

Национальный исследовательский университет ИТМО
Факультет программной инженерии и компьютерной техники

Методы и средства программной инженерии.

Лабораторная работа №4.

Вариант: 89996

Выполнили:

Газизов Анвар Васильевич

Романов Артём Максимович

Поток: 1.4

Преподаватель: Цопа Е. А.

Санкт-Петербург

2022

1. 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, 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 имена всех потоков, выполняющихся при запуске 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 объем 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 отчет, 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.

2. Исходный код разработанных MBean-классов и сопутствующих классов.

```
public class PointCounter extends NotificationBroadcasterSupport implements
PointCounterMXBean {
    private int sequenceNumber = 0;
    private int count = 0;
    private int hitCount = 0;

    private static final PointCounter pointCounter = new PointCounter();

    public static PointCounter getInstance() {
        return pointCounter;
    }

    @Override
    public int getCount() {
        return count;
    }

    @Override
    public int getHitCount() {
        return hitCount;
    }

    @Override
    public void check(boolean result) {
        count++;
        if (result) hitCount++;
        if (count % 10 == 0) {
            sendNotification(new Notification("multiple_of_10", this, sequenceNumber++,
            System.currentTimeMillis(), "The number of points is " + count));
        }
    }
}
```

```

    }
}

public void setCount(int count) {
    this.count = count;
}

public void setHitCount(int hitCount) {
    this.hitCount = hitCount;
}
}

-----
public class MissPercentage implements MissPercentageMBean {

    private double missRate;

    private static final MissPercentage missPercentage = new MissPercentage();

    public static MissPercentage getInstance() {
        return missPercentage;
    }

    @Override
    public void calculatePercentage(int count, int hit) {
        missRate = (double) (count - hit) / count * 100;
    }

    public void setMissRate(double missRate) {
        this.missRate = missRate;
    }

    @Override
    public double getMissPercentage() {
        return missRate;
    }
}

-----
public class JmxContextListener implements ServletContextListener {
    private static final String pointCounterName =
"nullnumber1.service.impl:type=mbean,name=PointCounter";
    private static final String missPercentageName =
"nullnumber1.service.impl:type=mbean,name=MissPercentage";

    @Override
    public void contextInitialized(ServletContextEvent servletContextEvent) {
        try {
            MBeanServer mBeanServer = ManagementFactory.getPlatformMBeanServer();

            ObjectName pointCounterObjectName = new ObjectName(pointCounterName);
            PointCounter pointCounter = PointCounter.getInstance();
            mBeanServer.registerMBean(pointCounter, pointCounterObjectName);

            ObjectName missPercentageObjectName = new ObjectName(missPercentageName);
            MissPercentage missPercentage = MissPercentage.getInstance();
            mBeanServer.registerMBean(missPercentage, missPercentageObjectName);
        }
    }
}

```

```

        MXBeanNotificationListener pointCounterListener = new
MXBeanNotificationListener("multiplicity");
        mBeanServer.addNotificationListener(pointCounterObjectName,
pointCounterListener, pointCounterListener.getNotificationFilter(), null);
    } catch (Exception exception) {
        log.error("Error registering MBean: " + exception);
    }
}
-----
public class MXBeanNotificationListener implements NotificationListener {
    private final String notificationType;

    public MXBeanNotificationListener(String notificationType) {
        this.notificationType = notificationType;
    }

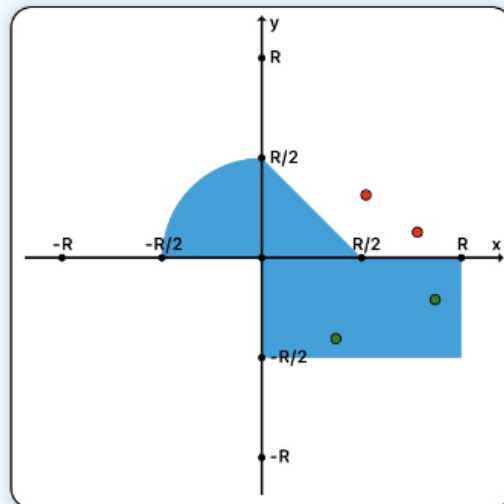
    @Override
    public void handleNotification(Notification notification, Object handback) {
        log.info(String.format("Listener %s: %s", notificationType,
notification.getMessage()));
    }

    public NotificationFilter getNotificationFilter() {
        NotificationFilterSupport filter = new NotificationFilterSupport();
        filter.enableType(notificationType);
        return filter;
    }
}

```

3. Скриншоты программы JConsole со снятыми показаниями, выводы по результатам мониторинга:

Показания MBeans:



Choose coordinate x:

-4 -3 -2 -1 0 1 2
3 4

Enter coordinate y:

0.0

Choose radius:

1 2 3 4
5

Submit

Clean

Coordina...	Coordina...	Radius	Hit	Local Time	Script Time
0.3711	-0.4046	1.0	true	2022-05-31 09:48...	0.1537 ms
0.5221	0.3145	1.0	false	2022-05-31 09:49...	0.0223 ms
0.7795	0.1281	1.0	false	2022-05-31 09:49...	0.0289 ms
0.8683	-0.2093	1.0	true	2022-05-31 09:49...	0.0217 ms

Attribute values

Name	Value
Count	4
HitCount	2

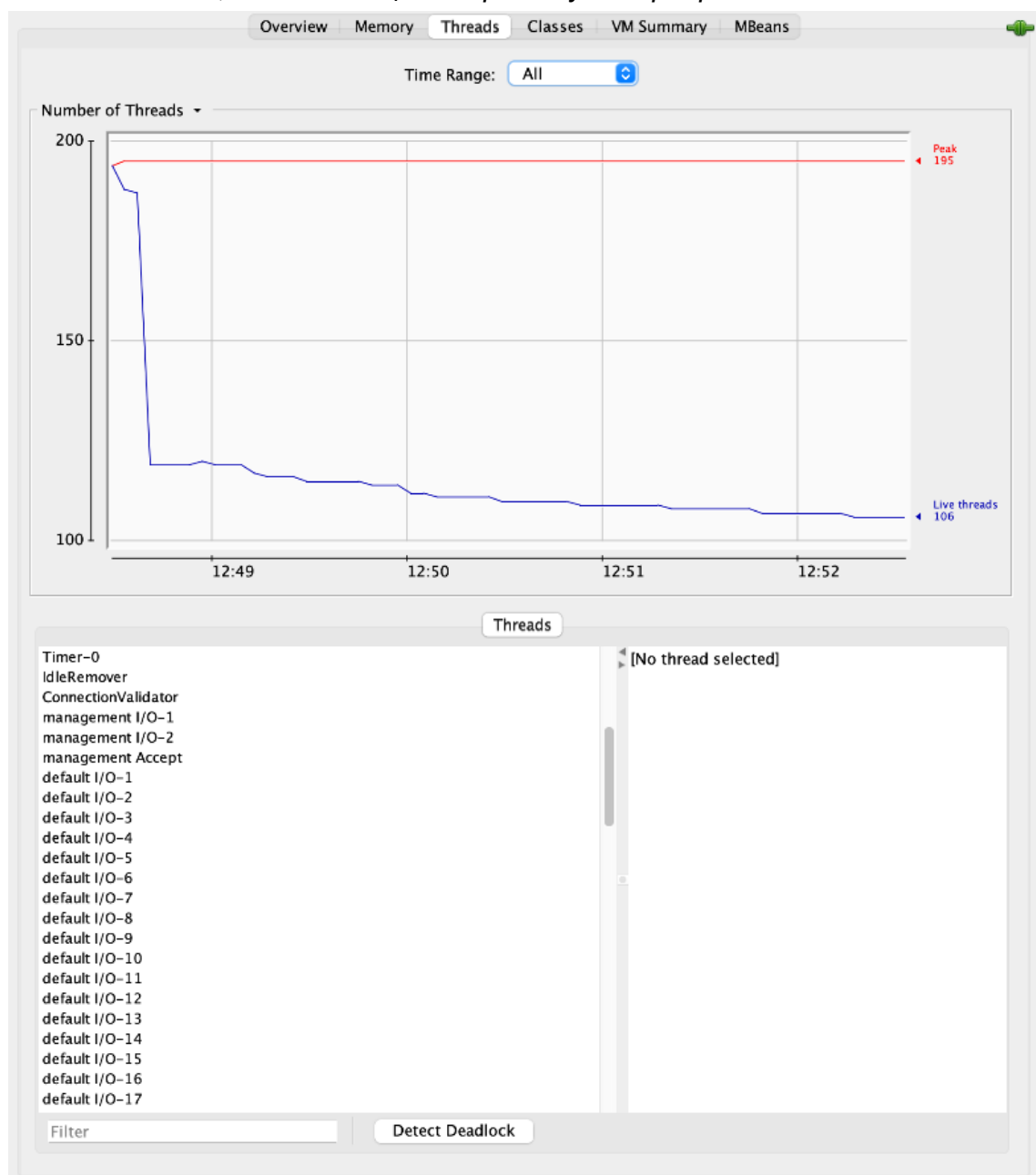
Attribute values

Name	Value
MissPercentage	50.0

Оповещения MBean:

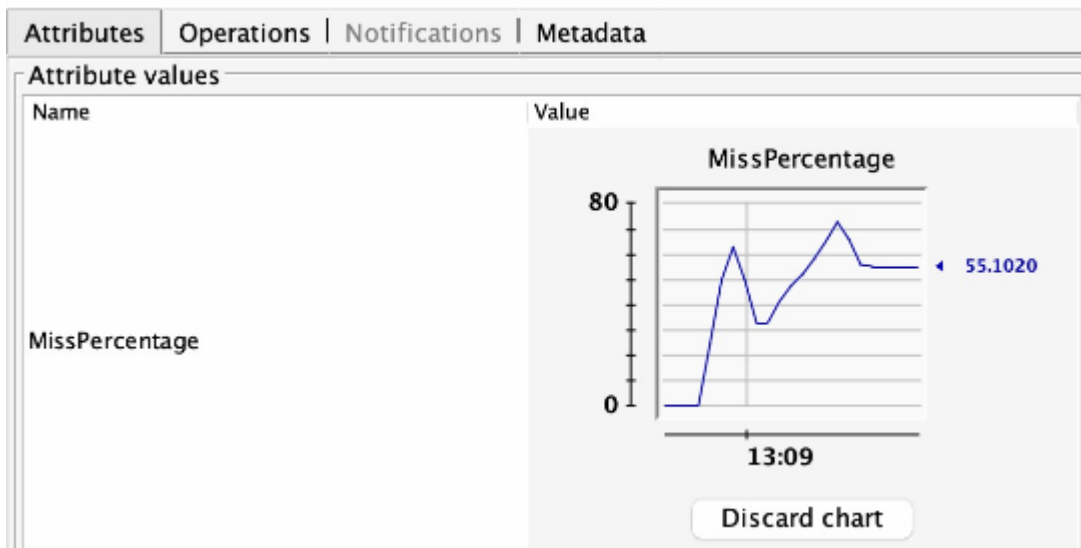
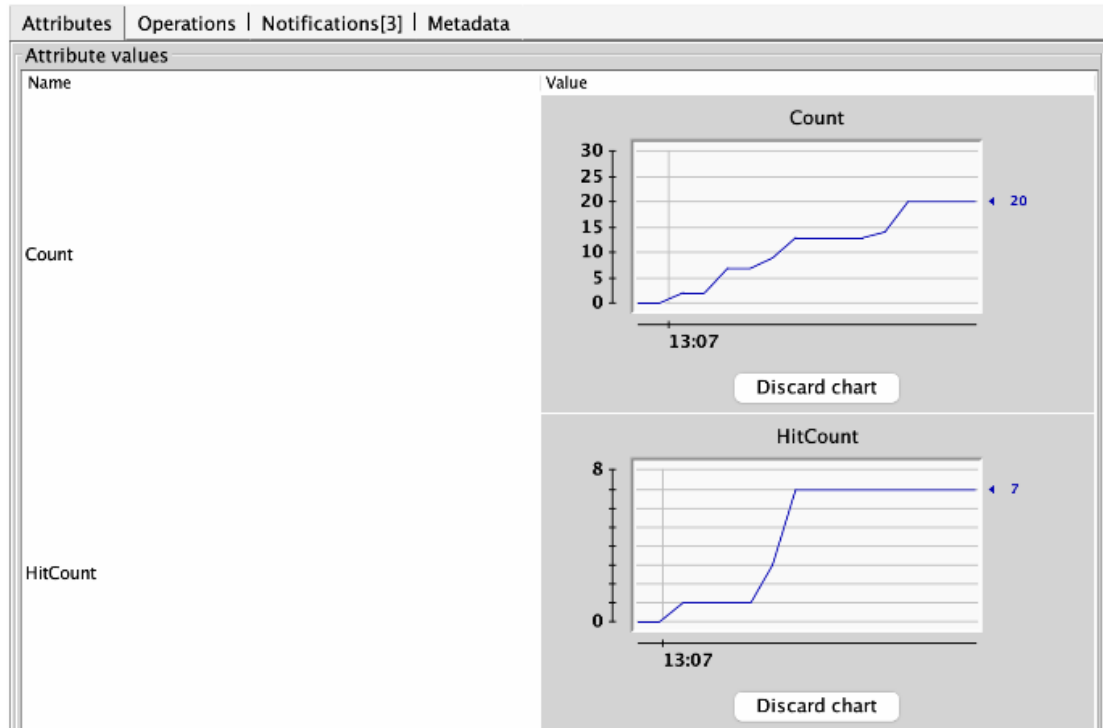
Notification buffer						
TimeStamp	Type	UserData	SeqNum	Message	Event	Source
12:53:30:999	multiple_of_10		4	The number of points is 50	javax.... nullnu...	
12:53:28:513	multiple_of_10		3	The number of points is 40	javax.... nullnu...	
12:53:27:046	multiple_of_10		2	The number of points is 30	javax.... nullnu...	
12:53:25:647	multiple_of_10		1	The number of points is 20	javax.... nullnu...	
12:53:22:449	multiple_of_10		0	The number of points is 10	javax.... nullnu...	

Имена потоков, выполняющихся при запуске программы:



4. Скриншоты программы VisualVM со снятыми показаниями, выводы по результатам профилирования:

Графики изменения показаний MBeans:



Класс, объекты которого занимают наибольший объем памяти JVM:

Наибольший объем памяти JVM занимают объекты `char[]`. Изучив GC Root, можно понять, что большая часть данных объектов использует JBoss. Также у немалой части объектов отсутствует GC Root, то есть они подлежат очистке.

org.jboss.modules.Main (pid 48581)

Heap Dump

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

Name	Count	Size	Retained (sort to get)
char[]	263,868 (15.1%)	29,632,422 B (27.1%)	n/a
java.util.HashMap\$Node	259,067 (14.8%)	11,398,948 B (10.4%)	n/a
java.lang.String	262,244 (15%)	7,342,832 B (6.7%)	n/a
java.util.HashMap\$Node[]	29,392 (1.7%)	6,537,000 B (6%)	n/a
java.lang.Object[]	63,556 (3.6%)	5,561,360 B (5.1%)	n/a
java.lang.reflect.Method	29,118 (1.7%)	4,251,228 B (3.9%)	n/a
java.util.HashMap	41,156 (2.3%)	2,633,984 B (2.4%)	n/a
java.util.jar.JarFile\$JarFileEntry	15,100 (0.9%)	2,053,600 B (1.9%)	n/a
java.lang.reflect.Parameter	27,216 (1.6%)	1,741,824 B (1.6%)	n/a
byte[]	8,674 (0.5%)	1,429,223 B (1.3%)	n/a
org.jboss.weld.annotated.slim.backed.BackedAnnotatedParameter	27,064 (1.5%)	1,407,328 B (1.3%)	n/a
java.util.LinkedHashMap\$Entry	20,579 (1.2%)	1,234,740 B (1.1%)	n/a
int[]	13,148 (0.8%)	1,233,656 B (1.1%)	n/a
java.util.concurrent.ConcurrentHashMap\$Node	25,258 (1.4%)	1,111,352 B (1%)	n/a
java.util.ArrayList	33,322 (1.9%)	1,066,304 B (1%)	n/a
org.jboss.vfs.spi.javaZipFileSystem\$ZipNode	16,083 (0.9%)	771,984 B (0.7%)	n/a
java.lang.Class[]	22,361 (1.3%)	742,888 B (0.7%)	n/a
java.lang.reflect.Constructor	5,995 (0.3%)	731,390 B (0.7%)	n/a
java.lang.reflect.Field	5,898 (0.3%)	666,474 B (0.6%)	n/a
org.jboss.weld.annotated.slim.backed.BackedAnnotatedMethod	11,176 (0.6%)	625,856 B (0.6%)	n/a
java.util.LinkedHashMap	7,531 (0.4%)	610,011 B (0.6%)	n/a

All Objects ▾ char[] ▾

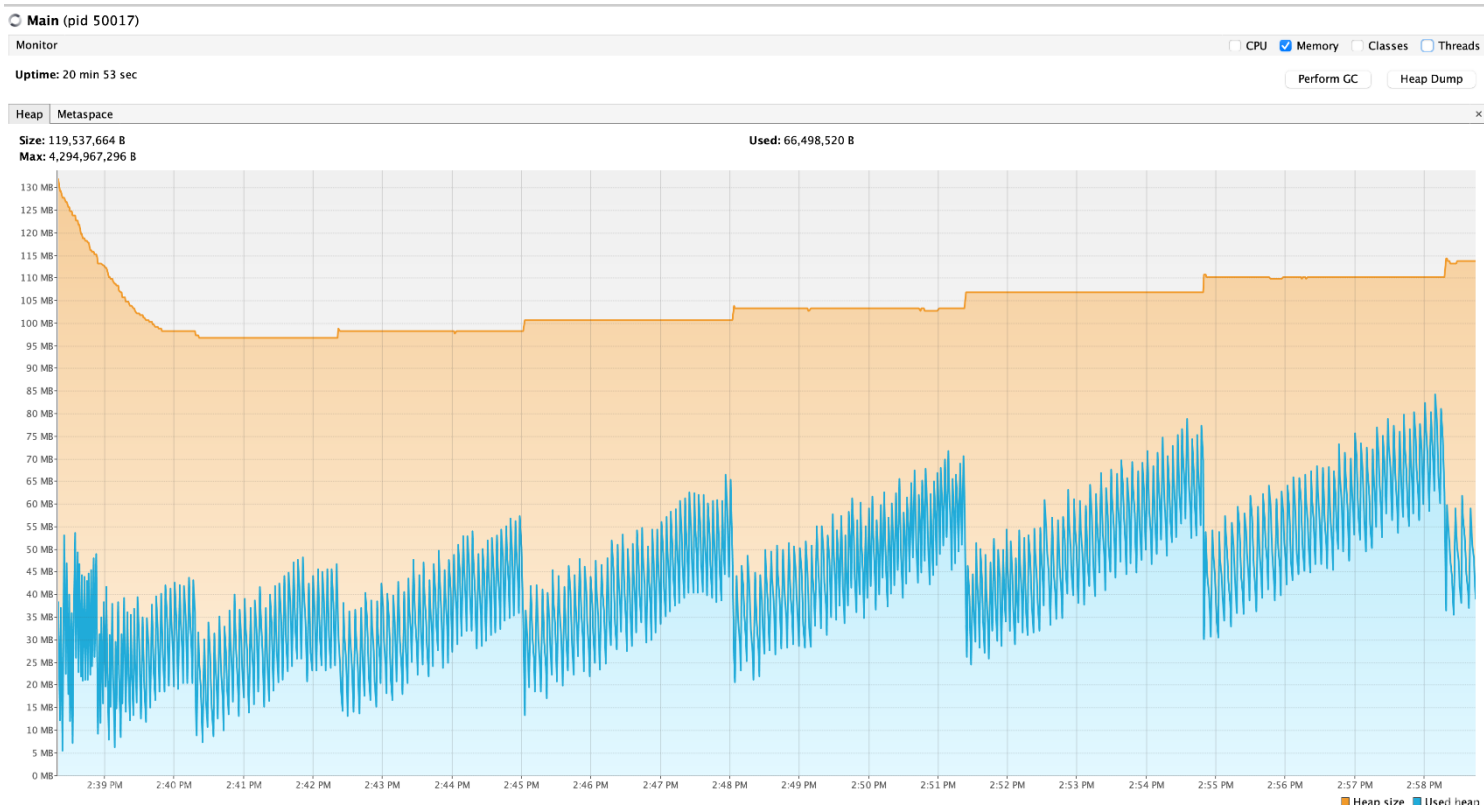
Class Filter: ▾ Filter ▾

GC Root

Name	Count
org.jboss.msc.service.ServiceContainerImpl\$ServiceThread#1 [GC root - Java frame, thread object] : MSC service thread 1-8	58,379 (3.3%)
<no GC root>	43,018 (2.5%)
org.jboss.modules.ModuleClassLoader#150 [GC root - Java frame]	39,134 (2.2%)
org.jboss.modules.ModuleClassLoader#106 [GC root - Java frame]	32,998 (1.9%)
java.lang.Thread#30 [GC root - Java frame, thread object] : pool-14-thread-1	18,022 (1%)
com.sun.jmx.remote.internal.ArrayNotificationBuffer#1 [GC root - Java frame, monitor used]	13,229 (0.8%)
class java.beans.ThreadGroupContext [GC root - sticky class] : ThreadGroupContext	4,067 (0.2%)
org.jboss.modules.ModuleClassLoader#108 [GC root - Java frame]	3,940 (0.2%)

5. Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти:

Исходя из полученных результатов мониторинга программы, можно сделать вывод, что после каждой мажорной сборки мусора, размер кучи возрастает по сравнению с предыдущей аналогичной операцией. Это говорит о том, что есть утечка памяти.



Чтобы локализовать место, в котором возникает утечка, мы сделали heap dump, в котором можно увидеть, что больше всего у нас объектов типа char. Далее за char, следует String, что логично. Ведь String состоит из char. Чтобы понять, где у нас столько строк, переключаемся в статистику по кол-ву instance-ов и видим, что в классе com.meterware.httpunit.javascript у нас есть объект ArrayList, в котором очень много элементов типа String с информацией об ошибках.

Main (pid 50017)

Heap Dump

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

Name	Size	Retained (sort to get)	
java.lang.Object[]#9120 : 810,325 items	6,482,624 B	(3.8%)	n/a
> <items>			
> <references>			
elementData in java.util.ArrayList#8 : 578,890 elements	32 B	(0%)	n/a
static _errorMessages in class com.meterware.httpunit.javascript.JavaScript	124 B	(0%)	n/a
> char[]#231 [GC root - Java frame] : ...	16,408 B	(0%)	n/a
> char[]#845 : ount: 578890[]_response = com.meterware.servletunit.ServletUnitHttpResponse@4c9cdeaf]...	16,408 B	(0%)	n/a
> char[]#846 : ...	16,408 B	(0%)	n/a

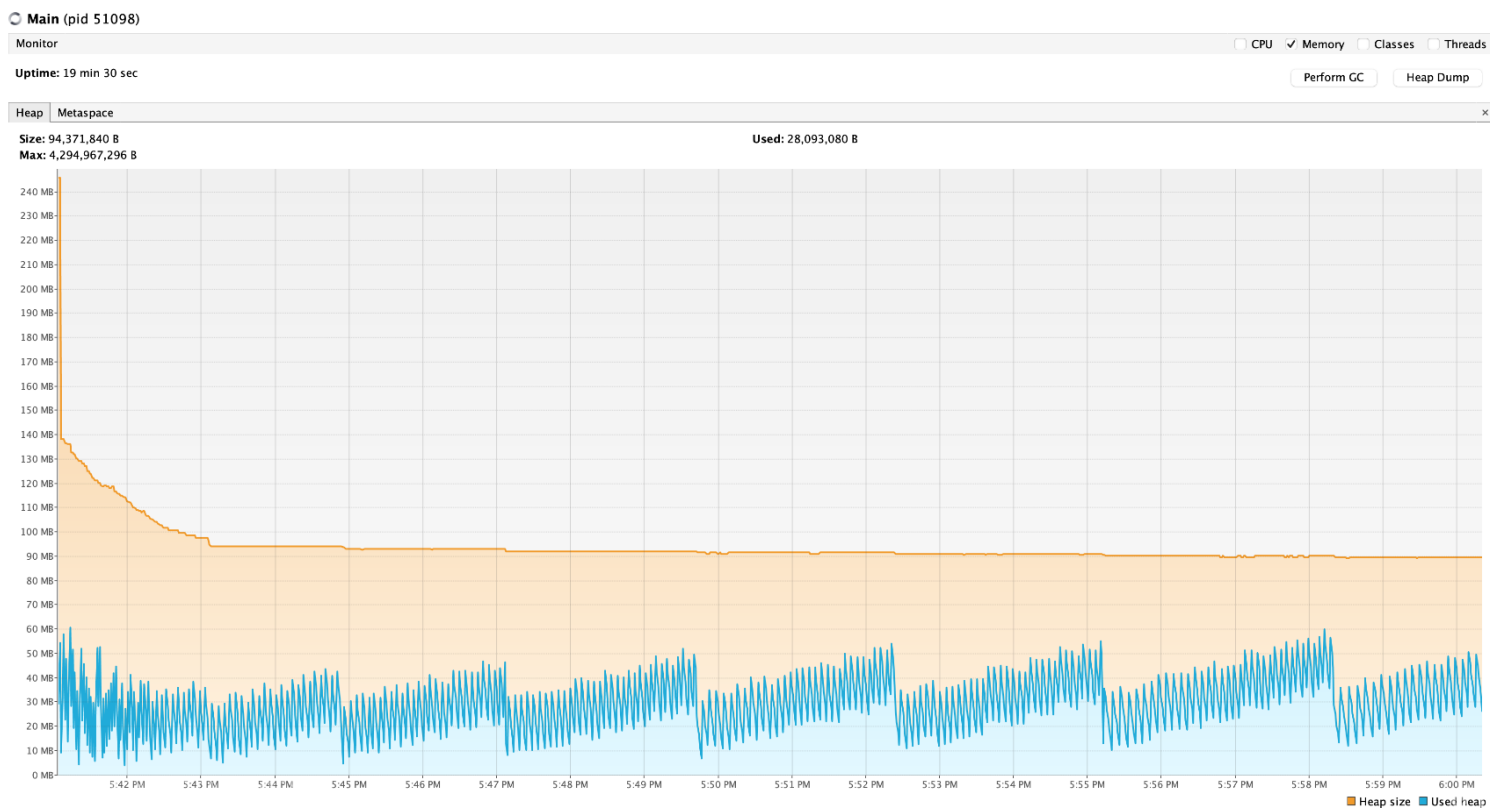
Изучив обращения к этому ArrayList-у, можно увидеть, что элементы в него добавляются, но никогда не очищаются.

```
private void handleScriptException( Exception e, String badScript ) {
    final String errorMessage = badScript + " failed: " + e;
    if (!(e instanceof EcmaError) && !(e instanceof EvaluatorException)) {
        e.printStackTrace();
        throw new RuntimeException( errorMessage );
    } else if (isThrowExceptionsOnError()) {
        e.printStackTrace();
        throw new ScriptException( errorMessage );
    } else {
        _errorMessages.add( errorMessage );
    }
}
```

Решение проблемы с утечкой памяти решается периодическим очищением коллекции с сообщениями об ошибках, когда её размер 100 элементов.

```
private void handleScriptException( Exception e, String badScript ) {  
    final String errorMessage = badScript + " failed: " + e;  
    if (!(e instanceof EcmaError) && !(e instanceof EvaluatorException)) {  
        e.printStackTrace();  
        throw new RuntimeException( errorMessage );  
    } else if (isThrowExceptionsOnError()) {  
        e.printStackTrace();  
        throw new ScriptException( errorMessage );  
    } else {  
        if (_errorMessagees.size() >= 100) clearErrorMessage();  
        _errorMessagees.add( errorMessage );  
    }  
}
```

График состояния кучи после устранения проблемы:



6. Выводы:

В результате выполнения данной лабораторной работы мы реализовали несколько MBeans, провели мониторинг и профилирование программы при помощи утилит JConsole и VisualVM, а также локализовали и устранили проблему с производительностью в выданной программе.