# 静的解析ツールを駆使してイケてるコードを書こう

Acroquest Technology
谷本 心

皆さんは、イケてるコードを書くために、普段からどのような取り組みを行なっているでしょうか。例えば、本連載のような雑誌記事や書籍を利用して、自分で勉強する方法があります。また、経験豊富なエンジニアに教えてもらう、レビューを受けることでイケてるコードの書き方を学んだりする、という方法を採用している人もいるでしょう。

しかし、これら以外にもイケてるコードの書き方を教えてくれる、あるいは問題コードを検出してくれるツールを使うという方法もあります。特に、プログラムを実行することなく、ソースコード自体を解析して問題を検出するツールのことを「静的解析ツール」と呼びます。今回は、この静的解析ツールにできることと、ツールの種類について紹介しましょう。

**リスト1●closeされない可能性があるコード**

```
public List<String> readFiles(String fileName)
  throws IOExeption {
  List<String> list = new ArrayList<>();
  BufferedReader reader = new BufferedReader(
    new FileReader(new File(fileName)));
  String line;
  while ((line = reader.readLine()) != null) {
    list.add(line);
  }
  reader.close();    ← このメソッドは必ず呼ばれるわけではない
  return list;
}
```

**リスト2●Findbugsの警告に従ってコードを改善した。確実にclose処理が行われる**

```
public List<String> read(String fileName)
  throws IOExeption {
  List<String> list = new ArrayList<>();
  BufferedReader reader = new BufferedReader(
    new FileReader(new File(fileName)));
  try {
    String line;
    while ((line = reader.readLine()) != null) {
      list.add(line);
    }
  } finally {
    reader.close();    ← finally句を利用して確実にclose処理が実行されるように変更した
  }
  return list;
}
```

## 静的解析ツールにできること

まずは、静的解析ツールを使って何ができるかを簡単な例でみてみましょう。**リスト1**のコードをみてください。リスト1では、処理の途中に例外が発生すると、BufferedReaderクラスのインスタンスであるreaderのcloseメソッドが呼ばれず、リソースの解放漏れを起こしてしまう可能性があります。

このようなコードから潜んでいるバグを発見するためには静的解析ツールが有効です。静的解析ツールの1つである「FindBugs」を適用すると、BufferedReaderがcloseされない可能性があることを検出し警告を出します(**図1**)。ポップアップされた個所に注目してください。「Main.read(String) may fall to close stream on exception」という警告メッセージが出力されています。「例外が発生した際に、ストリームのクローズに失敗するかもしれない」という意味です。

FindBugsは、Javaのクラスファイルを解析して、このreadFilesメソッドの中でBufferedReaderをnewしていることと、それがcloseされていないことを検出しています。リスト1を改善する方法は、これまでの連載で紹介しています。Java SE 7で導入されたtry-with-resourcesを利用する、finallyでcloseメソッドを実行する、というどちらかで解決できます。改善後のコードは**リスト2**のようになります。リスト2のように記述すれば、FindBugsによる警告は解消します(**図2**)。

このような問題の検出は、静的解析ツールでできることの一つです。静的解析ツールを利用すれば、大きく分けてさらに5つのことが可能になります(**図3**)。それぞれの機能については、後で静的解析ツールを紹介する際に詳しく説明します。

## なぜ静的解析ツールを使うのか?

静的解析ツールを使うことで、コードの典型的な問題を検出できて、コードの品質が向上することがわかりました。この「品質向上」は、静的解析ツールを使う大きな理由の一つです。しかし筆者は、品質向上以外にも静的解析ツールを使うメリットはあると考えています。その理由が、(1)ソースコードの品質を向上させる、(2)ソースコードのスタイルや水準を統一する、(3)ソースコードの複雑さを抑える、(4)他のエンジニアと差を付けられる、という4点です。それでは、1つずつみていきましょう。

### ◆ソースコードの品質を向上させる

コードの品質を向上させることは、動作確認や単体テスト、結合テストでも可能だと考える人もいるでしょう。しかし、決してそうではありません。静的解析ツールを使うことで「通常のテストでは検出しづらい問題」を発見できるのです。

例えば、効率を低下させるコードや無駄な処理などは、テストでは気づきにくい問題です。テストケースに合った結果を返せば、コードの効率性までは判断できないことが多いからです。さらに、マルチスレッド処理でのタイミングの問題や、再現が難しい状況下での異常動作なども、簡単なテストではなかなか検出できません。静的解析ツールを利用することで、このような問題の一部を検出できるのです。

さらに、コードの問題を静的解析ツールで把握できるので、開発者自身が学習できます。筆者自身も静的解析ツールに指摘されて初めて知った問題コードが多数あります。静的解析ツールは、典型的な問題を検出するためのノウハウを蓄積しているので、経験豊富なエンジニアからレビューを受けるのと同様の効果があると言えるでしょう。

**図1●FindBugsを利用して解析した様子**

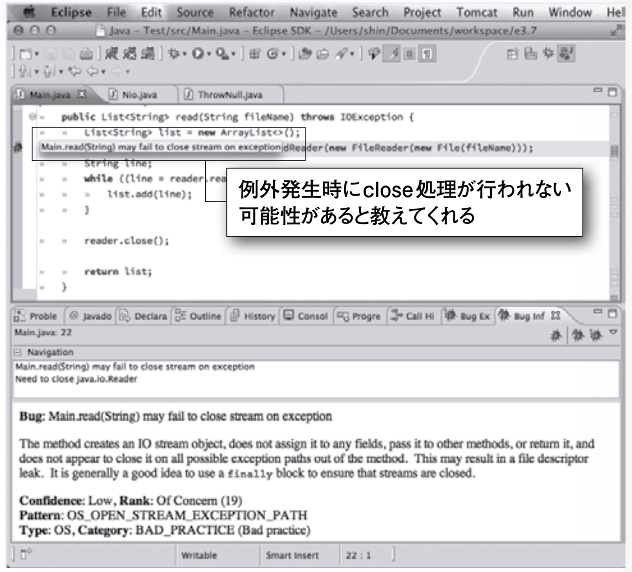


**図2●FindBugsによる警告が出なくなった**
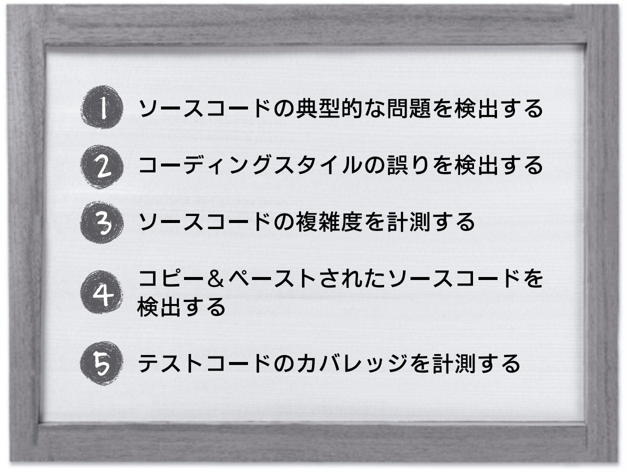
**図3●静的解析ツールでできること**