# 代码覆盖率

## 1.代码覆盖率工具比较

用过几款java单体覆盖率工具。稍稍列举一下：

**Jcoverage**

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

优点：1 支持ANT TASK

2 支持Eclipse Plugin

缺点 1 商用软件，要money的

2 貌似以前instrument内存管理的很郁闷，轻松就会outofmemory

**Emma**

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

优点：1 开源软件

2 支持ANT TASK

缺点： ANT TASK的参数指定较弱，coverage参数需要追加配置文件。

**Cobertura**

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

优点：1 开源软件

2 支持MARVEN

缺点： 可定制参数太少

从文件的输出上来看，还是以xml,html生成为主。

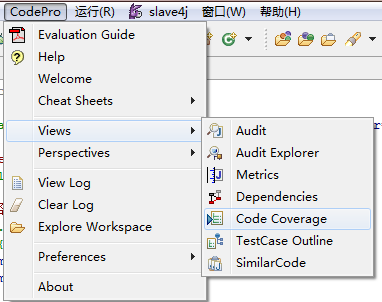
## 2. Codepro analytiX 代码覆盖率

安装eclipse插件，在这里不细说了。更新地址：

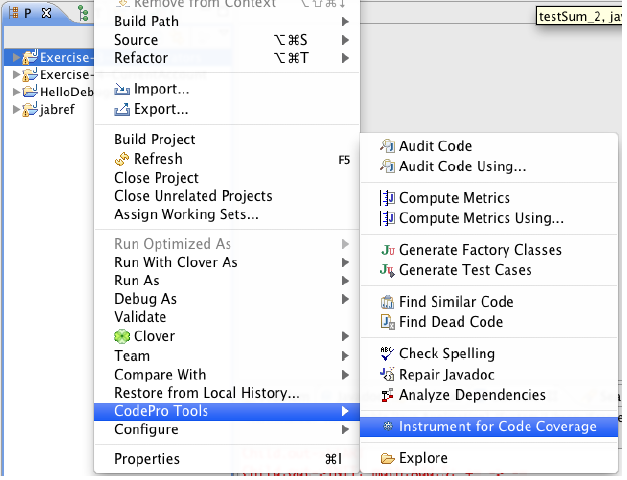
CodePro AnalytiX - <http://dl.google.com/eclipse/inst/codepro/latest/3.6>

可以参考Eclipse 常用插件图解汇总.docx 文档

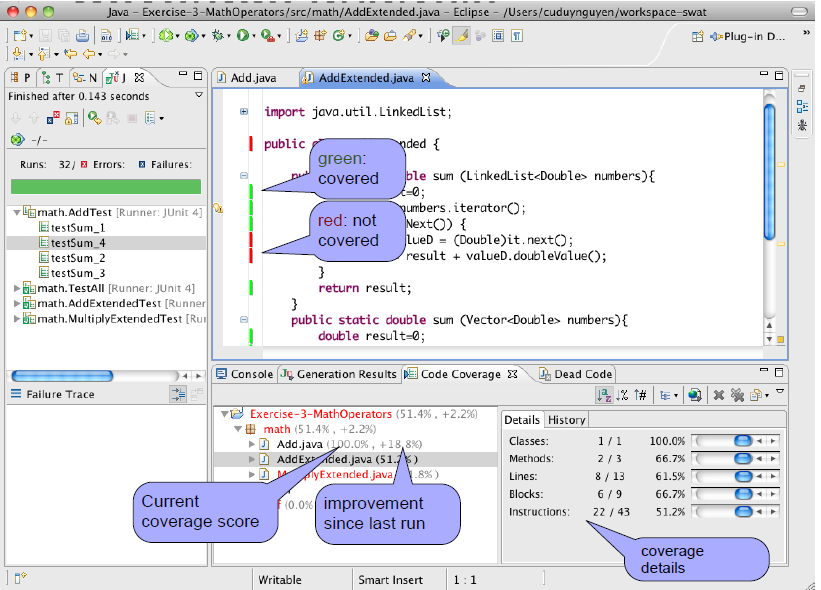
在菜单选择CodePro>Views>Code Coverage 选择统计结果视图



在项目上点击鼠标右键，CodePro Tools>Instrument for Code Coverage 如下图所示



正常运行测试用例，在Code Coverage 视图会出现结果



使用CodePro AnalytiX的优点：是Eclipse的集成插件，不需要想Cobertura建立项目，简单容易上手，对提高代码测试覆盖率节省了事件。

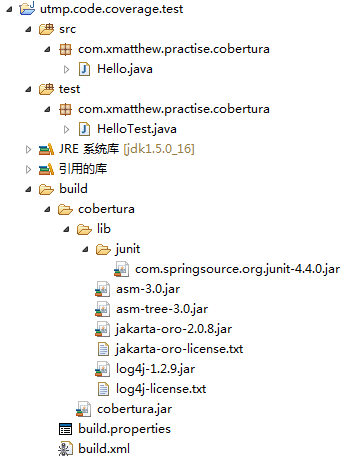
缺点：无法过滤类，在报告中包含了整个测试项目的测试点类。因此生成报告需要借助于Cobertura来完成。

总结：使用 CodePro AnalytiX来辅助编写测试用例，达到质量目标。中创中间件的质量目标是：行覆盖率90%，分支覆盖率75%。然后，使用Cobertura生成测试用例的报告。

## 3.Cobertura例子

首先创建一个Java项目，项目名：utmp.code.coverage.test

结构如下图所示：



接下，创建一个Ant构建配置文件，帮我们实现自动化的编译，单元测试以及cobertura来测量单元测试的代码覆盖情况。

Ant的 build.xml文件内容如下：

增加测试点，在红色1处添加，增加测试用例在红色2处添加。

<?xml version="1.0" encoding="UTF-8" ?>

<project name="cobertura.utmp.code.coverage" default="coverage" basedir=".">

<!-- 引用 build.properties文件 配置路径信息-->

<property file="build.properties" />

<!-- 设置 cobertura路径 -->

<path id="cobertura.classpath">

<fileset dir="${cobertura.dir}">

<include name="cobertura.jar" />

<include name="lib/\*\*/\*.jar" />

</fileset>

</path>

<!-- 项目依赖的jar包 -->

<!--path id="project.classpath">

<fileset dir="${project.lib.dir}">

<include name="\*.jar" />

</fileset>

</path -->

<!-- 配置 cobatura ant 扩展任务 -->

<taskdef classpathref="cobertura.classpath" resource="tasks.properties" />

<!-- 创建编译结果目录 -->

<target name="init">

<mkdir dir="${classes.dir}" />

<mkdir dir="${instrumented.dir}" />

<mkdir dir="${reports.xml.dir}" />

<mkdir dir="${reports.html.dir}" />

<mkdir dir="${coverage.xml.dir}" />

<mkdir dir="${coverage.html.dir}" />

</target>

<!-- 编译源代码 -->

<target name="compile" depends="init">

<javac srcdir="${src.dir}:${testsrc.dir}" destdir="${classes.dir}" debug="true">

<!-- classpath refid="project.classpath" / -->

<classpath refid="cobertura.classpath" />

</javac>

</target>

<target name="instrument" depends="init,compile">

<!--Remove the coverage data file and any old instrumentation.-->

<delete file="cobertura.ser" />

<delete dir="${instrumented.dir}" />

<!-- Instrument the application classes,

writing the instrumented classes into ${build.instrumented.dir}.

-->

<cobertura-instrument todir="${instrumented.dir}">

<!-- The following line causes instrument to ignore any source line

containing a reference to log4j,

for the purposes of coverage reporting.

-->

<!-- 排除的类 -->

<ignore regex="org.apache.log4j.\*" />

<fileset dir="${classes.dir}">

<!-- Instrument all the application classes,

but don't instrument the test classes.

-->

<!-- 包含的类 -->

<include name="\*\*/\*.class" />

**<!-- 1.添加测试点类，可以参考下面的写法 而不需要上面 \*\*/\*.class -->**

<!-- include name="com/cvicse/inforguard/sp/userman/service/impl/UserManImpl.class" / -->

<!-- 排除的类 -->

<exclude name="\*\*/\*Test.class" />

</fileset>

</cobertura-instrument>

</target>

<!-- 单元测试 -->

<target name="test" depends="init,compile">

<junit fork="yes" dir="${basedir}" failureProperty="test.failed">

<!--

Note the classpath order: instrumented classes are before the

original (uninstrumented) classes. This is important.

-->

<classpath location="${instrumented.dir}" />

<classpath location="${classes.dir}" />

<!—

The instrumented classes reference classes used by the

Cobertura runtime, so Cobertura and its dependencies

must be on your classpath.

-->

<classpath refid="cobertura.classpath" />

<!-- classpath refid="project.classpath" / -->

<formatter type="xml" />

<test name="${testcase}" todir="${reports.xml.dir}" if="testcase" />

<batchtest todir="${reports.xml.dir}" unless="testcase">

<fileset dir="${testsrc.dir}">

<include name="\*\*/\*Test.java" />

**<!-- 2.增加测试用例类，可以参考下面的例子写 -->**

<!--include name="com/cvicse/inforguard/sp/userman/service/impl/UserManImplTest.java" / -->

</fileset>

</batchtest>

</junit>

<junitreport todir="${reports.xml.dir}">

<fileset dir="${reports.xml.dir}">

<include name="TEST-\*.xml" />

</fileset>

<report format="frames" todir="${reports.html.dir}" />

</junitreport>

</target>

<target name="coverage-check">

<cobertura-check branchrate="34" totallinerate="100" />

</target>

<!-- 生成 coverage xml格式报告 -->

<target name="coverage-report">

<!--

Generate an XML file containing the coverage data using

the "srcdir" attribute.

-->

<cobertura-report srcdir="${src.dir}" destdir="${coverage.xml.dir}" format="xml" />

</target>

<!-- 生成 coverage html格式报告 -->

<target name="alternate-coverage-report">

<!-- Generate a series of HTML files containing the coverage

data in a user-readable form using nested source filesets.

-->

<cobertura-report destdir="${coverage.html.dir}">

<fileset dir="${src.dir}">

<include name="\*\*/\*.java" />

</fileset>

</cobertura-report>

</target>

<!-- 清除编译生成的结果文件 -->

<target name="clean" description="Remove all files created by the build/test process.">

<delete dir="${classes.dir}" />

<delete dir="${instrumented.dir}" />

<delete dir="${reports.dir}" />

<delete file="cobertura.log" />

<delete file="cobertura.ser" />

</target>

<target name="coverage" depends="clean,init,compile,instrument,test,coverage-report,alternate-coverage-report" description="Compile, instrument ourself, run the tests and generate JUnit and coverage reports." />

</project>

build.properties文件：

#The source code for the examples can be found in this directory

src.dir=../src

testsrc.dir=../test

#Test items to be dependent on the path of jar package

#project.lib.dir=E:/workspace 7/sp\_common\lib

#The path to cobertura.jar

cobertura.dir=cobertura

#Classes generated by the javac compiler are deposited in this directory

#classes.dir=../bin

classes.dir=classes

#Instrumented classes are deposited into this directory

instrumented.dir=instrumented

#All reports go into this directory

reports.dir=reports

#Unit test reports from JUnit are deposited into this directory

reports.xml.dir=${reports.dir}/junit-xml

reports.html.dir=${reports.dir}/junit-html

#Coverage reports are deposited into these directories

coverage.xml.dir=${reports.dir}/cobertura-xml

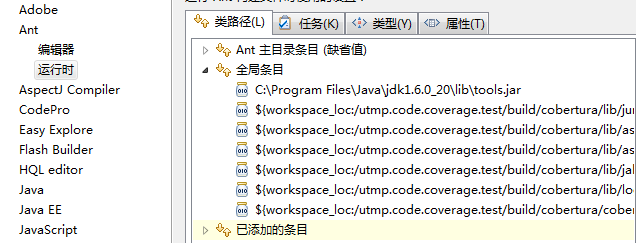
coverage.html.dir=${reports.dir}/cobertura-html

coverage.summaryxml.dir=${reports.dir}/cobertura-summary-xml

测试点Java类和测试用例不在下面贴出来了。请参考例子中的代码

接下来，我们运行 Cobertura来进行测试。

运行 Cobertura,测试在Eclipse运行时，要在Ant插件的运行类库环境中加入cobertura相关类库。



新增cobertura的类库