Java 空指针异常的单元测试生成

殷任政
南京邮电大学

2025 年 6 月 13 日

摘要

在 Java 程序中，空指针异常（Null pointer exception ，NPE）是最为常见、致命的错误之一 [1, 2, 3, 4, 5, 6] 。NPE 广泛出现在软件开发的实践中，对空指针进行的不当操作往往致使程序崩溃，然而现在的传统单元测试生成工具 EvoSuite 和 Randoop 虽然有时能达到较高的代码覆盖率，但并不足以有效探查出代码中的 NPE 问题 [7]。针对该现状，《Effective Unit Test Generation for Java Null Pointer Exceptions》[7] 的作者在结合传统自动化测试工具 EvoSuite 的基础上提出了一种专精于 NPE 检测的测试用例生成方法 NpeTest。

1 研究问题

尽管一定程度上实现了代码的高覆盖率，传统工具 EvoSuite 和 Randoop 并未能有效检测出 NPE 错误，这是由于这两种工具的目标主要是追求高代码覆盖率，而要触发 NPE ，不仅要让测试覆盖到能够触发 NPE 的语句，还要满足特定的触发条件，例如在执行到解引用的相关语句时，引用指针仍然为 null 的情况，这正是传统方法容易忽略之处。NpeTest 基于 EvoSuite，结合了静态分析与动态分析，首先通过静态分析大致确定能够引发 NPE 的语句、代码块所在位置，再在测试过程中动态剔除那些已经触及 NPE 的方法，使测试生成器更加充分地关注尚未得到探查的 NPE 相关的代码区域。

2 研究方法

2.1 静态分析器

在作者提出的方法中，静态分析器主要用于识别测试类（class under test ，CUT）中的所有 NPE 易受影响区域和方法，以及优先确定给定测试用例中要变异的语句。具体而言，给定一个 CUT ，首先从代码库中构造每个方法的控制流图（control flow graph ，CFG），其中每个节点代表一个原子语句。基于 CFG ，计算每个方法 m 的目标表达

式 exp 和所在行号 loc 的集合 Texp（即（exp ，loc） Texp）。如果表达式满足以下条件中的至少一个，则将其收集到集合 Texp 中：(1) 以“E1 .E2”的形式表示的表达式 E1 。(2) 调用一个方法很有可能触发 NPE 的方法。(3) 当返回类型为引用类型时的返回变量。以图 1 中展示的两个方法为例，得到的目标集合如下所示:

Texp (addEdge) : (getEdge (src) ， 3), (src ， 4), (getEdge (target) ， 5)

Texp (getEdge) : (vertexMap, 2), (vertexMap, 5), (ec, 7) .

```
1  public boolean addEdge(V src, V target, E e) {
2    if (src == null) return false;
3    getEdge(src).addEdge(e);
4    if (!src.equals(target)) {
5       getEdge(target).addEdge(e);
6    }
7    return true; }
```

```
1  public Edge<V, E> getEdge(V vertex) {
2    Edge<V, E> ec = vertexMap.get(vertex);
3    if (ec == null) {
4       ec = new Edge<>(edgeFactory, vertex);
5       vertexMap.put(vertex, ec);
6    }
7    return ec; }
```

图 1: addEdge 方法与 getEdge 方法

利用集合 Texp，为每个目标表达式构建一个有限的路径信息集合。方法 m 的每条路径信息是一个元组（path，isNull，exp）。其中 path 是一系列语句，isNull 是从引用变量到布尔值的映射变量，表示引用变量 x 的非空性，该映射遵循的规则如图 2 所示。exp 是来自 Texp(m) 的单个目标表达式。以上述 Texp (getEdge) 中的表达式 (ec, 7) 为例，可以构建出如下指向 ec 的路径：path1 →(ec，7):[ec = new Edge< >(edgeFactory，vertex); return ec]，path2 →(ec，7):[!(ec == null); return ec]，

$$
\begin{aligned}
\Theta(x := \mathtt{null}, isNull) &= isNull[x \mapsto true] \\
\Theta(x := y, isNull) &= isNull[x \mapsto isNull[y]] \\
\Theta(x := new\ C(), isNull) &= isNull[x \mapsto false] \\
\Theta(x := call(\mathtt{m}'), isNull) &= isNull[x \mapsto Ret_{null}(\mathtt{m}')] \\
\Theta(x == \mathtt{null}, isNull) &= isNull[x \mapsto true], \\
\Theta(\neg(x == \mathtt{null}), isNull) &= isNull[x \mapsto false],
\end{aligned}
$$

图 2: isNull 的映射规则

根据 isNull 的映射规则，在这两条路径的入口处，ec 的 isNull 属性都将被设为 false, 因此这两条指向 return ec 的路径都会导致返回非空的对象，从而 getEdge 方法永远不会返回 null，即 Retnull 为 false。对所有方法进行可空路径识别后，可以根据可空路径数量占比计算每个方法 NPE 可能性的得分，随后用于变异测试用例的选择，调用得分更高的方法的测试用例更有可能被选中进行变异。

2.2 动态监控

基于 EvoSuite 的测试用例执行监控，NpeTest 能够动态获取每个测试用例执行的一组方法调用，以及执行过程中产生的异常信息。具体而言，NpeTest 为每个方法维护一个目标表达式集合 ((exp，loc) ∈ Texp)。如果在测试用例执行过程中发生 NPE，NpeTest 会从运行时异常错误日志中收集方法 m 和 NPE 触发的错误位置 loc 的信息，并从 Texp(m) 中移除相应的目标表达式 exp。通过动态剔除已检测到所有 NPE 的方法 m（即 Texp(m) 为空），动态分析器使单元测试生成器能够更多地关注那些尚未充分探索的与 NPE 相关的代码。

3 评估

作者将 NpeTest 与现有的单元测试生成工具进行了性能的比较，选择了 EvoSuite 和 Randoop 作为基准，这两种工具因其卓越的性能而在单元测试生成领域被广泛采用 [8, 9] 。实验设置方面，为每种工具进行了 25 次评估实验，在发生 NPE 的基准类中使用了 5 分钟的时间预算，每个基准测试总共使用了 125 分钟的 CPU 时间。所有实验均在运行 Ubuntu 20.04 的 Linux 机器上进行，该机器配备 64 个 CPU 和 256GB 内存，搭载 AMD Ryzen Threadripper3990X 64 核处理器。EvoSuite 使用了开发者团队设定的默认参数。基准测试项目则是从相关文献以及 BugSwarm[10]、Defects4J[11] 中收集的共 195 个有缺陷的项目, 排除因实验环境等不能重现的测试后，最终收集了 96 个包含 108 个已知 NPE 的有缺陷项目用于基准测试套件。就实验结果而言，NpeTest 对于成功生成的检测已知 NPE 的测试用例，其平均的重现率分别比 Randoop 和 EvoSuite 高 45.2% 和 22.4%。在 25 次试验中检测到的 NPE 数量方面，NpeTest 发现了 73 个 NPE，而 Randoop 和 EvoSuite 分别发现了 25 个和 59 个 NPE。这些数据经过检验具有统计学意义。此外实验数据还表明，尽管 NpeTest 的代码覆盖率较低，且相较于其他两种工具忽略了许多非 NPE 问题，但在探测 NPE 方面，NpeTest 表现出了其独特的优越性以及性能。

4 小结

NPE 是 Java 应用程序中最常见且致命的错误之一。尽管自动化生成 Java 测试用例的方法众多，但通过测试发现 NPE 仍然是一个挑战，并具有重大意义。本文介绍了一种专精于生成 NPE 缺陷检测测试用例的方法 NpeTest，通过结合传统自动化测试工具 EvoSuite 以及静态和动态分析方法，指导测试用例生成器更高效地探索 NPE 易发区域。实验结果表明，NpeTest 可以显著提高现有最先进单元测试生成器的 NPE 检测能力。作者同时强调了在工业软件开发中采用综合方法检测 NPE 的重要性。尽管有严格的测试和开发流程，本文提到的三种测试工具仍能够检测出大量先前未知的 NPE ，表明了手动测试和代码审查的局限性。此外，作者通过访谈揭示了在实际开发中，尽管出

发点正确，但在实现时也可能会无意中引入漏洞。通过将 NpeTest 等自动化测试工具集成到开发中，除了手动测试和代码审查之外，开发人员可以更主动高效地识别和解决潜在的 NPE。

参考文献

[1] R. Coelho, L. Almeida, G. Gousios, A. van Deursen, Unveiling exception han dling bug hazards in android based on github and google code issues, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp.134 - 145. doi:10 . 1109/MSR.2015.20.

[2] A. S. Boujarwah and K. Saleh. Compiler test case generation methods: a survey and assessment. Information and Software Technology, 39(9):617 -625, 1997.

[3] R. L. Sauder . A general test data generator for COBOL. In AFIPS Joint Computer Conferences, pages 317 -323, May 1962.

[4] K. V. Hanford. Automatic generation of test cases. IBM Systems Journal, 9(4):242 -257, Dec. 1970.

[5] P. Purdom. A sentence generator for testing parsers. BIT Numerical Mathematics, 12(3):366 -375, 1972.

[6] C. J. Burgess and M. Saidi. The automatic generation of test cases for optimizing Fortran compilers. Information and Software Technology, 38(2):111 -119, 1996.

[7] Myungho Lee, Jiseong Bak, Seokhyeon Moon, Yoon-Chan Jhi, and Hakjoo Oh. 2024. Effective Unit Test Generation for Java Null Pointer Exceptions. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24). Association for Computing Machinery, New York, NY, USA, 1044-1056. https://doi.org/10.1145/3691620.3695484

[8] C. Pacheco, S. K. Lahiri, M. D. Ernst, T. Ball, Feedback-directed random test generation, in: 29th International Conference on Software Engineering (ICSE ' 07), 2007, pp. 75 –84. doi:10 . 1109/ICSE.2007.37.

[9] G. Fraser, A. Arcuri, Whole test suite generation, IEEE Transactions on Software Engineering 39 (2013) 276 –291. doi:10 . 1109/TSE.2012. 14.

[10] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoud hury. 2017. Directed Greybox Fuzzing. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX,USA, Octo-

ber 30 - November 03, 2017, Bhavani Thuraisingham, David Evans,Tal Malkin, and Dongyan Xu (Eds.). ACM, 2329 –2344. https://doi.org/10.1145/3133956.3134020

[11] Andrea Fioraldi, Dominik Christian Maier, Heiko Eißfeldt, and Marc Heuse.2020. AFL++ : Combining Incremental Steps of Fuzzing Research. In 14th USENIX Workshop on Offensive Technologies, WOOT 2020, August 11, 2020, Yuval Yarom and Sarah Zennou (Eds.). USENIX Association. https://www.usenix.org/conference/woot20/presentation/fioraldi