

Университет ИТМО

Факультет ПИиКТ

Дисциплина: МиСПИ

Лабораторная работа №4.

Вариант 35

Выполнил:

Абульфатов Руслан Мехтиевич,

группа Р32312

Преподаватель:

Исаев Илья Владимирович

Задание:

Лабораторная работа №4

Внимание! У разных вариантов разный текст задания!

1. Для своей программы из лабораторной работы #3 по дисциплине "Веб-программирование" реализовать:

- MBean, считающий общее число установленных пользователем точек, а также число точек, попадающих в область. В случае, если количество установленных пользователем точек стало кратно 10, разработанный MBean должен отправлять оповещение об этом событии.
- MBean, определяющий процентное отношение "промахов" к общему числу кликов пользователя по координатной плоскости.

2. С помощью утилиты JConsole провести мониторинг программы:

- Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
- Определить количество классов, загруженных в JVM в процессе выполнения программы.

3. С помощью утилиты VisualVM провести мониторинг и профилирование программы:

- Снять график изменения показаний MBean-классов, разработанных в ходе выполнения задания 1, с течением времени.
- Определить имя потока, потребляющего наибольший процент времени CPU.

4. С помощью утилиты VisualVM и профилировщика IDE NetBeans, Eclipse или Idea локализовать и устранить проблемы с производительностью в программе.

По результатам локализации и устранения проблемы необходимо составить отчёт, в котором должна содержаться следующая информация:

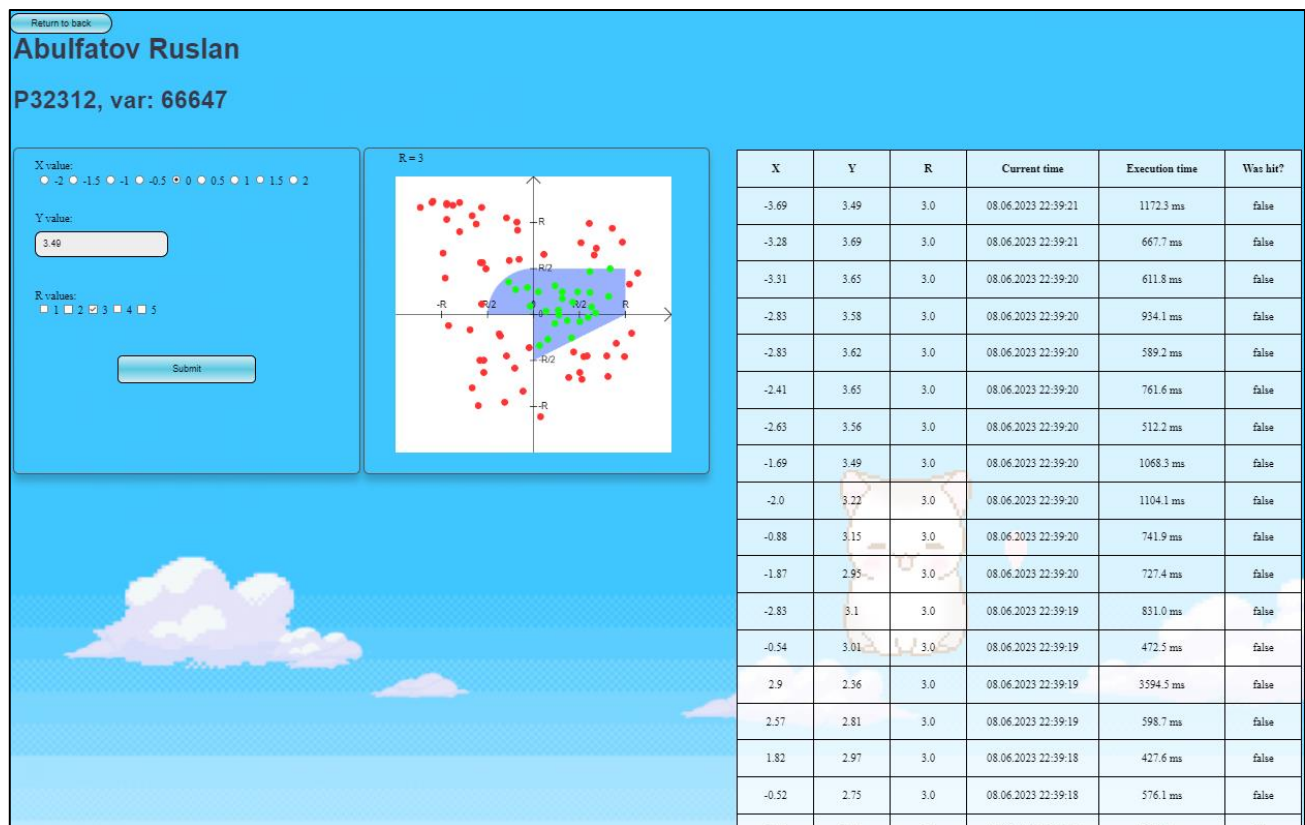
- Описание выявленной проблемы.
- Описание путей устранения выявленной проблемы.
- Подробное (со скриншотами) описание алгоритма действий, который позволил выявить и локализовать проблему.

Студент должен обеспечить возможность воспроизведения процесса поиска и локализации проблемы по требованию преподавателя.

Исходный код MBeans: -----

JConsole

Запущенное приложение на WildFly



Атрибуты MBean - CountPoints

Java Monitoring & Management Console - pid: 17132 jboss-modules.jar -mp C:\Users\User\Desktop\wildfly-28.0.1.Final\modules org.jboss.as.stand...

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

Attribute values

Name	Value
CountHitPoints	26
CountMissPoints	58
CountPoints	84
PercentMiss	0.6904762

org.jboss.mbeans

CountPoint

Attributes

- PercentMiss
- CountPoints
- CountHitPoints
- CountMissPoints

Operations

Notifications[2]

org.xnio

Уведомления MBean - CountPoints

Java Monitoring & Management Console - pid: 17132 jboss-modules.jar - mp C:\Users\User\Desktop\wildfly-28.0.1.Final\modules org.jboss.as.stand...

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

Notification buffer

TimeStamp	Type	UserData	SeqNum	Message	Event	Source
22:39:20:603	count_points		2	Count points is multiple of ten.	javax.management.Notificati...	mbeans:type=CountPoint
22:39:19:547	count_points		2	Count points is multiple of ten.	javax.management.Notificati...	mbeans:type=CountPoint

Tree view:

- JMImplementation
 - com.sun.management
 - java.lang
 - java.nio
 - java.util.logging
 - jboss.as
 - jboss.as.expr
 - jboss.jta
 - jboss.modules
 - jboss.msc
 - jboss.remoting.endpoint
 - jboss.remoting.handler
 - jboss.root
 - jboss.threads
 - jboss.vvs
 - jdk.management.jfr
 - mbeans
 - CountPoint
 - Attributes
 - PercentMiss
 - CountPoints
 - CountHitPoints
 - CountMissPoints
 - Operations
 - Notifications[2]
 - org.xnio

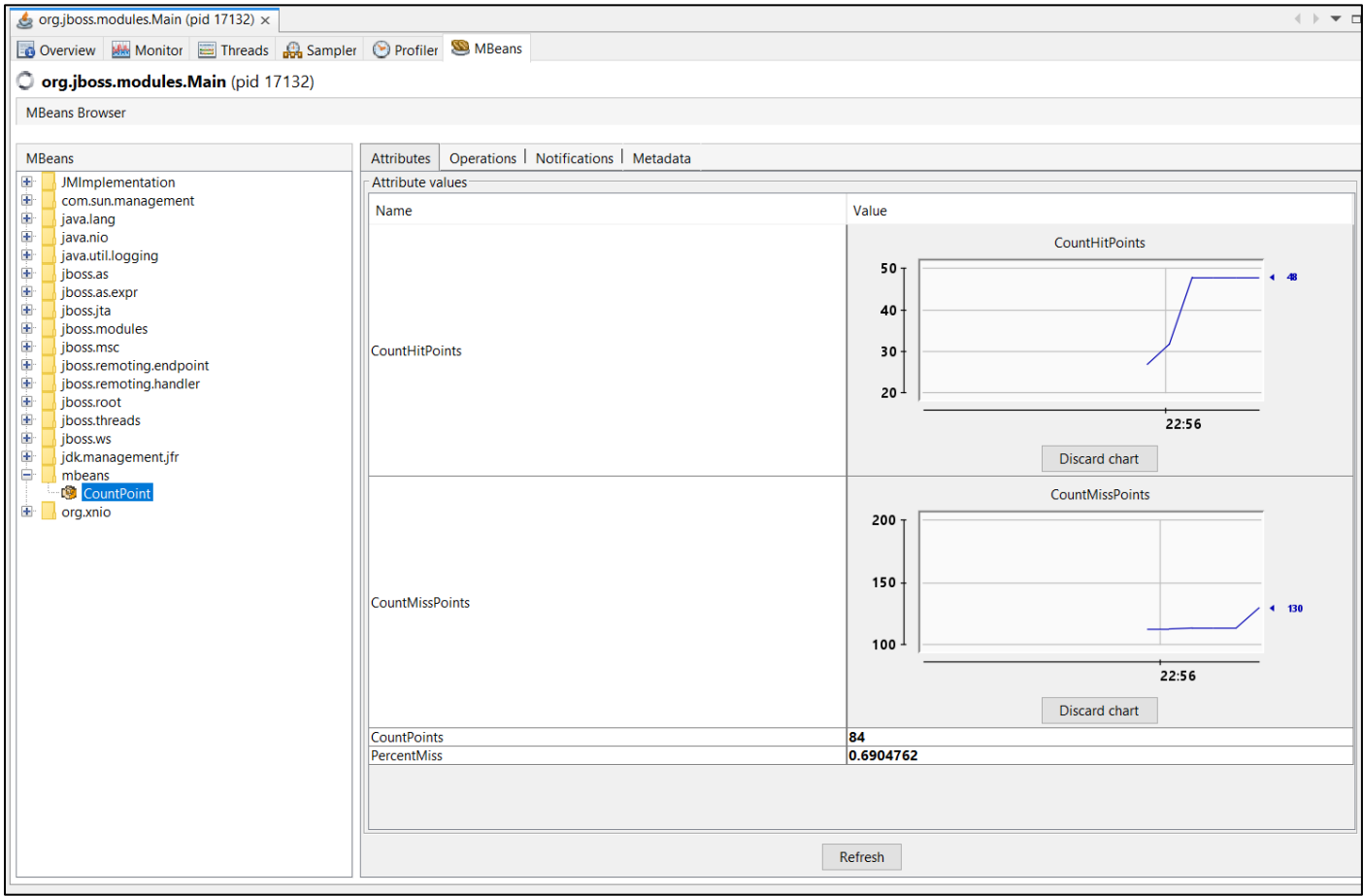
Вся информация о запущенном процессе WildFly

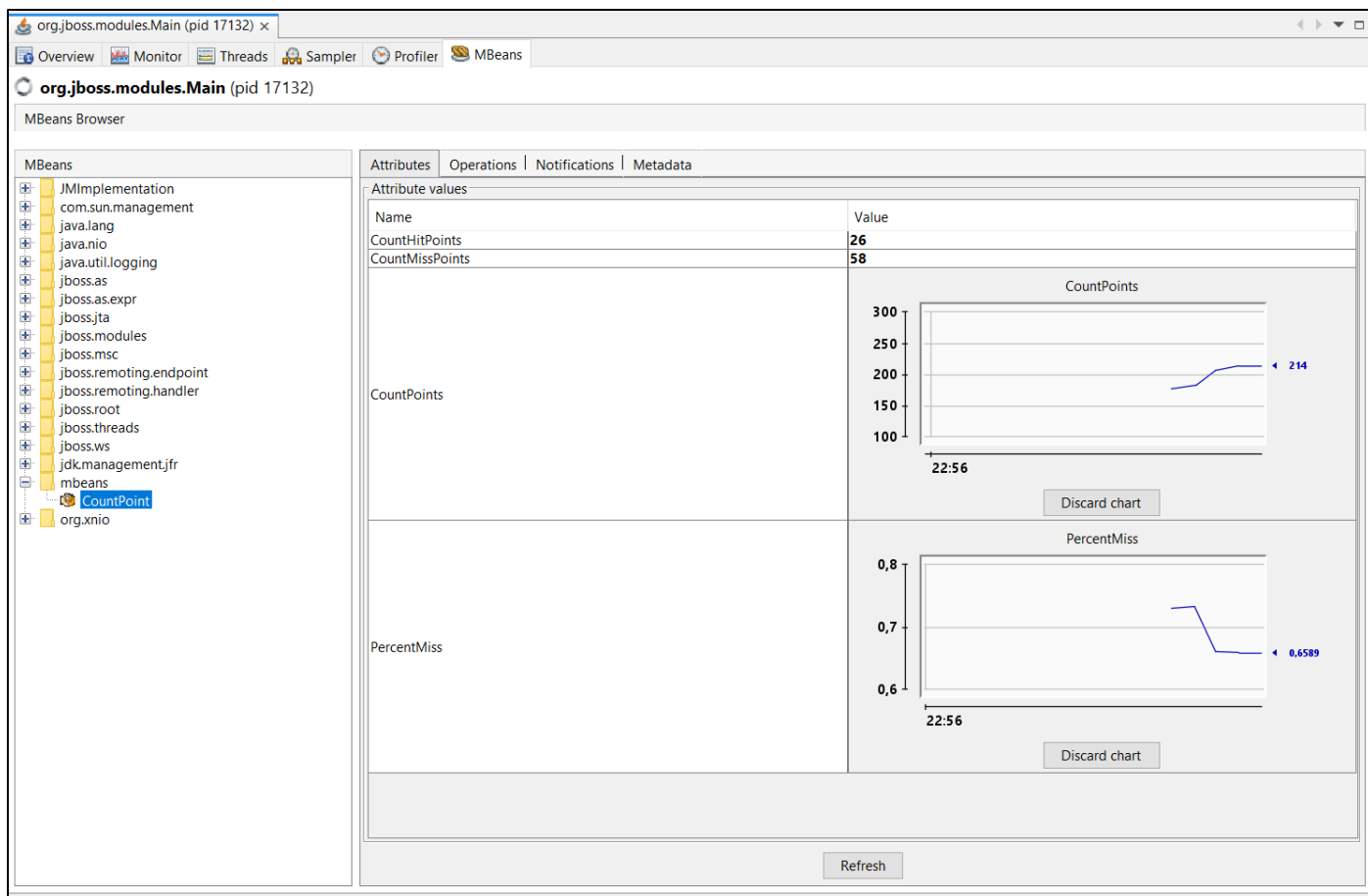
Connection name: pid: 17132 jboss-modules.jar -mp C:\Users\User\Desktop\wildfly-28.0.1.Final\modules org.jboss.as.standalone -Djboss.home.dir=C:\Users\User\Desktop\wildfly-28.0.1.Final		Uptime: 15 minutes
Virtual Machine: Java HotSpot(TM) 64-Bit Server VM version 20.0.1+9-29		Process CPU time: 1 minute
Vendor: Oracle Corporation		JIT compiler: HotSpot 64-Bit Tiered Compilers
Name: 17132@DESKTOP-NHEUAE9		Total compile time: 49,812 seconds
Live threads: 73 Peak: 155 Daemon threads: 24 Total threads started: 171	Current classes loaded: 18 567 Total classes loaded: 18 753 Total classes unloaded: 186	
Current heap size: 66 682 kbytes Maximum heap size: 524 288 kbytes		Committed memory: 124 928 kbytes Pending finalization: 0 objects
Garbage collector: Name = 'G1 Young Generation', Collections = 36, Total time spent = 0,801 seconds Garbage collector: Name = 'G1 Concurrent GC', Collections = 20, Total time spent = 1,018 seconds Garbage collector: Name = 'G1 Old Generation', Collections = 0, Total time spent = 0,000 seconds		
Operating System: Windows 10 10.0 Architecture: amd64 Number of processors: 8 Committed virtual memory: 431 368 kbytes		Total physical memory: 14 611 768 kbytes Free physical memory: 384 728 kbytes Total swap space: 31 222 452 kbytes Free swap space: 3 544 056 kbytes
VM arguments: -Dprogram.name=standalone.bat -Xms64M -Xmx512M -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256m -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true --add-exports=java.desktop/sun.awt=ALL-UNNAMED --add-exports=java.naming/com.sun.jndi.idap=ALL-UNNAMED --add-exports=java.naming/com.sun.jndi.url.idap=ALL-UNNAMED --add-exports=java.naming/com.sun.jndi.url.idaps=ALL-UNNAMED --add-exports=jdk.naming.dns/com.sun.jndi.dns=ALL-UNNAMED --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UNNAMED --add-opens=java.base/java.lang.reflect=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.base/java.net=ALL-UNNAMED --add-opens=java.base/java.security=ALL-UNNAMED --add-opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-opens=java.management/javax.management=ALL-UNNAMED --add-opens=java.naming/javax.naming=ALL-UNNAMED -Djava.security.manager=allow -Dorg.jboss.boot.log.file=C:\Users\User\Desktop\wildfly-28.0.1.Final\standalone\log\server.log -Dlogging.configuration=file:C:\Users\User\Desktop\wildfly-28.0.1.Final\standalone\configuration\logging.properties Class path: C:\Users\User\Desktop\wildfly-28.0.1.Final\jboss-modules.jar Library path: C:\Program Files\Java\jdk-20\bin;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0;C:\WINDOWS\System32\OpenSSH;C:\Program Files\Git\cmd;C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files\PuTTY;C:\Program Files\nodejs;C:\Program Files\GTK3-Runtime Win64\bin;C:\Program Files\Docker\Docker\resources\bin;C:\Users\User\Desktop\apache-ant-1.10.13\bin;C:\Users\User\AppData\Local\Microsoft\WindowsApps;C:\Program Files\JetBrains\PyCharm Community Edition 2022.1.3\bin;C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.1.3\bin;C:\Users\User\AppData\Roaming\npm;C:\Program Files\Azure Data Studio\bin;C:\Users\User\AppData\Local\Programs\Python\Python310\Scripts;C:\Program Files\PostgreSQL\14\bin;C:\Users\User\Desktop\apache-maven-4.0.0-alpha-5\bin;C:\Program Files\Java\jdk-20\bin;.		
Boot class path: Unavailable		

Количество загруженный классов – 18567 и 186 не загружены. Всего было загружено – 18753 класса.

VisualVM:

Отслеживание переменных из созданных MBean с течением времени.





Имя потока, потребляющего наибольший процент времени CPU.

org.jboss.modules.Main (pid 17132) x

Overview Monitor Threads Sampler Profiler MBeans

org.jboss.modules.Main (pid 17132)

Sampler

Settings

Sample: CPU Memory Stop

Status: CPU sampling in progress

CPU samples Thread CPU time

Results: Statistics: Threads Count: 75 Total Time (CPU): 37 359 ms

Thread Dump

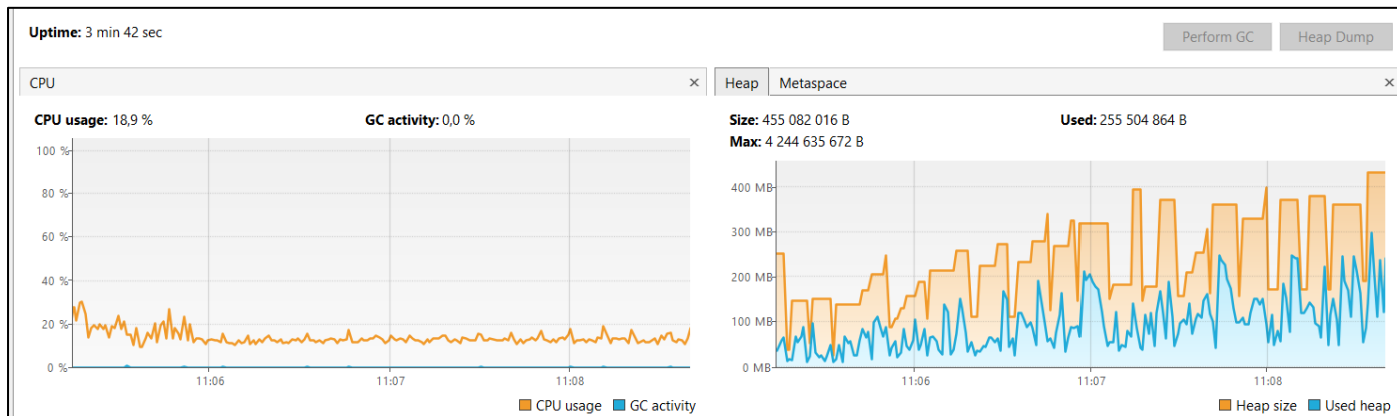
Name	Thread Time (CPU)	Thread Time (CPU) / sec
default task-1	10 937 ms (29,3 %)	0,0 ms (0 %)
RMI TCP Accept-0	4 484 ms (12 %)	0,0 ms (0 %)
RMI TCP Connection(13)-26.212.163.107	4 187 ms (11,2 %)	15,5 ms (1,6 %)
RMI TCP Connection(6)-26.212.163.107	4 125 ms (11 %)	0,0 ms (0 %)
DestroyJavaVM	2 234 ms (6 %)	0,0 ms (0 %)
RMI TCP Connection(14)-26.212.163.107	1 406 ms (3,8 %)	31,1 ms (3,1 %)
MSC service thread 1-6	1 203 ms (3,2 %)	0,0 ms (0 %)
MSC service thread 1-2	1 156 ms (3,1 %)	0,0 ms (0 %)
MSC service thread 1-8	1 140 ms (3,1 %)	0,0 ms (0 %)
MSC service thread 1-1	1 078 ms (2,9 %)	0,0 ms (0 %)
ServerService Thread Pool -- 82	953 ms (2,6 %)	0,0 ms (0 %)
MSC service thread 1-3	890 ms (2,4 %)	0,0 ms (0 %)
DeploymentScanner-threads - 2	843 ms (2,3 %)	0,0 ms (0 %)
Attach Listener	593 ms (1,6 %)	0,0 ms (0 %)
MSC service thread 1-5	437 ms (1,2 %)	0,0 ms (0 %)
DeploymentScanner-threads - 1	390 ms (1 %)	0,0 ms (0 %)
MSC service thread 1-7	343 ms (0,9 %)	0,0 ms (0 %)
MSC service thread 1-4	109 ms (0,3 %)	0,0 ms (0 %)
Weld Thread Pool -- 2	109 ms (0,3 %)	0,0 ms (0 %)
JFR Periodic Tasks	109 ms (0,3 %)	0,0 ms (0 %)
Periodic Recovery	78,1 ms (0,2 %)	0,0 ms (0 %)
ServerService Thread Pool -- 1	62,5 ms (0,2 %)	0,0 ms (0 %)
default I/O-12	62,5 ms (0,2 %)	0,0 ms (0 %)

VisualVM + Idea:

1. Описание выявленной проблемы

За 4 минуты работы программы, были сняты следующие показатели:

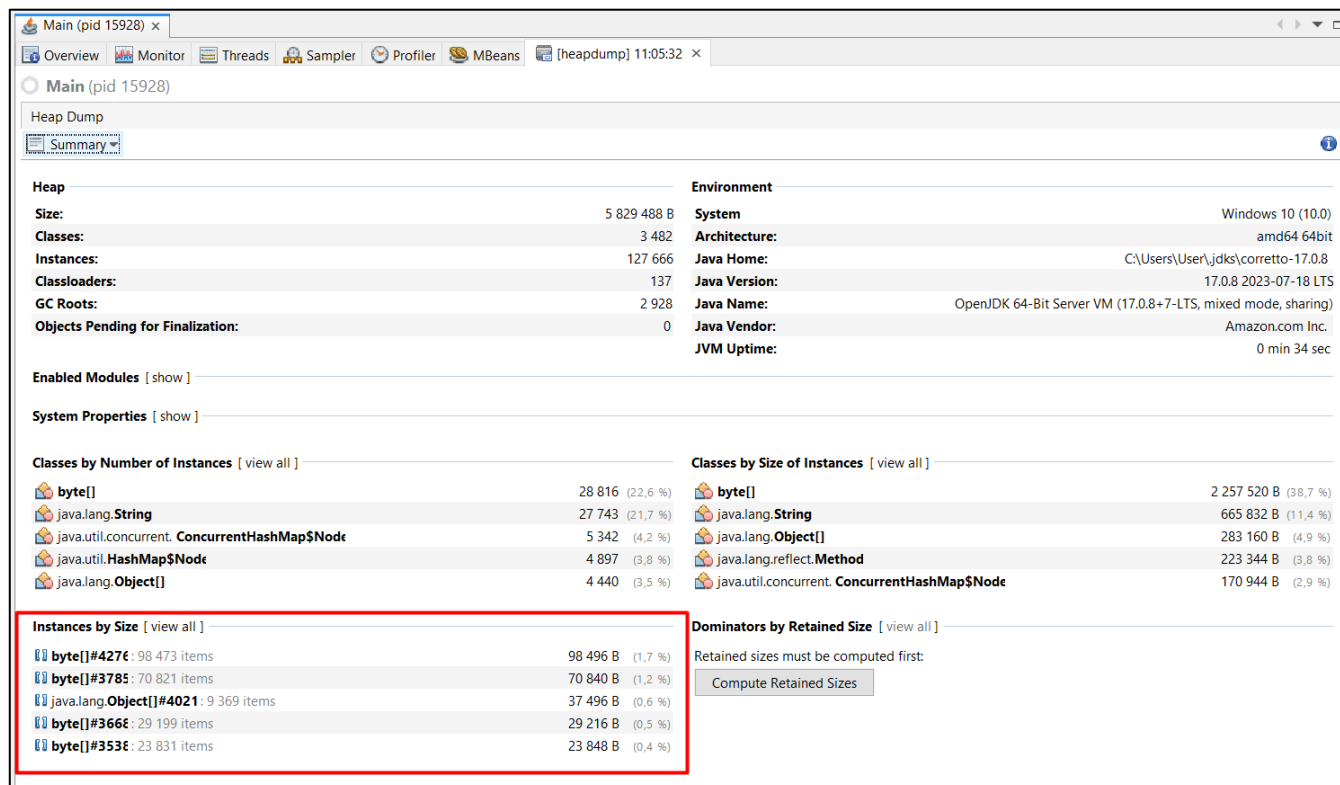
- нагрузка на процессор стабильна
- нагрузка по памяти увеличивается



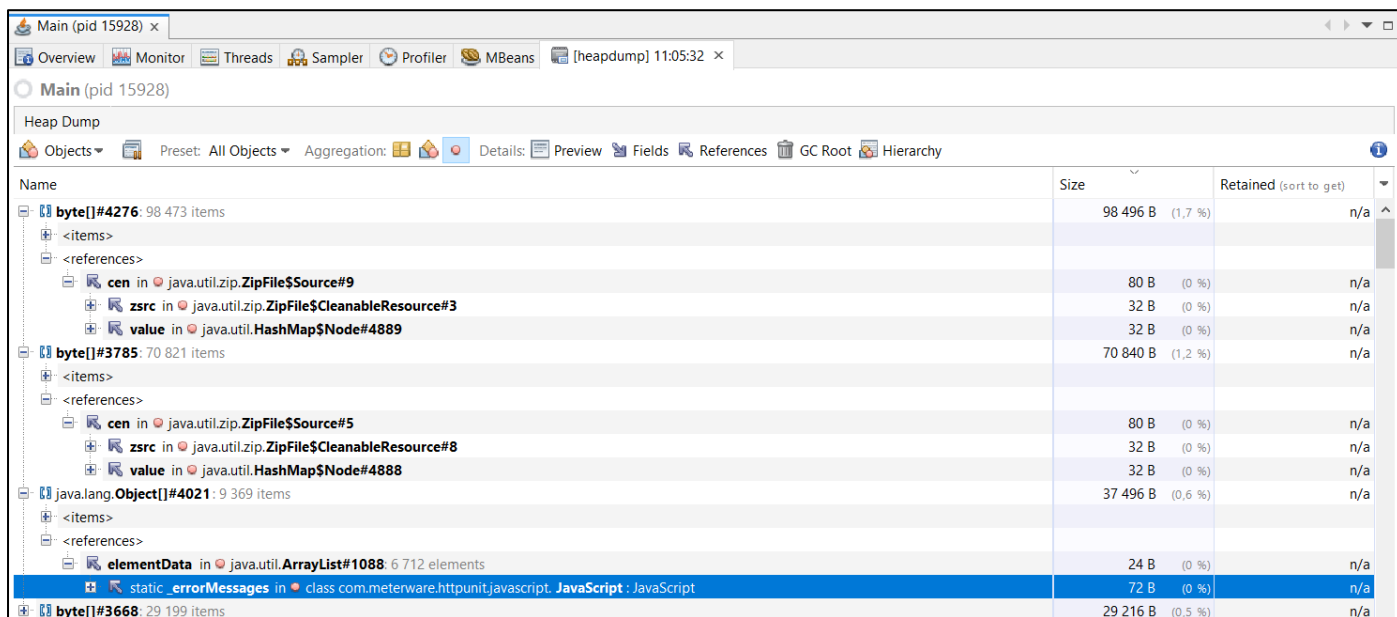
Из этого можно предположить, что идет некорректное взаимодействие с памятью при работе программы.

2. Описание путей устранения проблемы

После остановки программы, посмотрим собранную статистику, чтобы понять каких объектов было больше в памяти.

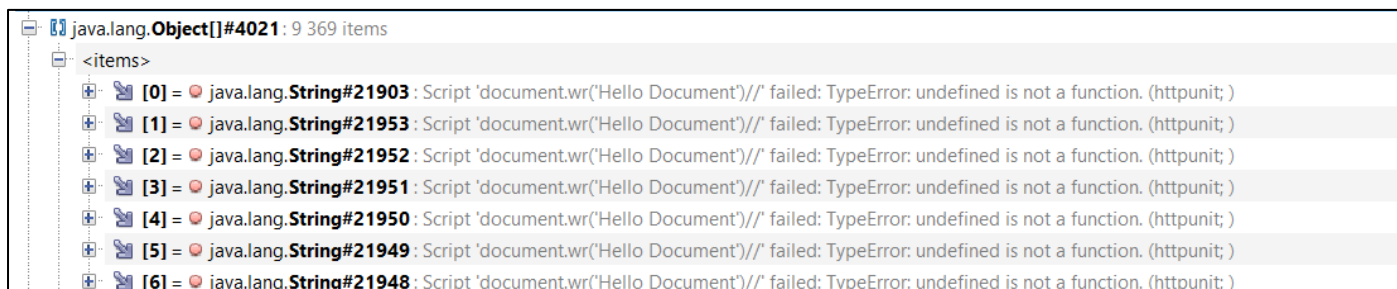


Попытаемся найти что нагружает память. Самый большой размер у переменной `_errorMessages` в классе `com.meterware.httpunit.javascript.JavaScript`.



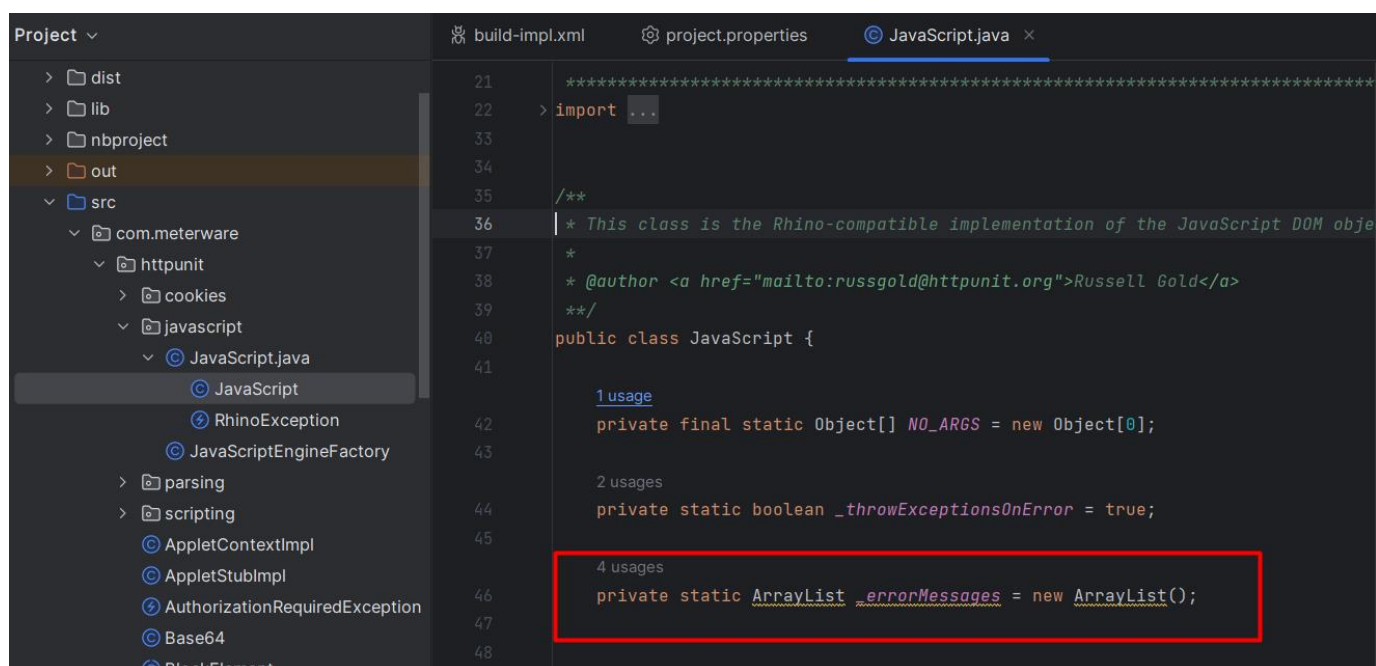
Name	Size	Retained (sort to get)
byte[] #4276: 98 473 items	98 496 B (1,7 %)	n/a
cen in java.util.zip.ZipFile\$Source#9	80 B (0 %)	n/a
zsrc in java.util.zip.ZipFile\$CleanableResource#3	32 B (0 %)	n/a
value in java.util.HashMap\$Node#4889	32 B (0 %)	n/a
byte[] #3785: 70 821 items	70 840 B (1,2 %)	n/a
cen in java.util.zip.ZipFile\$Source#5	80 B (0 %)	n/a
zsrc in java.util.zip.ZipFile\$CleanableResource#8	32 B (0 %)	n/a
value in java.util.HashMap\$Node#4888	32 B (0 %)	n/a
java.lang.Object[] #4021: 9 369 items	37 496 B (0,6 %)	n/a
elementData in java.util.ArrayList#1088: 6 712 elements	24 B (0 %)	n/a
static _errorMessages in class com.meterware.httpunit.javascript. JavaScript : JavaScript	72 B (0 %)	n/a
byte[] #3668: 29 199 items	29 216 B (0,5 %)	n/a

Если посмотреть данные из вкладки `items`, то можно сделать вывод, что данная переменная используется для накопления логов ошибок во время работы программы.



Index	Value
[0]	java.lang.String#21903: Script 'document.wr('Hello Document')/' failed: TypeError: undefined is not a function. (httpunit;)
[1]	java.lang.String#21953: Script 'document.wr('Hello Document')/' failed: TypeError: undefined is not a function. (httpunit;)
[2]	java.lang.String#21952: Script 'document.wr('Hello Document')/' failed: TypeError: undefined is not a function. (httpunit;)
[3]	java.lang.String#21951: Script 'document.wr('Hello Document')/' failed: TypeError: undefined is not a function. (httpunit;)
[4]	java.lang.String#21950: Script 'document.wr('Hello Document')/' failed: TypeError: undefined is not a function. (httpunit;)
[5]	java.lang.String#21949: Script 'document.wr('Hello Document')/' failed: TypeError: undefined is not a function. (httpunit;)
[6]	java.lang.String#21948: Script 'document.wr('Hello Document')/' failed: TypeError: undefined is not a function. (httpunit;)

Найдем данную переменную в коде.



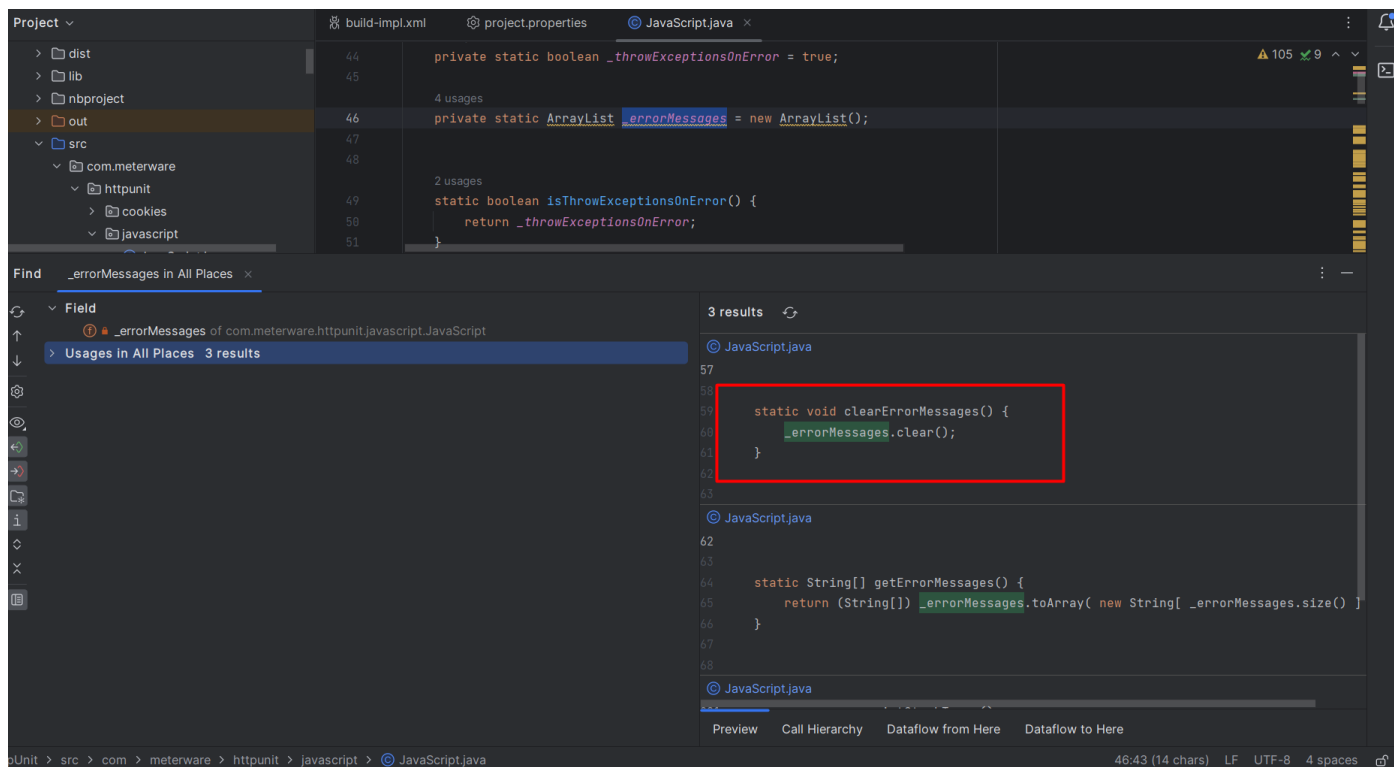
```
Project > build-impl.xml project.properties JavaScript.java x
> dist
> lib
> nbproject
> out
> src
  > com.meterware
    > httpunit
      > cookies
      > javascript
        > JavaScript.java
          > JavaScript
            > RhinoException
            > JavaScriptEngineFactory
          > parsing
          > scripting
            > AppletContextImpl
            > AppletStubImpl
            > AuthorizationRequiredException
            > Base64
            > BlockElement

21 *****
22 > import ...
33
34
35 /**
36 * This class is the Rhino-compatible implementation of the JavaScript DOM object
37 *
38 * @author <a href="mailto:russgold@httpunit.org">Russell Gold</a>
39 */
40 public class JavaScript {
41
42     1 usage
43     private final static Object[] NO_ARGS = new Object[0];
44
45     2 usages
46     private static boolean _throwExceptionsOnError = true;
47
48     4 usages
49     private static ArrayList _errorMessages = new ArrayList();
50 }
```

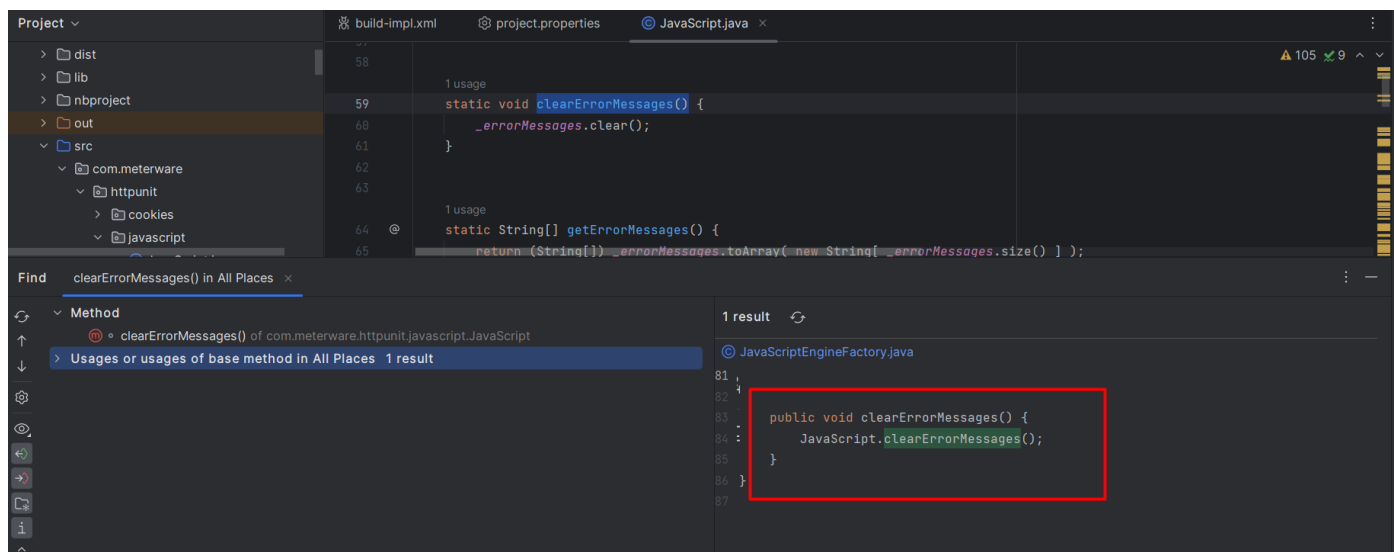
Она является статичной, следовательно существует все время, и не может быть очищена автоматически, только вручную.

3. Описание алгоритма решения проблемы

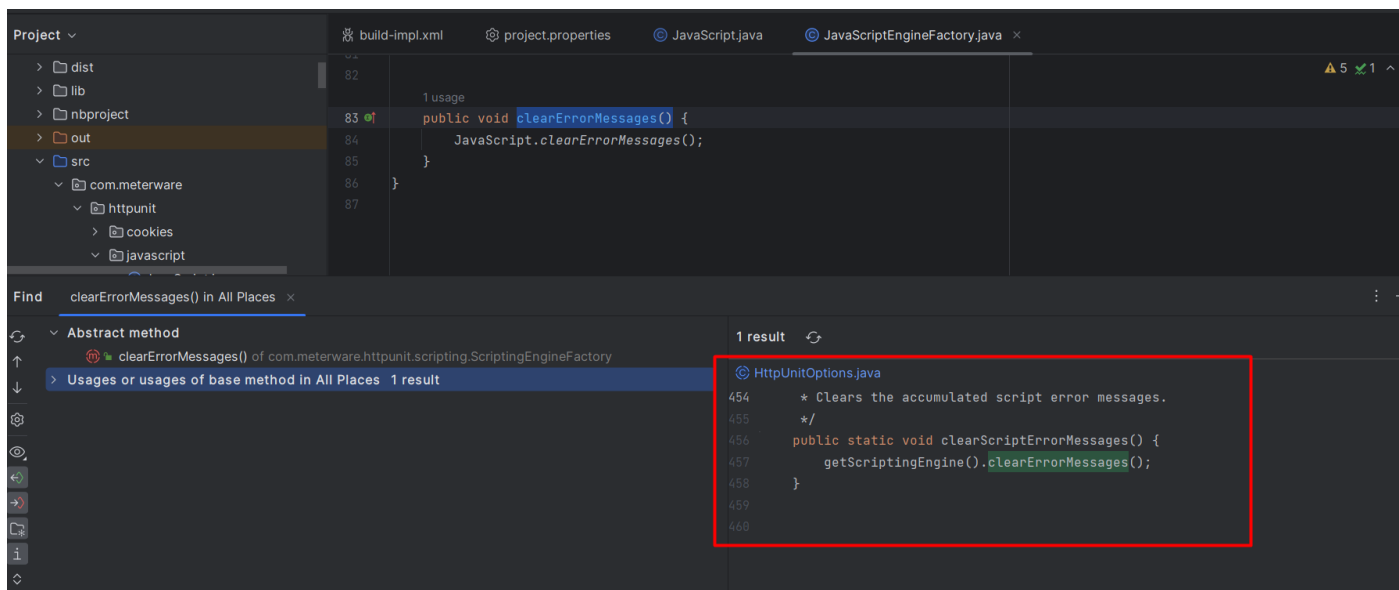
Посмотрим где в данном классе используется переменная `_errorMessages`.



Она используется в 3х местах, и даже есть метод для удаления всех элементов списка. Посмотрим где вызывается он.



Теперь он используется только в одном месте. Исследуем дальше.



Он также используется только в одном месте.



Благодаря IDE мы сразу можем увидеть, что данный метод нигде не вызывается.

Добавим принудительную очистку в главный метод.

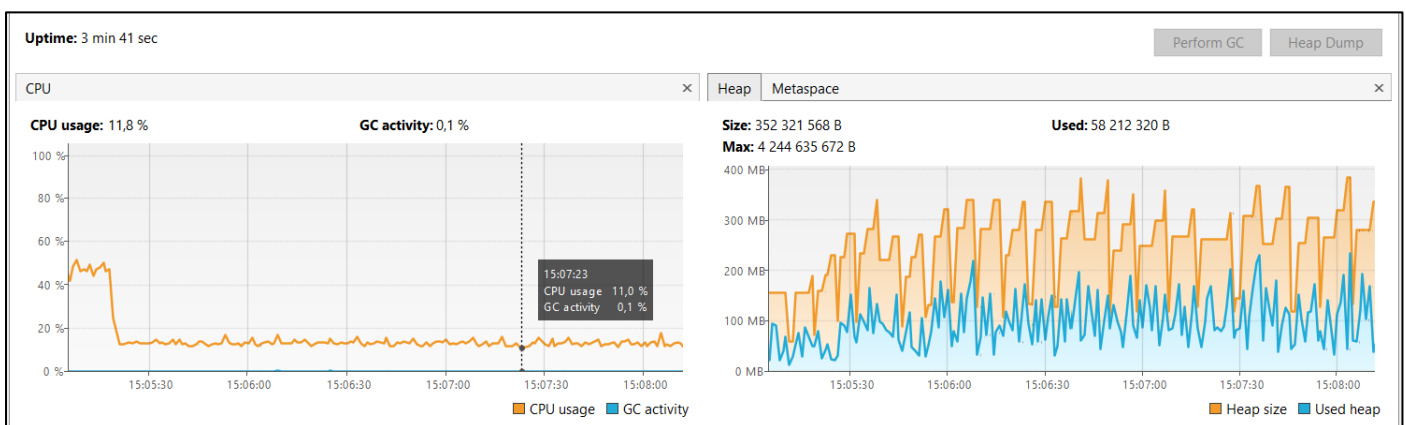
```

public static void main(String[] args) {
    try {
        HttpUnitOptions.setExceptionsThrownOnScriptError(false);
        ServletRunner sr = new ServletRunner();
        sr.registerServlet( resourceName: "myServlet", HelloWorld.class.getName());
        ServletUnitClient sc = sr.newClient();
        int number = 1;
        WebRequest request = new GetMethodWebRequest( urlString: "http://test.meterware.com/myServlet");
        while (true) {
            WebResponse response = sc.getResponse(request);
            System.out.println("Count: " + number++ + response);
            java.lang.Thread.sleep( millis: 200);

            HttpUnitOptions.clearScriptErrorMessages();
        }
    } catch (InterruptedException ex) {
        Logger.getLogger( name: "global").log(Level.SEVERE, msg: null, ex);
    } catch (MalformedURLException ex) {
        Logger.getLogger( name: "global").log(Level.SEVERE, msg: null, ex);
    } catch (IOException ex) {
        Logger.getLogger( name: "global").log(Level.SEVERE, msg: null, ex);
    } catch (SAXException ex) {
        Logger.getLogger( name: "global").log(Level.SEVERE, msg: null, ex);
    }
}

```

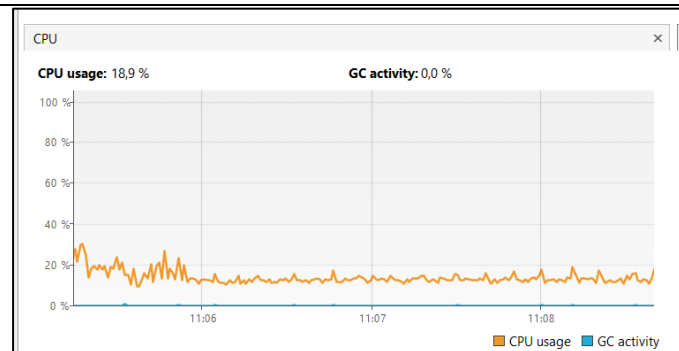
Проверим новый данные в VisualVM.



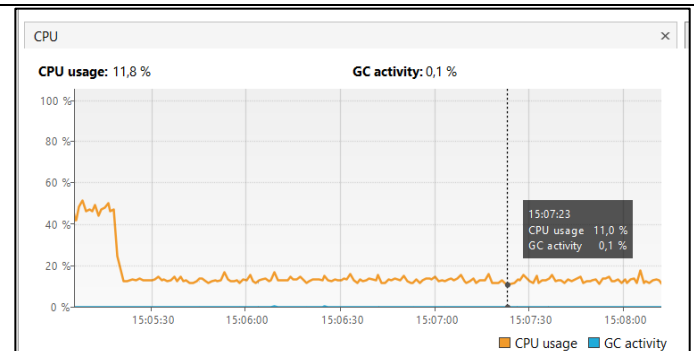
Проведем сравнение старых и новых показателей.

CPU.

Было

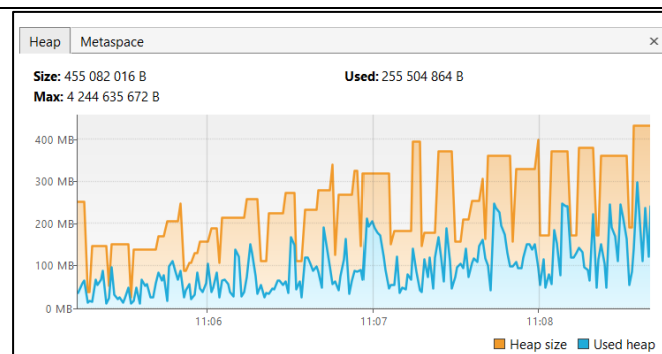


Стало

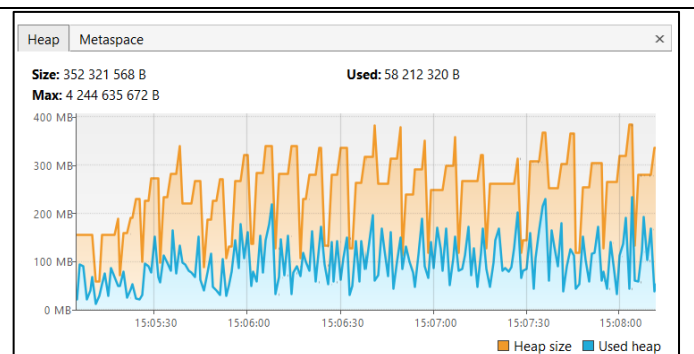


Memory.

Было



Стало



Показатели пришли к более горизонтальному виду.

Вывод:

Были произведен мониторинг и профилирование работы программы с помощью утилит JConsole и VisualVM. Утилиты позволяют отслеживать показания MBean'ов, графики изменения показаний с течением времени, вызывать операции, а также отслеживать уведомления. VisualVM также позволяет получить более подробную информацию о нагрузке на CPU/RAM. Например, узнать какие потоки программы нагружают CPU и как долго они это делают.