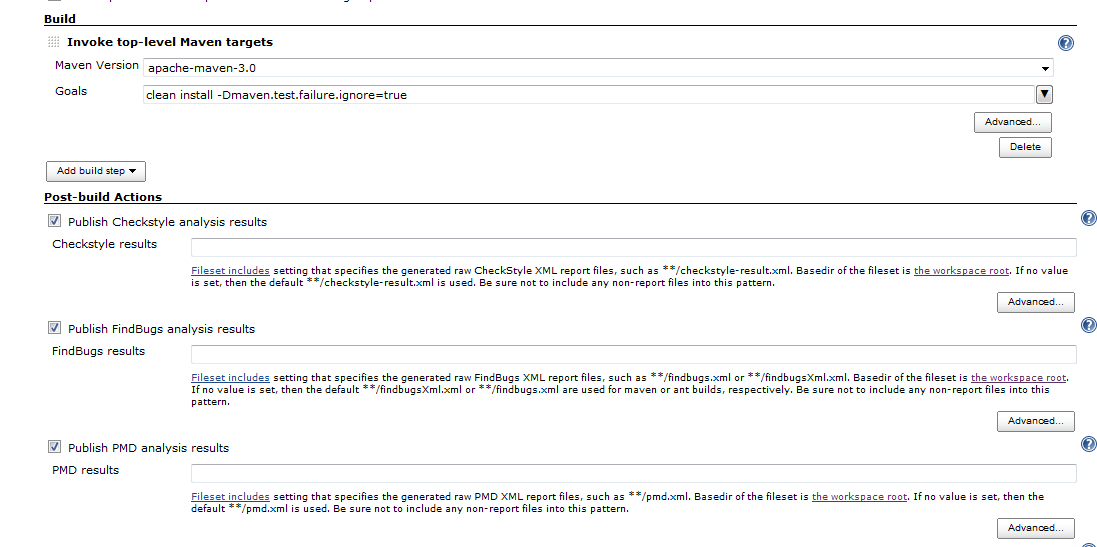
如果用户为一个Hudson 任务配置测试报告，就可以同时配置构建命令忽略测试，Maven 构建命令可以更改为clean install -Dmaven.test.failure.ignore=true，这样失败的测试就不会导致构建失败，也就是说构建的状态不会是红色，同时，由于Hudson 能够解析测试报告并发现失败的测试，构建的状态也不会是健康的蓝色。结果就是，用户会看到黄色的任务状态，表示构建不稳定。这种配置方式能够帮助用户区分失败的构建与不稳定

的构建。



Maven工程必须添加checkstyle、findbugs、PMD以及Site插件

如果不指定路径的话，他们会使用默认的规则。

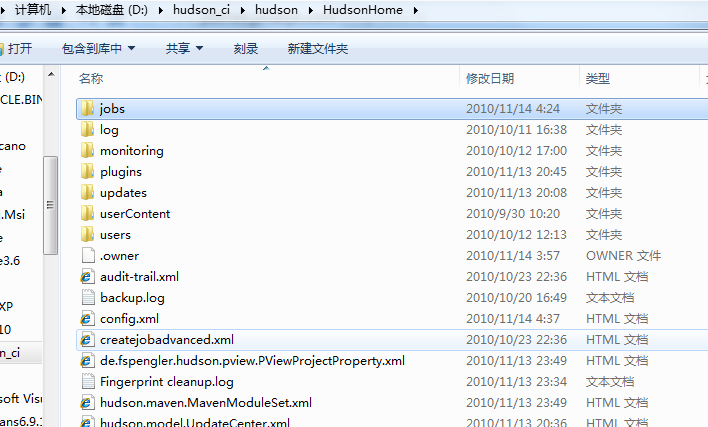




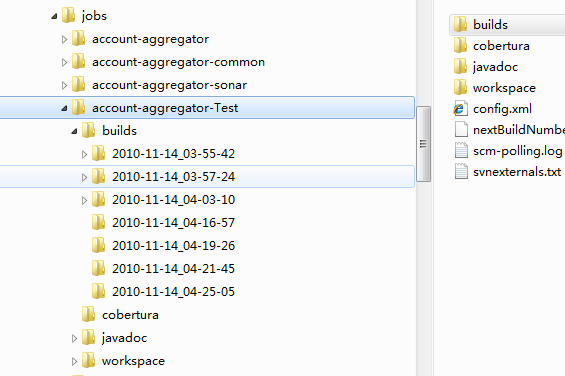
# 高级详解(Advanced)

## Hudson工作目录

* **\*.xml**：这些XML 文件是Hudson 核心及相关插件的配置，如config.xml 配置了全局的JDK、任务视图等信息，*hudson.tasks.Maven.xml* 配置了Maven 安装信息，*hudson.tasks.Mailer.xml* 配置了邮件服务器信息，等等。
* **war**：如果用户独立运行hudson.war，那么其内容会被释放到该目录中后再启动。
* **users**：Hudson 所存储的用户信息。
* **userContent**：用户可以将任意内容放到该目录下后通过Hudson 服务页面的子路径访问，如：http://127.0.0.1:9090/hudson/userContent/。
* **updates**：这里存储了各类可更新的插件信息。
* **plugins**：所有Hudson 插件都被安装在该目录而不会影响到Hudson 的核心。
* **jobs**：该目录包含了所有Hudson 任务的配置、存储的构建、归档的构建输出等内容。



* 每个任务都会包含如*config.xml*、*nextBuildNumber*、*scm-polling.log* 等文件，这其中的
* *config.xml* 包含了该任务的所有配置，如SCM 地址、轮询频率等等。
* 每个任务目录下会包含一个 workspace 子目录，这就是该任务的工作区，这里有最近一次构建所包含的源代码及相关输出。
* 任务目录下还有一个 builds 子目录，该目录包含了所有Hudson 记录的历史构建，每个构建对应了一个目录，这些目录都是以构建所发生的时间命名的，如*2010-04-15\_14-56-08*，
* 每个构建目录包含了一些文件记录其成功失败信息、构建日志、测试报告、变更记录等等。
* 如果用户为该任务配置了文件归档，那么每次构建归档的内容都会存储在*archive* 子目录下。



## Hudson安全控制

### 基于用户控制

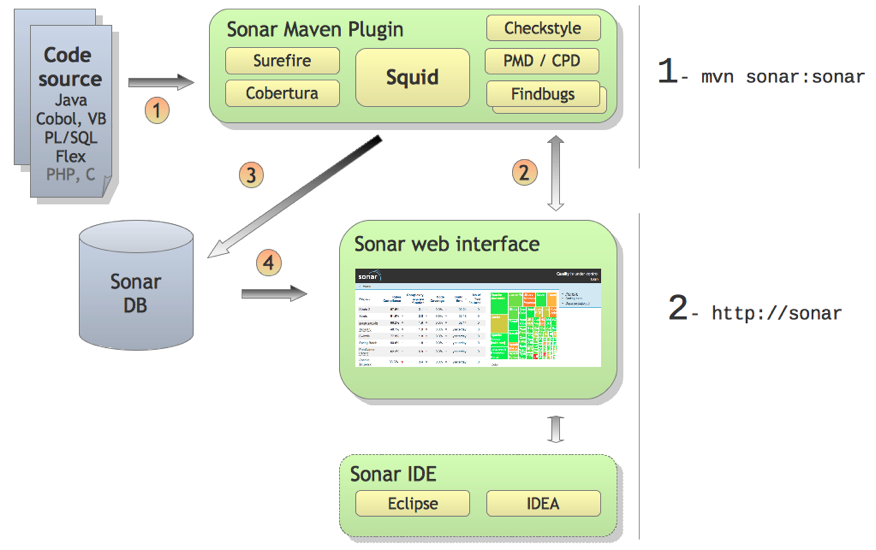
### 基于项目控制

## Hudson最佳实践

* Hudson一直处于安全状态。
* 定期备份Hudson主目录。
* 确保版本仓库中取出干净的源代码。
* 运行普通构建后持续运行自动化测试构建并生成报告。
* 在最空闲的磁盘分区设置Hudson主目录。
* 大项目时分段构建生命周期。
* 不再使用的构件先归档，再删除。
* 为每一个项目建立单独的视图

# Sonar简介

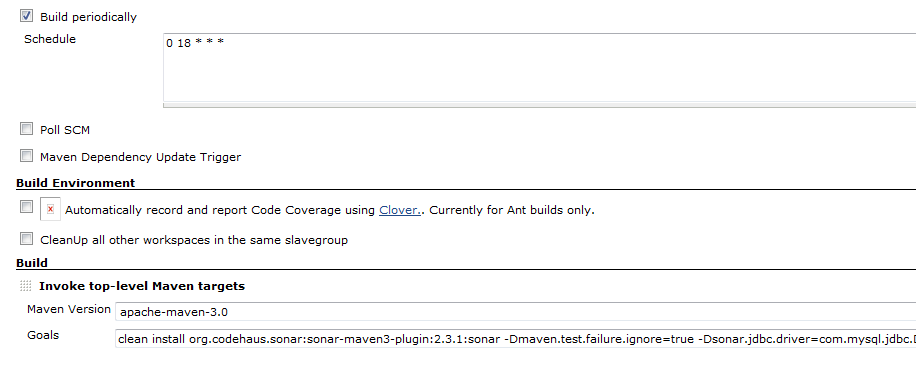
## 1.Sonar结构图



## 2.Sonar任务

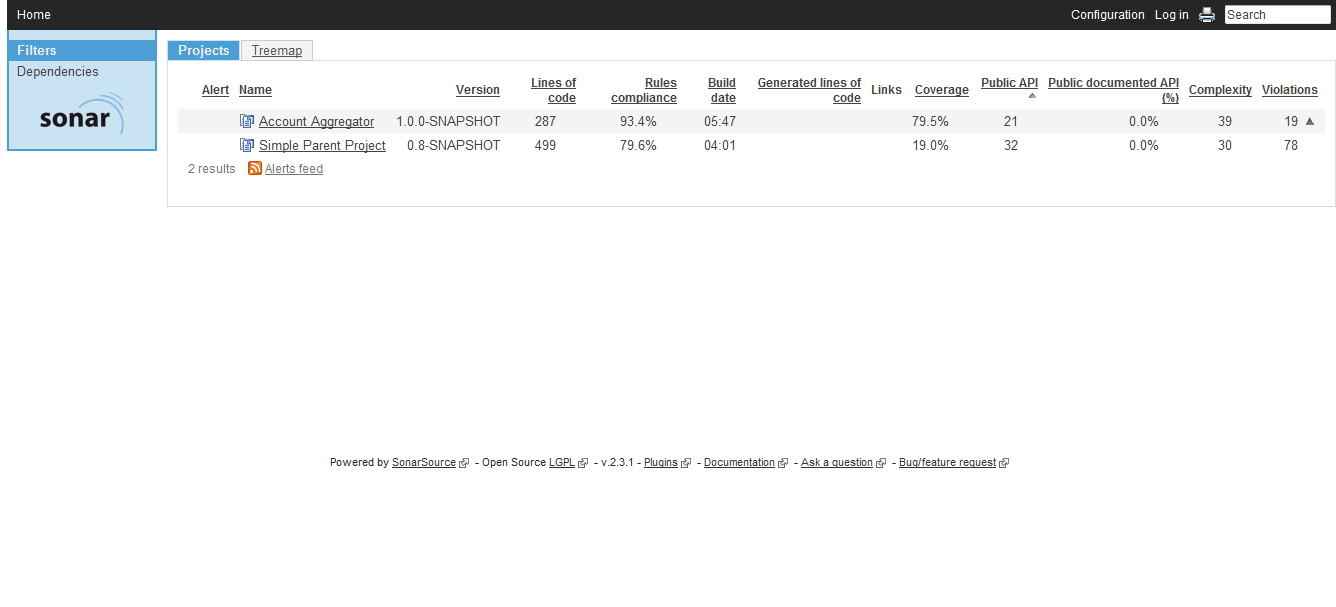
0 18 \* \* \* 每天6点执行构建

clean install org.codehaus.sonar:sonar-maven3-plugin:2.3.1:sonar -Dmaven.test.failure.ignore=true -Dsonar.jdbc.driver=com.mysql.jdbc.Driver -Dsonar.jdbc.url=jdbc:mysql://127.0.0.1:3306/sonar?useUnicode=true&characterEncoding=utf8 -Dsonar.jdbc.username=sonar -Dsonar.jdbc.password=sonar -Dsonar.host.url=http://127.0.0.1:9000/



## 3.Sonar分析(一)

生成报告后进入 Sonar查看。



## Sonar分析(二)

页面描述：

Lines of code 287：表示该项目有287行代码。

Comments [0.0%](http://127.0.0.1:9000/drilldown/measures/1?metric=comment_lines_density) ：公共API文档为0。

Complexity [2.2](http://127.0.0.1:9000/drilldown/measures/1?metric=function_complexity) ：代码的圈复杂度2.2。

Rules compliance [93.4%](http://127.0.0.1:9000/drilldown/measures/1?highlight=weighted_violations&metric=weighted_violations) ：项目的匹配程度为93.4%。

Violations 19：项目违例的数量为19。

需要注意的几个参数：

代码行：项目的代码行数。这个度量仅仅测量不包含注释和空行的Java代码总数。

类总数：包含在项目内部的类、接口、枚举和注解。

方法数：统计中不会包含get和set方法。但是会把构造函数计算在内。

Sonar会按优先顺序进行统计违例：失败([Blocker](http://127.0.0.1:9000/drilldown/violations/1?priority=BLOCKER))、严重的([Critical](http://127.0.0.1:9000/drilldown/violations/1?priority=CRITICAL))、重要的([Major](http://127.0.0.1:9000/drilldown/violations/1?priority=MAJOR))、一般的([Minor](http://127.0.0.1:9000/drilldown/violations/1?priority=MINOR))、信息类([Info](http://127.0.0.1:9000/drilldown/violations/1?priority=INFO))。

圈复杂度：也可称为McCabe复杂度。添加一些对如 'if', 'for', 'while'之类的统计。除getter和setter外的其它方法复杂度起始值都是1。每个下面的语句和关键字增加都会增加1个复杂度：

for

while

case

catch

throw

return (如果不是方法的最后一句的话。)

&&

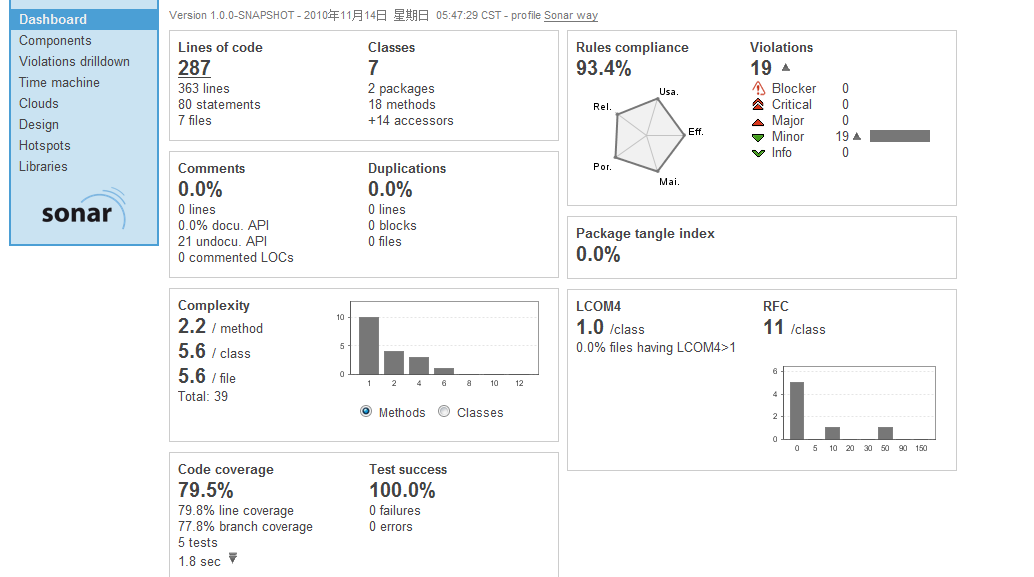
||

?

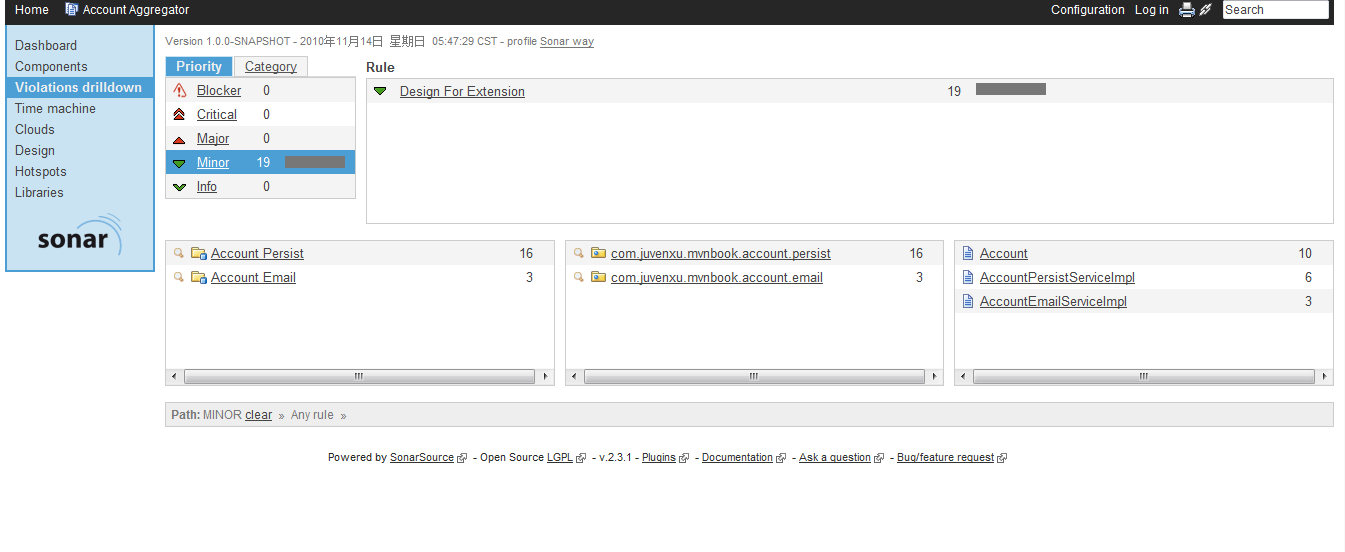
类的圈复杂度：计算一个项目里类的平均圈复杂度。

方法的圈复杂度：计算一个项目里所有方法的平均圈复杂度。

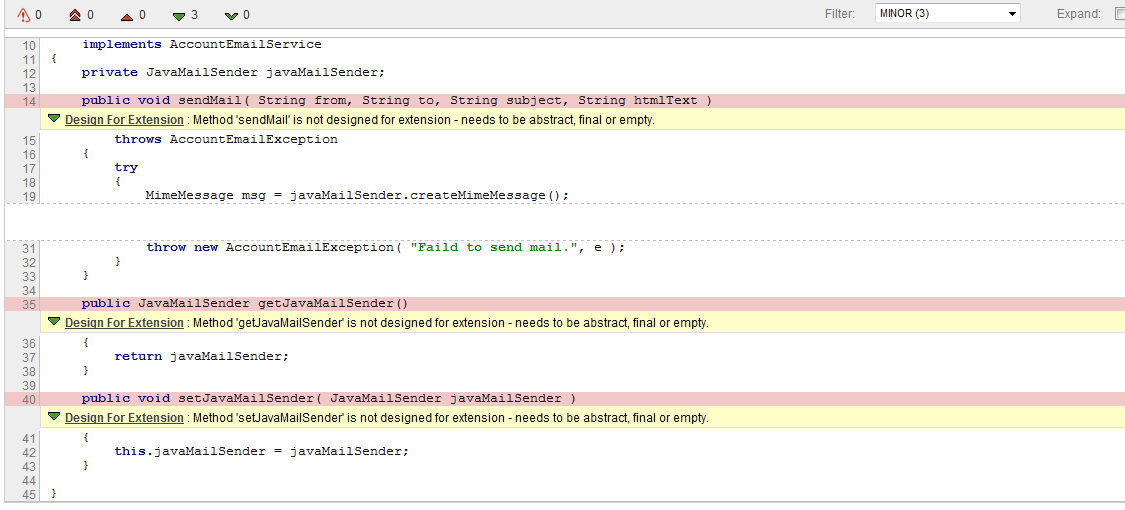
公共API文档统计：统计类、方法和公共属性的Javadoc数量。



## Sonar分析(三)



## Sonar分析(四)



## Sonar分析(五)

点击Sonar主页 选择EditFilter 增加 你需要在项目首页显示的字段。

必须登录。

示例如下：

metrics Size -> Lines of code, Complexity -> Complexity, Documentation -> Public Document API (%) and Rules -> Violations.

