

Федеральное государственное автономное образовательное учреждение высшего образования «**Национальный исследовательский университет ИТМО**»

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа по дисциплине «Методы и средства программной инженерии» №4

Вариант: 1682

Преподаватель:
Абузов Ярослав Александрович

Выполнил: Васильченко Роман

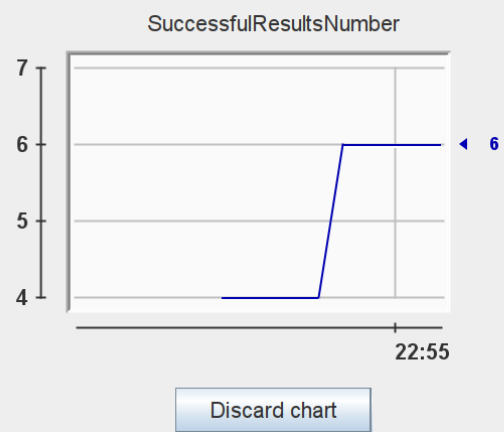
Группа: P32081

Санкт-Петербург, 2023г

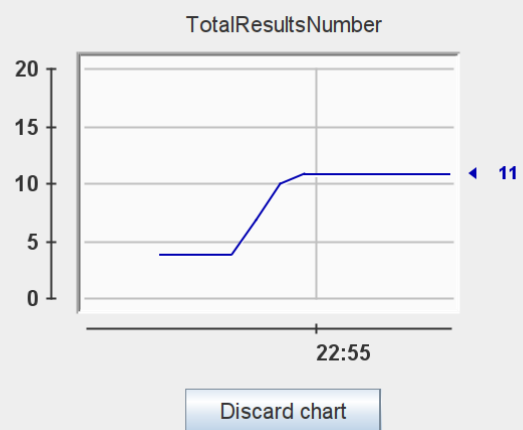
1. Задание
Показания JConsole

Attribute value	
SuccessfulResultsNumber	2
<input type="button" value="Refresh"/>	
MBeanAttributeInfo	
Name	Value
Attribute:	
Name	SuccessfulResultsNumber
Description	SuccessfulResultsNumber
Readable	true
Writable	false
Is	false
Type	int

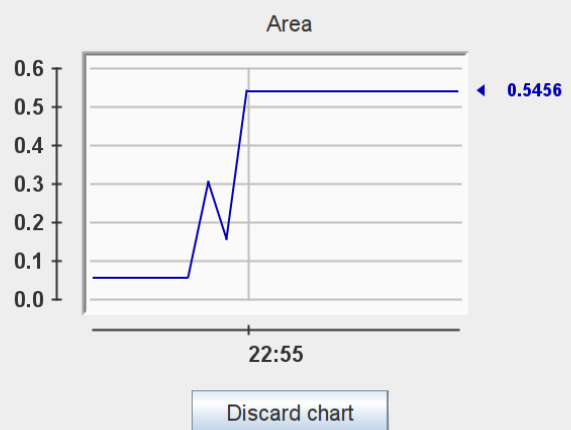
SuccessfulResultsNumber

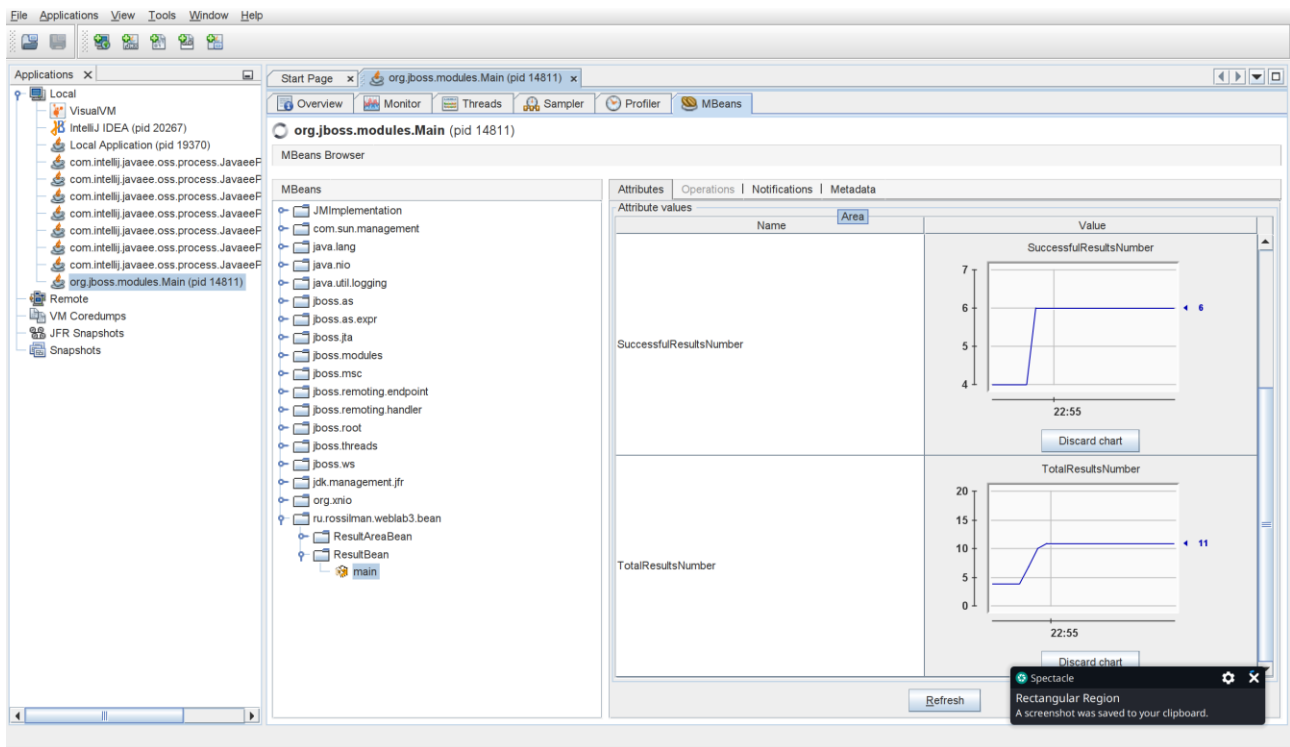


TotalResultsNumber



Area





org.jboss.modules.Main (pid 14811)

Sampler

Sample: CPU Mem... Stop

Status: CPU sampling in progress

CPU samples: Thread CPU time

Results: Statistics: Threads Count: 103 Total Time (CPU): 11,169 ms

Thread Dump

Name	Thread Time (CPU)	Thread Time (CPU) / sec
JFR Periodic Tasks	2,298 ms (20.6%)	2.57 ms (0.3%)
MSC service thread 1-7	1,385 ms (12.4%)	0.0 ms (0%)
DestroyJavaVM	849 ms (7.6%)	0.0 ms (0%)
RMI TCP Connection(9)-127.0.0.1	800 ms (7.2%)	39.9 ms (4%)
RMI TCP Connection(8)-127.0.0.1	743 ms (6.7%)	0.030 ms (0%)
ServerService Thread Pool -- 81	670 ms (6%)	0.0 ms (0%)
MSC service thread 1-3	628 ms (5.6%)	0.0 ms (0%)
RMI TCP Connection(10)-127.0.0.1	620 ms (5.6%)	0.0 ms (0%)
default task-1	598 ms (5.4%)	0.0 ms (0%)
MSC service thread 1-8	555 ms (5%)	0.0 ms (0%)
DeploymentScanner-threads - 1	346 ms (3.1%)	1.5 ms (0.1%)
MSC service thread 1-4	278 ms (2.5%)	0.0 ms (0%)
Attach Listener	204 ms (1.8%)	0.0 ms (0%)
MSC service thread 1-6	203 ms (1.8%)	0.0 ms (0%)
MSC service thread 1-5	168 ms (1.5%)	0.0 ms (0%)
MSC service thread 1-1	130 ms (1.2%)	0.0 ms (0%)
External Management Request Threads -- 1	73.2 ms (0.7%)	0.0 ms (0%)
management task-1	72.7 ms (0.7%)	0.0 ms (0%)
DeploymentScanner-threads - 2	50.7 ms (0.5%)	0.023 ms (0%)
MSC service thread 1-2	48.3 ms (0.4%)	0.0 ms (0%)
Weld Thread Pool -- 7	45.5 ms (0.4%)	0.0 ms (0%)
management I/O-2	34.6 ms (0.3%)	0.0 ms (0%)
Weld Thread Pool -- 4	33.3 ms (0.3%)	0.0 ms (0%)
Weld Thread Pool -- 6	20.6 ms (0.2%)	0.0 ms (0%)

Наибольший процент времени CPU занимает поток JFR Periodic Tasks.

JFR (Java Flight Recorder) - это инструмент для сбора диагностической и профилировочной информации о работающем приложении Java и JVM. "JFR Periodic Tasks" - это служебный поток, который используется JFR для выполнения периодических задач, таких как запись статистики и состояния JVM, в соответствии с конфигурацией профилирования.

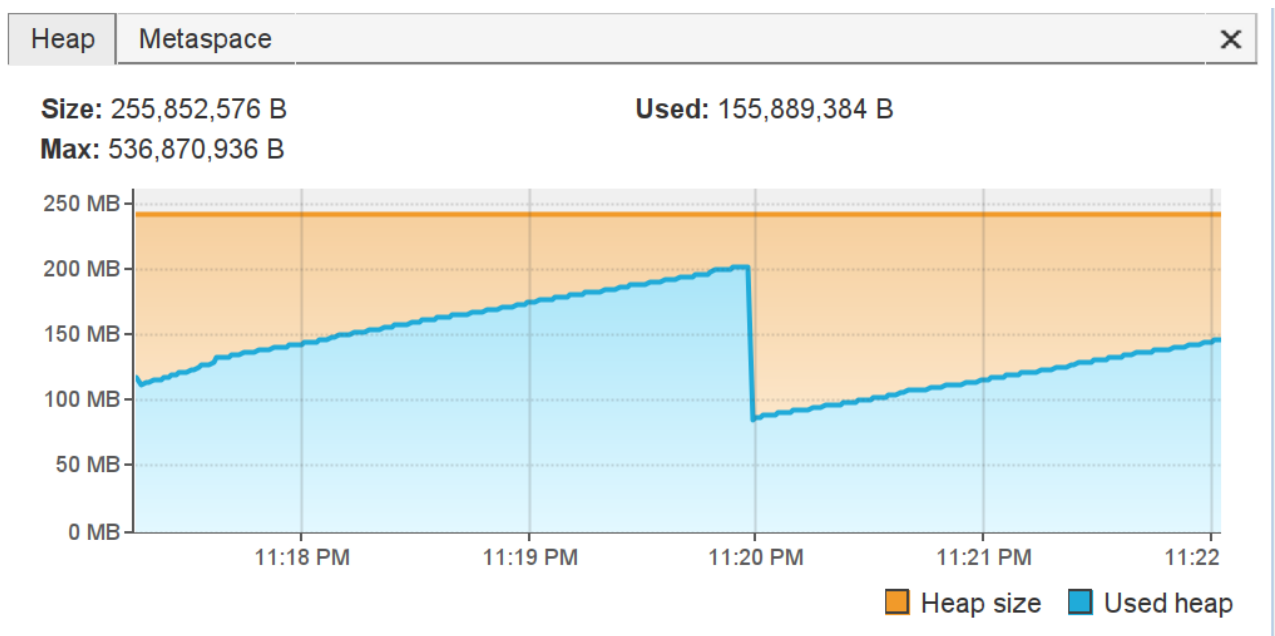
Отладка

Предполагаемая утечка памяти

```
@Override
public double getArea() throws InterruptedException {

    while(results.size() < 3) {
        java.lang.Thread.sleep(millis: 300);
        return getArea();
    }
}
```

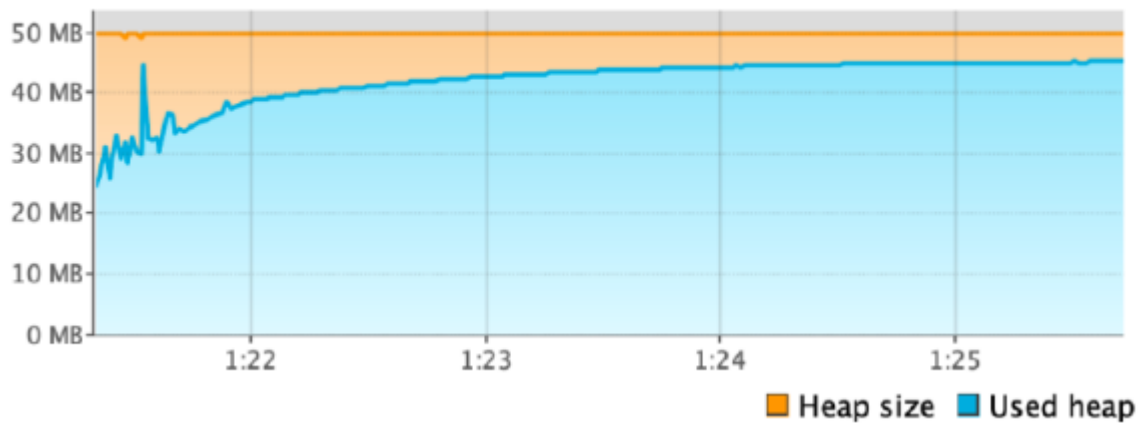
Рассмотрим график памяти, при sleep(0)



Видно, как память увеличивается, пока сборщик мусора не очищает ее. Однако при низких значениях выделенной памяти для Heap значения будут следующими

Size: 52 428 800 B
Max: 52 428 800 B

Used: 47 926 640 B



Что может вызвать ошибку переполнения. Также это можно заметить на Heap Histogram

Name	Live Bytes	Live Objects
byte[]	33,992,224 B (21.2%)	388,584 (10.7%)
java.lang.Object[]	25,463,040 B (15.9%)	656,457 (18.1%)
java.util.TreeMap\$Entry	16,479,320 B (10.3%)	411,983 (11.4%)
java.util.HashMap\$Node	8,339,296 B (5.2%)	260,603 (7.2%)
int[]	8,010,456 B (5%)	21,522 (0.6%)
java.lang.String	6,253,752 B (3.9%)	260,573 (7.2%)
java.util.HashMap\$Node[]	4,872,152 B (3%)	47,442 (1.3%)
org.jboss.jandex.MethodInternal	2,951,592 B (1.8%)	52,707 (1.5%)
java.util.TreeMap\$KeyIterator	2,697,344 B (1.7%)	84,292 (2.3%)
java.util.LinkedHashMap\$Entry	2,655,680 B (1.7%)	66,392 (1.8%)
java.lang.Class	2,392,568 B (1.5%)	19,913 (0.5%)
java.util.HashMap	2,047,968 B (1.3%)	42,666 (1.2%)
java.util.TreeMap	1,878,096 B (1.2%)	39,127 (1.1%)
char[]	1,752,680 B (1.1%)	5,524 (0.2%)
java.lang.management.ThreadInfo	1,588,808 B (1%)	15,277 (0.4%)
java.util.ArrayList	1,328,040 B (0.8%)	55,335 (1.5%)
java.util.LinkedHashMap	1,299,368 B (0.8%)	23,203 (0.6%)
java.io.jar.JarFile\$JarFileEntry	1,180,608 B (0.7%)	11,352 (0.3%)
java.io.SerializableCallbackContext	968,424 B (0.6%)	40,351 (1.1%)
java.lang.management.ManagementFactory\$DataSupport	970,888 B (0.6%)	38,337 (1.1%)

Решение проблемы:

В качестве решения можно не ожидать от пользователя нажатия на 3 кнопки для отображения площади, а просто выводить 0, пока не будут нажаты эти 3 кнопки.

```
@Override
public double getArea() {

    if (results.size() < 3) {
        return 0;
    }
}
```

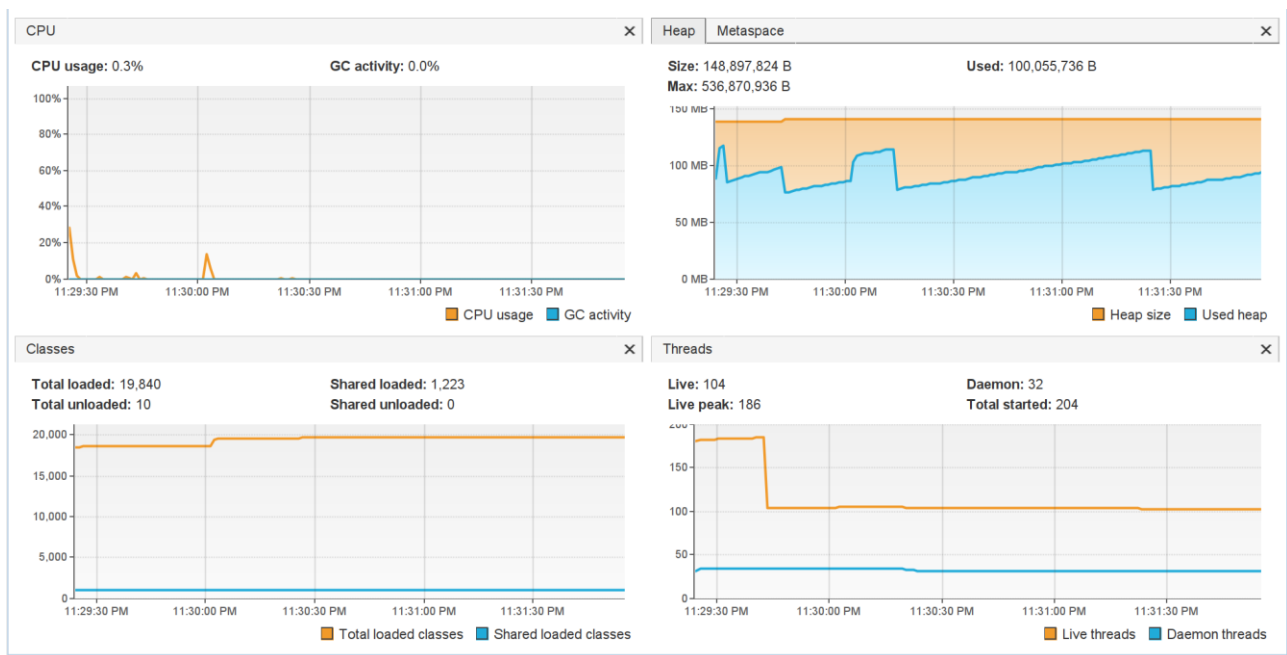
Также для дальнейшей оптимизации был просмотрен код и было выявлено следующее:

```
@Inject
ResultStorage storage;
final List<Result> resul
```

Использование storage в ResultAreaBean является нецелесообразным, так как мы получаем значения из ResultBean.

Проверка на дальнейшую утечку:

Для проверки был запущен проект и нажаты 2 кнопки, а также включена проверка на площадь, которая ранее выдавала увеличение памяти. Сейчас на картинке можно увидеть, что память используется более равномерно и очистки сборщика мусора происходят реже.



Следовательно утечка памяти была решена.

2. Вывод

В ходе выполнения этого проекта, я получил практический опыт работы с утилитами JConsole и VisualVM для мониторинга и профилирования Java-приложений. Это помогло мне разобраться в деталях работы JVM и научиться определять, какие компоненты приложения влияют на его производительность. Благодаря этому, я смог успешно локализовать и устранить проблемы, связанные с производительностью, на основе собранных данных и анализа.