

Университет ИТМО

Факультет программной инженерии и компьютерных технологий

Лабораторная работа №4 по Методам и Средствам Программной Инженерии

Выполнил: Богатов Александр Сергеевич

Группа: Р3233

Вариант: 6699

Преподаватель: Цопа Евгений Алексеевич

Санкт-Петербург

2022

Задание:

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

- MBean, считающий общее число установленных пользователем точек, а также число точек, не попадающих в область. В случае, если количество установленных пользователем точек стало кратно 15, 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 класса, объекты которого занимают наибольший объём памяти JVM; определить пользовательский класс, в экземплярах которого находятся эти объекты.

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.

MBean методы:

```
@Override
public long getResultAmount () {
    loadResults ();
    return results.size();
}
```

```

@Override
public long getSVGResultAmount() {
    loadResults();
    return results.stream().filter(result -> result.getType().equals("fromSVG")).count();
}

@Override
public long getMissAmount() {
    loadResults();
    return results.stream().filter(result -> result.getHit() == false).count();
}

@Override
public long checkPointAmountDivisor() {
    long amount = getResultAmount();
    long misses = getMissAmount();
    resultAmount = amount;
    missAmount = misses;
    if (amount % 15 == 0) {
        sendNotification(new Notification("Result amount can be divided by 15", this.getClass().getName(),
sequenceNumber++, "Overall number of results: " + results.size() + "\n Missed results: " + misses));
    }
    return amount;
}

@Override
public double getMissPercentage() {
    long amount = getSVGResultAmount();
    long misses = getMissAmount();
    svgAmount = amount;
    missAmount = misses;
    if (amount != 0) {
        missPercentage = misses * 100 / amount;
        return misses * 100 / amount;
    }
    missPercentage = 0;
}

```

```
        return 0;
    }
}
```

MXBeanContextListener:

```
package management;

import data.ResultBean;

import javax.management.*;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import java.lang.management.ManagementFactory;

public class MXBeanContextListener implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        try {
            MBeanServer = ManagementFactory.getPlatformMBeanServer();
            ResultBean bean = new ResultBean();
            ObjectName beanObjName = new ObjectName("data:type=mbeans,name=result");
            mBeanServer.registerMBean(bean, beanObjName);
            MXBeanListener beanListener = new MXBeanListener();
            mBeanServer.addNotificationListener(beanObjName, beanListener,
            beanListener.getNotificationFilter(), null);
        } catch (InstanceAlreadyExistsException | MalformedObjectNameException | MBeanRegistrationException |
            NotCompliantMBeanException e) {
            e.printStackTrace();
        } catch (InstanceNotFoundException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
```

```

    try {
        ManagementFactory.getPlatformMBeanServer().unregisterMBean(new
ObjectName("data:type=mbeans,name=result"));
    } catch (MBeanRegistrationException | InstanceNotFoundException | MalformedObjectNameException e) {
        e.printStackTrace();
    }
}
}

```

Показания MBeans:

Attribute values	
Name	Value
MissAmount	46
MissPercentage	47.0
ResultAmount	150
SVGResultAmount	96

Notification buffer					
TimeStamp	Type	UserData	SeqNum	Message	Event
23:11:12:209	Result amount can be divided by 15		0	Overall number of results: 135 Missed results: 40	javax.management.Notification[source=data.R... data.ResultBean

Изменение атрибутов MBean:

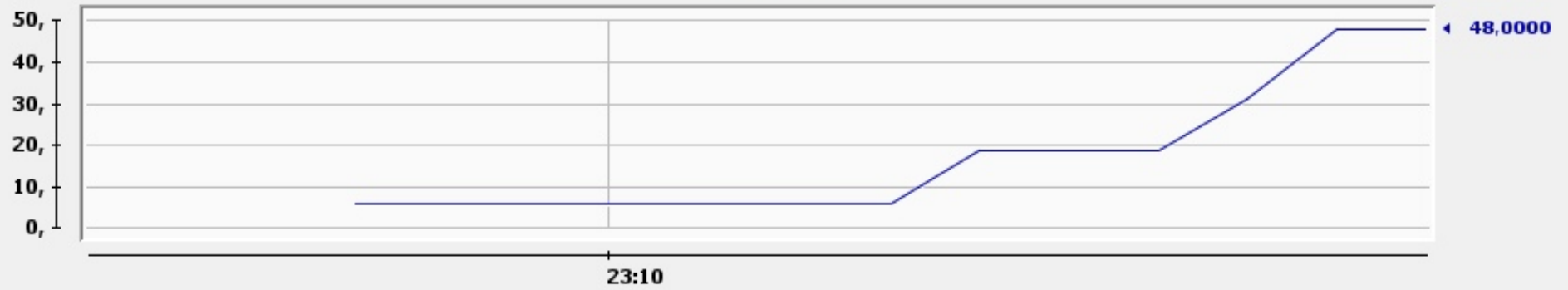
Попользуемся разработанным сайтом какое-то время и посмотрим на то, как изменяются значения атрибутов MBeans.

MissAmount

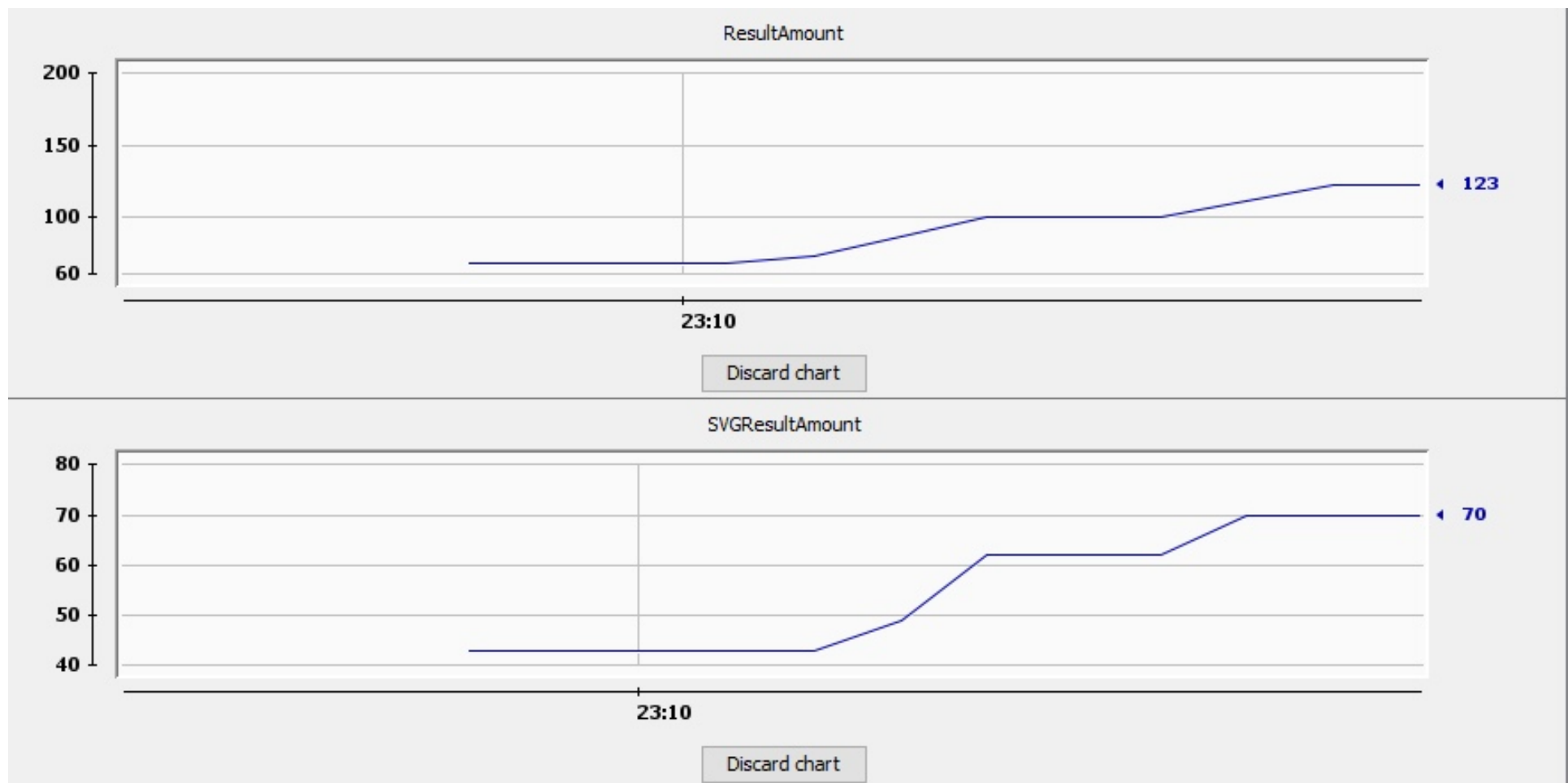


Discard chart

MissPercentage



Discard chart



Количество классов в JVM:

Информацию о числе загруженных и выгруженных из JVM классов можно найти во вкладке Classes.

Time: 2022-05-25 23:25:42
Current classes loaded: 34 412
Total classes loaded: 34 418
Total classes unloaded: 6

Класс, занимающий больше всего памяти:

Сделав дамп кучи, выясняем, что больше всего места в JVM занимают объекты `byte[]`. Посмотрим корни GC и выясним, что наибольшее число объектов используются JBoss'ом, а также что у многих объектов уже нет корня GC и они подлежат очистке.

org.jboss.modules.Main (pid 15496)

Heap Dump

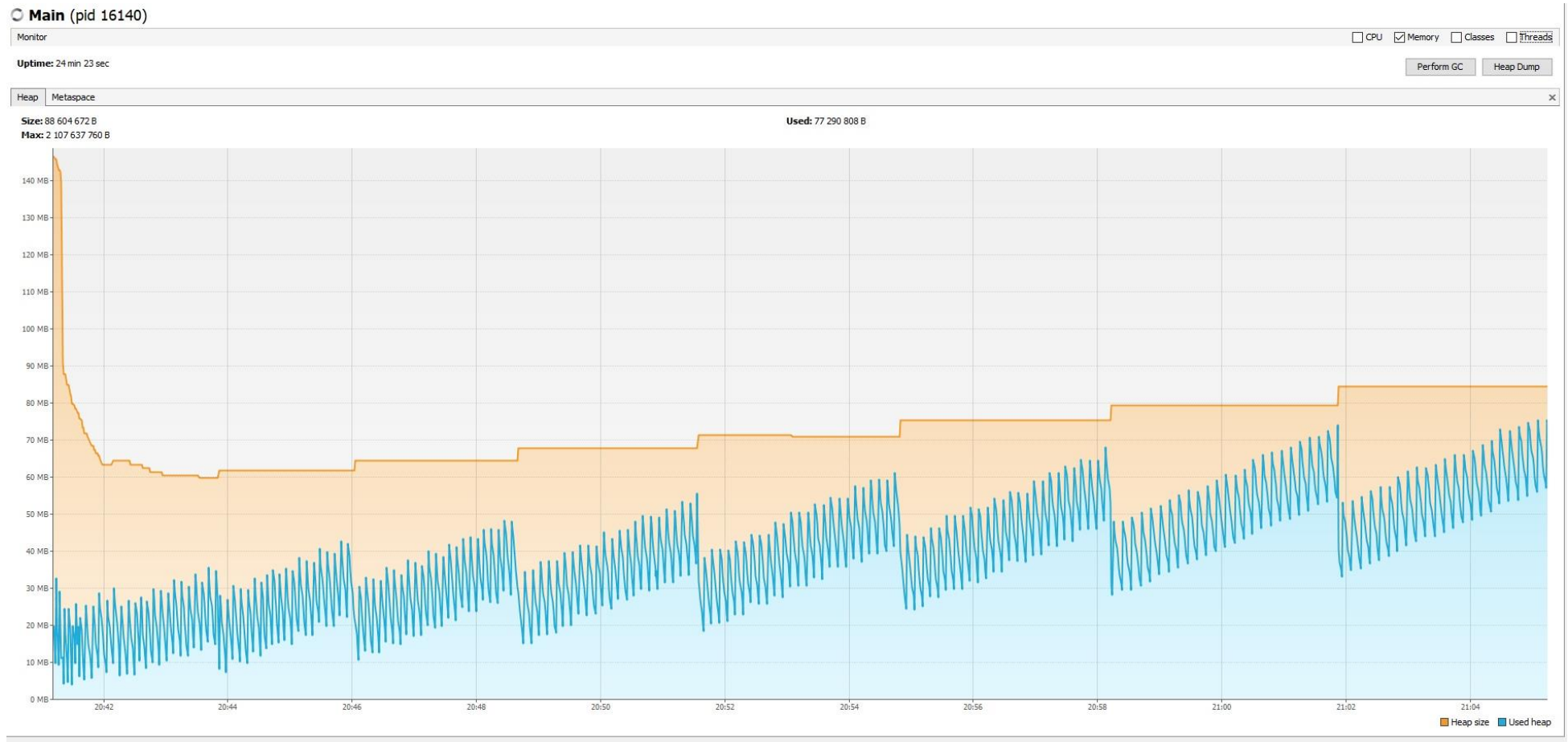
Objects ▾ | Preset: All Objects ▾ | Aggregation: | Details: | Preview | Fields | References | GC Root | Hierarchy

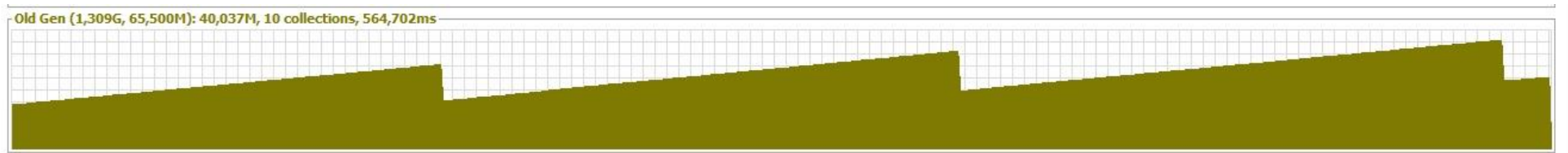
Name	Count	Size	Retained (sort to get)
byte[]	282 986 (16,6 %)	31 893 859 B (27,7 %)	n/a
java.util.HashMap\$Node	240 570 (14,1 %)	10 585 080 B (9,2 %)	n/a
java.lang.String	275 308 (16,1 %)	8 259 240 B (7,2 %)	n/a
java.util.HashMap\$Node[]	27 532 (1,6 %)	6 195 896 B (5,4 %)	n/a
java.lang.Object[]	51 682 (3 %)	4 745 640 B (4,1 %)	n/a

Name	Count
org.jboss.modules.ModuleClassLoader#120 [GC root - Java frame]	67 086 (3,9 %)
<no GC root>	63 196 (3,7 %)
org.jboss.msc.service.ServiceContainerImpl\$ServiceThread#1 [GC root - Java frame, thread object] : MSC service thread 1-1	48 116 (2,8 %)
com.sun.jmx.remote.internal.ArrayNotificationBuffer#1 [GC root - Java frame, monitor used]	14 920 (0,9 %)
java.util.TimerThread#4 [GC root - Java frame, thread object] : Timer-2	13 902 (0,8 %)
java.util.TimerThread#1 [GC root - Java frame, thread object] : Timer-3	5 823 (0,3 %)
jdk.internal.loader.ClassLoaders\$AppClassLoader#1 [GC root - Java frame]	5 523 (0,3 %)
java.lang.Thread#18 [GC root - Java frame, thread object] : pool-5-thread-1	5 220 (0,3 %)
org.wildfly.common.ref.References\$ReaperThread#2 [GC root - Java frame, thread object] : Reference Reaper #1	4 567 (0,3 %)
java.lang.Thread#15 [GC root - Java frame, thread object] : RxSchedulerPurge-1	4 311 (0,3 %)
class java.beans.ThreadGroupContext [GC root - sticky class] : ThreadGroupContext	4 237 (0,2 %)
java.util.TimerThread#6 [GC root - Java frame, thread object] : Timer-1	3 555 (0,2 %)
java.util.TreeSet#440 [GC root - Java frame] : 1 element	3 278 (0,2 %)
class sun.net.www.protocol.jar.JarFileFactory [GC root - sticky class] : JarFileFactory	3 048 (0,2 %)

Исправление утечки:

Проведем наблюдение за кучей программы: занятое место в куче с каждой мажорной очисткой становится больше, что свидетельствует об утечке памяти.





Сделав дамп кучи видим, что много места занимают объекты `char[]`. Посмотрим на корни GC объектов и обнаружим там пользовательский класс `com.meterware.httpunit.javascript`.

The screenshot shows the Eclipse IDE with the 'Classes' and 'References' tabs open. The 'Classes' tab displays a list of classes, including `char[]`, `java.lang.String`, `java.util.TreeMap$Entry`, `java.lang.Long`, and `java.util.TreeMap`. The 'References' tab shows the references to the `char[]` class, including the `this` field, the `value` field, and the `elementData` field.

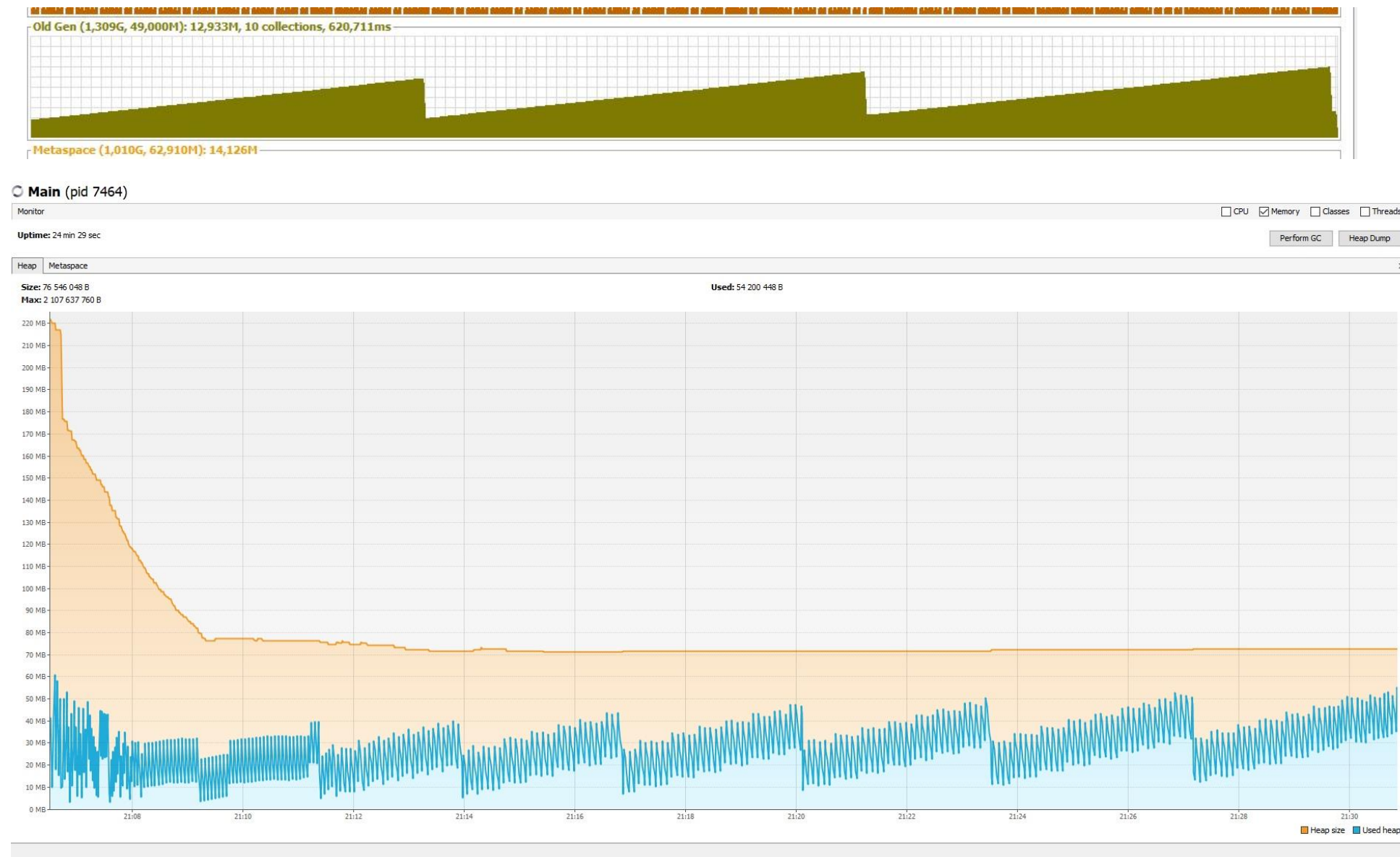
Class Name	Instances [%]	Instances	Size
<code>char[]</code>	21.8%	56 375	10 917 138 (53.8%)
<code>java.lang.String</code>	21.7%	56 109	1 571 052 (7.7%)
<code>java.util.TreeMap\$Entry</code>	13.6%	35 089	2 000 073 (9.9%)
<code>java.lang.Long</code>	8.6%	22 221	533 304 (2.6%)
<code>java.util.TreeMap</code>	4.3%	10 978	878 240 (4.3%)

Field	Type	Value
<code>this</code>	<code>char[]</code>	#4625 Script 'document.wr('Hello Do...
<code>value</code>	<code>String</code>	#4750 Script 'document.wr('Hello Do...
<code>[123]</code>	<code>Object[]</code>	#5350 47 427 items
<code>elementData</code>	<code>ArrayList</code>	#28
<code>_errorMessages</code>	<code>JavaScript</code>	class JavaScript
<code>[54]</code>	<code>Object[]</code>	#361 640 items
<code>elementData</code>	<code>Vector</code>	#24
<code>classes (Java frame)</code>	<code>Launcher\$AppClassLoader</code>	#1

Среди полей класса есть Array List хранящий ошибки исполнения JS, но никогда не очищающийся. Также содержимое поля никогда не используется в программе. Изменим метод так, чтобы в списке максимально хранилось 50 последних ошибок.

```
    } else {  
        if (_errorMessages.size() >= 50) {  
            _errorMessages.remove( index: 0);  
        }  
        _errorMessages.add( errorMessage );  
    }
```

Наблюдаем за исполнением и видим улучшения.



Вывод:

При выполнении данной лабораторной работы были изучены методы мониторинга и профилирования Java приложений с помощью утилит JConsole и VisualVM.