Java理论与实践:平衡测试，第2部分（一） PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/145/2021_2022_Java_E7_90_86_E8_AE_BA_c104_145095.htm 这一简短系列的 第 1 部分 介绍了如何进行有效测试，它构建了 FindBugs 插件来查找一个简单的 bug 模式（只需调用 System.gc() 即可）。Bug 模式会标识有问题的编码实践，它们常常位于 bug 所在的区域。当然，并不是所有出现 bug 模式的地方都一定出现 bug，但这并不能抹杀 bug 模式检测器的巨大作用。一个有效 bug 模式检测器的主要功能是发现更高百分比的可疑代码，使该模式具有更大的使用价值。创建 bug 模式检测器可以提高使用价值；创建检测器之后，无论是现在还是将来，您都可以在您需要的任何代码上运行它，并且您可能对发现的问题感到惊讶。例如，第 1 部分 中的简单检测器显示了对 System.gc() 的调用，在 JDK 1.4.2 中，它隐藏在 JPEG 图像 I/O 库中。 编写检测器可以查找对特定静态方法的调用，这并不困难，但是大多数的 bug 检测器都包含相当多的分析和实现。在这一期的文章中，您将开发一个称为 RuntimeException capture 的更小 bug 模式的检测器（目前，FindBugs 发行版中已包含此 bug 检测器。） RuntimeException 捕获 用 Java&#8482. 语言进行异常处理的一个优点是：异常是一些对象，try-catch 机制了解异常类型的分层结构，并在客户机如何处理错误处理方面提供实际灵活性。例如，如果不能找到文件，则 FileInputStream 构造函数会抛出 FileNotFoundException，该异常是 IOException 的一个子类。此传统用法允许客户机处理未发现文件的条件，这些条件是从其他与文件相关的条件中分离出来的（如果

他们喜欢单独捕获 FileNotFoundException）。但是，他们还可以使用捕获 IOException 的方法处理所有与文件相关的错误条件。 另一方面，异常处理的主要缺陷是：在正确使用异常时，易于建立带有三行或四行业务逻辑以及 20 或 30 行异常处理的方法。因为错误恢复代码在测试时容易出现错误并且执行困难，使一部分专门用于异常处理的代码无所适从并容易出错。这种情况的典型示例如清单所示，其中带有两行“真的 ”代码的方法需要三个独立的捕获块，每个捕获块都执行完全相同的操作 记录该异常： 清单 1. 多个相同的捕获块public void addInstance(String className) { try { Class clazz = Class.forName(className). objectSet.add(clazz.newInstance()). } catch (IllegalAccessException e) { logger.log("Exception in addInstance", e). } catch (InstantiationException e) { logger.log("Exception in addInstance", e). } catch (ClassNotFoundException e) { logger.log("Exception in addInstance", e). }} 请参见清单 1，您可能尝试将三个捕获块合并成捕获 Exception 的单独捕获块，因为每个捕获块的捕获恢复操作是相同的。乍一看，该策略似乎是一个好方法 但代码副本有错误，所以整合这些复制路径应该是一种改进。不过，此“ 改进 ”常常会带来意想不到的结果。因为 RuntimeException 扩展了 Exception，将三个捕获块合并成一个捕获块（如清单 2 所示），所以这会更改语义，现在，未经检查的异常将被记录（而不传播）。此 bug 模式（其中 RuntimeException 容易被超大捕获块捕获）也称为 RuntimeException 捕获。 