Основы программной инженерии

Раевский Григорий

17.04.2024

Отчет по лабораторной работе 4

Содержание

Задание

1. Для своей программы из лабораторной работы 3 по дисциплине "Веб-программирование"реализовать:
   (a) MBean, считающий общее число установленных пользователем точек, а также число точек, попадающих в область. В случае, если количество установленных пользователем точек стало кратно 5, разработанный MBean должен отправлять оповещение об этом событии.
   (b) MBean, определяющий средний интервал между кликами пользователя по координатной плоскости.
2. С помощью утилиты JConsole провести мониторинг программы:
   (a) Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
   (b) Определить значение переменной classpath для данной JVM.
3. С помощью утилиты VisualVM провести мониторинг и профилирование программы:
   (a) Снять график изменения показаний MBean-классов, разработанных в ходе выполнения задания 1, с течением времени.
   (b) 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:
   (a) Описание выявленной проблемы.
   (b) Описание путей устранения выявленной проблемы.
   (c) Подробное (со скриншотами) описание алгоритма действий, который позволил выявить и локализовать проблему.
   (d) Студент должен обеспечить возможность воспроизведения процесса поиска и локализации проблемы по требованию преподавателя.

Отчёт по работе должен содержать:

1. Текст задания.
2. Исходный код разработанных MBean-классов и сопутствующих классов.

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

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

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

6. Выводы по работе.

Вопросы к защите лабораторной работы:

1. Мониторинг и профилирование. Основные понятия. Отличия мониторинга от профилирования.

2. Инфраструктура для организации мониторинга и профилирования в составе JDK. JMX.

3. MBeans. Основные понятия. Архитектура фреймворка.

4. Утилита JConsole. Возможности, область применения.

5. Утилита Visual VM. Возможности, область применения.

6. Удалённый мониторинг и профилирование приложений на платформе Java.

Код MBean и сопутствующие классы

Общее количество точек и кратность 5

```
1  package org.example.MBean.classes;
2
3  import org.example.MBean.interfaces.DotsHitMBean;
4
5  import javax.management.Notification;
6  import javax.management.NotificationBroadcasterSupport;
7
8  public class DotsHit extends NotificationBroadcasterSupport implements DotsHitMBean {
9      private int totalDots = 0;
10     private int totalHits = 0;
11     private long counter = 0;
12
13     @Override
14     public int getTotalDots(){
```

```
15          return totalDots;
16      }
17
18      @Override
19      public int getTotalHits(){
20          return totalHits;
21      }
22
23      @Override
24      public void resetCounters(){
25          totalDots = 0;
26          totalHits = 0;
27      }
28
29      public void addPoint(boolean isHit) {
30          totalDots++;
31          if (isHit) {
32              totalHits++;
33          }
34          if (totalDots % 5 == 0) {
35              Notification notification = new Notification(
36                      "org.example.points.notification", this, counter++,
37                      System.currentTimeMillis(), "Total points reached multiple of 5.");
38              sendNotification(notification);
39          }
40      }
41  }
```

Листинг 1: DotsHit.java

Интервал между кликами

```
1  package org.example.MBean.classes;
2
3  import org.example.MBean.interfaces.ClickIntervalMBean;
4
5  import java.util.LinkedList;
```

```
6  import java.util.Queue;
7
8  public class ClickInterval implements ClickIntervalMBean {
9      private final Queue<Long> clickTimes = new LinkedList<>();
10     private final static int MAX_HISTORY = 100;
11
12     @Override
13     public double getAverageInterval() {
14         if (clickTimes.size() < 2) {
15             return 0;
16         }
17         long totalInterval = 0;
18         Long prevTime = clickTimes.peek();
19         for (Long time : clickTimes) {
20             if (prevTime != time) {
21                 totalInterval += (time - prevTime);
22                 prevTime = time;
23             }
24         }
25         return totalInterval / (double) (clickTimes.size() - 1);
26     }
27
28     @Override
29     public void reset() {
30         clickTimes.clear();
31     }
32
33     public void recordClick() {
34         long now = System.currentTimeMillis();
35         clickTimes.add(now);
36         if (clickTimes.size() > MAX_HISTORY) {
37             clickTimes.poll();
38         }
39     }
40 }
```

Листинг 2: ClickInterval.java

Слушатель уведомлений

```
1  package org.example.MBean;
2
3  import javax.management.Notification;
4  import javax.management.NotificationListener;
5
6  public class DotsHitListener implements NotificationListener {
7      @Override
8      public void handleNotification(Notification notification, Object handback){
9          System.out.println("!!!Notification!!! " + notification.getMessage());
10     }
11 }
```

Листинг 3: ClickInterval.java

Модифицированный PointBean

```
1  package org.example;
2
3
4  import org.example.MBean.classes.ClickInterval;
5  import org.example.MBean.classes.DotsHit;
6  import org.example.MBean.DotsHitListener;
7
8  import javax.annotation.PostConstruct;
9  import javax.faces.bean.ApplicationScoped;
10 import javax.faces.bean.ManagedBean;
11 import javax.faces.context.FacesContext;
12 import javax.management.*;
13 import javax.persistence.*;
14 import javax.transaction.Transactional;
15 import java.lang.management.ManagementFactory;
16 import java.time.Instant;
17 import java.time.LocalDateTime;
18 import java.time.ZoneOffset;
19 import java.time.format.DateTimeFormatter;
20 import java.util.List;
```

```
21 import java.util.Map;
22 import java.util.logging.Level;
23 import java.util.logging.Logger;
24
25
26
27 @ManagedBean(name = "bean")
28 @ApplicationScoped
29 public class PointBean {
30
31     private double x;
32
33     private double y;
34
35     private double r;
36     private int timezoneOffset;
37
38     private DotsHit dotsHit;
39     private ClickInterval clickInterval;
40
41     private static final EntityManagerFactory EMF = Persistence.createEntityManagerFactory("lab3");
42
43     @PostConstruct
44     public void init(){
45         registerMBeans();
46         addNotificationListener();
47     }
48
49     private void registerMBeans(){
50         try {
51             MBeanServer mBeanServer = ManagementFactory.getPlatformMBeanServer();
52
53
54             ObjectName dotsHitName = new ObjectName("org.example.MBean.classes:name=dotsHit");
55             dotsHit = new DotsHit();
56             try {
57                 mBeanServer.registerMBean(dotsHit, dotsHitName);
```

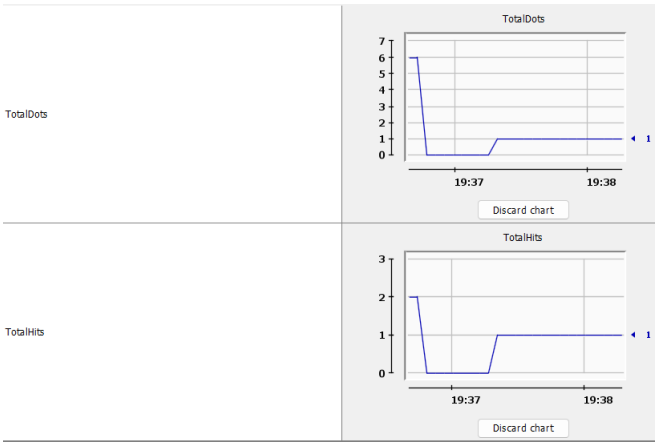
```
58              }catch (MBeanRegistrationException | InstanceAlreadyExistsException |
    NotCompliantMBeanException e){
59                  System.err.println("Error registering bean " + e.getMessage());
60              }
61
62              ObjectName clickIntervalName = new ObjectName("org.example.MBean.classes:name=clickInterval
    ");
63              clickInterval = new ClickInterval();
64              try {
65                  mBeanServer.registerMBean(clickInterval,clickIntervalName);
66              }catch (MBeanRegistrationException | InstanceAlreadyExistsException |
    NotCompliantMBeanException e){
67                  System.err.println("Error registering bean " + e.getMessage());
68              }
69          }catch (Exception e){
70              System.err.println("Error creating MBean server " + e.getMessage());
71          }
72      }
73
74      private void addNotificationListener() {
75          try {
76              MBeanServer mBeanServer = ManagementFactory.getPlatformMBeanServer();
77              ObjectName dotsHitName = new ObjectName("org.example.MBean.classes:name=dotsHit");
78              NotificationListener listener = new DotsHitListener();
79              NotificationFilter filter = new NotificationFilterSupport();
80              mBeanServer.addNotificationListener(dotsHitName, listener, filter, null);
81          } catch (Exception e) {
82              System.err.println("Error adding notification listener: " + e.getMessage());
83          }
84      }
85      ...
86
87      @Transactional
88      public String checkPoint() {
89          EntityManager entityManager = EMF.createEntityManager();
90          EntityTransaction transaction = entityManager.getTransaction();
91          try {
```

```
92              ...
93              dotsHit.addPoint(hit);
94              ...
95
96          } catch (PersistenceException e) {
97              Logger.getLogger(PointBean.class.getName()).log(Level.SEVERE, "
                                            ", e.getMessage());
98          } catch (IllegalArgumentException e) {
99              Logger.getLogger(PointBean.class.getName()).log(Level.WARNING, "
                                        ", e.getMessage());
100         } catch (Exception e) {
101             Logger.getLogger(PointBean.class.getName()).log(Level.SEVERE, "
                     ", e.getMessage());
102         } finally {
103             entityManager.close();
104         }
105
106         return "main";
107     }
108
109     public void handleCanvasClick() {
110         ...
111         clickInterval.recordClick();
112         checkPoint();
113     }
114 }
```

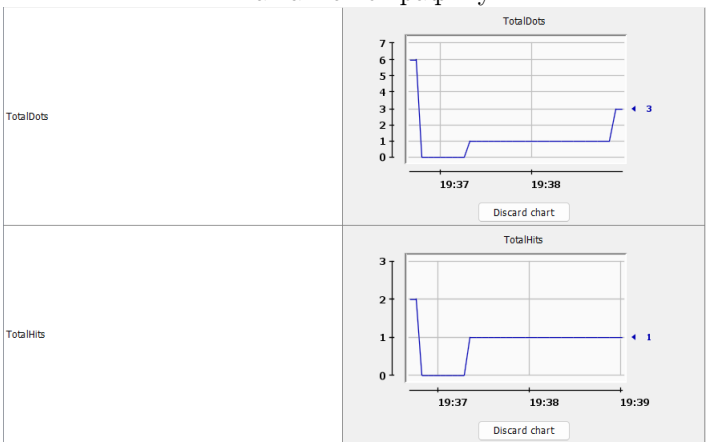
Листинг 4: PointBean.java

Показания JConsole

DotsHit



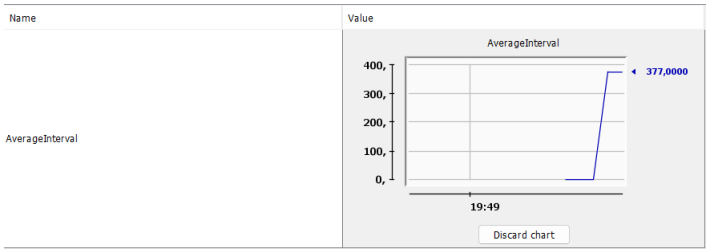
1 нажатие по графику



2 нажатия мимо графика

3 нажатия по графику

ClickInterval



Небольшой интервал между 2 короткими нажатиями

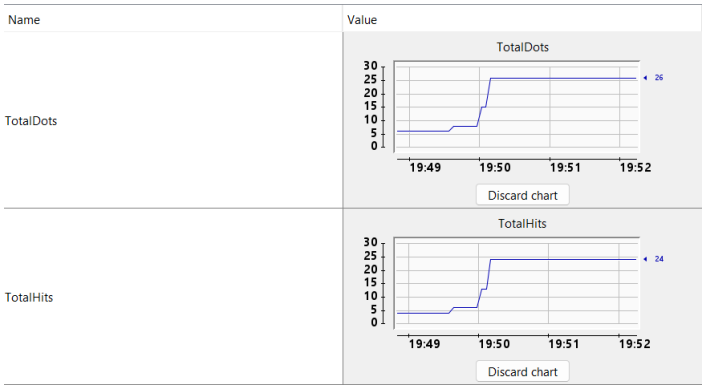
Большой интервал, много нажатий, интервал, еще нажатия

classpath

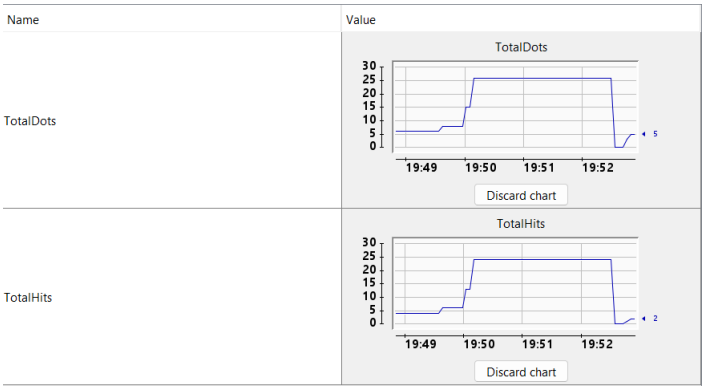
Значение classpath для данной JVM: `E:\prog\wildfly\wildfly-20.0.1.Final\jboss-modules.jar`.

Показания VisualVM

DotsHit

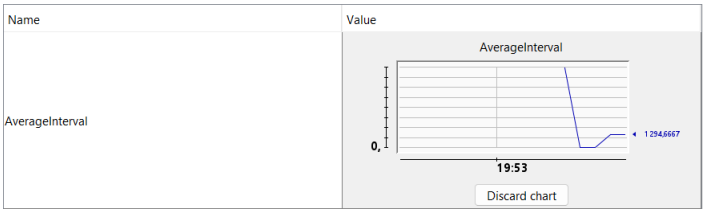


Много нажатий

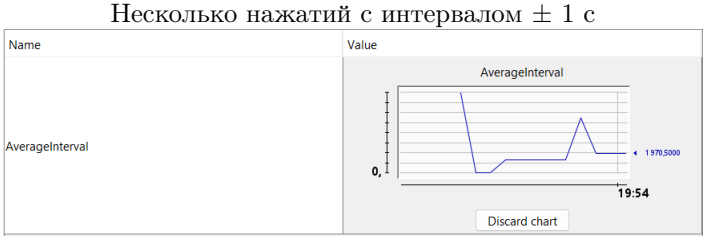


Сброс, несколько нажатий

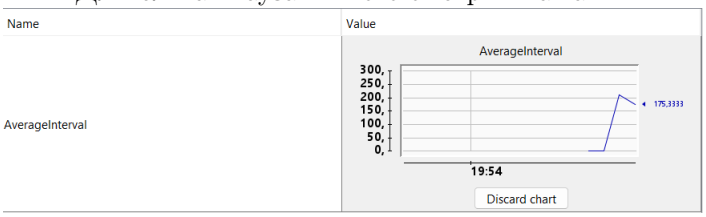
ClickInterval



Несколько нажатий с интервалом ± 1 с



Длительная пауза и много быстрых нажатий



Сброс и много быстрых нажатий

Поток, потребляющий наибольший процент времени CPU

Наибольший процент времени потребляет поток **AttachListener**.

Проблема в программе

Описание выявленной проблемы

В ходе исследования программы с помощью VisualVM и Eclipse Memory Analyzer была выявлена проблема накопления сообщений с ошибками, что вызывало бесконечный рост Heap (утечка памяти).

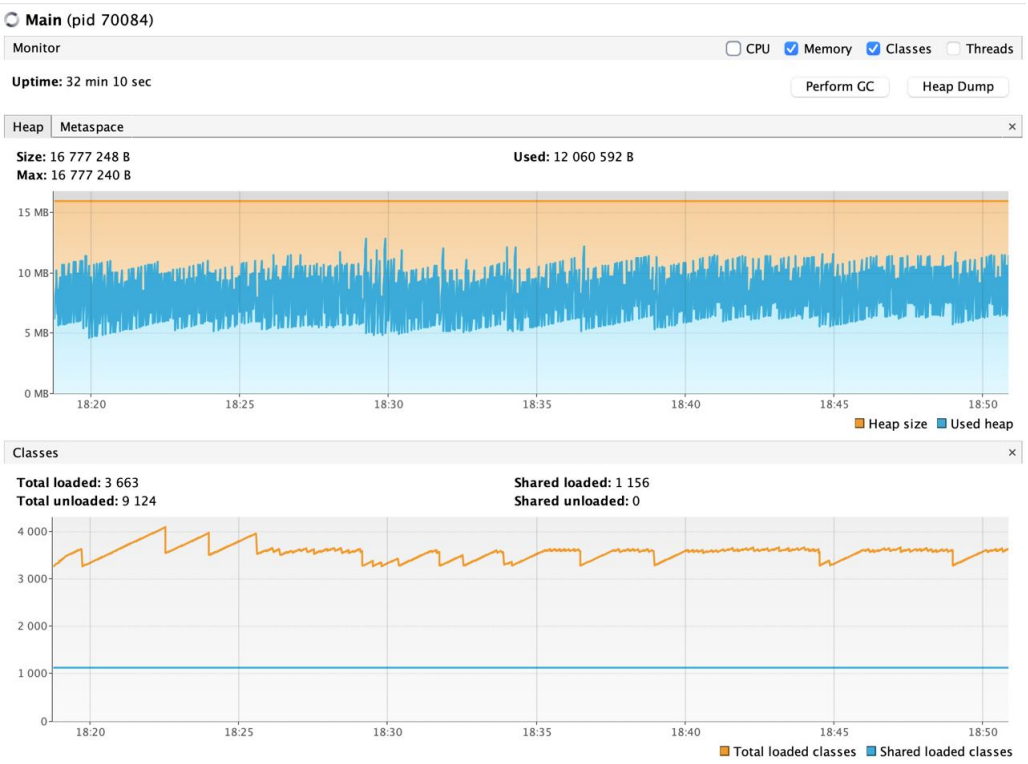
Проблема заключалась в классе **JavaScript**, а именно, в ArrayList, хранящим сообщения об ошибках. Метод clearErrorMessage из JavaScript используется в классе HttpUnitOptions (через JavaSciptEngineFactory), однако метод **clearScriptErrorMessages** не используется, что приводит к накоплению сообщений и росту кучи.

В качестве решения можно добавить проверку количества сообщений перед добавлением. А если размер листа превысил заданный размер — удалять самое старое сообщение:

```
1        private void handleScriptException( Exception e, String badScript ) {
2            final String errorMessage = badScript + " failed: " + e;
3            if (!(e instanceof EcmaError) && !(e instanceof EvaluatorException)) {
4                e.printStackTrace();
5                throw new RuntimeException( errorMessage );
6            } else if (isThrowExceptionsOnError()) {
7                e.printStackTrace();
8                throw new ScriptException( errorMessage );
9            } else {
10               // fix
11               if(_errorMessages.size() > 20){
12                   _errorMessages.remove(0);
13               }
14               _errorMessages.add( errorMessage );
15           }
16       }
```

Листинг 5: DotsHit.java

До решения проблемы



Размер занятого места постепенно растет

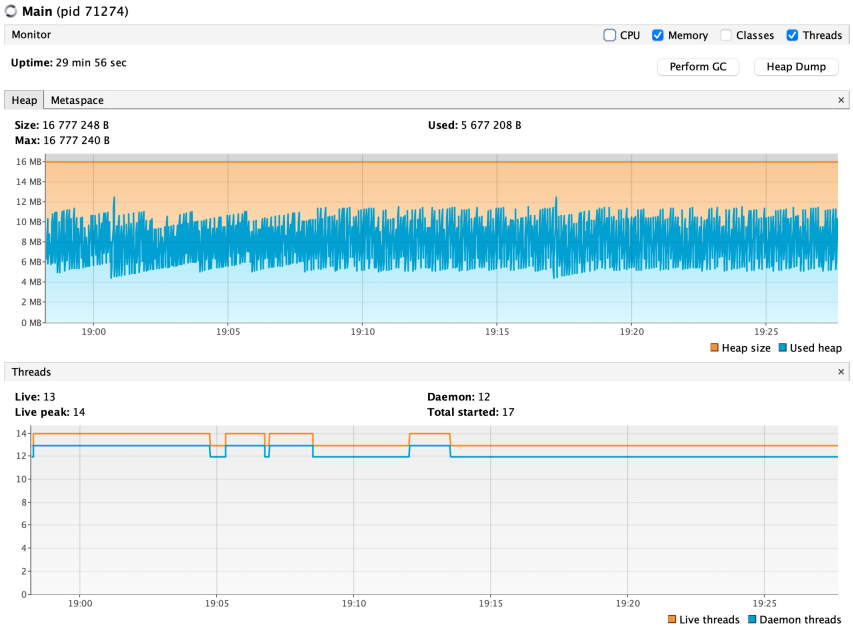
java.util.**ArrayList**	1 030	(0,9 %)	24 720 B	(0,4 %)	1 843 528 B	(33,1 %)
java.util.**ArrayList#1030** : 435 elements			24 B	(0 %)	1 364 584 B	(24,5 %)
<fields>						
elementData = java.lang.**Object[]#757** : 474 items			1 912 B	(0 %)	1 364 560 B	(24,5 %)
[51] = class com.meterware.httpunit.javascript.**JavaScript** : JavaScript			72 B	(0 %)	1 306 056 B	(23,4 %)
[389] = class org.cyberneko.html.**HTMLElements** : HTMLElements			72 B	(0 %)	12 632 B	(0,2 %)
[31] = class org.mozilla.javascript.optimizer.**Codegen** : Codegen			72 B	(0 %)	10 712 B	(0,2 %)
[321] = class org.apache.html.dom.**HTMLDocumentImpl** : HTMLD			72 B	(0 %)	6 872 B	(0,1 %)
[400] = class org.mozilla.javascript.**Parser** : Parser			72 B	(0 %)	5 544 B	(0,1 %)
[113] = class org.mozilla.javascript.**NativeDate** : NativeDate			72 B	(0 %)	3 456 B	(0,1 %)
[8] = class org.mozilla.javascript.**Context** : Context			72 B	(0 %)	3 424 B	(0,1 %)
[282] = class org.apache.xerces.util.**URI** : URI			72 B	(0 %)	3 288 B	(0,1 %)

Размер ArrayList с ошибками очень большой

Class Name	Shallow Heap	Retained Heap
java.lang.Object[9869] @ 0x7ff402090	37 496	1 117 928
elementData java.util.ArrayList @ 0x7ff08db10	24	1 117 952
_errorMessages class com.meterware.httpunit.javascript.JavaScript @ 0x7ff08d9a0	16	1 118 008
[51] java.lang.Object[474] @ 0x7ff0f87f8	1 912	1 163 792
elementData java.util.ArrayList @ 0x7ffe5ab70	24	1 163 816
classes jdk.internal.loader.ClassLoaders$AppClassLoader @ 0x7ffe58a10	96	1 234 408

Eclipse Memory Tool так же помечает список как утечку памяти

После решения проблемы



Размер занятого места постепенно растет

java.util.**ArrayList**	1 030	(1 %)	24 720 B	(0,6 %)	542 856 B	(12,8 %)
java.util.**ArrayList#409** : 151 elements			24 B	(0 %)	78 200 B	(1,8 %)
<fields>						
elementData = java.lang.**Object[]#1577** : 211 items			864 B	(0 %)	78 176 B	(1,8 %)
static **<resolved_references>** = java.lang.**Object[]#3448** : 6 items			40 B	(0 %)	208 B	(0 %)
static **DEFAULTCAPACITY_EMPTY_ELEMENTDATA** = java.lang.**Obje**			16 B	(0 %)	16 B	(0 %)
static **EMPTY_ELEMENTDATA** = java.lang.**Object[]#741** : 0 items			16 B	(0 %)	16 B	(0 %)
size = int **151**			-		-	
modCount = int **151**			-		-	
static **DEFAULT_CAPACITY** = int **10**			-		-	
static **serialVersionUID** = long **8683452581122892189**			-		-	
static **<classLoader>** = null			-		-	
<references>						
java.util.**ArrayList#1002** : 2 elements			24 B	(0 %)	73 104 B	(1,7 %)
<fields>						
elementData = java.lang.**Object[]#3258** : 10 items			56 B	(0 %)	73 080 B	(1,7 %)
static **<resolved_references>** = java.lang.**Object[]#3448** : 6 items			40 B	(0 %)	208 B	(0 %)
static **DEFAULTCAPACITY_EMPTY_ELEMENTDATA** = java.lang.**Obje**			16 B	(0 %)	16 B	(0 %)
static **EMPTY_ELEMENTDATA** = java.lang.**Object[]#741** : 0 items			16 B	(0 %)	16 B	(0 %)
size = int **2**			-		-	
modCount = int **2**			-		-	
static **DEFAULT_CAPACITY** = int **10**			-		-	
static **serialVersionUID** = long **8683452581122892189**			-		-	
static **<classLoader>** = null			-		-	
<references>						
java.util.**ArrayList#1030** : 434 elements			24 B	(0 %)	66 096 B	(1,6 %)
java.util.**ArrayList#1029** : 57 elements			24 B	(0 %)	11 944 B	(0,3 %)

Размер ArrayList с ошибками очень большой

Eclipse Memory Tool перестал помечать утечку памяти.

Выводы

В процессе выполнения лабораторной работы я познакомился с различными методами мониторинга Java приложения. Мной были созданы простые MBean классы и проведен их мониторинг. Я так же поработал с отладочными системами VisualVM, JConsole и Eclipse Memory Tool. Я получил опыт в поиске и решении проблем с приложениями.