代码覆盖率

开发人员在单元测试期间执行代码覆盖，以验证代码实现，尽可能多执行代码语句。大多数代码覆盖率工具都使用静态工具，将监视执行的语句插入代码中的必要位置。尽管添加检测代码会导致总体应用程序大小和执行时间增加，但与通过执行检测代码生成的信息相比，开销却很小。输出包含一个详细描述测试套件测试范围的报告。

覆盖率类型

代码覆盖率的一些常见子类型为：

语句覆盖：又称行覆盖(LineCoverage)，段覆盖(SegmentCoverage)，基本块覆盖(BasicBlockCoverage)，这是最常用也是最常见的一种覆盖方式，就是度量被测代码中每个可执行语句是否被执行到了。选择足够多的测试数据，使被测程序中每条语句至少执行一次。语句覆盖是很弱的逻辑覆盖，它只管覆盖代码中的执行语句，却不考虑各种分支的组合等等。测试效果不明显，很难更多地发现代码中的问题。

判定覆盖(BranchCoverage)，所有边界覆盖(All-EdgesCoverage)，基本路径覆盖(BasicPathCoverage)，判定路径覆盖(Decision-Decision-Path)。 比语句覆盖稍强的覆盖标准，它**度量程序中每一个判定的分支是否都被测试到了。**设计足够的测试用例，使得程序中的每个判定至少都获得一次“真值”或“假值”，或者说使得程序中的每一个取“真”分支和取“假”分支至少经历一次，因此判定覆盖又称为分支覆盖。

JavaScript

```
if(a && b) {
  //  分支1
}else {
  //  分支2
}

// a && b = true
// a && b = false
```

条件覆盖：它度量判定中的每个子表达式结果 true 和 false 是否被测试到了。在设计程序中，一个判定语句是由多个条件组合而成的复合判定。为了更彻底地实现逻辑覆盖，可以采用条件覆盖（Condition Coverage）的标准。条件覆盖的含义是：构造一组测试用例，使得每一判定语句中每个逻辑条件的可能值至少满足一次。条件覆盖与判定覆盖非常容易混淆，条件覆盖不是将判定中的每个条件表达式的结果进行排列组合，而是只要每个条件表达式的结果 true 和 false 测试到了就 OK 了。因此，我们可以这样推论：完全的条件覆盖并不能保证完全的判定覆盖

JavaScript

```
if(a && b) {
  //  分支1
}else {
  //  分支2
}
// a = true
// a = false
// b = true
// b = false
```

条件判定组合覆盖：多条件覆盖多条件覆盖也称条件组合覆盖，它的含义是：**设计足够的测试用例，使得每个判定中条件的各种可能组合都至少出现一次**。显然满足多条件覆盖的测试用例是一定满足判定覆盖、条件覆盖和条件判定组合覆盖的。

JavaScript

```
if(a && b) {
  //  分支1
}else {
  //  分支2
}

// a = true  && b = true
// a = true  && b = false
// a = false  && b = true
// a = false  && b = false
```

路径覆盖：又称断言覆盖(PredicateCoverage)。它度量了是否函数的每一个分支都被执行了。 就是所有可能的分支都执行一遍，有多个分支嵌套时，需要对多个分支进行排列组合，可想而知，测试路径随着分支的数量指数级别增加。路径覆盖被很多人认为是“最强的覆盖”。

- 项目 成本 与 质量
- 测试本质： 发现 Bug

测试方法

三种主要类型

- 代码检测：这里的源代码是在添加检测语句之后编译的。编译应使用常规工具链完成，编译成功将导致生成检测装配。例如，为了检查在代码中执行特定功能所花费的时间，可以在功能的“开始”和“结束”中添加检测语句。
- 运行时检测：与代码检测方法相反，此处的信息是从运行时环境（即在执行代码时）收集的。
- 中间代码检测：在这种检测类型中，通过向已编译的类文件中添加字节码来生成检测类。

覆盖率工具

- Coverage.py：这是 Python 的代码覆盖工具。顾名思义，它可以分析您的源代码并确定已执行代码的百分比。它是用 Python 开发的。
- Serenity BDD：支持 Java 和 Groovy 编程语言，Serenity BDD 是一个流行的开源库，主要用于更快地编写出色的质量验收测试。它可以与 JUnit，Cucumber 和 JBehave 一起使用。Serenity BDD 可以轻松地与 Maven，Cradle，JIRA 和 Ant 集成。

- JaCoCo：JaCoco 是 Java 的代码覆盖工具。尽管还有其他选项，例如 Cobertura 和 EMMA，但由于长时间没有更新，因此不推荐使用这些工具。除了积极开发 JaCoCo 之外，使用 JaCoCo 的另一个优势是可以与 CI/CD 和项目管理工具（例如 Maven，Jenkins，Gradle 等）无缝集成。
- JCov：JCov 是一个测试框架不可知代码覆盖工具。它可以轻松地与 Oracle 的测试基础架构 JavaTest 和 JTReg 集成。尽管尚未积极开发，但对即时检测和脱机检测的支持是使用 JCov 的主要优势。
- PITest：这是一个突变测试框架。它有快、可扩展，并与当前测试和构建工具集成好的优点。传统的测试覆盖率（即行，语句，分支等）仅衡量测试执行的代码。 它不会检查测试是否真正能够检测到所执行代码中的错误。 因此，它只能识别绝对未经测试的代码。PITest 是一种非常流行的代码覆盖工具，用于 Java 和 JVM 的变异测试。它通过修改测试代码来完成突变测试的工作，并且现在已经在修改后的代码上执行了单元测试。PITest 易于使用，快速且正在积极开发中。它还与流行的 CI/CD 工具集成在一起使用。
- Istanbu JS： JS 覆盖率

Jest 覆盖率实战

index.js

```JavaScript
module.exports = function main(a, b) {
  if (a && b) {
    return x();
  } else {
    return y();
  }
};

function x() {}

function y() {}
```

index.spec.js

```
const main = require("../index");
describe("覆盖率演示", () => {
  it("main true", () => {
    main(true);
  });

  it("main false", () => {
    main(false);
  });
});
```

Bash

```
jest --converage
```