

developerWorks®

技术主题

软件下载

社区

技术讲座

# 使用 EMMA 测量测试覆盖率

[梁衍轩](#) ([liangyx@cn.ibm.com](mailto:liangyx@cn.ibm.com)), 软件工程师, IBM 中国开发中心上海全球化实验室


[钱建平](#) ([qjianping02@hotmail.com](mailto:qjianping02@hotmail.com)), 学生, 南京大学软件学院

简介: 本文主要通过一个示例项目介绍如何在集成了 **Ant** 和 **Junit** 的基础上, 利用 **EMMA** 来收集单元测试对代码的覆盖率。

发布日期: 2006 年 6 月 15 日

级别: 初级

访问情况 2513 次浏览

建议: 

 标记本文!

## 介绍测试代码覆盖率的重要性

测试驱动开发 (TDD) 是极限编程的一个重要特点, 它具有很多优点, 并被越来越多的开发人员所接受。在测试驱动开发过程中, 程序员经历了编写测试用例, 实现功能, 重构代码这个不断迭代的过程。实践证明, 这个过程能显著提高我们的生产效率, 并产生高质量的代码。它还能给我们以自信, 让我们放心的重构自己的代码。

测试代码确实能够保证代码的质量, 但如果你以为自己已经写了一堆测试用例, 并都能运行通过时, 就能高枕无忧了, 那么你错了。隐藏的 **Bug** 也许只是在等待时机让你的系统崩溃。这是什么原因呢? 聪明的你肯定已经想到, 测试代码是用来保证功能代码的质量的, 但测试代码的质量如何, 我们不得而知。我们需要知道, 我们辛苦编写的测试代码到底覆盖了多少功能代码, 这就是我写这篇文章的出发点, 我将介绍一种测试代码覆盖率的工具 - **EMMA**。

内容
<ul style="list-style-type: none"><li>介绍测试代码覆盖率的重要性</li><li>介绍 <b>EMMA</b></li><li>示例项目</li><li>显示报告</li><li>隐藏在报告背后的问题</li><li>结束语</li><li>下载</li><li>参考资料</li><li>作者简介</li><li>建议</li></ul>

 [回页首](#)

## 介绍 EMMA

**EMMA** 是一个用于检测和报告 **JAVA** 代码覆盖率的开源工具。它不但能很好的用于小型项目, 很方便得出覆盖率报告, 而且适用于大型企业级别的项目。

**EMMA** 有许多优点, 首先你能免费得到它, 并把它用于自己项目的开发。它支持许多种级别的覆盖率指标: 包, 类, 方法, 语句块 (**basic block**) 和行, 特别是它能测出某一行是否只是被部分覆盖, 如条件语句短路的情况。它能生成 **text**, **xml**, **html** 等形式的报告, 以满足不同的需求, 其 **html** 报告提供下钻功能, 我们能够从 **package** 开始一步步链接到我们所关注的某个方法。**EMMA** 能和 **Makefile** 和 **Ant** 集成, 便于应用于大型项目。特别还须指出的一点是, **EMMA** 的效率很高, 这对于大型项目来说很重要。

**EMMA** 是通过向 **.class** 文件中插入字节码的方式来跟踪记录被运行代码信息的。**EMMA** 支持两种模式: **On the fly** 和 **Offline** 模式。

**On the fly** 模式往加载的类中加入字节码, 相当于用 **EMMA** 实现的 **application class loader** 替代原来的 **application class loader**。

**Offline** 模式在类被加载前, 加入字节码。

**On the fly** 模式比较方便, 缺点也比较明显, 如它不能为被 **boot class loader** 加载的类生成覆盖率报告, 也不能为像 **J2EE** 容器那种自己有独特 **class loader** 的类生成覆盖率报告。这时, 我们能求助于 **Offline** 模式。

**EMMA** 也支持两种运行方式: **Command line** 和 **Ant**。

命令行一般和 **On the fly** 模式一起适用, 对于简单的项目能够快速产生覆盖率报告。通过 **Ant task** 来运行 **EMMA** 的话, 特别适用于大型的项目。

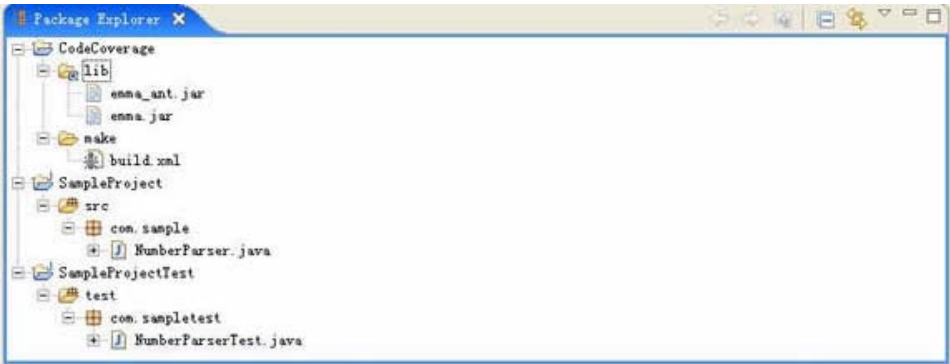
本文后面提供的实例主要是演示如何集成 **EMMA** 和 **Ant**, 通过 **Offline** 模式产生覆盖率报告。

 [回页首](#)

## 示例项目

示例工程 **SampleProject** 是个小型的项目, 有一个类 **NumberParser**, 主要功能是把一个字符串解析成 **float** 型。下面是整个工程的目录结构。

图1. 示例项目的目录结构



下面，我们开始来为我们的工程编写 Ant 脚本。

清单1设置一些属性，包括源文件，二进制文件，JUnit 报告，覆盖率报告等的路径

```
<!-- 设置Java类被注入字节码后存放的路径-->
<property name="bin.instrument.dir" location="../instrbin" />
<!-- 设置覆盖率元数据和报告的路径-->
<property name="coverage.dir" location="../coverage" />
<!-- 设置junit报告的路径 -->
<property name="junitReport.dir" location="../junitReport" />
<!-- 设置主题代码bin路径-->
<property name="bin.main.dir" location="../srcbin" />
<!-- 设置测试代码bin路径-->
<property name="bin.test.dir" location="../testbin" />
<!-- 设置主题代码源路径-->
<property name="src.main.dir" location="../../SampleProject/src" />
<!-- 设置测试代码源路径-->
<property name="src.test.dir" location="../../SampleProjectTest/test" />
<!-- 指示需要注入字节码的Java类的路径-->
<path id="classpath.main">
    <pathelement location="{bin.main.dir}" />
</path>
<!-- 指示 emma.jar 和emma_ant.jar 的路径-->
<path id="emma.lib">
    <pathelement location="{libs}/emma.jar" />
    <pathelement location="{libs}/emma_ant.jar" />
</path>
<!-- 允许emma-->
<property name="emma.enabled" value="true" />
```

其中目录`{bin.instrument.dir}`存放被注入字节码的类，"emma.lib" 指向 emma 资源所在的位置。

清单2为 ANT 定义 EMMA 任务

```
<!-- 为ANT添加EMMA任务-->
<taskdef resource="emma_ant.properties" classpathref="emma.lib" />
```

清单3编译源代码和测试代码

```
<target name="compile-src.main">
    <mkdir dir="{bin.main.dir}" />
    <javac destdir="{bin.main.dir}" debug="on">
        <src path="{src.main.dir}" />
    </javac>
    <copy todir="{bin.main.dir}">
        <fileset dir="{src.main.dir}">
            <exclude name="**/*.java" />
        </fileset>
    </copy>
</target>

<target name="compile-src.test">
    <mkdir dir="{bin.test.dir}" />
    <javac destdir="{bin.test.dir}" debug="on">
        <src path="{src.test.dir}" />
        <classpath location="{bin.main.dir}" />
    </javac>
    <copy todir="{bin.test.dir}">
```

```
        <fileset dir="${src.test.dir}">
            <exclude name="**/*.java" />
        </fileset>
    </copy>
</target>
```

编译分两阶段，先编译源代码，然后再编译测试用例代码。

清单4 在所要测试类的代码中插入字节码

```
<!-- 对编译在路径bin.main.dir中的Java类注入字节码，
并且把注入字节码的新Java类存放到路径bin.instrument.dir-->
<!-- 覆盖率的元数据存放在路径coverage.dir中-->
<target name="instrument">
    <mkdir dir="${bin.instrument.dir}" />
    <mkdir dir="${coverage.dir}" />
    <emma enabled="${emma.enabled}">
        <instr instrpathref="classpath.main"
            destdir="${bin.instrument.dir}"
            metadatafile="${coverage.dir}/metadata.emma"
            merge="true">

        </instr>
    </emma>
    <copy todir="${bin.instrument.dir}">
        <fileset dir="${bin.main.dir}">
            <exclude name="**/*.java" />
        </fileset>
    </copy>
</target>
```

当\${emma.enabled}为 true 时，才生成插入字节码的类。<instr>中指定了要 instrument 的类的地址，instrumented 后类存放的地址，以及 metadata 存放的地址。

清单5 运行测试用例，得到一些生成报告的元数据

```
<!-- 执行测试用例同时生成JUnit测试报告和emma代码覆盖率报告-->
<target name="test">
    <mkdir dir="${junitReport.dir}" />
    <junit fork="true" forkmode="once"
        printsummary="withOutAndErr"
        errorproperty="test.error"
        showoutput="on">
        <!-- 指明代码覆盖率的元数据的存放位置-->
        <jvmarg
            value="- Demma.coverage.out.file=${coverage.dir}/metadata.emma" />
        <jvmarg value="- Demma.coverage.out.merge=true" />
        <classpath location="${bin.instrument.dir}" />
        <classpath location="${bin.test.dir}" />
        <classpath refid="emma.lib" />

        <formatter type="xml" />
        <!-- 执行所有以Test结尾的JUnit测试用例-->
        <batchtest todir="${junitReport.dir}" haltonfailure="no">
            <fileset dir="${bin.test.dir}">
                <include name="**/*Test.class" />
            </fileset>
        </batchtest>
    </junit>
</target>
```

在运行测试用例前，需要设置 jvmarg。所有的测试用例都跑在 instrumented 的类上面。

清单6 生成 JUnit 报告

```
<target name="gen-report-junit">
    <!-- 生成JUnit测试报告-->
    <junitreport todir="${junitReport.dir}">
        <fileset dir="${junitReport.dir}">
            <include name="*" />
        </fileset>
        <report format="frames" todir="${junitReport.dir}" />
    </junitreport>
```

```
</target>
```

清单7 生成覆盖率报告

```
<!-- 生成代码覆盖率报告 -->
<target name="gen-report-coverage">
  <!-- 如果属性emma.enabled的值是true，就生成代码覆盖率报告 -->
  <emma enabled="${emma.enabled}">
    <report sourcepath="${src.main.dir}"
      sort="+block,+name,+method,+class"
      metrics="method: 70, block: 80, line: 80, class: 100">
      <fileset dir="${coverage.dir}">
        <include name="*.emma" />
      </fileset>
      <html outfile="${coverage.dir}/coverage.html"
        depth="method" columns="name, class, method, block, line" />
    </report>
  </emma>
</target>
```

<report>中 sourcepath 指明源代码所在的位置，以便能够显示每行代码的覆盖情况。Sort指明生成列表的排列顺序，"+"表示升序， "-"表示降序。Metrics 可为每个度量指明一个覆盖率阈值，若未达到该阈值，则该行会被标记出来（前提是报告的形式支持这个功能，如 HTML）。<html>指明以 HTML 形式生成报告，Depth 指明报告的详细程度，columns 指明生成列表列名的排列顺序。

 [回页首](#)

## 显示报告

我们已经写好了Ant脚本，接下来你就可以运行该脚本了。这里假设你已经搭好了运行 Ant 和 JUnit 的环境，直接到脚本所在目录，在命令行敲入 Ant 即可。

下面是各个层次的报告：

图2 整个项目层次的报告

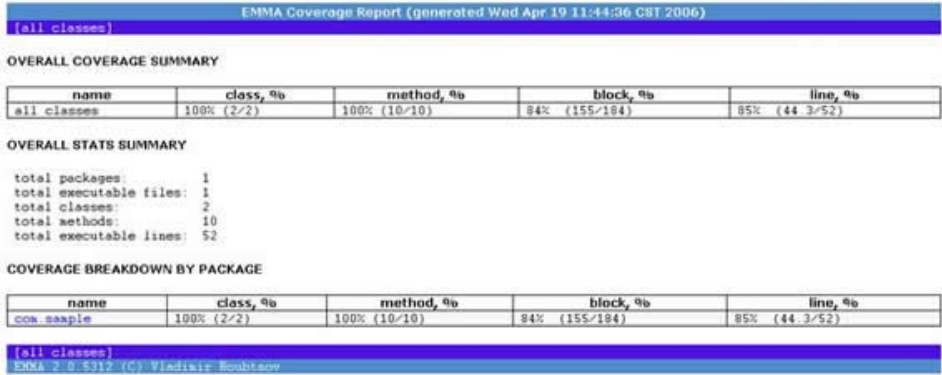


图3 包层次的报告

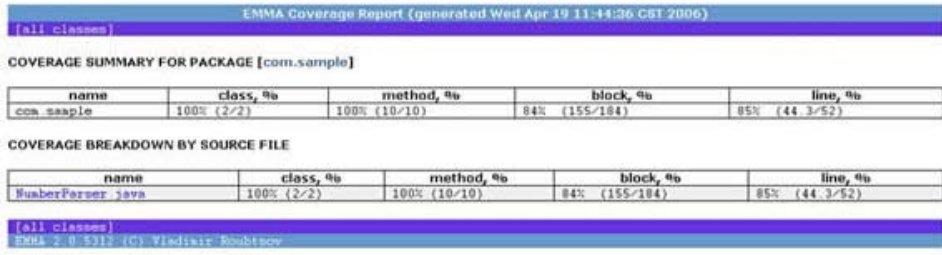


图4 类层次的报告

EMMA Coverage Report (generated Wed Apr 19 11:44:36 CST 2006)				
[all classes][com.example]				
COVERAGE SUMMARY FOR SOURCE FILE [NumberParser.java]				
name	class, %	method, %	block, %	line, %
NumberParser.java	100% (2/2)	100% (10/10)	84% (155/184)	85% (44.3/52)
COVERAGE BREAKDOWN BY CLASS AND METHOD				
name	class, %	method, %	block, %	line, %
class NumberParser	100% (1/1)	100% (6/6)	79% (112/141)	81% (32.3/40)
getFraction (StringIterator): float		100% (1/1)	31% (9/29)	44% (4/9)
getSign (StringIterator): void		100% (1/1)	83% (19/23)	82% (7.4/9)
getInteger (StringIterator): int		100% (1/1)	92% (24/26)	89% (8/9)
parse (String): float		100% (1/1)	93% (39/42)	99% (7.9/8)
NumberParser (): void		100% (1/1)	100% (6/6)	100% (2/2)
parseSingleNumber (char): int		100% (1/1)	100% (15/15)	100% (3/3)
class StringIterator	100% (1/1)	100% (4/4)	100% (43/43)	100% (12/12)
StringIterator (String): void		100% (1/1)	100% (9/9)	100% (4/4)
hasNext (): boolean		100% (1/1)	100% (10/10)	100% (3/3)
next (): char		100% (1/1)	100% (14/14)	100% (2/2)
pushBack (): void		100% (1/1)	100% (10/10)	100% (3/3)

图5 用颜色标记的源代码

```
21 public float parse(String number) {
22     if (number.equals("")||null == number ) {
23         throw new IllegalArgumentException(
24             "Nuber string should not be empty or null");
25     }
26     StringIterator stringIterator = new StringIterator(number);
27     getSign(stringIterator);
28     int integer = getInteger(stringIterator);
29     float fraction = getFraction(stringIterator);
30     float total = integer + fraction;
31     return isPositive ? total : (-1) * total;
32 }
33
34 /**
35  * Gets the sign from StringIterator. If there's a sign in the
36  * StringIterator, the boolean isPositive will be marked
37  *
38  * @param si
39  *     the StringIterator
40  */
41 private void getSign(StringIterator si) {
42     char sign = si.next();
43     if (sign == '+') {
44         isPositive = true;
45     } else if (sign == '-') {
46         isPositive = false;
47     } else {
48         isPositive = true;
49         si.pushBack();
50     }
51 }
```

你会发现有三种颜色，绿色，红色和黄色，它们分别表示该行：被测试到，未被测试到，以及部分被测试到。红色或黄色的部分是需  
要引起你注意的，bug 也许就隐藏在这部分代码中，你所需做的就是设计一些测试用例，使它们运行以前未被执行到的语句。如上面  
那张图给出了我们一些信息，String 中含有"+"号的情况未被测试到，还有"isPositive"只被测试到 true 或 false 的一种情况，你需要相  
应的增加一些测试用例。运行新加的测试用例，你也许会发现一些新的 bug，并修正这些 bug。

### 隐藏在报告背后的问题

对于这个简单的例子，你会发现，我们很容易达到 100% 的测试覆盖率，你也许会松口气说：啊，我把所有情况都测试到了，这下放心了。  
在这里很遗憾的告诉你，EMMA 的功能是有限的，它不支持决策覆盖和路径覆盖。事实上，对于一个稍复杂的工程进行穷尽的测试是不可能  
的。

清单8 决策覆盖和路径覆盖的代码示例

```
/**
 * Parses the given string to a float number
 *
 * @param number
 *     the given string
 * @return the float number related with the string
 *
 * @throws IllegalArgumentException
 *     if the string is empty, null or can not parse to a float
 */
public float parse(String number) {
    if (number.equals("")||number == null ) {
```

```
        throw new IllegalArgumentException(
            "Number string should not be empty or null");
    }
    StringIterator stringIterator = new StringIterator(number);
    getSign(stringIterator);
    int integer = getInteger(stringIterator);
    float fraction = getFraction(stringIterator);
    float total = integer + fraction;
    return isPositive ? total : (-1) * total;
}
```

清单9 决策覆盖和路径覆盖的测试用例

```
public void test_parse () {
    NumberParser np = new NumberParser();
    String number = "";
    try {
        np.parse(number);
        fail("should throw IAE");
    } catch (IllegalArgumentException e) {
        // pass
    }
    number = "22.010";
    float parsedNumber = np.parse(number);
    assertEquals((float) 22.010, parsedNumber);

    number = "-22.010";
    parsedNumber = np.parse(number);
    assertEquals((float) 22.010, parsedNumber);
}
```

运行 **Ant** 脚本，生成报告，你会发现，测试用例都运行通过了，测试覆盖报告也表明代码所有的行都被执行到了。但细心的读者肯定早已看到上面代码存在 **Bug**。若传进 **parse** 的 **string** 为 **null** 的话，并不是如我们所愿，得到 **IllegalArgumentException**，而是抛出了 **NullPointerException**。

虽然下面那行是绿色的，但它只表明每个条件语句都被执行到了，并不能说明每个条件都取到**true**和**false**两种情况。在我们设计的测试用例中，“**null == number**”只取到 **false** 一种情况。我们需要在我们的测试用例中加入对 **string** 情况是 **null** 的测试。

图6 决策覆盖和路径覆盖率报告

```
| 22|          if (number.equals(""))||null == number ) {
```

清单10 修正代码的 Bug

```
        if (null == number || "".equals(number)) {
```

 [回页首](#)

## 结束语

为你的项目生成覆盖率报告，**EMMA** 是个不错的选择。通过覆盖率报告，我们能发现并修复一些隐藏的 **bug**，我们的软件会变得更强壮。

 [回页首](#)

## 下载

名字	大小	下载方法
SampleProjects.zip	419 K	<a href="#">HTTP</a>

 [关于下载方法的信息](#)

## 参考资料

1. [从 SourceForge 下载 EMMA](#)
2. [得到示例工程。](#)

- 3. Dennis M. Sosnoski 介绍Hansel和Gretel的文章Classworking工具箱: [用Hansel和Gretel覆盖代码](#)。
- 4. Elliotte Rusty Harold 介绍Cobertura的文章 [用Cobertura测量测试覆盖率](#)。
- 5. [Andrew Glover](#)的追求代码质量: 不要被覆盖报告所迷惑让我们看到不能在覆盖报告中显示的问题。
- 6. [Elliotte Rusty Harold](#) 介绍Jester的文章用 [Jester](#) 对测试进行测试。

作者简介

梁衍轩 CSTE，PMP，现就职于 IBM 中国开发中心上海全球化实验室，对 Java SE、全球化技术、Unicode，软件测试已经项目管理有很深的研究。您可以通过[lijangyx@cn.ibm.com](mailto:lijangyx@cn.ibm.com)联系到他。

钱建平 就读于南京大学软件学院，喜欢钻研算法以及Java编程，您可以通过[qjianping02@hotmail.com](mailto:qjianping02@hotmail.com)联系到他。

建议



[🏠 回页首](#)

打印此页面    分享此页面    关注 developerWorks

技术主题

AIX and UNIX  
Information Management  
Lotus  
Rational  
WebSphere  
  
Cloud computing

Java technology  
Linux  
Open source  
SOA and web services  
Web development  
XML  
[更多...](#)

查找软件  
IBM 产品  
评估方式（下载，在线试用，Beta 版，云）  
行业  
  
技术讲座

[社区](#)  
群组  
博客  
Wiki  
文件  
使用条款  
报告滥用  
[更多...](#)

关于 developerWorks  
反馈意见  
在线投稿  
投稿指南  
网站导航  
请求转载内容  
  
相关资源  
ISV 资源 (英语)  
IBM 教育学院教育培养计划

[IBM](#)  
解决方案  
软件  
支持门户  
产品文档  
红皮书 (英语)  
隐私条约  
浏览辅助