

# 上机作业A6 SPECjvm2008基准测试

## 1.访问 SPECjvm2008官方网站，阅读 User's Guide、Run and Report Rules、Known Issues、FAQ 四份文档了解SPEC组织和一个标准的基准测试集的基本情况。

SPEC (Standard Performance Evaluation Corporation, 标准性能评估组织) 是一个全球性的第三方非营利性组织，致力于建立、维护和认证一套应用于计算机的标准化基准评测套件，SPEC组织开发基准测试套件并经过检验然后在SPEC网站上公开测试结果。

测试集基本情况：（摘自User's Guide）

In a given run, each SPECjvm2008 sub-benchmark produces a result in ops/min (operations per minute) that reflects the rate at which the system was able to complete invocations of the workload of that sub-benchmark. At the conclusion of a run, SPECjvm2008 computes a single quantity intended to reflect the overall performance of the system on all the sub-benchmarks executed during the run. The basic method used to compute the combined result is to compute a geometric mean. However, because it is desired to reflect performance on various application areas more or less equally, the computation done is a little more complex than a straight geometric mean of the sub-benchmark results.

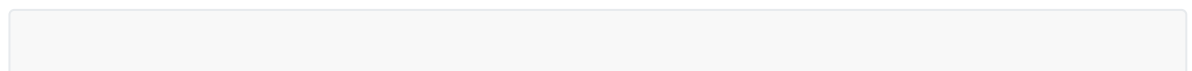
In order to include multiple sub-benchmarks that represent the same general application area while still treating various application areas equally, an intermediate result is computed for certain groups of the sub-benchmarks before they are combined into the overall throughput result. In particular, for these groups of sub-benchmarks

- COMPILER: compiler.compiler, compiler.sunflow
- CRYPTO: crypto.aes, crypto.rsa, crypto.signverify
- SCIMARK: scimark.fft.large, scimark.lu.large, scimark.sor.large, scimark.sparse.large, scimark.fft.small, scimark.lu.small, scimark.sor.small, scimark.sparse.small, scimark.monte\_carlo
- STARTUP: {all sub-benchmarks having names beginning with startup. } See Appendix A for the complete list.
- XML: xml.transform, xml.validation

## 2.下载和安装 SPECjvm2008，根据上述文档的指引，跑一次完整的基准测试（大约 2~3小时），记录安装和测试的过程和结果，了解一个标准的基准测试集的基本执行过程。

1.安装 OpenJDK 1.8.0\_41 RI Binaries版本

首先在官网<https://jdk.java.net/java-se-ri/8-MR3> 下载压缩包，点击高亮部分下载到本地。



jdk.java.net/java-se-ri/8-MR3

应用 collections — Con... Iterables vs. Iterat... How to Implemen... Ch. 1 Solutions - L... Issues · Firmiana1... Teaching - Ming... Yuntao Du bookstore/fe at m...

# jdk.java.net

## Java Platform, Standard Edition 8 Reference Implementations

The official Reference Implementations for Java SE 8 (JSR 337) are based solely upon open-source code available from the JDK 8 Project in the OpenJDK Community. This Reference Implementation applies to JSR 337 Maintenance Release 3 (Feb 2020). Reference Implementation for Maintenance Release 1 (Mar 2015) and Maintenance Release 2 (Mar 2019) contains RIs for these releases. Binaries are provided for both the Linux x64 and Windows i586 platforms and Compact Profiles for Linux i586.

**These binaries are for reference use only!**

These binaries are provided for use by implementers of the Java SE 8 Platform Specification and are for reference purposes only. These Reference Implementations have been approved through the Java Community Process. Binaries for development and production will be available from Oracle and in most popular Linux distributions.

**RI Binaries (build 1.8.0\_41-b04) under the GNU General Public License version 2**

- Oracle Linux 7.6 x64 Java Development Kit (md5) 167 MB
- Windows 10 i586 Java Development Kit (md5) 92 MB
- Oracle Linux 7.6 i586 Java Runtime Environment for Compact Profiles
  - Compact Profile 1 (md5) 15 MB
  - Compact Profile 2 (md5) 18 MB
  - Compact Profile 3 (md5) 20 MB
- Full JRE (md5) 40 MB

**RI Source Code**

The source code of the RI binaries is available under the GPLv2 in a single zip file (md5) 123 MB.

**International use restrictions**

Due to limited intellectual property protection and enforcement in certain countries, the JDK source code may only be distributed to an authorized list of countries. You will not be able to access the source code if you are downloading

**GA Releases**  
JDK 17  
JMC 8

**Early-Access Releases**  
JDK 18  
Loom  
Metropolis  
Panama  
Valhalla

**Reference Implementations**  
Java SE 17  
Java SE 16  
Java SE 15  
Java SE 14  
Java SE 13  
Java SE 12  
Java SE 11  
Java SE 10  
Java SE 9  
Java SE 8  
Java SE 7

**Feedback**  
Report a bug

**Archive**

将压缩包文件传送到/usr/java文件夹下并解压：

```
# 一般将java安装在/usr/java/default目录下
sudo mv /home/zhangkeer/a6/openjdk-8u41-b04-linux-x64-14_jan_2020.tar.gz
/usr/java/
cd /usr
mkdir java
cd java
sudo tar -zxvf openjdk-8u41-b04-linux-x64-14_jan_2020.tar.gz
# 上面tar解压后的文件放在了/usr/java/java-se-8u41-ri/目录下
# 重命名目录，最终让JDK放在/usr/java/default目录下
sudo mv java-se-8u41-ri/ default/
```

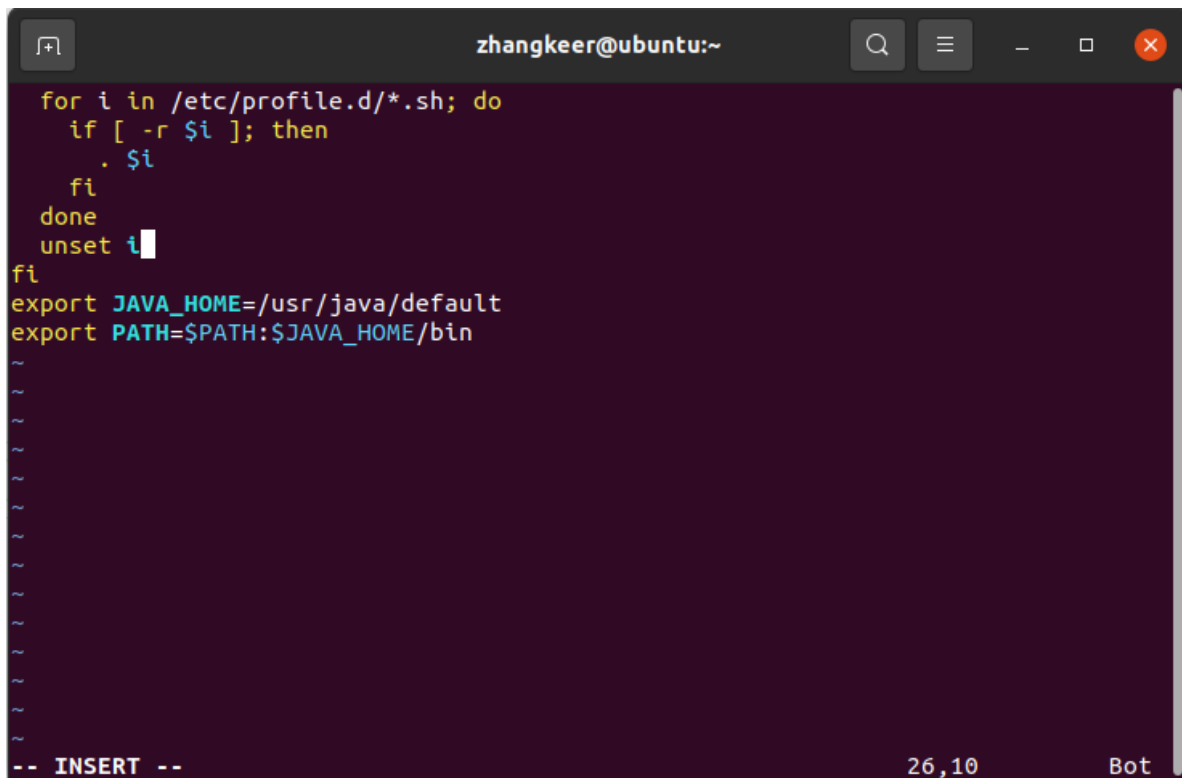
```
zhangkeer@ubuntu:/usr/java$ sudo tar -zxvf openjdk-8u41-b04-linux-x64-14_jan_2020.tar.gz
java-se-8u41-ri/
java-se-8u41-ri/src.zip
java-se-8u41-ri/sample/
java-se-8u41-ri/sample/nbproject/
java-se-8u41-ri/sample/nbproject/project.xml
java-se-8u41-ri/sample/scripting/
java-se-8u41-ri/sample/scripting/scriptpad/
java-se-8u41-ri/sample/scripting/scriptpad/src/
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/memmonitor.js
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/memory.sh
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/memory.bat
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/textcolor.js
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/insertfile.js
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/mail.js
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/memory.js
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/browse.js
java-se-8u41-ri/sample/scripting/scriptpad/src/scripts/README.txt
```

```
zhangkeer@ubuntu:/usr/java$ ls
java-se-8u41-ri  openjdk-8u41-b04-linux-x64-14_jan_2020.tar.gz
zhangkeer@ubuntu:/usr/java$ mv java-se-8u41-ri/ default/
mv: cannot move 'java-se-8u41-ri/' to 'default/': Permission denied
zhangkeer@ubuntu:/usr/java$ sudo mv java-se-8u41-ri/ default/
```

配置JAVA\_HOME环境变量：

在这一步遇到了vi无法使用方向键移动光标的情况，很不方便；因此卸载了ubuntu原装的vim tiny版本，安装vim full版本（参考<https://blog.csdn.net/xujingcheng123/article/details/83444800?spm=1001.2014.3001.5501>）

```
# 打开环境变量文件
sudo vi /etc/profile
# 子文件末尾追加2行
export JAVA_HOME=/usr/java/default
export PATH=$PATH:$JAVA_HOME/bin
# source一下，重新加载环境变量
source /etc/profile
```

A terminal window titled 'zhangkeer@ubuntu:~' showing the contents of the /etc/profile file being edited in vi. The file contains a loop that sources all .sh files in /etc/profile.d/, followed by the export of JAVA\_HOME and PATH. The terminal is in INSERT mode, indicated by '-- INSERT --' at the bottom left. The cursor is at the end of the PATH export line. The bottom right shows '26,10' and 'Bot'.

下载、解压JDK和配置环境变量都成功后可以查看java 版本：

```
zhangkeer@ubuntu:~$ java -version
openjdk version "1.8.0_41"
OpenJDK Runtime Environment (build 1.8.0_41-b04)
OpenJDK 64-Bit Server VM (build 25.40-b25, mixed mode)
zhangkeer@ubuntu:~$
```

注意：配置环境变量前我从网上得知，在ubuntu中，使用source /etc/profile命令可以使新建的环境变量立刻生效而不用重新启动系统，但是当我使用source /etc/profile这个命令之后，新的环境变量只能在一个终端里面有效（这个终端会失去高亮），而当我把这个终端关闭重新再打开另一个终端时，刚才有效的环境变量就没了。网上也有一部分网友产生了这种情况，在我重新启动虚拟机后，这个问题解决。

## 2.安装SPECjvm2008

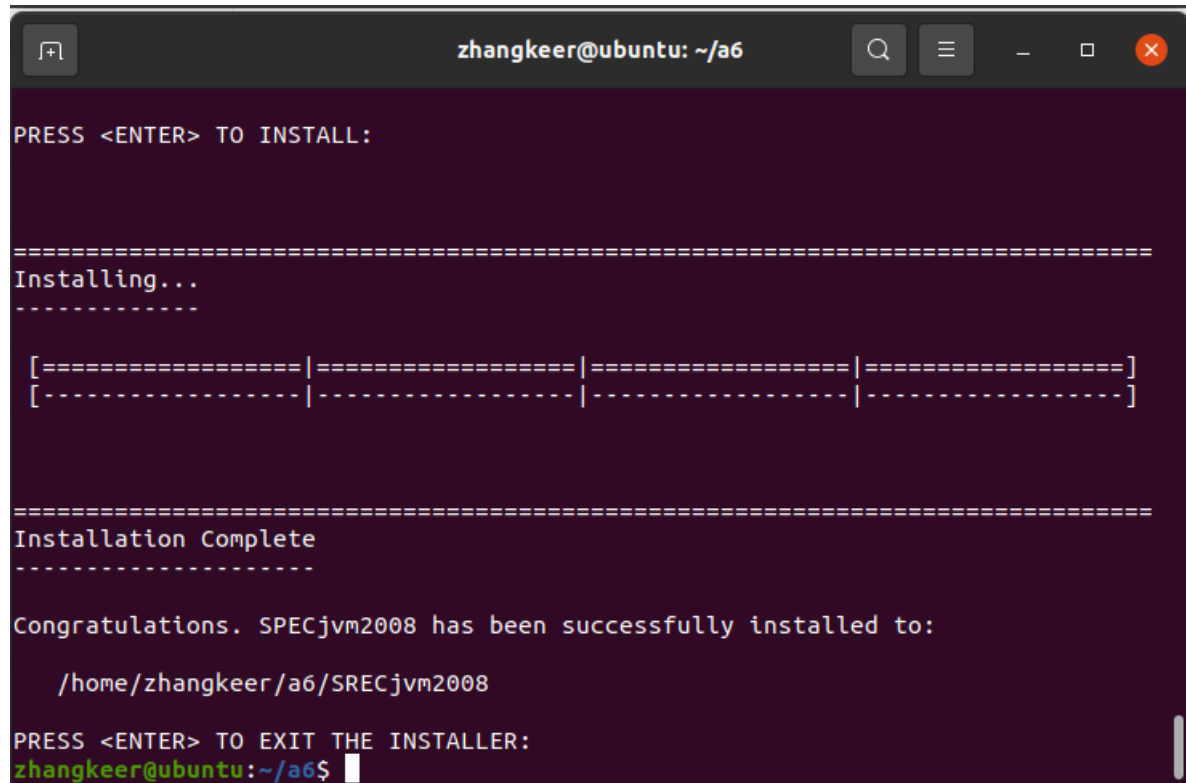
将在官网下载好的 SPECjvm2008\_1\_01\_setup.jar 传送到我的虚拟机中。

在jar包所在的终端输入命令：

```
java -jar SPECjvm2008_1_01_setup.jar -i console
```

接下来一路enter即可。安装路径我手动输入了/home/zhangkeer/a6/SPECjvm2008

安装结束：



```
zhangkeer@ubuntu: ~/a6
PRESS <ENTER> TO INSTALL:

=====
Installing...
-----

[=====|=====|=====|=====]
[-----|-----|-----|-----]

=====
Installation Complete
-----

Congratulations. SPECjvm2008 has been successfully installed to:

    /home/zhangkeer/a6/SPECjvm2008

PRESS <ENTER> TO EXIT THE INSTALLER:
zhangkeer@ubuntu: ~/a6$
```

测试是否安装成功：

```
./run-specjvm.sh startup.helloworld -ikv
```

startup.helloworld 是测试helloworld程序的启动时间，选择这个测试用例是考虑到运行速度比较快。

-ikv的意思是跳过签名检查。

```

zhangkeer@ubuntu:~/a6/SPECjvm2008$ ./run-specjvm.sh startup.helloworld -ikv
SPECjvm2008 Base
Properties file: none
Benchmarks: startup.helloworld

WARNING: Run will not be compliant.
Not a compliant sequence of benchmarks for publication.
Property specjvm.run.checksum.validation must be true for publication.

-----

Benchmark: check
Run mode: static run
Test type: functional
Threads: 1
Iterations: 1
Run length: 1 operation

Iteration 1 (1 operation) begins: Thu Nov 25 05:49:53 PST 2021
Iteration 1 (1 operation) ends: Thu Nov 25 05:49:53 PST 2021
Iteration 1 (1 operation) result: PASSED

Valid run!

-----

Benchmark: startup.helloworld
Run mode: static run
Test type: single
Threads: 1
Iterations: 1
Run length: 1 operation

Iteration 1 (1 operation) begins: Thu Nov 25 05:49:53 PST 2021
Iteration 1 (1 operation) ends: Thu Nov 25 05:49:54 PST 2021
Iteration 1 (1 operation) result: 218.18 ops/m

Valid run!
Score on startup.helloworld: 218.18 ops/m

Results are stored in:
/home/zhangkeer/a6/SPECjvm2008/results/SPECjvm2008.001/SPECjvm2008.001.raw
Generating reports in:
/home/zhangkeer/a6/SPECjvm2008/results/SPECjvm2008.001

Noncompliant composite result: 218.18 ops/m

```

### 3.跑一次完整的基准测试

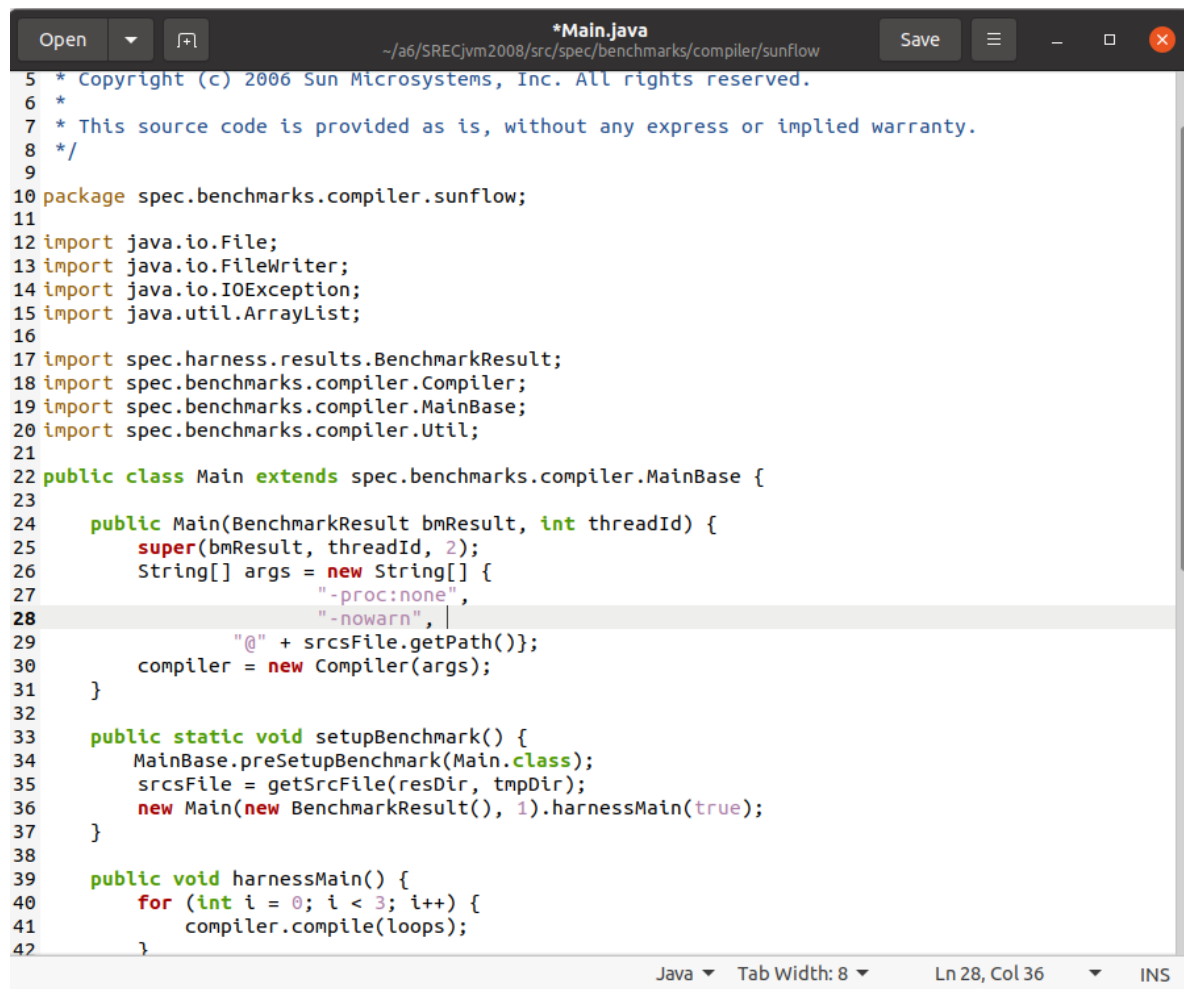
只需运行 Base类别:

```
java -jar SPECjvm2008.jar --base
```

考虑到startup.compiler.sunflow等会出现堵塞现象，重新编译SPECjvm来避免堵塞

(1)修改Main.java文件

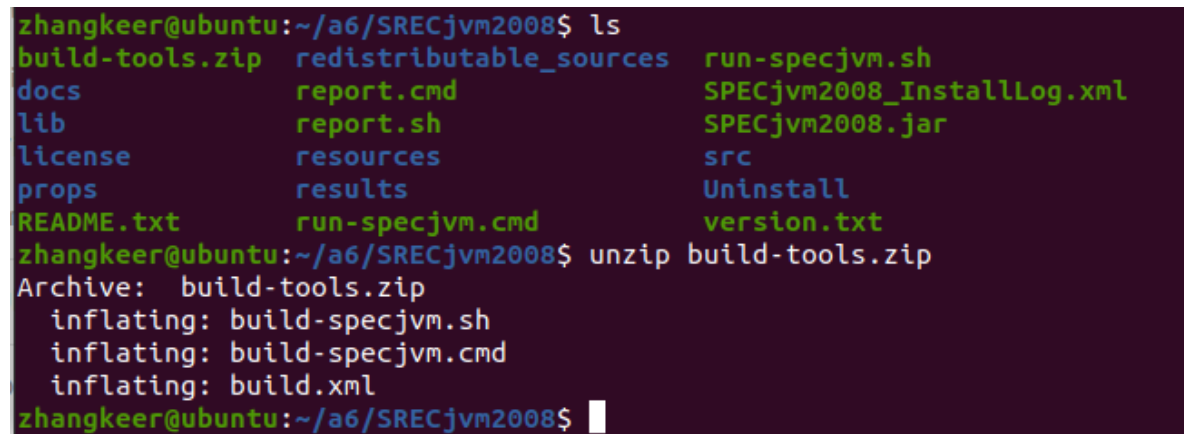
在第27行“-proc:none”，下一行增加“-nowarn”，



```
5 * Copyright (c) 2006 Sun Microsystems, Inc. All rights reserved.
6 *
7 * This source code is provided as is, without any express or implied warranty.
8 */
9
10 package spec.benchmarks.compiler.sunflow;
11
12 import java.io.File;
13 import java.io.FileWriter;
14 import java.io.IOException;
15 import java.util.ArrayList;
16
17 import spec.harness.results.BenchmarkResult;
18 import spec.benchmarks.compiler.Compiler;
19 import spec.benchmarks.compiler.MainBase;
20 import spec.benchmarks.compiler.Util;
21
22 public class Main extends spec.benchmarks.compiler.MainBase {
23
24     public Main(BenchmarkResult bmResult, int threadId) {
25         super(bmResult, threadId, 2);
26         String[] args = new String[] {
27             "-proc:none",
28             "-nowarn", |
29             "@" + srcsFile.getPath());
30         compiler = new Compiler(args);
31     }
32
33     public static void setupBenchmark() {
34         MainBase.preSetupBenchmark(Main.class);
35         srcsFile = getSrcFile(resDir, tmpDir);
36         new Main(new BenchmarkResult(), 1).harnessMain(true);
37     }
38
39     public void harnessMain() {
40         for (int i = 0; i < 3; i++) {
41             compiler.compile(loops);
42         }
43     }
44 }
```

保存退出。

(2)解压build-tools.zip文件



```
zhangkeer@ubuntu:~/a6/SRECjvm2008$ ls
build-tools.zip  redistributable_sources  run-specjvm.sh
docs             report.cmd              SPECjvm2008_InstallLog.xml
lib             report.sh              SPECjvm2008.jar
license         resources              src
props          results              Uninstall
README.txt     run-specjvm.cmd       version.txt
zhangkeer@ubuntu:~/a6/SRECjvm2008$ unzip build-tools.zip
Archive:  build-tools.zip
  inflating: build-specjvm.sh
  inflating: build-specjvm.cmd
  inflating: build.xml
zhangkeer@ubuntu:~/a6/SRECjvm2008$
```

(3)重新编译SPECjvm

```
bash ./build-specjvm.sh
```

```
zhangkeer@ubuntu: ~/a6/SRECjvm2008

[copy] Copying 2 files to /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008/props
[copy] Copying 145 files to /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008/resources
[copy] Copied 37 empty directories to 1 empty directory under /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008/resources
[copy] Copying 3 files to /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008/license
[copy] Copying 1 file to /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008
[copy] Copying 1 file to /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008
[copy] Copying 1 file to /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008
[copy] Copying 1 file to /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008
[copy] Copying 1 file to /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008
[zip] Building zip: /home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008/build-tools.zip

BUILD SUCCESSFUL
Total time: 8 seconds
zhangkeer@ubuntu:~/a6/SRECjvm2008$
```

(4)进入build/release/SPECjvm2008/目录，在此目录下执行测试任务

运行成功记录：

```
Nov 25 21:43
zhangkeer@ubuntu: ~/a6/SRECjvm2008/build/release/SPECjvm2008

Benchmark:  xml.transform
Run mode:   timed run
Test type:  multi
Threads:    2
Warmup:     120s
Iterations: 1
Run length: 240s

Warmup (120s) begins: Thu Nov 25 08:19:56 PST 2021
Warmup (120s) ends:   Thu Nov 25 08:21:17 PST 2021
Warmup (120s) result: 204.48 ops/m

Iteration 1 (240s) begins: Thu Nov 25 08:21:57 PST 2021
Iteration 1 (240s) ends:   Thu Nov 25 08:25:57 PST 2021
Iteration 1 (240s) result: 215.84 ops/m

Valid run!
Score on xml.transform: 215.84 ops/m

-----
Benchmark:  xml.validation
Run mode:   timed run
Test type:  multi
Threads:    2
Warmup:     120s
Iterations: 1
Run length: 240s

Warmup (120s) begins: Thu Nov 25 08:25:58 PST 2021
Warmup (120s) ends:   Thu Nov 25 08:27:58 PST 2021
Warmup (120s) result: 398.23 ops/m

Iteration 1 (240s) begins: Thu Nov 25 08:27:58 PST 2021
Iteration 1 (240s) ends:   Thu Nov 25 08:31:58 PST 2021
Iteration 1 (240s) result: 411.07 ops/m

Valid run!
Score on xml.validation: 411.07 ops/m

Results are stored in:
/home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008/results/SPECjvm2008.001/SPECjvm2008.001.raw
Generating reports in:
/home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008/results/SPECjvm2008.001

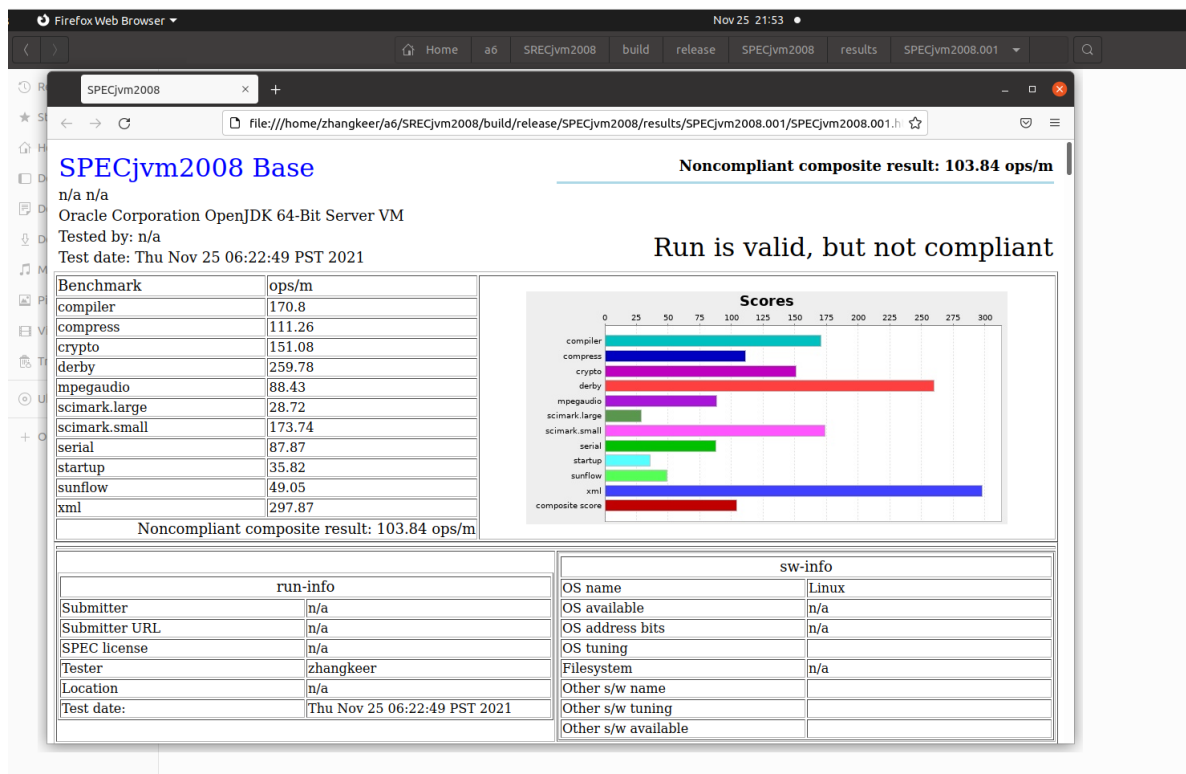
Noncompliant composite result: 103.84 ops/m
zhangkeer@ubuntu:~/a6/SRECjvm2008/build/release/SPECjvm2008$
```

Generating reports in:

/home/zhangkeer/a6/SRECjvm2008/build/release/SPECjvm2008/results/SPECjvm2008.001

可以打开html文件查看：





### 3. 查阅过去官方发布结果，对比最近任意一次发布结果（可从表格中任选一项与自己执行结果的差异，并尝试解释原因。

我选择了官方结果的这一项与我的执行结果进行比较：

Tester	System Name	Cores	Chips	Processor		Threads/ Core	JVM Name	JVM Version	Results	
				HW	Threads				Base	Peak
Sun Microsystems, Inc.	Sun Fire X4450	16	4		16	1	Java HotSpot(TM) 64-Bit Server VM on Solaris	1.6_0_06 Performance Release	260.08	--
Sun Microsystems	Sun Fire X4450	24	4		24	1	Java HotSpot(TM) 64-Bit Server VM	1.6_0_06 Performance Release	283.79	--
Sun Microsystems Inc.	Sun Blade X6270	8	2	(2 chips, 4 cores/chip, 2 threads/core)	16	2	Java HotSpot(TM) 64-Bit Server VM	1.6_0_14 Performance Release	317.13	--
Apple Inc.	iMac (Early 2009)	2	1		2	1	Java HotSpot(TM) Client VM	1.5_0_20-141 mixed mode	--	26.37
Apple Inc.	iMac (Early 2009)	2	1		2	1	Java HotSpot(TM) Client VM	14.1-b02-90 mixed mode	--	37.37
Apple Inc.	iMac (Early 2009)	2	1		2	1	Java HotSpot(TM) 64-Bit Server VM	14.1-b02-90 mixed mode	--	51.6
Oracle Corporation	SPARC T3-2	32	2		256	8	Java HotSpot(TM) 64-Bit Server VM on Solaris	1.6_0_21 Performance Release	--	320.52
Oracle Corporation	SPARC T4-2	16	2		128	8	Java HotSpot(TM) 64-Bit Server VM	22.0-b08 mixed mode	--	454.25
Oracle Corporation	Netra SPARC T4-2	16	2		128	8	Java HotSpot(TM) 64-Bit Server VM	22.0-b10 mixed mode	--	454.52
Huawei Technologies Co	RH 2285	12	2	(2 chips, 6 cores/chip, 2 threads/core)	24	2	Java HotSpot(TM) 64-Bit Server VM on Linux	Java(TM) SE Runtime Environment (build 1.7.0_02-b13)	335.78	--
Sugon	Sugon I620-G20(Intel Xeon E5-2660 v3)	20	2		40	2	OpenJDK 64-Bit Server VM	24.45-b08 mixed mode	853.15	--
Sugon	Sugon I840-G25(Intel Xeon E7-4830 v2)	40	4		40	1	OpenJDK Java 1.7.0 64-bit Server VM	24.45-b08 mixed mode	897.23	--

Last update: Wednesday, 30 September 2015, 12:32

选择这一项的原因是使用的JVM与我使用的比较相似，

官方使用的JVM为 OpenJDK 64-Bit Server VM, 24.45-b08 mixed mode

我使用的JVM为 OpenJDK 64-Bit Server VM, 25.40-b25 mixed mode

只有版本有略微差异。

(1)关于报告，首先最大的差异在于我的右上角有一行"Run is valid, but not compliant"，表示运行有效但不合规。原因是我使用了自己编译的SPECjvm2008，在修改Main.java文件的第27行“-proc:none”，下一行增加了“-nowarn”，。



需要这样修改的原因是，参考<https://zhuanlan.zhihu.com/p/258483799>，进程重定向了子进程的 stderr，但是却没有读取对应的数据，导致一旦出现大量的警告信息时，子进程对 stderr 的管道写入被堵塞，从而导致测试进程整个被停滞。

修改 Main.java 文件，在编译的时候加入 `-nowarn` 选项，可以让 `javac` 不输出警告信息，从而避免堵塞。而上述警告信息主要是因为 SPECjvm2008 内置的编译器与被测试 Java 的编译器版本不符。

虽然测试报告上会显示不合规，但我认为这对基准测试的结果影响不大。

(2) 报告的另一个明显差异在于，官方的 Base result 高达 853.15 ops/m，而我的 Base Result 只有 103.84 ops/m。

我认为产生这一差异的主要原因是硬件条件的不同。

在 Summary Report 中，可以看到官方使用的物理 CPU 个数以及每个物理 CPU 的核数分别是 2 和 20；

## SPECjvm2008 SPEC Summary Report

Tester	<a href="#">Sugon</a>
JVM Vendor	<a href="#">Red Hat, Inc.</a>
JVM	OpenJDK 64-Bit Server VM 24.45-b08 mixed mode
Hardware Company	<a href="#">Sugon</a>
Hardware Model	Sugon I620-G20(Intel Xeon E5-2660 v3)
Number of chips	2
Number of cores	20
Number of logical CPUs	40
Base result	853.15 SPECjvm2008 Base ops/m
Peak result	
Base report	<a href="#">html</a> , <a href="#">txt</a>
Peak report	

First published at SPEC.org on 23-Feb-2015

而我的虚拟机使用的物理 CPU 个数和每个物理 CPU 的核数分别是 2 和 1：

```
#查看物理CPU个数
cat /proc/cpuinfo | grep "physical id" | sort | uniq | wc -l
#查看每个CPU的核数
cat /proc/cpuinfo | grep "cpu cores" | uniq
```

```
zhangkeer@ubuntu:~/a6/SPECjvm2008/build/release/SPECjvm2008$ cat /proc/cpuinfo | grep "physical id" | sort | uniq | wc -l
2
zhangkeer@ubuntu:~/a6/SPECjvm2008/build/release/SPECjvm2008$ cat /proc/cpuinfo | grep "cpu cores" | uniq
cpu cores      : 1
zhangkeer@ubuntu:~/a6/SPECjvm2008/build/release/SPECjvm2008$
```

(3) 关于 Base Report 的具体差异：

与官方文件相比，我的 report 缺少了很多基础信息，比如：hw-info 几乎全部都是未知。

hw-info	
HW vendor	n/a
HW vendor's URL	n/a
HW model	n/a
HW available	n/a
CPU vendor	n/a
CPU vendor's URL	n/a
CPU name	n/a
CPU frequency	n/a
# of logical cpus	n/a
# of chips	n/a
# of cores	n/a
Cores per chip	n/a
Threads per core	n/a
Threading enabled	n/a
HW address bits	n/a
Primary cache	n/a
Secondary cache	n/a
Other cache	n/a
Memory size	n/a
Memory details	n/a
Other HW details	n/a

关于这些信息，user guide里有：

The system information in the results reports is categorized as general information about the run (such as time and date), information about the JRE on which the benchmark was run, information about the OS, and information about the hardware. All of the system information properties that SPECjvm2008 is capable of reporting are documented in the example reporter properties file (props/specjvm.reporter.properties).

官方报告和我的报告都包含了下列每一项测试：（按顺序列出，都没有出入）

compiler.compiler 在规定时间内，多线程迭代测试普通java编译，得出 ops/m  
 compiler.sunflow 在规定时间内，多线程迭代测试sunflow图像渲染，得出 ops/m  
 compress 在规定时间内，多线程迭代测试压缩，得出 ops/m  
 crypto.aes 在规定时间内，多线程迭代测试AES/DES加解密算法，得出 ops/m  
 crypto.rsa 在规定时间内，多线程迭代测试RSA加解密算法，得出 ops/m  
 crypto.signverify 在规定时间内，多线程迭代测试使用MD5withRSA，SHA1withRSA，SHA1withDSA，SHA256withRSA来签名，识别，得出 ops/m  
 derby 在规定时间内，迭代测试数据库相关逻辑，包括数据库锁，BigDecimal计算等，最后得出 ops/m  
 mpegaudio 在规定时间内，多线程迭代测试mpeg音频解码，得出 ops/m  
 scimark.fft.large 在规定时间内，多线程迭代测试快速傅立叶变换，使用32M大数据集，最后得出 ops/m  
 scimark.lu.large 在规定时间内，多线程迭代测试LU分解，使用32M大数据集，最后得出 ops/m  
 scimark.sor.large 在规定时间内，多线程迭代测试jacobi逐次超松弛迭代法，使用32M大数据集，最后得出 ops/m  
 scimark.sparse.large 在规定时间内，多线程迭代测试稀疏矩阵乘积，使用32M大数据集，最后得出 ops/m

scimark.monte\_carlo 在规定时间内，多线程迭代测试蒙特卡罗算法，得出 ops/m  
scimark.fft.small 在规定时间内，多线程迭代测试快速傅立叶变换，使用512K小数据集，最后得出 ops/m  
scimark.lu.small 在规定时间内，多线程迭代测试LU分解，使用512KB小数据集，最后得出 ops/m  
scimark.sor.small 在规定时间内，多线程迭代测试jacobi逐次超松弛迭代法，使用512KB小数据集，最后得出 ops/m  
scimark.sparse.small 在规定时间内，多线程迭代测试稀疏矩阵乘积，使用512KB小数据集，最后得出 ops/m  
serial 在规定时间内，多线程迭代测试通过socket传输java序列化对象到对端反序列化（基于jboss serialization benchmark），得出 ops/m  
startup.helloworld 测试helloworld程序从运行开始到结束所需的时间  
startup.compiler.compiler 普通java编译所需要的时间  
startup.compiler.sunflow 编译sunflow图像渲染引擎所需要的时间  
startup.compress 测试压缩程序，单次压缩所需的时间  
startup.crypto.aes 测试AES/DES加密算法，单次加解密所需的时间输入数据长度为 100 bytes ， 713KB  
startup.crypto.rsa测试RSA加密算法，单次加解密需要的时间输入数据长度为 100 bytes, 16KB  
startup.crypto.signverify 测试单次使用MD5withRSA, SHA1withRSA, SHA1withDSA, SHA256withRSA来签名，识别所需要的时间。输入数据长度为 1KB, 65KB, 1MB  
startup.mpegaudio 单次mpeg音频解码所需的时间  
startup.scimark.fft 单次快速傅立叶变换所需的时间  
startup.scimark.lu 单次LU分解所需的时间  
startup.scimark.monte\_carlo 单次运行蒙特卡罗算法所需的时间  
startup.scimark.sor 单次运行jacobi逐次超松弛迭代法所需的时间  
startup.scimark.sparse 单次稀疏矩阵乘积所需的时间  
startup.serial 单次通过socket传输java序列化对象到对端反序列化完成所需的时间（基于jboss serialization benchmark）  
startup.sunflow 单次图片渲染处理所需的时间  
startup.xml.transform 单次xml转换所需的时间，转换包括dom,sax,stream方式  
startup.xml.validation 单次xml schema校验所需的时间  
sunflow 在规定时间内，利用sunflow多线程迭代测试图片渲染，得出 ops/m  
xml.transform 在规定时间内，多线程迭代测试xml转换，得出 ops/m  
xml.validation 在规定时间内，多线程迭代测试xml schema验证，得出 ops/m

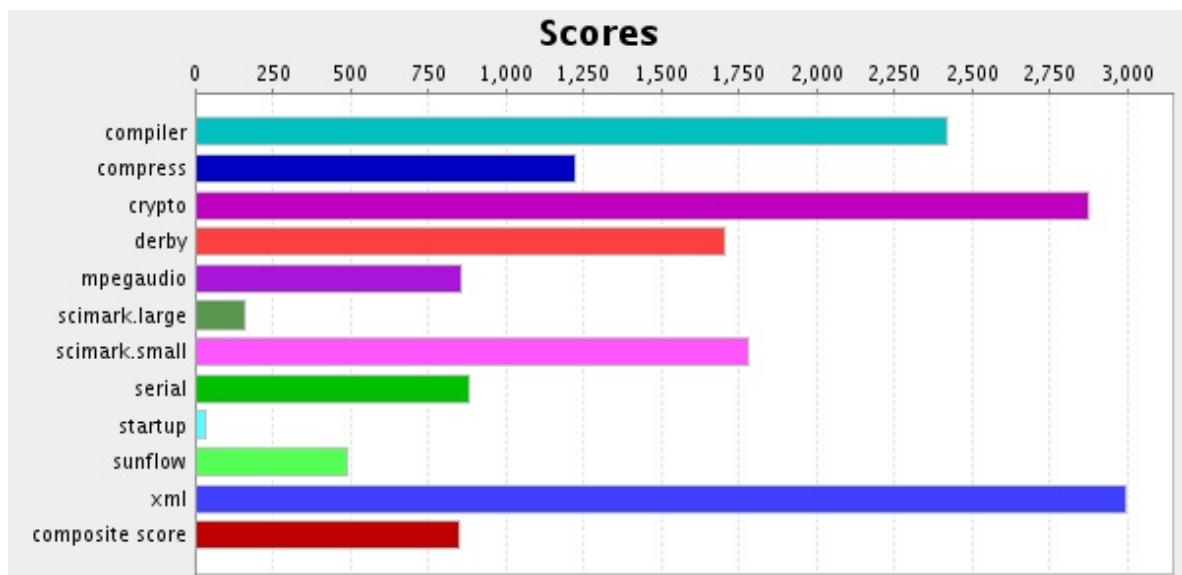
对于大部分不以startup开头的测试，我的和官方的ops/m的差异都在10倍左右（会有某一项高达30倍，也会有某一项低至三倍左右，但基本情况是：官方的ops/m是我的十倍）。仔细观察这些测试在测什么，会发现不以startup开头的大都是“多线程迭代测试”，CPU的核数相差十倍，性能也相差十倍，我认为这个解释是合理的。

而对于以startup开头的测试，我的ops/m与官方的结果基本相等，可见单次处理所需时间是基本一致的。

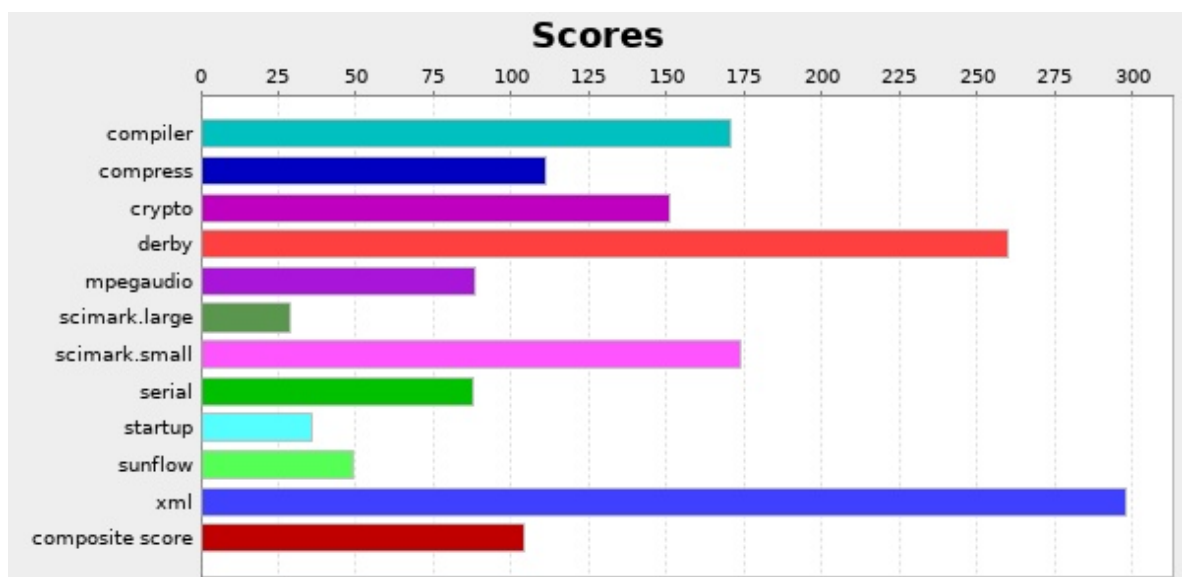
对于运行的所有测试，涉及多线程迭代的相差10倍左右，单次运行的差异不大，因此最终运行的base result相差8倍左右。

总体而言，各部分的得分差异也是类似的：

官方结果：



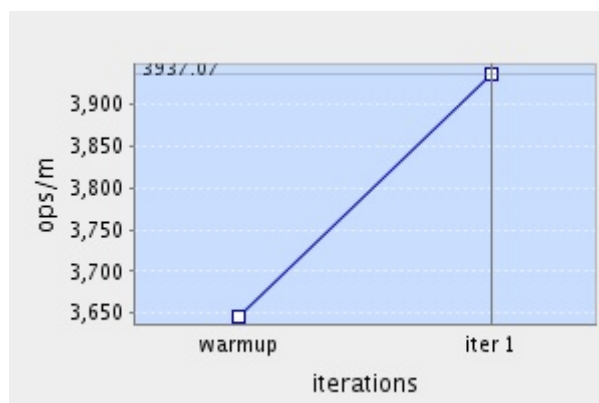
我的结果：



除此之外，对于每个测试的从warmup到iter1的ops/m的变化，大部分情况下该曲线也是类似的。

以compiler.compiler为例：

官方结果：



我的结果：

