Vi 学习报告

其实用vim挺久了， 在没有增加自己的配置的时候，vi还是很难用的，然后一般在刚开始用的时候因为要增加配置文件和记忆不少命令，所以很多人在开始的时候就用不下去了。

因为vim也是vi增强版，所以我都是直接使用vim。下面也是我学习使用vim的报告。

但是作为一个使用vim有挺长一段时间的人来说，vim是真的很好用。对我来说，一个软件很重要的一点是“流畅”。而vim相对于其他许多的IDE来说其流畅程度是首屈一指的。我比较喜欢的vim的分屏功能，这个很重要。因为linux本身在终端的拷贝粘贴还是比较麻烦（ctrl+shift+c/v）.并且一般在没有使用鼠标的情况下这样也不利于操作。而vim的y，p的拷贝粘贴功能只能在同一个vim窗口下面使用。然后在一些情况下我们要同时对两个文件同时操作的话，或者要看着一个文件来写另一个文件，分屏就成了一个很不错的选择。

下面是分屏的一些命令:

分屏启动Vim

1. 使用大写的O参数来垂直分屏 vim -On file1 file2 ..
2. 使用小写的o参数来水平分屏 vim -on file1 file2 ...注释: n是数字，表示分成几个屏。

关闭分屏

1. 关闭当前窗口 Ctrl+W c
2. 关闭当前窗口，如果只剩最后一个了，则退出Vim Ctrl+W q

分屏

1. 上下分割当前打开的文件 Ctrl+W s
2. 上下分割，并打开一个新的文件 :sp filename
3. 左右分割当前打开的文件 Ctrl+W v
4. 左右分割，并打开一个新的文件 :vsp filename

移动光标

Vi中的光标键是h, j, k, l，要在各个屏间切换，只需要先按一下Ctrl+W

1. 把光标移到右边的屏 Ctrl+W l
2. 把光标移到左边的屏中 Ctrl+W h
3. 把光标移到上边的屏中 Ctrl+W k
4. 把光标移到下边的屏中 Ctrl+W j
5. 把光标移到下一个的屏中 Ctrl+W w

移动分屏

这个功能还是使用了Vim的光标键，只不过都是大写。当然了，如果你的分屏很乱很复杂的话，这个功能可能会出现一些非常奇怪的症状。

1. 向右移动 Ctrl+W L
2. 向左移动 Ctrl+W H
3. 向上移动 Ctrl+W K
4. 向下移动 Ctrl+W J

屏幕尺寸

下面是改变尺寸的一些操作，主要是高度，对于宽度你可以使用Ctrl+W <或是>，但这可能需要最新的版本才支持。

1. 让所有的屏都有一样的高度 Ctrl+W =
2. 增加高度 Ctrl+W +
3. 减少高度 Ctrl+W -

当然，当要操作的文件很多了，直接分屏会出现窗口太小不好操作的情况，这个时候我们可以直接在打开vim的时候用 vim file1 file2... 然后在vim下进入到命令模式然后按 :n 选择下一个文件， :N选择下一个文件。

下面说一下我在使用vim的时候比较常用的操作：vim有一个比较常用的功能是它能在你没有保存就因意外退出的时候它会自动生成一个.swp后缀的文件，这个文件会在你再次启动编辑这份代码的时候提示你要不要重新载入，如下：

这个时候我们可以选择(o)只读打开，(e)不管备份文件直接编辑，(d)把备份文件删除，(q)退出，(a)退出。其实q和a没多大区别，可以默认为一致。
这个功能是vim会在退出之前进行保存，就算你是直接kill掉进程，它也能保存下来。

下面一个是比较小的功能，我们一般会在打开文件的时候就在 vim命令后面跟上文件名，那如果代开的时候忘文件名了或者中途想改文件名，那么可以在命令模式下按 :w filename来保存。在保存的时候如果出现 read-only 的提示的情况下，如果有提示可以通过增加！来保存，那么可以通过 在命令后面增加！来进行操作。

vim还有一个比较有趣的命令是块选择，我们一般使用 v 进行字选择和 V 进行行选择，那如果我们想进行整块的操作，比如在多行的后面加上相同的一个词，比如如下代码：

```
print lista
print listb
print listc
print listd
```

如果我们想打印每个list的大小，也就是在每行后面加上.size()，这个时候我们就可以用块选择，也就是 ctrl+v 然后使用方向键（或hjkl）来选择块，然后直接粘贴即可。这个功能还是很常用的。

下面的n表示数字

还有一个比较好用的功能是 n+cmd. 表示对那个命令操作多少次。比如 12space 就是光标向后移动12个位置，nx表示删除从光标开始的多少个字母。ndd表示删除多少行等，这样一个命令还是很经常用的。

其实vim有很多强大的命令还有比如ZZ表示有修改就保存退出没有修改就直接退出等，这些命令不能说看一眼就全部记住，都是要经常用了才能记住。在linux平台上来说，vim是在是一个很好的选择，不同的*unix平台可能有很多软件不同，但是它们都会有一个vi，我们在刚开始系统的时候如果要修改一些配置文件，那么学会了vi就会方便很多。另外vim的轻便性，快捷行，对于一个键盘手来说，他的快捷键功能也是一个非常好的选择。

下面说说vim的配置，我们能在根目录下找到一个 .vimrc 的隐藏文件(la -a)，然后可以根据vim的一些配置项对它进行配置，比如语法高亮，自动缩进等。我们还可以安装插件，比如Bundle. 这样我们就能有不同语言的自动补全，错误提示等功能。但是我们在使用插件的时候要注意vim的版本，版本太低的话很多插件是无法使用的。

嗯这些就是我的vim学习报告，也不能把每个命令的学习都写上来，这些是我觉得比较实用且好用的东西。

JAVA 学习报告

总的来说，java和c++还是有不少相似之处的。就常语法来说，比如变量类型，面向对象思想等。所以我在学习java的时候主要看的是java和c++不一样的地方。

一个是java没有指针这个东西，其实我是比较喜欢用指针的，所以一开始还不是很适应。java没有指针，并不是说java不需要指针，相反，java比c/c++更依赖指针。java里面的引用就相当于指针，只不过java的指针相对于编程者来说是透明的。java会自动去处理无指向内存。也就是说如果一段内存没有一个变量指向它了，那么java就会自动释放那段内存，这样就不会出现内存泄露问题。

再就是java的继承，一个java类只能继承一个父类。这是为了防止多重继承容易引发的意义不明确问题，例如：有一个类C，如果允许它同时继承A类与B类（class C extends A,B{}），假如A、B两个类都有同一种方法fun()，如果定义：C c = new C();
那么c.fun()应该调用哪一个父类的fun()方法？无法给出答案，因此Java语言禁止多重继承。但是java的一个类可以实现多个接口，因为接口里面的方法都是抽象的，所以不会出现上面的问题。这样也弥补了不能继承多个父类的问题。

还有一个就是java的包引用，在eclipse或者直接在命令行，ant中，刚开始的时候包的引用都给我带来了比较大的问题，eclipse里面比较好解决，直接导入到项目即可。而在命令行，我们可以加一个-cp来将包的路径设置到classpath里面。在ant里面也是，将包的路径设置到classpath里。

对于java文件，有一个就是一个文件只能有一个public类，这个类的名字要和文件名一个样。还有其他的比如HashMap等数据结构的使用就不说了。

总的来说，java和c++还是有比较多的相似之处。所以学习起来还是比较快的。

Ant 学习报告

ant是一个基于java的build工具。和c++的make有点相似之处，但是相对于make来说，ant在格式限制方面没那么严格。另外，因为ant是基于java写的，所以ant也可以跨平台使用。它与make那种基于shell命令的扩展模式不同。

ant是一个将项目自动化完成项目编译，打包，测试等的工具。相当于一个批处理作业。它的结构是xml格式，我们可以在里面增加许多target任务，然后ant就会根据任务之间

的依赖关系进行项目处理。因为ant的跨平台性和简单的操作，它还能集成到一些开发环境里面。

对于ant的语法，今天看一个同学的错误的时候发现了问题，其实也就是java包的导入问题。在进行junit测试的时候，总是出现test的包没有导入的问题，然后直接把包放到项目目录下也不行。后面我发现需要将path增加到classpath里面，也就是使用path的classpathref属性将path的reference设置到classpath里面，这样ant运行的时候就能找到对应的包了。

然后是写ant测试文件，我们一般要给project增加一个default，也就是默认执行的target。然后要注意每个target之间的依赖关系。

在ant里面，我一般建立这样几个文件夹，lib放置jar包文件，src放置原文件，build/classes放置构建的.classes文件，然后如果有junit test的话，就增加一个文件夹build/test放置产生的test文件和一个 build/test/report文件夹放置测试报告。这样的话项目结构就会比价清晰。

另外我们在运行测试的时候可以用ant targetname 来运行对应的target。比如初始化，清除等。

在对于ant具体的语法，没有太多需要说的地方，我们主要每个target之间的依赖关系，沥青文件结构即可。

Junit 学习报告

junit是一个回归测试框架，junit是程序员测试，即白盒测试。目前来说很多的开发环境都集成了junit，比如eclipse。写junit测试很多时候是在正式写代码前写的。那么这就需要你在写代码之前思考好代码的方法，功能，逻辑。而且，测试和代码都是增量式的，这样可以及时测试，及时发现问题，从而减少回归测试的难度。

对于junit的使用，在我们目前的测试里面主要用到的是@Test，@After，@Before和@Ignore 这些功能，一般是调用一个方法，我们直接计算一个方法的返回值应该是什么，然后用assertEquals来判断实际返回值和预期是否一致即可。junit会返回测试的结果，看有哪些是与预期不一致的，这样我们就能根据结果来修稿代码。

我们可以通过修改test 的runner来设置（即@RunWith (Parameterized.class)）来设置成可以一次测试多组数据，这样就不用为每组数据建立一个test方法。

在默认的情况下，test方法的执行顺序是随机的，那么我们可以设置rule来设定执行顺序。另外，我们可以通过设置@Before，@BeforeClass，@After，@AfterClass来设置每个test之前和之后要执行的函数（@Before，@After），只在第一个test之前和最后一个test之后执行的函数（@BeforeClass，@AfterClass）等。

一般来说我们跑一次test就是跑一个test文件，但是我们也可以跑多个test文件，比如我们为源码每个类都建立了一个test文件，然后不想跑那么多次的test，那我们就可以使用Suite批量执行Test。

junit的学习报告就这些，这里说的也只是一些比较容易出错和比较常用的地方，其他的一般知识点就没说了。

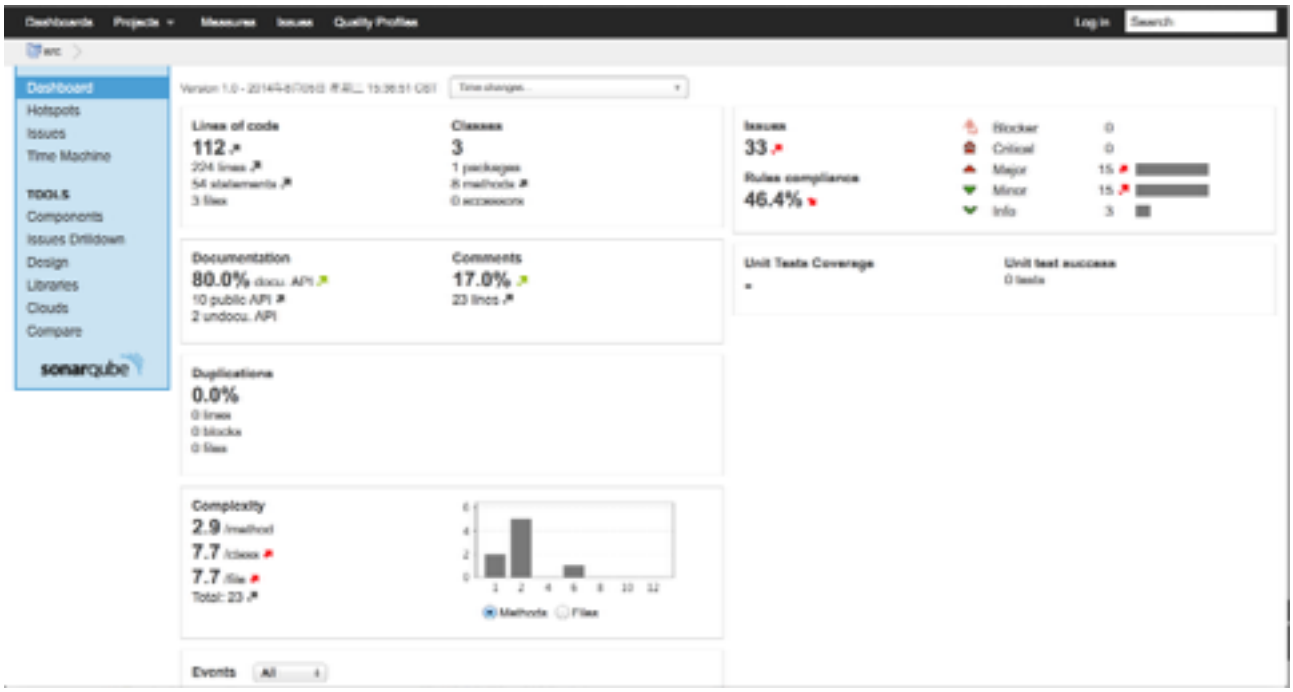
SonarQube 学习报告

sonarqube是一款静态代码分析的软件，能够发现代码中潜在的缺陷，比如异常吞没、注释或多或少、不恰当的类声明等，为代码的重构提供了很多指导.

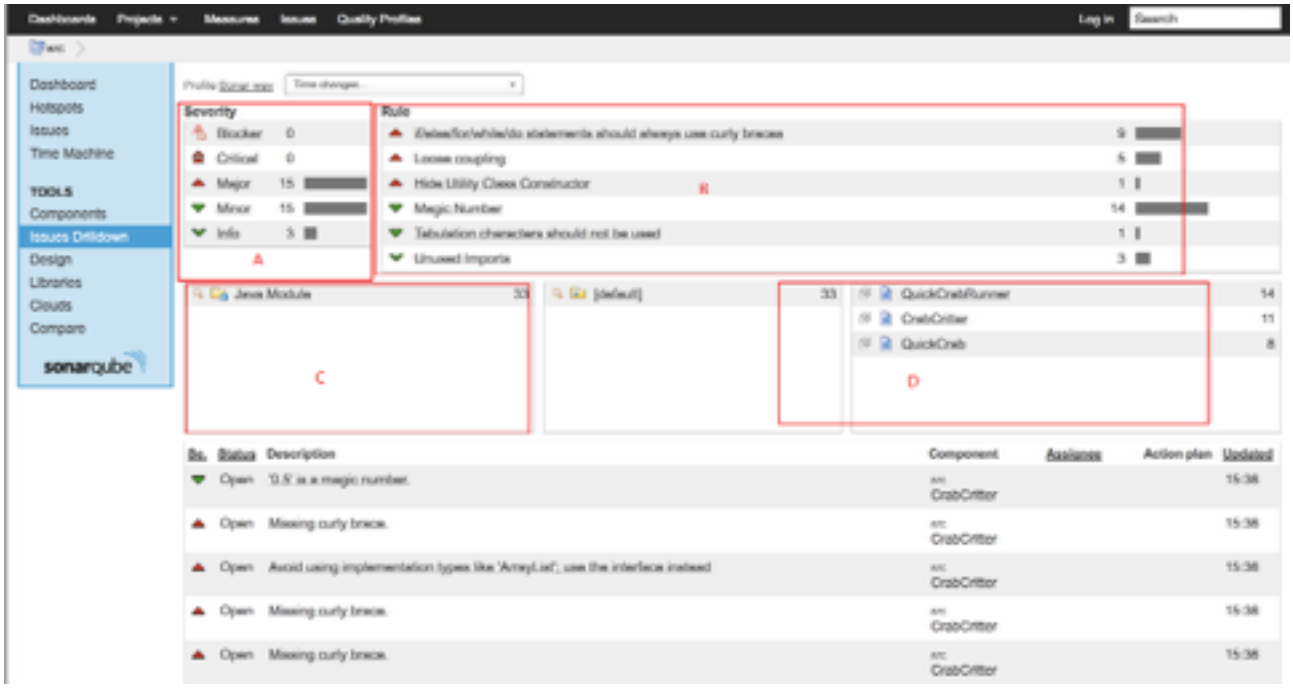
首先我们能用sonarqube对代码的风格进行分析，就目前我测试的代码来说，它会有不推荐使用空格代替tab，不要直接使用数字，分析注释程度，找出为使用的import和变量，方法等。sonar还能使用如PWD这样的工具分析潜在的bug。

下面是GridWorld的一个分析报告的Dashboard，它是查看任意项目分析结果的入口，展示了该项目各项指标总的分析数据结果，其中包括项目的重复率，注释的比例，单元测试

覆盖率以及 Issue 的比例等。通过 Dashboard 就可以对项目总体的质量情况有一个全局性的认识。因为没有安装其他插件，所以没有更多详细的分析结果。



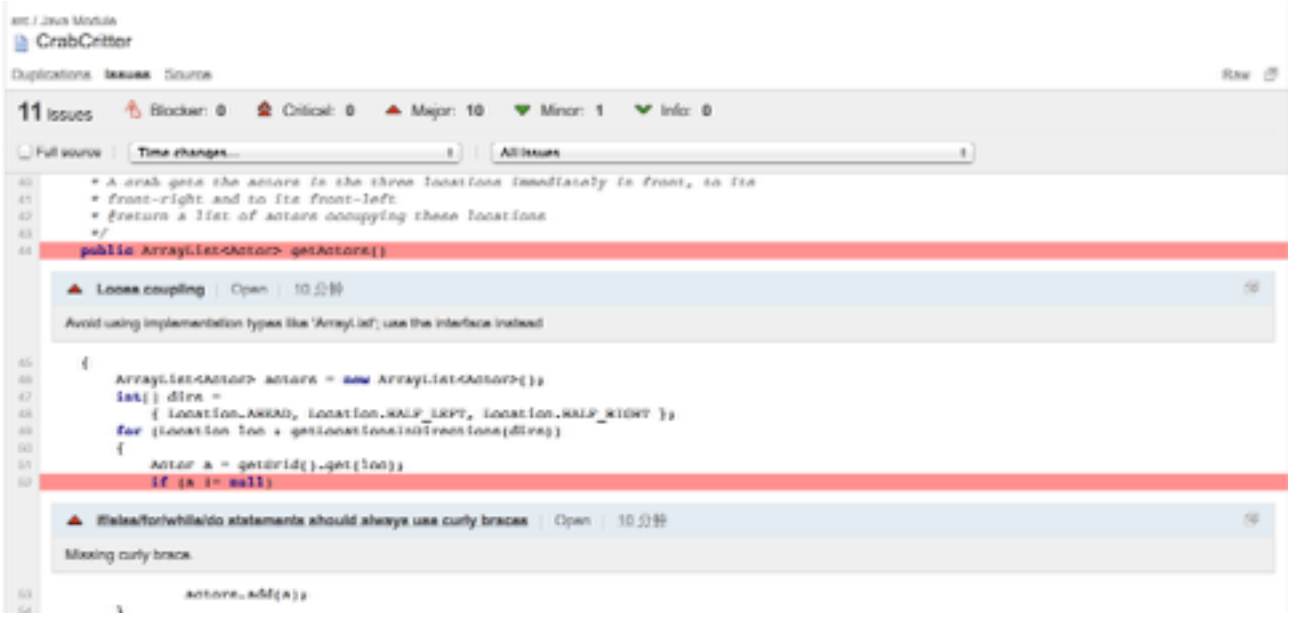
下面以Issue 为例，对其分析结果进行解析。



上图的区域 A 展示了整个项目中不同严重程度的 Issues 数量；区域 B 则显示了项目代码具体所违反的各种编码规则以及违反规则的次数；点击区域 B 中相应的规则，在区域 C 中就会显示该项目中违反相应规则的所有模块及其违反规则的次数；选择区域 C 中

的某一模块，在区域 D 中就会显示该模块内违反规则的所有文件；如果要查看代码文件到底如何违反了某一规则，只需在区域 D 中点击相应的文件，则会显示如图 16 所示的详细情况。

在下图中，可以看到违反相应规则的具体代码，用红色标出；点击所违反的具体规则，SonarQube 则会解释代码违反规则的具体原因，并给出相应的解决方案以供参考，这对于定位问题和解决问题都有很大的帮助。



目前这些就是我对sonarqube的学习报告了。