# Опция -XX:+PrintCompilation

Java HotSpot JIT компилятор — устройство, мониторинг и настройка (часть 1) <https://habr.com/ru/post/536288/>

D:\Java\SberCourse\Practice\Task13\_JVMJITGC>java -XX:+PrintCompilation -classpath ./out/production/Task13\_JVMJITGC jit.Main

Пример результата работы программы

116 102 % 4 java.util.HashMap::resize @ 151 (356 bytes)

122 103 % 3 jit.Main::main @ 10 (61 bytes)

124 104 3 jit.Main::main (61 bytes)

125 73 2 java.lang.AbstractStringBuilder::append (55 bytes) made not entrant

127 72 2 java.lang.StringBuilder::append (8 bytes) made not entrant

132 105 4 java.util.HashMap::resize (356 bytes)

**timestamp** — (116) это время со старта JVM.

**compilation\_id** — (102) это внутренний ID задачи. Обычно он последовательно увеличивается в каждом сообщении, но иногда порядок может нарушаться. Это может произойти в случае, если существует несколько потоков компиляции, работающих параллельно.

**attributes** — (%) это набор из пяти символов, несущих дополнительную информацию о скомпилированном коде. Существуют следующие атрибуты:

* % — OSR (on-stack replacement) (need to switch to compiled method in the middle of interpreted method execution, big loops);
* s — метод является синхронизированным (synchronized);
* ! — метод содержит обработчик исключений;
* b — компиляция произошла в блокирующем режиме;
* n — скомпилированный метод является оберткой нативного метода.

made non entrant compilation was wrong/incomplete, no future callers will use this version

made zombie code is not in use and ready for GC

**tiered\_level** – (3) содержит номер уровня, на котором был скомпилирован код или может быть пустым, если многоуровневая компиляция выключена. Существует 5 уровней компиляции:

0 — интерпретируемый код;

1 — C1 с полной оптимизацией (без профилирования);

2 — C1 с учетом количества вызовов методов и итераций циклов;

3 — С1 с профилированием;

4 — С2.

C1 – компилятор java –client. Отличия 1) быстрая генерация кода приемлемого качества; 2) базовая оптимизация; 3) не требуется профилирование программы; 3) начало компиляции после 1500 вызовов метода.

C2 – компилятор java -server: 1) высоко оптимизированный код; 2) агрессивные методы оптимизации использующие профилирование программы; 3) начало компиляции после 10000 вызовов метода.

**method\_name** – (java.util.HashMap::resize @151) содержит название скомпилированного метода или название метода, содержащего скомпилированный цикл.

**size** – (356 bytes) содержит размер скомпилированного байт-кода, не размер полученного машинного кода. Размер указан в байтах.

**deopt** – (made not entrant) содержит название проведенной деоптимизации и может содержать такие сообщения как «made not entrant» и «made zombie». Появляется не в каждом сообщении.

<https://stackoverflow.com/questions/2930838/java-printcompilation-output-whats-the-meaning-of-made-not-entrant-and-made>

Zombie methods are methods whose code has been made invalid by class loading. Generally the server compiler makes aggressive inlining decisions of non-final methods. As long as the inlined method is never overridden the code is correct. When a subclass is loaded and the method overridden, the compiled code is broken for all future calls to it. The code gets declared "not entrant" (no future callers to the broken code), but sometimes existing callers can keep using the code. In the case of inlining, that's not good enough; existing callers' stack frames are "deoptimized" when they return to the code from nested calls (or just if they are running in the code). When no more stack frames hold PC's into the broken code it's declared a "zombie" - ready for removal once the GC gets around to it.

# Как победить latency

Что дает анализ компилятора

<https://algodma.wordpress.com/2020/12/31/%D0%BA%D0%B0%D0%BA-%D0%BF%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80%D0%B5%D1%82%D1%8C-%D1%87%D1%82%D0%BE-%D0%B8-%D0%BA%D0%B0%D0%BA-%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D1%82-jit/>

Когда код хорошо прогрет, т.е. весь [«горячий код» обработан JIT-компилятором](https://algodma.wordpress.com/2019/04/11/jit-%d0%ba%d0%be%d0%bc%d0%bf%d0%b8%d0%bb%d1%8f%d1%82%d0%be%d1%80-%d0%b8-%d0%ba%d0%b0%d0%ba-%d0%be%d0%bd-%d0%bd%d0%b0%d0%bc-%d0%bf%d0%be%d0%bc%d0%be%d0%b6%d0%b5%d1%82-%d0%bf%d0%be%d0%b1%d0%b5%d0%b4/) и превращен в машинный нативный код, эти строки перестают появляться в консоли. Это служит хорошим сигналом, что ваше приложение [достигло «нирваны»](https://algodma.wordpress.com/2019/04/18/%d0%b4%d0%be%d1%81%d1%82%d0%b8%d0%b6%d0%b5%d0%bd%d0%b8%d0%b5-4-%d0%be%d0%b3%d0%be-%d1%83%d1%80%d0%be%d0%b2%d0%bd%d1%8f-%d1%81-%d0%bf%d0%be%d0%bc%d0%be%d1%89%d1%8c%d1%8e-%d0%bf%d0%be%d0%b4%d0%be%d0%b3/) и работает на полной скорости.

JIT-компилятор и как он нам поможет победить latency

<https://algodma.wordpress.com/2019/04/11/jit-%d0%ba%d0%be%d0%bc%d0%bf%d0%b8%d0%bb%d1%8f%d1%82%d0%be%d1%80-%d0%b8-%d0%ba%d0%b0%d0%ba-%d0%be%d0%bd-%d0%bd%d0%b0%d0%bc-%d0%bf%d0%be%d0%bc%d0%be%d0%b6%d0%b5%d1%82-%d0%bf%d0%be%d0%b1%d0%b5%d0%b4/>