06 | 你真的懂测试覆盖率吗?

2018-07-11 茹炳晟

讲述：茹炳晟
时长 12:47 大小 5.86M

在上一篇文章中，我为你介绍了软件测试各个阶段的自动化技术，在前面的文章中我也提到了测试覆盖率的概念，你当时可能有点不明白，那么今天我就和你详细聊聊测试覆盖率这个主题。

测试覆盖率通常被用来衡量测试的充分性和完整性，从广义的角度来讲，测试覆盖率主要分为两大类，一类是面向项目的需求覆盖率，另一类是更偏向技术的代码覆盖率。

需求覆盖率

需求覆盖率是指测试对需求的覆盖程度，通常的做法是将每一条分解后的软件需求和对应的测试建立一对多的映射关系，最终目标是保证测试可以覆盖每个需求，以保证软件产品的质量。

我们通常采用 ALM，Doors 和 TestLink 等需求管理工具来建立需求和测试的对应关系，并以此计算测试覆盖率。

需求覆盖率统计方法属于传统瀑布模型下的软件工程实践，传统瀑布模型追求自上而下地制定计划、分析需求、设计软件、编写代码、测试和运维等，在流程上是重量级的，已经很难适应当今互联网时代下的敏捷开发实践。

所以，互联网测试项目中很少直接基于需求来衡量测试覆盖率，而是将软件需求转换成测试需求，然后基于测试需求再来设计测试点。

因此，现在人们口中的测试覆盖率，通常默认指代码覆盖率，而不是需求覆盖率。

代码覆盖率

简单来说，代码覆盖率是指，至少被执行了一次的条目数占整个条目数的百分比。

如果“条目数”是语句，对应的就是代码行覆盖率；如果“条目数”是函数，对应的就是函数覆盖率；如果“条目数”是路径，那么对应的就是路径覆盖率。依此类推，你就可以得到绝大多数常见的代码覆盖率类型的定义。

这里我给你简单介绍一下最常用的三种代码覆盖率指标。

行覆盖率又称为语句覆盖率，指已经被执行到的语句占总可执行语句（不包含类似 C++ 的头文件声明、代码注释、空行等等）的百分比。这是最常用也是要求最低的覆盖率指标。实际项目中通常会结合判定覆盖率或者条件覆盖率一起使用。

判定覆盖又称分支覆盖，用以度量程序中每一个判定的分支是否都被测试到了，即代码中每个判断的取真分支和取假分支是否各被覆盖至少各一次。比如，对于 if(a>0 && b>0)，就要求覆盖“a>0 && b>0”为 TURE 和 FALSE 各一次。

条件覆盖是指，判定中的每个条件的可能取值至少满足一次，度量判定中的每个条件的结果 TRUE 和 FALSE 是否都被测试到了。比如，对于 if(a>0 && b>0)，就要求“a>0”取 TRUE 和 FALSE 各一次，同时要求“b>0”取 TRUE 和 FALSE 各一次。

代码覆盖率的价值

现在很多项目都在单元测试以及集成测试阶段统计代码覆盖率，但是我想说的是，统计代码覆盖率仅仅是手段，你必须透过现象看到事物的本质，才能从根本上保证软件整体的质量。

统计代码覆盖率的根本目的是找出潜在的遗漏测试用例，并有针对性的进行补充，同时还可以识别出代码中那些由于需求变更等原因造成的不可达的废弃代码。

通常我们希望代码覆盖率越高越好，代码覆盖率越高越能说明你的测试用例设计是充分且完备的，但你也会发现测试的成本会随着代码覆盖率的提高以类似指数级的方式迅速增加。

如果想达到 70% 的代码覆盖率，你可能只需要 30 分钟的时间成本。但如果你想把代码覆盖率提高到 90%，那么为了这额外的 20%，你可能花的时间就远不止 30 分钟了。更进一步，你如果想达到 100% 的代码覆盖率，可想而知你花费的代价就会更大了。

那么，为什么代码覆盖率的提高，需要付出越来越大的代价呢？因为在后期，你需要大量的桩代码、Mock 代码和全局变量的配合来控制执行路径。

所以，在软件企业中，只有单元测试阶段对代码覆盖率有较高的要求。因为从技术实现上讲，单元测试可以最大化地利用打桩技术来提高覆盖率。而你如果想在集成测试或者是 GUI 测试阶段将代码覆盖率提高到一定量级，那你所要付出的代价是巨大的，而且在很多情况下根本就实现不了。

代码覆盖率的局限性

我先来问你一个问题，如果你通过努力，已经把某个函数的 MC/DC 代码覆盖率（MC/DC 覆盖率是最高标准的代码覆盖率指标，除了直接关系人生命安全的软件以外，很少会有项目会有严格的 MC/DC 覆盖率要求）做到了 100%，软件质量是否就真的高枕无忧、万无一失了呢？

很不幸，即使你所设计的测试用例已经达到 100% 的代码覆盖率，软件产品的质量也做不到万无一失。其根本原因在于代码覆盖率的计算是基于现有代码的，并不能发现那些“未考虑某些输入”以及“未处理某些情况”形成的缺陷。

我给你举个极端的例子，如果一个被测函数里面只有一行代码，只要这个函数被调用过了，那么衡量这一行代码质量的所有覆盖率指标都会是 100%，但是这个函数是否真正实现了应该需要实现的功能呢？

显然，代码覆盖率反映的仅仅是已有代码的哪些逻辑被执行过了，哪些逻辑还没有被执行过。以此为依据，你可以补充测试用例，可以去测试那些还没有覆盖到的执行路径。但也是仅此而已，对于那些压根还没有代码实现的部分，基于代码覆盖率的统计指标就无能为力了。

总结来讲，高的代码覆盖率不一定能保证软件的质量，但是低的代码覆盖率一定不能能保证软件的质量。

好了，现在你已经了解了代码覆盖率的概念、价值和局限性，那么接下来，我就以 Java 代码覆盖率工具为例，给你解释一下代码覆盖率工具的内部实现原理以及一些关键技术。

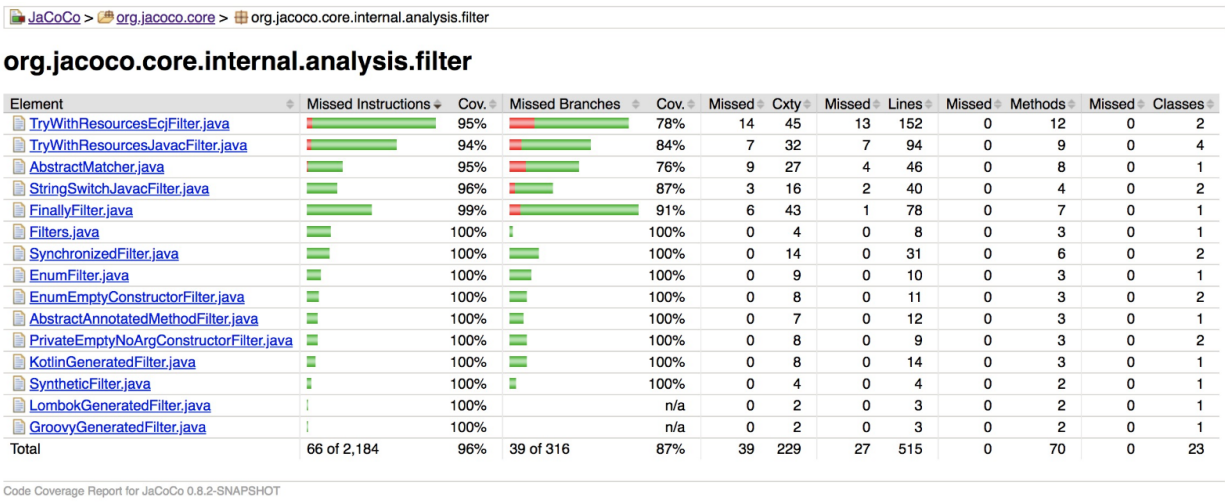
当你理解了这部分内容，以后再面对各个不同开发语言的不同代码覆盖率工具时，就可以做到胸有成竹地根据具体的项目性质，选择最合适的代码覆盖率工具了。

代码覆盖率工具

JaCoCo 是一款 Java 代码的主流开源覆盖率工具，可以很方便地嵌入到 Ant、Maven 中，并且和很多主流的持续集成工具以及代码静态检查工具，比如 Jekins 和 Sonar 等，都有很好的集成。

首先，我先带你看看 JaCoCo 的代码覆盖率报告长什么样子。

如图 1 所示为 JaCoCo 的整体代码覆盖率统计报告，包括了每个 Java 代码文件的行覆盖率以及分支覆盖率统计，并给出了每个 Java 代码文件的行数、方法数和类数等具体信息。


JaCoCo > org.jacoco.core > org.jacoco.core.internal.analysis.filter

org.jacoco.core.internal.analysis.filter

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
TryWithResourcesEcjFilter.java		95%		78%	14	45	13	152	0	12	0	2
TryWithResourcesJavacFilter.java		94%		84%	7	32	7	94	0	9	0	4
AbstractMatcher.java		95%		76%	9	27	4	46	0	8	0	1
StringSwitchJavacFilter.java		96%		87%	3	16	2	40	0	4	0	2
FinallyFilter.java		99%		91%	6	43	1	78	0	7	0	1
Filters.java		100%		100%	0	4	0	8	0	3	0	1
SynchronizedFilter.java		100%		100%	0	14	0	31	0	6	0	2
EnumFilter.java		100%		100%	0	9	0	10	0	3	0	1
EnumEmptyConstructorFilter.java		100%		100%	0	8	0	11	0	3	0	2
AbstractAnnotatedMethodFilter.java		100%		100%	0	7	0	12	0	3	0	1
PrivateEmptyNoArgConstructorFilter.java		100%		100%	0	8	0	9	0	3	0	2
KotlinGeneratedFilter.java		100%		100%	0	8	0	14	0	3	0	1
SyntheticFilter.java		100%		100%	0	4	0	4	0	2	0	1
LombokGeneratedFilter.java		100%		n/a	0	2	0	3	0	2	0	1
GroovyGeneratedFilter.java		100%		n/a	0	2	0	3	0	2	0	1
Total	66 of 2,184	96%	39 of 316	87%	39	229	27	515	0	70	0	23

Code Coverage Report for JaCoCo 0.8.2-SNAPSHOT

图 1 JaCoCo 代码覆盖率统计报告实例

如图 2 所示为每个 Java 文件内部详细的代码覆盖率情况，图中绿色的行表示已经被覆盖，红色的行表示尚未被覆盖，黄色的行表示部分覆盖；左侧绿色菱形块表示该分支已经被完全覆盖、黄色菱形块表示该分支仅被部分覆盖。

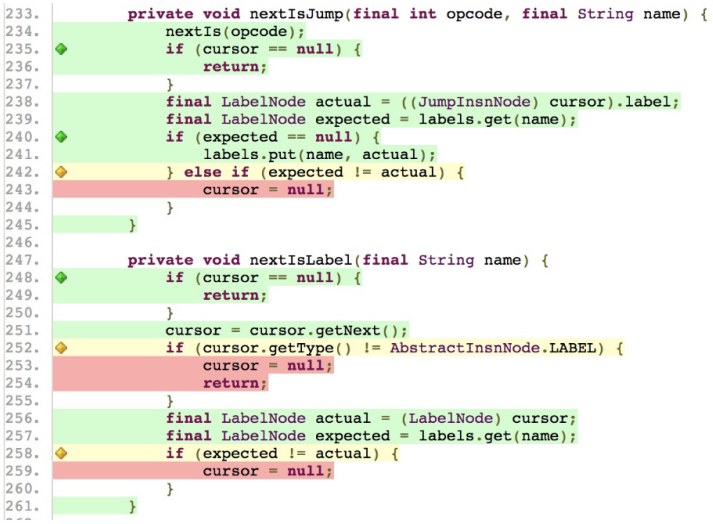


图 2 JaCoCo 详细代码覆盖率实例

显然，通过这个详尽的报告，你就可以知道代码真实的执行情况、哪些代码未被覆盖。以此为基础，你再去设计测试用例就会更有针对性了。

代码覆盖率工具的实现原理

JaCoCo 的详细报告，让你惊叹于代码覆盖率工具的强大。但你有没有仔细想过，这样的统计信息如何被获取到的呢?

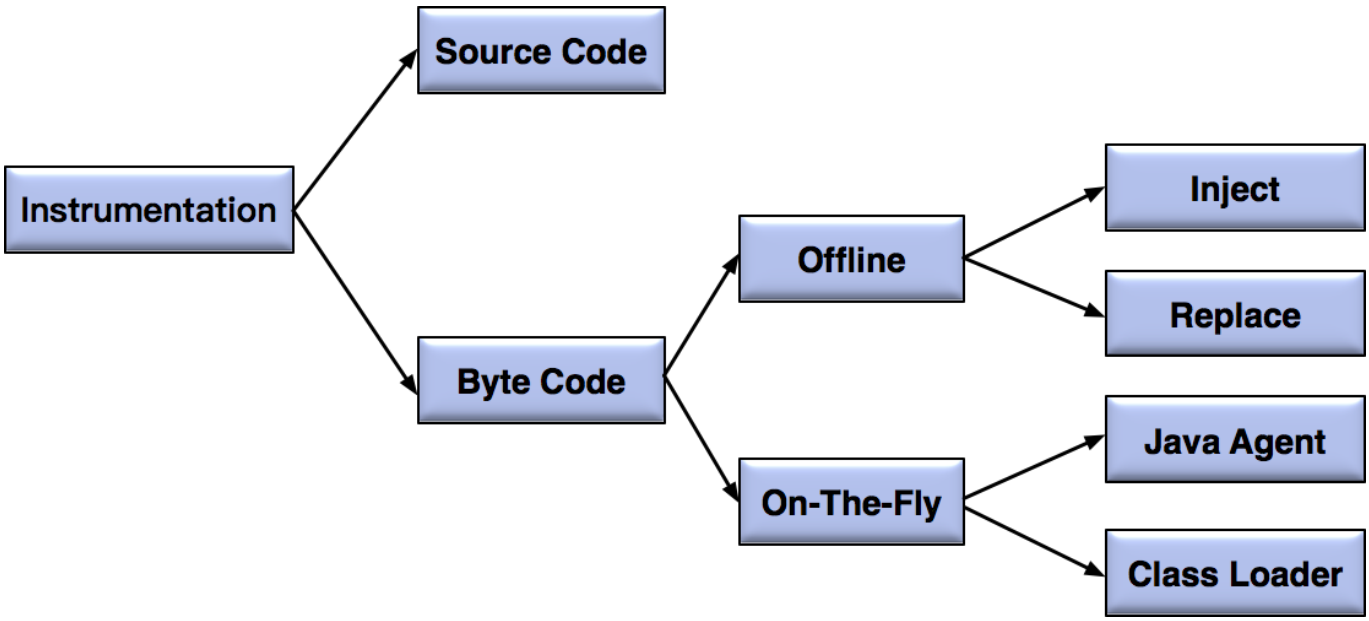


图 3 统计代码覆盖率的不同注入实现技术

实现代码覆盖率的统计，最基本的方法就是注入（Instrumentation）。简单地说，注入就是在被测代码中自动插入用于覆盖率统计的探针（Probe）代码，并保证插入的探针代码不会给原代码带来任何影响。

对于 Java 代码来讲，根据注入目标的不同，可以分为源代码（Source Code）注入和字节码（Byte Code）注入两大类。基于 JVM 本身特性以及执行效率的原因，目前主流的工具基本都是使用字节码注入，注入的具体实现采用 ASM 技术。

ASM 是一个 Java 字节码操纵框架，能被用来动态生成类或者增强既有类的功能，可以直接产生 class 文件，也可以在类被加载入 JVM 之前动态改变类行为。

根据注入发生的时间点，字节码注入又可以分为两大模式：On-The-Fly 注入模式和 Offline 注入模式。

第一，On-The-Fly 注入模式

On-The-Fly 模式的特点在于无需修改源代码，也无需提前进行字节码插桩。它适用于支持 Java Agent 的运行环境。

这样做的优点是，可以在系统不停机的情况下，实时收集代码覆盖率信息。缺点是运行环境必须允许使用 Java Agent。

实现 On-The-Fly 模式，主要有两种技术方案：

1. 开发自定义的类装载器（Class Loader）实现类装载策略，每次类加载前，需要在 class 文件中插入探针，早期的 Emma 就是使用这种方案实现的探针插入；
2. 借助 Java Agent，利用执行在 main() 方法之前的拦截器方法 premain() 来插入探针，实际使用过程中需要在 JVM 的启动参数中添加“-javaagent”并指定用于实时字节码注入的代理程序，这样代理程序在装载每个 class 文件前，先判断是否已经插入了探针，如果没有则需要将探针插入 class 文件中，目前主流的 JaCoCo 就是使用了这个方式。

第二，Offline 注入模式

Offline 模式也无需修改源代码，但是需要在测试开始之前先对文件进行插桩，并事先生成插过桩的 class 文件。它适用于不支持 Java Agent 的运行环境，以及无法使用自定义类装载器的场景。

这样做的优点是，JVM 启动时不再需要使用 Java Agent 额外开启代理，缺点是无法实时获取代码覆盖率信息，只能在系统停机时下获取。

Offline 模式根据是生成新的 class 文件还是直接修改原 class 文件，又可以分为 Replace 和 Inject 两种不同模式。

和 On-The-Fly 注入模式不同，Replace 和 Inject 的实现是，在测试运行前就已经通过 ASM 将探针插入了 class 文件，而在测试的运行过程中不需要任何额外的处理。
Cobertura 就是使用 Offline 模式的典型代表。

总结

测试覆盖率通常被用来衡量测试的充分性和完整性，包括面向项目的需求覆盖率和更偏向技术的代码覆盖率。而需求覆盖率的统计方式不再适用于现在的敏捷开发模式，所以现在谈到测试覆盖率，大多是指代码覆盖率。

但是，高的代码覆盖率不一定能保证软件的质量，因为代码覆盖率是基于现有代码，无法发现那些“未考虑某些输入”以及“未处理某些情况”形成的缺陷。

另外，对于代码覆盖率的统计工具，我希望你不仅仅是会用的层次，而是能够理解它们的原理，知其然知其所以然，才能更好地利用这些工具完成你的测试工作。

思考题

你在实际工作中，是否还接触过 C/C++，JavaScript 等语言的代码覆盖率工具，比如 GCC Coverage、JSCoverage 和 Istanbul 等？如果接触过的话，请你谈谈自己使用的感受以及遇到过的“坑”。

欢迎你给我留言。

上一篇 05 | 你知道软件开发各阶段都有哪些自动化测试技术吗？

下一篇 07 | 如何高效填写软件缺陷报告？

精选留言 (48)

写留言

海罗沃德

2018-07-20

4

实际项目中，无论覆盖率多高，没有根据需求正确的写assert其实也是无法利用测试用例发现bug，提高代码质量，在实际的测试用例中，正向的case一般比较容易写，难得是测试

error handling和模拟各种异常情况下的代码行为

展开

作者回复: 你的两个观点都非常正确👍

lucky_zi...
2018-07-15

2

安全产品嵌入式C的单元测试对覆盖率要求很高，语句，分支，MCDC都要达到100%的覆盖率要求，单元测试工具一般会有统计要求，比如RTRT能统计一部分覆盖率，但MCDC没法统计；遇到的问题其实还是一个业界普遍的问题，就是覆盖率达到了，问题却发现的少，没有代码走查发现的多，但为完成覆盖率却耗时很多，所以包括公司领导在内的很多同事都认为覆盖率意义不大，觉得单元测试没有意义。这种局面貌似国内普遍现象，老...

展开

作者回复: 一看就知道你也是这个领域有过很多经验的业内人士，代码走查的确是个不错的时间，尤其是敏捷模式推崇的结对编程，但是走查往往只能发现静态或者一部分动态问题，而且走查的效果很大程度取决于个人的能力，不能系统化和体系化，代码覆盖率的局限性主要是不能发现有需求但是代码没有实现的场景，但是对于已有代码还是具有很高参考价值的，至少知道你那些分支和语句覆盖到了，另外mcdc现在只是在人生命相关的软件和系统中才会采用，很才有其它企业会用这个指标

仡
2018-07-12

2

听到这一期，很多技术都是JAVA相关的～目前正在学习Python，Python自动化框架有什么好的建议么？谢谢～

作者回复: 其实更多的还是框架的思想，至于是用什么语言来实现都是可以的，最近基于python的自动化框架比较流行，主要是python自己的方便性

sylan215
2019-02-12

1

1.我们目前的业务主要是 C++ 代码，曾经有测试开发调研过一些代码覆盖率工具，但是由于代码量庞大，且和目前业务场景不符，导致没有后续；

2.非常同意茹老师说的「高的代码覆盖率不一定能保证软件的质量，但是低的代码覆盖率一定不能保证软件的质量」，所以我们一直坚持的理念是，注重效果，能够保证质量的手段就是好手段；...

展开

Jia 1

2018-08-28

还有一个很重要的覆盖率，Mutation测试覆盖率。主要是用来测试Test case的质量，而不是代码本身的质量。一个Test case suite质量高的话，对于代码出现的人为变异（可以理解为错误代码）都能很好的检测出来。

展开

Struggling 1

2018-07-18

打卡～～之前的公司单元测试都是开发写，不过我也帮开发写过单元测试，正如老师所说单元测试覆盖率达到一定程度想要再提升非常难，而且并不是覆盖率越高就表明质量就没问题，个人觉得如果能从需求功能的角度去写单元测试，更能发挥单元测试的作用。

小康师傅 1

2018-07-17

目前在后台和安卓端使用过jacoco，还是有很好的作用的。一方面可以让开发同学了解自己的业务逻辑设计是否有问题，另一方面可以让测试同学了解自己的用例设计是否完善。覆盖率高不一定质量就没有问题，但是覆盖率低质量肯定不能保证。

lalio 1

2018-07-15

最近在研究go语言的代码覆盖率工具集成到公司的CI，尝试用的go test 的cover，不过了解下来发现这款官方出的工具只统计到了语句覆盖，用了godoc实现（本人英文水平有限，目前还没吃透原理），没有jacoco这款覆盖的种类全，统计报告也比较单一，想在cover的统计数据基础上做一些优化，老师有没有好的建议

展开

作者回复: go test我没有用过，如果你想自己扩展开发代码覆盖率工具的话，技术要求还是比较高的，首先需要知道工具的注入原理，然后要搞清楚出内部的代码覆盖率统计的数据结构，然后再去修改工具的源代码，总体来看，技术要求比较高

Nic辉少

2018-07-11

1

目前还没有接触过单元测试和代码方面的测试，每天努力坚持学一点，思考一点，吸收一点。

作者回复: 好样的，学习的过程就是螺旋上升的过程，可能会有点痛苦，但是你会发现自己在潜移默化的进步，一个月后你再回头看看，会发现自己已经站在了一个全新的高度了，加油

丹

2018-07-11

1

你好，我想请教一些问题，我们公司属于迭代开发，更新比较快，没有技术文档，这种情况下感觉测试用例不能覆盖全面，这种情况应该怎么去写测试用例

作者回复: 很多敏捷团队可能没有完全文档化的测试用例，所以会比较依赖团队成熟度以及一些BDD和TDD的工具，关于什么是bdd和tdd，后面会专门来讲，这两者都属于迭代开发下的敏捷实践

宸浩

2018-07-11

1

老师，我想问下测试接口的话，怎么能保证覆盖完全呢，敏捷项目中涉及到的接口覆盖率，麻烦普及一下呗

作者回复: 这个会在后面的api测试中专门来讲。其实接口测试完全可以用代码覆盖率来衡量测试的完备性。

FamilyLi

2018-07-11

1

对于安卓智能手机的代码覆盖率有什么测试或者工具

展开

作者回复: Android SDK内置了Emma test coverage，你可以直接用，不过我还是比较推荐Jacoco

假装乐
2018-07-11

1

jococo是哪年出的？当初因为没有找到合适的，把cobertura改成了agent模式的

作者回复: jacoco的早期版本2009年就有了，那时还不是很成熟，早期的版本我也没有用过

Rachel_fan...
2018-07-11

1

jacoco大多用于自动化测试得到覆盖率，但如果想知道单条case（手工or自动化）的覆盖率不知道有啥好的方法吗？

作者回复: jacoco其实不管你的测试是手工跑的还是自动跑的，它都能统计哪些代码被执行到了，哪些没有。单条case的话也是一样的

口水窝
2019-03-04

测试工作中没有接触过代码覆盖率的工具，所以测试的工作不仅仅是测试业务逻辑，更多的时候本着质量的意识，从广度、深度去挖掘，以前没有接触过，去学习，去自己搭建环境，去实践，对于测试的理解更上一层楼！

但是在测试报告的编写中，知道测试用例覆盖率和需求覆盖率，那时候用简单的加减法算的，没用具体工具。

展开

向着光亮那...
2018-11-16

想问下，对于Android写的SDK，功能测试怎么统计覆盖率呢？

展开

作者回复: 同样是基于传统的代码覆盖率工具，没有任何的区别

郭小菜

2018-10-20

老师，jacoco可以直接应用于线上吗？比如说根据用户的实际操作看代码的覆盖情况？

小老鼠

2018-10-08

用插桩会不会影响程序性能。

展开

作者回复: 严格说会，但是单元测试基本可以忽略不计，除非是那些本身性能灵感的方法

碎碎念

2018-08-23

覆盖率分为项目的需要覆盖和代码覆盖，而目前常说的覆盖率更多的是指代码覆盖率。覆盖率这是检查代码的完备性，并不能说明覆盖率高软件的质量就高，初学者，多多指教

展开

才子

2018-08-21

真正写好单元测试，其代码量会是源代码的数倍。现在开发人员也不愿意写单元测试或者写的不完全。老师说的对，代码覆盖率只是一个指标，并不能代表产品的质量。我们单位内以此作为质量的标杆，很不合理。而且有时候会为了满足覆盖率而作假，这是很不合理的。