

Санкт-Петербургский *Национальный Исследовательский*
Университет ИТМО
Факультет программной инженерии и компьютерной техники

Методы и средства программной инженерии
Лабораторная работа №4
Вариант 1439

Выполнили:

студент группы Р3231

Нестеров Иван Алексеевич

студент группы Р3232

Стуков Егор Александрович

Преподаватель:

Цопа Евгений Алексеевич

г. Санкт-Петербург

2022 г.

Цель работы:

1. Для своей программы из лабораторной работы #2 по дисциплине "Веб-программирование" реализовать:

- MBean, считающий общее число установленных пользователем точек, а также число точек, не попадающих в область. В случае, если пользователь совершил 3 "промаха" подряд, разработанный MBean должен отправлять оповещение об этом событии.
- MBean, определяющий средний интервал между кликами 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 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.

Ход работы:

1. Реализация MBean-классов и сопутствующих классов

1. DotChecker

```
package com.zloyegor.mispi.lab4.management;

import javax.management.Notification;
import javax.management.NotificationBroadcasterSupport;
import java.util.concurrent.atomic.AtomicInteger;

public class DotChecker extends NotificationBroadcasterSupport implements DotCheckerMBean {

    private AtomicInteger totalDotAmount = new AtomicInteger(0);
    private AtomicInteger missesInRow = new AtomicInteger(0);
    private AtomicInteger notificationSequence = new AtomicInteger(1);

    private static DotChecker dotChecker = new DotChecker();

    public static DotChecker getInstance() {
        return dotChecker;
    }

    @Override
    public void process(boolean result) {
        totalDotAmount.getAndIncrement();
        if (result) {
            missesInRow.set(0);
        } else {
            missesInRow.getAndIncrement();
            if (missesInRow.get() >= 3) {
                Notification notification = new Notification("Dot miss", this,
                    notificationSequence.get(), "User got " + missesInRow + " misses in a row");
                sendNotification(notification);
                notificationSequence.getAndIncrement();
            }
        }
    }
}
```

2. TimeSpanQualifier

```
package com.zloyegor.mispi.lab4.management;

import java.util.Date;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicLong;

public class TimeSpanQualifier implements TimeSpanQualifierMBean{
```

```

private AtomicLong initTime = new AtomicLong(new Date().getTime() / 1000);
private AtomicLong lastClickTime = new AtomicLong(initTime.get());
private AtomicInteger dotAmount = new AtomicInteger(0);

private static TimeSpanQualifier timeSpanQualifier = new TimeSpanQualifier();

public static TimeSpanQualifier getInstance() {
    return timeSpanQualifier;
}

@Override
public void process() {
    lastClickTime.set(new Date().getTime() / 1000);
    dotAmount.getAndIncrement();
}

@Override
public long getTimeSpan() {
    if (dotAmount.get() > 0)
        return (lastClickTime.get() - initTime.get()) / dotAmount.get();
    else
        return 0;
}
}

```

3. MBeanListener

```

package com.zloyegor.mispi.lab4.management;

import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationFilterSupport;
import javax.management.NotificationListener;

public class MBeanListener implements NotificationListener {

    private final String notificationType;

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

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

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

```

```

        return filter;
    }
}

```

4. JmxContextListener

```

package com.zloyegor.mispi.lab4;

import com.zloyegor.mispi.lab4.management.DotChecker;
import com.zloyegor.mispi.lab4.management.MBeanListener;
import com.zloyegor.mispi.lab4.management.TimeSpanQualifier;

import javax.management.MBeanServer;
import javax.management.ObjectName;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import java.lang.management.ManagementFactory;

public class JmxContextListener implements ServletContextListener {

    private static final String dotCheckerMBeanName =
"com.zloyegor.mispi.lab4:type=mbeans,name=dot_checker";
    private static final String timeSpanQualifierMBeanName =
"com.zloyegor.mispi.lab4:type=mbeans,name=time_span_qualifier";

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

            ObjectName dotCheckerObjectName = new ObjectName(dotCheckerMBeanName);
            DotChecker dotChecker = DotChecker.getInstance();
            mBeanServer.registerMBean(dotChecker, dotCheckerObjectName);

            ObjectName timeSpanQualifierObjectName = new ObjectName(timeSpanQualifierMBeanName);
            TimeSpanQualifier timeSpanQualifier = TimeSpanQualifier.getInstance();
            mBeanServer.registerMBean(timeSpanQualifier, timeSpanQualifierObjectName);

            MBeanListener dotCheckerListener = new MBeanListener("dotChecker");
            mBeanServer.addNotificationListener(dotCheckerObjectName, dotCheckerListener,
dotCheckerListener.getNotificationFilter(), null);
        } catch (Exception e) {
            System.err.println("Error registering MBean: " + e);
        }
    }

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

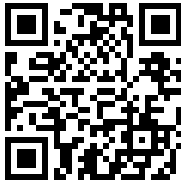
```

```
ObjectName dotCheckerObjectName = new ObjectName(dotCheckerMBeanName);
mBeanServer.unregisterMBean(dotCheckerObjectName);

ObjectName timeSpanQualifierObjectName = new ObjectName(timeSpanQualifierMBeanName);
mBeanServer.unregisterMBean(timeSpanQualifierObjectName);

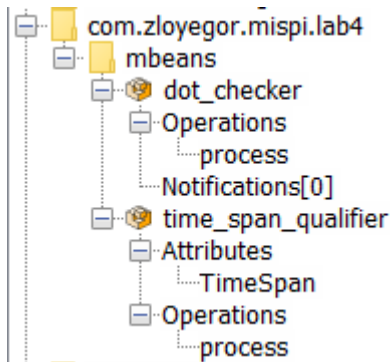
} catch (Exception e) {
    System.err.println("Error unregistering MBean: " + e);
}
}
```

Ссылка на GitHub:



<https://github.com/ZloyEgor/MISPI-Lab4>

2. Мониторинг программы с помощью утилиты JConsole



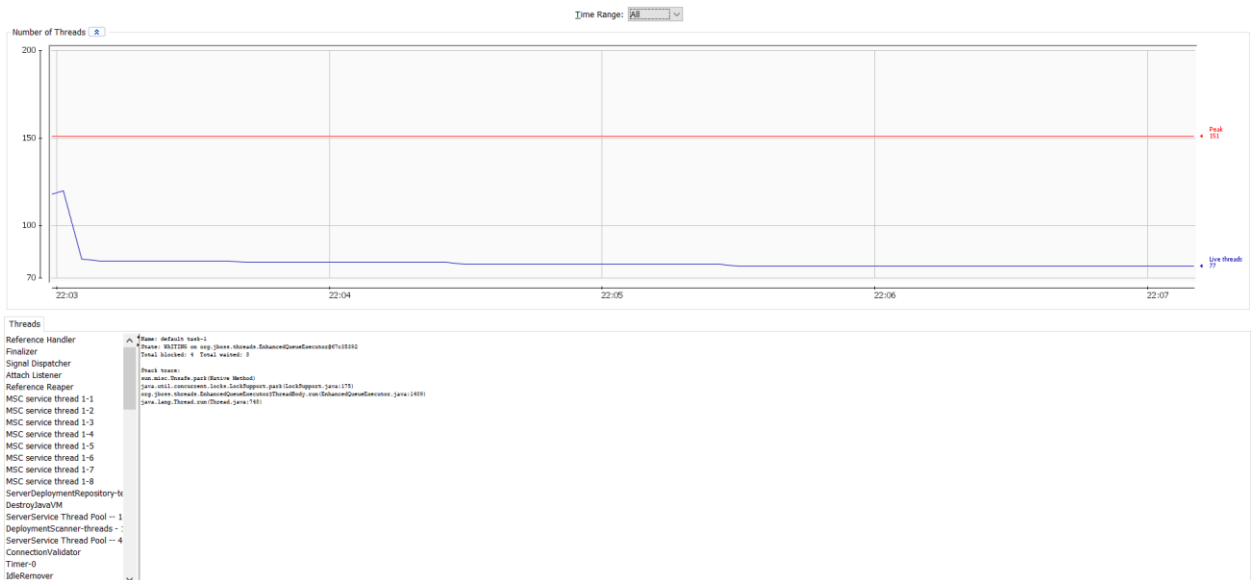
Запустив утилиту JConsole и открыв вкладку MBeans, можно развернуть папку программы и увидеть два управляемых бина, реализованных в рамках данной лабораторной работы.

Notification buffer						
TimeStamp	Type	UserData	SeqNum	Message	Event	Source
22:06:39:220	Dot miss		3	User got 4 misses in a row	javax.management.Notification[source...]	com.zloyegor.mispi.lab4:type=mbeans,na...
22:06:38:680	Dot miss		2	User got 3 misses in a row	javax.management.Notification[source...]	com.zloyegor.mispi.lab4:type=mbeans,na...
22:06:21:333	Dot miss		1	User got 3 misses in a row	javax.management.Notification[source...]	com.zloyegor.mispi.lab4:type=mbeans,na...

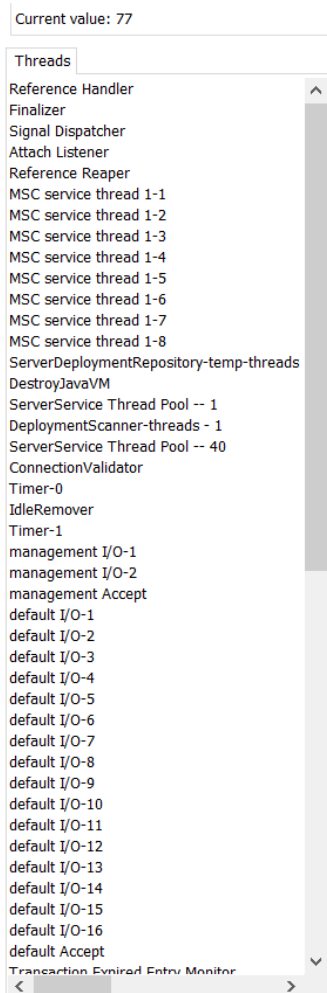
Управляемый бин, названный как “dot_checker”, не имеет атрибутов – его задача сводится к отправке сообщений о очереди промахов пользователя. Можем видеть, что каждое оповещение бина имеет время TimeStamp, порядковый номер SeqNum, а также сообщение Message.

Attribute value	
Name	Value
TimeSpan	29
Refresh	
MBeanAttributeInfo	
Name	Value
Attribute:	
Name	TimeSpan
Description	Attribute exposed for management
Readable	true
Writable	false
Is	false
Type	long

Управляемый бин “time_span_qualifier” имеет read-only атрибут TimeSpan, определяющий средний интервал между кликами пользователя по координатной плоскости в секундах.



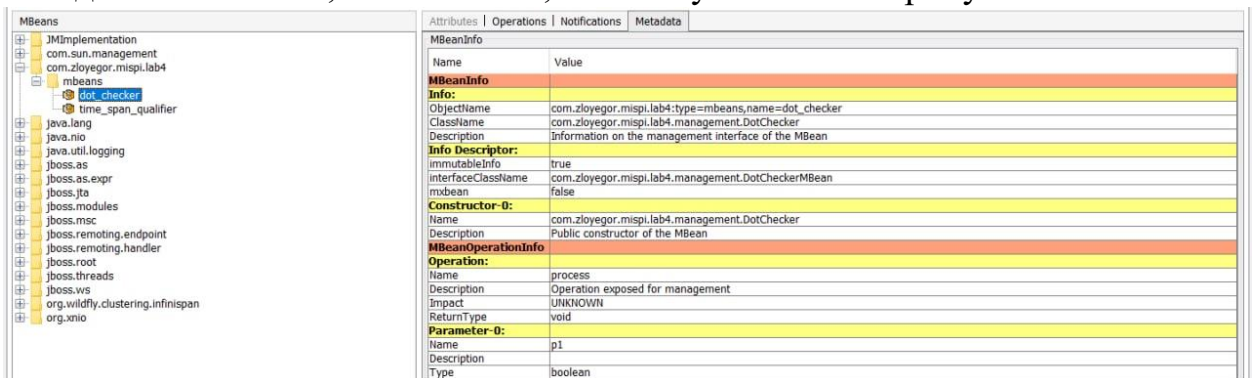
Во вкладке Threads можно увидеть список потоков, выполняющихся при работе программы



Развернув соответствующую вкладку, можно увидеть список всех живых потоков программы.

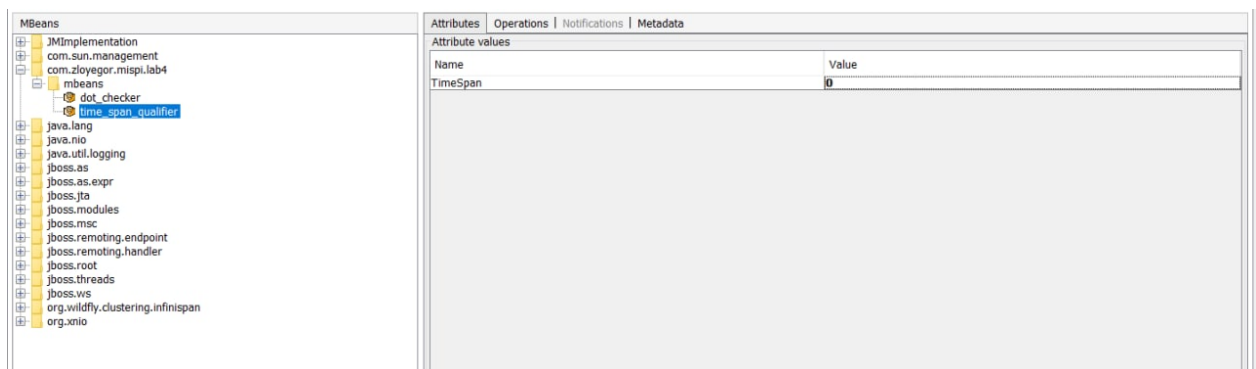
3. Мониторинг и профилирование программы:

Загрузим плагин MBeans в VisualVM для анализа показателей разработанных ранее Bean-классов. Увидим, что у первого из бинов, DotChecker, недоступна вкладка «Attributes», что логично, так как у бина нет атрибутов как таковых:



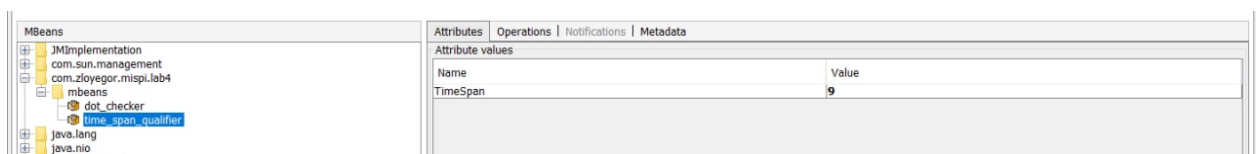
Name	Value
MBeanInfo	
Info:	
ObjectName	com.zoyegor.mispi.lab4:type=mbeans,name=dot_checker
ClassName	com.zoyegor.mispi.lab4.management.DotChecker
Description	Information on the management interface of the MBean
Info Descriptor:	
immutableInfo	true
interfaceClassName	com.zoyegor.mispi.lab4.management.DotCheckerMBean
mbean	false
Constructor-0:	
Name	com.zoyegor.mispi.lab4.management.DotChecker
Description	Public constructor of the MBean
MBeanOperationInfo	
Operation:	
Name	process
Description	Operation exposed for management
Impact	UNKNOWN
ReturnType	void
Parameter-0:	
Name	p1
Description	
Type	boolean

Однако анализ следующего бина представляется более интересным. Он, как было сказано выше, считает среднее интервал между кликами пользователя по координатной плоскости с момента запуска программы, и искомое значение можно увидеть в соответствующей бину вкладке Attributes.



Name	Value
Attribute values	
TimeSpan	0

Сделаем несколько кликов по координатной плоскости и значение изменится:



Name	Value
Attribute values	
TimeSpan	9

Перейдем к следующему пункту задания. Сделаем дамп кучи, и откроем вкладку с информацией о классах. По умолчанию они отсортированы в порядке убывания занимаемой классами памяти. На первом месте интересующий нас, HashMap\$Node класс, который в коде при написании лабораторной работы по веб-программированию не использовался. Откуда можно предположить, что данный класс используется веб-контейнером для запуска программы. Действительно, откроем раздел Instances, вычислим nearest GC root, чтобы вывести на экран самый близкий корневой объект сборки "мусора", то есть найти источник использования данного класса:

Heap Dump

Summary Classes Instances OQL Console

java.util.HashMap\$Node Instances: 218,220 | Instance size: 44 | Total size: 9,601,680 | [Compute Retained Sizes](#)

Instances

Field

Field	Type	Value
this	HashMap\$Node	#1
next	<object>	null
value	LogContext\$StrongLevelRef	#1
key	String	#12896 FATAL
hash	int	66666269
<resolved_references>	Object[]	#2513 1 item
<classLoader>	<object>	null

References

Field	Type	Value
value	AtomicReference	#1 HashMap#5
levelMapReference	LogContext	#1
context	LoggerNode	#8
loggerNode	Logger	#1 javax.management.snmp.daemon
SNMP_ADAPTOR_LOGGER (sticky class)	JmxProperties	class JmxProperties
referent	Finalizer	#204 Logger: javax.management.sn...
referent	CopyOnWriteWeakMap\$Node	#8 LoggerNode#8
referent	Finalizer	#205 LoggerNode#8
context	LoggerNode	#847
context	LoggerNode	#848
context	LoggerNode	#849
context	LoggerNode	#844
context	LoggerNode	#845
context	LoggerNode	#846
context	LoggerNode	#841

<No details>

Становится видно, что класс используется классом JmxProperties. JMX – java management extensions – технология, используемая, в частности веб-контейнером JBoss WildFly для запуска сервера приложений, что подтверждает сделанное предположение.

4. Анализ утечек памяти:

Выберем нужный процесс (запущенную лабу), откроем раздел Monitor и посмотрим за графиком использования кучи. Становится видно, что с одинаковой частотой объем данных в кучи увеличивается, а потом падает. Дополнительные вызовы Garbage collector (правая часть графика) показывает, что данные действительно могут быть удалены без последствий для работы программы. Отсюда можно предположить, что программа постоянно загружает одни и те же данные. Обратимся к коду. Видим, что программа в

бесконечном цикле создает объект ответа от сервера, что кажется логичным:

```
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( uriString: "http://test.meterware.com/myServlet");
    while (true) {
        WebResponse response = sc.getResponse(request);
        System.out.println("Count: " + number++ + response);
        java.lang.Thread.sleep( millis: 200);
    }
} catch (InterruptedException ex) {
    Logger.getLogger("global").log(Level.SEVERE, msg: null, ex);
} catch (MalformedURLException ex) {
    Logger.getLogger("global").log(Level.SEVERE, msg: null, ex);
} catch (IOException ex) {
    Logger.getLogger("global").log(Level.SEVERE, msg: null, ex);
} catch (SAXException ex) {
```

Однако дальнейшие поиски приводят к коду сервлета, отвечающего за создание ответа от сервера. И данный сервлет возвращает одну и ту же статику:

```
HelloWorld.java x
22
23 @Override
24 public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
25     PrintWriter out = response.getWriter();
26     response.setContentType("text/html");
27     out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
28                 \"Transitional//EN\">\n" +
29                 "<HTML>\n" +
30                 "<HEAD><TITLE>Hello World</TITLE></HEAD>\n" +
31                 "<BODY>\n" +
32                 "<H1>Hello World</H1>\n");
33     out.println("<script language=\"JavaScript\" type=\"text/javascript\">");
34     out.println("<!--");
35     out.println("document.wr('Hello Document')");
36     out.println("//--");
37     out.println("</script>");
38     out.println("</BODY></HTML>");
39 }
40 }
```

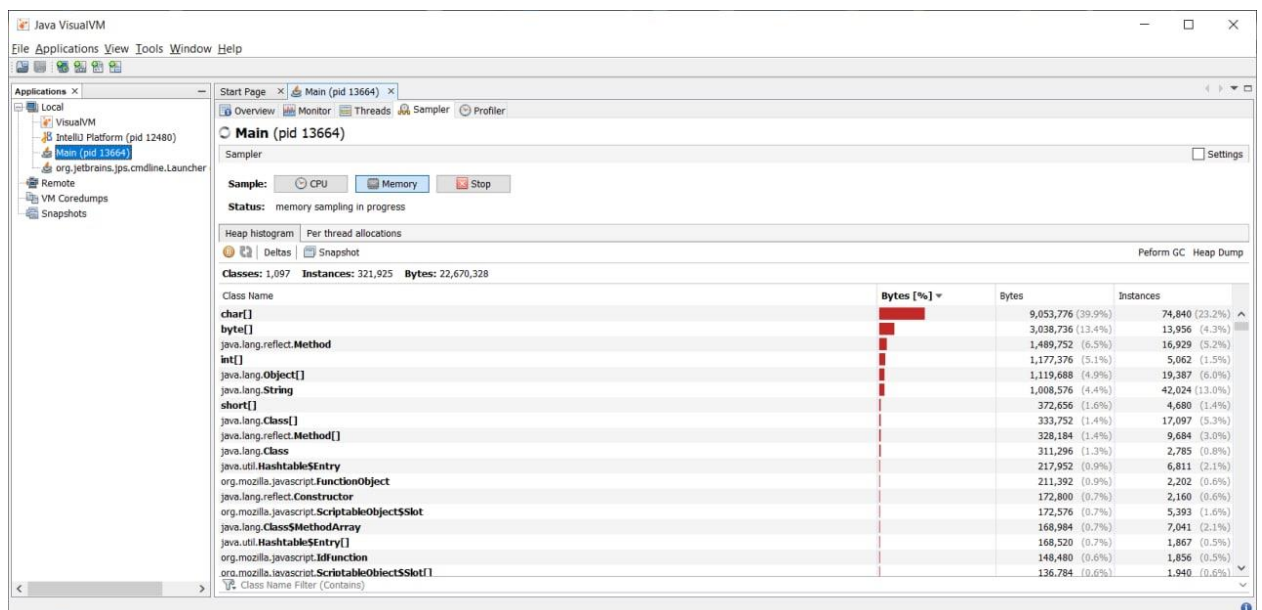
При этом использование оператора new заставляет JVM каждый раз выделять память в куче для «нового» объекта статики. Это подтверждается тем, что каждый раз выводится новая ссылка на объект:

```

Count: 28087[ _response = com.meterware.servletunit.ServletUnitHttpResponse@5da54c8a]
Count: 28088[ _response = com.meterware.servletunit.ServletUnitHttpResponse@76fbcd4]
Count: 28089[ _response = com.meterware.servletunit.ServletUnitHttpResponse@5975dbfb]
Count: 28090[ _response = com.meterware.servletunit.ServletUnitHttpResponse@57e0ab14]
Count: 28091[ _response = com.meterware.servletunit.ServletUnitHttpResponse@da29829]
Count: 28092[ _response = com.meterware.servletunit.ServletUnitHttpResponse@12708387]
Count: 28093[ _response = com.meterware.servletunit.ServletUnitHttpResponse@19706691]
Count: 28094[ _response = com.meterware.servletunit.ServletUnitHttpResponse@9939119]
Count: 28095[ _response = com.meterware.servletunit.ServletUnitHttpResponse@2d3393fc]
Count: 28096[ _response = com.meterware.servletunit.ServletUnitHttpResponse@71af7abb]
Count: 28097[ _response = com.meterware.servletunit.ServletUnitHttpResponse@725d6d32]
Count: 28098[ _response = com.meterware.servletunit.ServletUnitHttpResponse@3bab03db]
Count: 28099[ _response = com.meterware.servletunit.ServletUnitHttpResponse@221399ab]
Count: 28100[ _response = com.meterware.servletunit.ServletUnitHttpResponse@223db2a9]
Count: 28101[ _response = com.meterware.servletunit.ServletUnitHttpResponse@524a9f24]
Count: 28102[ _response = com.meterware.servletunit.ServletUnitHttpResponse@46cbca44]
Count: 28103[ _response = com.meterware.servletunit.ServletUnitHttpResponse@44d557e2]

```

Кроме того, дамп кучи показывает, что память активно используется массивами символов, что так же подтверждает наш вывод:



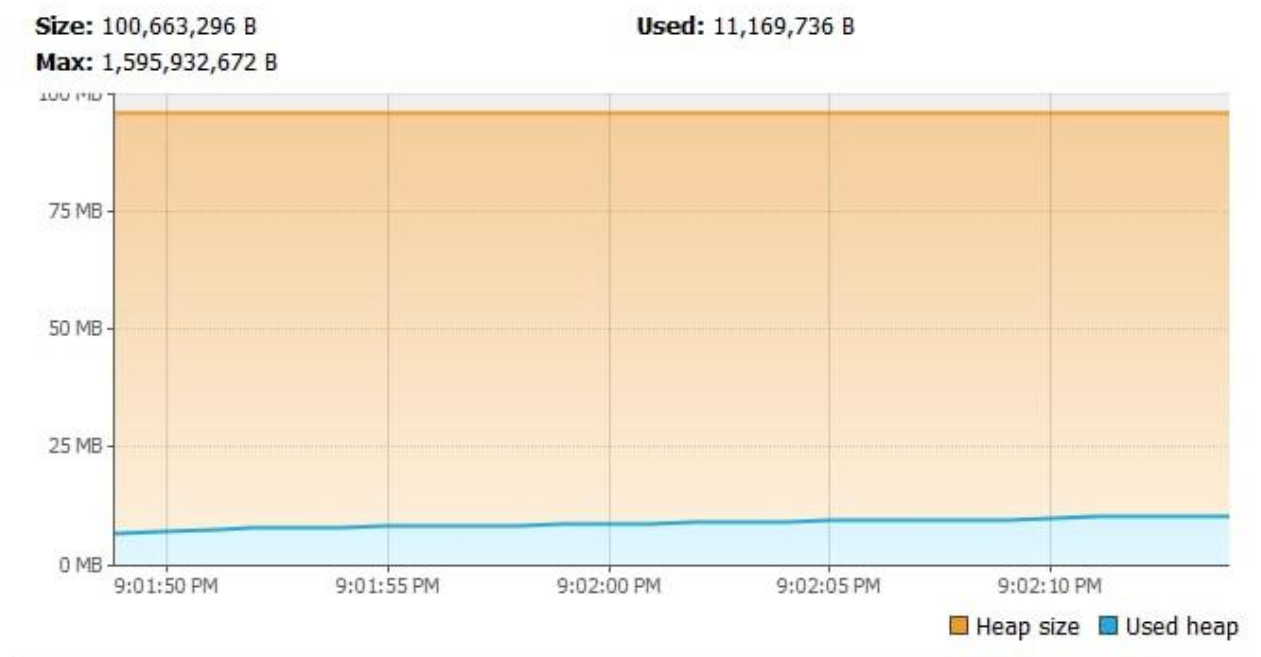
Для ликвидации утечки памяти перенесем строку с созданием объекта ответа из тела цикла «наружу», до начала работы цикла:

```

39 public static void main(String[] args) {
40     try {
41         HttpUnitOptions.setExceptionsThrownOnScriptError(false);
42         ServletRunner sr = new ServletRunner();
43         sr.registerServlet( resourceName: "myServlet", HelloWorld.class.getName());
44         ServletUnitClient sc = sr.newClient();
45         int number = 1;
46         WebRequest request = new GetMethodWebRequest( urlString: "http://test.meterware.com/myServlet");
47         WebResponse response = sc.getResponse(request);
48         while (true) {
49             System.out.println("Count: " + number++ + response);
50             java.lang.Thread.sleep( millis: 200);
51         }
52     } catch (InterruptedException ex) {
53         Logger.getLogger("global").log(Level.SEVERE, msg: null, ex);
54     } catch (MalformedURLException ex) {
55         Logger.getLogger("global").log(Level.SEVERE, msg: null, ex);
56     } catch (IOException ex) {
57         Logger.getLogger("global").log(Level.SEVERE, msg: null, ex);
58     } catch (SAXException ex) {
59         Logger.getLogger("global").log(Level.SEVERE, msg: null, ex);
60     }
61 }
62 }
63

```

Посмотрим на графики в VisualVM и убедимся, что проблема решена:



Теперь выводится ссылка на одну и ту же область кучи, одинаковый ответ не создается каждый раз заново:


```
Count: 741[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 742[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 743[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 744[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 745[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 746[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 747[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 748[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 749[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 750[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
Count: 751[ _response = com.meterware.servletunit.ServletUnitHttpResponse@18ef96]
```

Вывод:

В ходе проделанной работы мы ознакомились с практикой написания MBeans в веб-приложениях, изучили работу технологии JConsole и VisualVM для профилирования программы. Данные утилиты позволили проанализировать дампы кучи, использование памяти JVM в целом, и то, как память используется отдельными классами в частности, найти и устранить утечку памяти при помощи VisualVM.