Testkit 使用指南

202028013229053 胡起

2020 年 12 月 15 日

1 文件清单

所有程序放在 ticketingsystem 目录中，trace.sh 文件放在 ticketingsystem 目录的上层目录中。如果程序有多重目录，那么将主 Java 程序放在 ticketingsystem 目录中。

文件清单如下：

- trace.sh 是 trace 生成脚本，用于正确性验证，不能更改。
- pom.xml 是依赖配置文件，使用mvn。
- .travis.yml 是 CI 配置文件，用于自动化测试。
- 文件夹.github 是 github 自动化测试配置文件。
- 文件夹src/main/java 为代码文件夹。
  1. TicketingSystem.java 是规范文件，不能更改。
  2. Trace.java 是 trace 生成程序，用于正确性验证，不能更改。
  3. TicketingDS.java 是并发数据结构的实现。
  4. ... 其他的自建类。
  5. PerformanceBenchmark.java 是 JMH 基准测试程序。
  6. jmh.benchmark.PerformanceBenchmarkRunner.java 是 JMH 基准测试启动文件。
- 文件夹src/test/java 为测试文件夹。
  1. ticketingsystem 存放基本测试单元。
     - UnitTest.java 为系统的单元测试，为单线程运行。
     - RandomTest.java 为系统的随机测试，通过多线程，随机购、退、查票。
     - MultiThreadTest.java 为多线程买、退票测试程序，通过多线程随机购、退票。
     - TraceVerifyTest.java 为 trace 单线程可线性化比对测试。
  2. verify 文件夹存放 trace 单线程可线性化比对测试资源文件
     - Trace.java.copy 为 Trace 调用文件，会自动替换原先的 Trace.java。
     - verify.jar 为单线程线性化测试包。

3. `linerChecker` 文件夹存放 trace 多线程可线性化比对测试。
   - `check.sh` 为启动脚本。
   - `checker.jar` 为多线程线性化测试包。

2 Testkit 使用说明

`Testkit` 是一个使用 `Maven` 创建并封装的测试环境，其中包括有正确性测试以及性能测试两大套件。项目的文件主体放在 `src/main/java` 目录下，你需要将你的文件放在 `src/main/java/ticketingsystem` 文件夹内，`PerformanceBenchmark.java` 以及 `jmh` 文件夹用于基准测试不能删除。

测试包含有 74 个正确性测试点，其包含有：

1. UnitTest，单元测试，用于测试单线程情况下买、退、查票的运行正确性，共 3 个测试点，15 个断言；
2. MultiThreadTest，多线程测试，用于测试买、退票正确性，会执行 50 次。每次测试 1 个测试点，4 个断言；
3. RandomTest，随机测试，用于测试随机环境下的稳定性，共 1 个测试点，2 个断言；
4. TraceVerifyTest，单线程可线性化 trace 测试工具，由老师提供，会执行 10 次。每次测试 1 个测试点，3 个断言；
5. MultiTraceVerifyTest，多线程可线性化 trace 测试工具，由陈学海同学提供，会执行 10 次。每次测试 1 个测试点，3 个断言。

包含一个性能测试工具，通过每次运行 64000 次操作，计算对应吞吐率。

你需要：

1. 保证整个项目的结构；
2. 在 `src/main/java/ticketingsystem` 文件夹下替换自己的实现；
3. 如果使用非 `java-11` 版本，请调整 `pom.xml` 。更改 `your java version` 为自己版本。

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>your java version</java.version>
    <maven.compiler.source>${java.version}</maven.compiler.source>
    <maven.compiler.target>${java.version}</maven.compiler.target>
    ...
</properties>
```

2.1 使用 maven

项目使用 maven 构建，运行前请安装 maven 。没有改变基本操作，常用的命令如下：

- 通过 mvn clean 清理生成文件。
- 通过 mvn package 打成 jar 包。
- 通过 mvn test 执行测试。
- ...

2.2 使用 Junit 进行正确性测试

> 注意：你需要安装 maven 才能执行，并且在执行过程中会自动安装相应依赖。

项目使用 Junit 进行正确性测试，你可以使用：

- mvn test 命令完成测试；
- 查看运行结果，会报告测试数量以及通过测试点数量。

2.3 使用 JMH 进行性能测试

> 注意：你需要安装 maven 才能执行，并且在执行过程中会自动安装相应依赖。

JMH 是 OpenJDK 团队开发的一款基准测试工具，一般用于代码的性能调优，精度甚至可以达到纳秒级别，适用于 java 以及其他基于 JVM 的语言。

项目使用 JMH 进行性能测试，每次测试会：

- 创建新的 TicketingDS 类，以保证每次测试可靠性；
- 每轮测试进行 5 次预热 (Warmup)，每次间隔 1 秒，目的是让 JVM 充分优化代码；
- 每轮测试 5 次记录 (Measurement)，每次间隔 5 秒，以减少 GC 带来的性能影响；
- 共测试 2 轮，取 10 次记录平均值。

你可以使用：

- mvn package 将项目打包；
- 在项目根目录下，运行：

```
java -cp ./target/trainTicketingSystem-1.0-SNAPSHOT.jar
↪  ticketingsystem.jmh.benchmark.PerformanceBenchmarkRunner
```

- 查看运行结果，结果单位为 ops/s ，即每秒操作数。这里的操作数与真实数量有差距，需要对数据乘上每次操作执行的买、退、查票动作数，即需要乘上 64000；
- 工具会生成 result.json 报告，可以使用 JMH-Visual-chart 等可视化工具查看。

2.4 使用 CI 自动化测试

项目支持 `github workflow` 以及 `travis-ci` 自动化测试，开箱即用。每次 `push` 都会自动触发测试。

- 文件夹`.github` 存放 github 自动化测试配置文件；
- `.travis.yml` 是 travis-ci 自动化测试配置文件。