* Cobertura大家想必都知道并正在使用了，这里我整合了关于这个工具的有用信息+一点说明。

**官网**

<http://cobertura.sourceforge.net/>

官网首页上的两篇文章：

* Measuretest coverage with Cobertura, by Elliotte Rusty Harold 仅仅是简单介绍了cobertura大概的样子，而主要谈了一些对于测试的认识。

另外文章中关于安装和使用的部分已经过时：

[http://www.ibm.com/developerworks/java/library/j-cobertura/](http://cobertura.sourceforge.net/)

* Inpursuit of code quality: Don't be fooled by the coverage report, byAndrew Glover 介绍了应该如何看待code coverage的问题，**非常值得一看**。

<http://www.ibm.com/developerworks/java/library/j-cq01316/index.html?ca=drs>

* 入门：[http://cobertura.sourceforge.net/introduction.html](http://www.ibm.com/developerworks/java/library/j-cq01316/index.html?ca=drs)

**Cobertura的三个指标：line coverage，Branch coverage和复杂度。**

* line coverage就是执行的代码行数占总行数的百分比。很容易理解。
* Branch coverage是指，如果你的代码中有if等分支时，覆盖分支的情况。例如：

if (i == 0 && j ==0)

begin

i++;

return i;

end

可以理解Branch 的总数是一个有一个入口，一个或多个出口的迷宫。只有将迷宫中的每个地方都走遍(有时有些代码可能需要走多次)，并从每一个出口走出迷宫的多个test case，才能将Branch coverage达到100%。

需要注意的是，上例被认为不是多出1个Branch而是4个。应于i和j不同的取值，会有条件1成立条件2不成立，条件1成立且条件2成立，条件1不成立条件2成立，都不成立的四种情况。

因此也可能出现line coverage已经100%，但Branch coverage还很低的情况(类似虽然经过了所有的路，但是最后总是从一个出口走出迷宫的情况)。因此要使Branch coverage达到100%将是比line coverage100%更艰巨的任务。同时如果代码中的分支比较多，就会发现需要覆盖的Branch相当惊人。Cobertura基本上是根据分支条件中的and和or关系来计算Branch的。

* 复杂度complexity

McCabe's cyclomatic code complexity algorithm is used to determinehow "complex" a given piece of code is. As code becomes morecomplex, it becomes more error prone. If a class has a highcomplexity number, then that class is a good target for additionaltest coverage.

McCabe是一个分析应用程序安全性及其质量的工具。

http://www.mccabe.com/

**Cobertura是如何实现的**

Cobertura是构建在JCoverage之上的开源软件。(JCoverage已经不再更新，因为开发人员都做商业JCoverage+去了。)

思考一下，要实现计算test suite的code coverage大概要怎样实现呢？

Cobertura是如何得知，你的代码是否被调用了呢？其实是通过更改编译过的bytecode，加入一些用于计数的statement，从而得出之后的代码是否被执行了。

(这种做法称为 add instruments to bytecode)

**jcoverage works by modifying your Java at the bytecode level. It also works against code that you do not have source code for.**

<http://www.jcoverage.com/>

和ASM<http://asm.ow2.org/>

**ASM is an all purpose Java bytecode manipulation and analysis framework. It can be used to modify existing classes or dynamically generate classes, directly in binary form.**

Cobertura uses ASM to modify your bytecode. There are a few otherpopular programs that use ASM; Groovy and Hibernate, to name two.You could have problems if Cobertura uses a different version ofasm and you add both versions to your classpath.

Cobertura only uses ASM when instrumenting. Cobertura *does notneed ASM in your classpath when running tests*.

**如何在Maven中安装及运行**

<http://java-tutorial.ch/software-testing/maven-cobertura>

**一个简单应用的例子**

Cobertura不仅仅可以方便的显示UT cases的代码覆盖率，也可以用来计算其他测试的代码覆盖率。

例如，现在有一个Integration Test 程序叫itf-app，我们希望得知这个集成测试程序对源程序app的代码覆盖率。

1，首先，将 instrumentation加入源程序app的classes。  
  
Cobertura works by inserting instrumentation instructions directly into your compiled Java classes. When these instructions are encountered by the Java Virtual Machine, the inserted code increments various counters so that it is possible to tell which instructions have been encountered and which have not.  
  
You instruct Ant to create instrumented versions of your classes using the instrument task. The example below assumes your classes are in the directory build/classes.  
  
< cobertura-instrument todir="build/instrumented-classes">  
<fileset dir="build/classes">  
<include name="\*\*/\*.class"/>  
</fileset>  
< /cobertura-instrument>

2，之后，使用被注入过的classes。

方式就是在itf-app运行时，在itf-app调用真正的app的class前，让它去调用被注入过了classes。  
  
Simply include a classpath entry for the instrumented classes before any reference to the original classes. This will ensure that the instrumented classes are loaded in preference to the original classes.  
  
< itf-app fork="yes">  
<classpath location="${build.instrumented.dir}"/>  
<classpath location="${build.classes.dir}"/>  
...  
< /itf-app>  
  
The instrumented classes found in ${build.instrumented.dir} will be loaded before those found in ${build.classes.dir} ensuring that the instrumented classes are used by JUnit.  
  
An instrumented class serializes information into the file **cobertura.ser**. Any existing information found in this file will be merged with the current information. In this way the instrumentation for several sessions of a running program can be merged together, producing a single coverage report. For example, the instrumentation from unit and functional tests can be merged together to produce a single coverage report.

此时运行itf-app，就会产生一个文件 **cobertura.ser**。该文件存储了代码被调用的情况。Cobertura就是根据该文件来生产报表的。

3，因此最后，找到这个文件，生成报表。

An instrumented class serializes coverage information to thefile cobertura.ser. Using the report tag,Cobertura can generate coverage reportsin eitherHTML orXML format.

The default format for a Cobertura report is HTML.  
  
< target name="coverage">  
<cobertura-report srcdir="${src.dir}" destdir="${build.coverage.dir}"/>  
< /target>

如果想生成XML报表，或者希望merge多个报表时，参考原文：

http://cobertura.sourceforge.net/introduction.html