Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Методы и средства программной инженерии.

Лабораторная работа №4.

Вариант: 89996

Выполнили:

Газизов Анвар Василович

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 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.

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

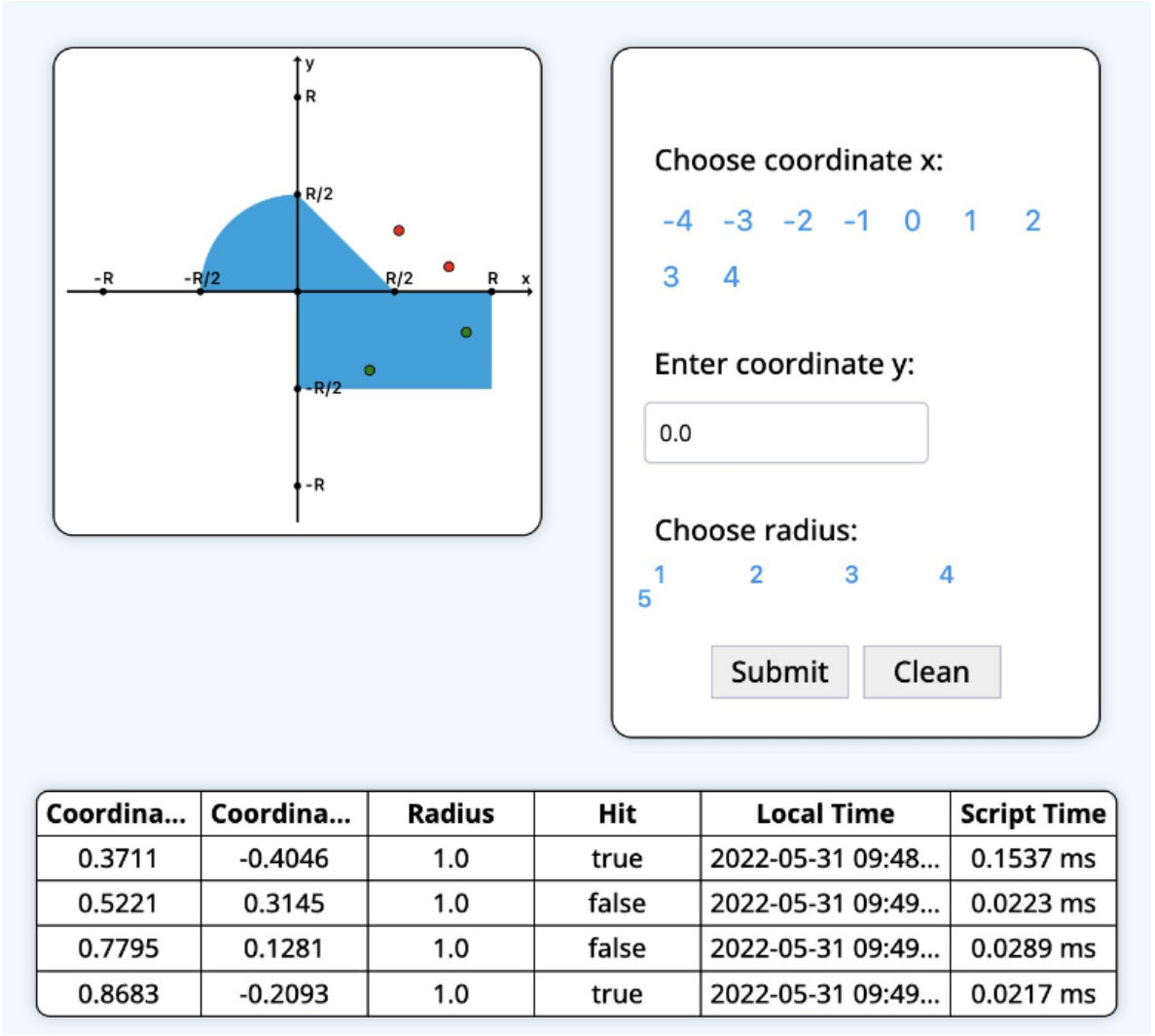
```
public class PointCounter extends NotificationBroadcasterSupport implements
PointCounterMXBean {
    private int sequenceNumber = 0;
    private int count = 0;
    private int hitCount = 0;

    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 MissPercentageMXBean {

    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 JConcole со снятыми показаниями, выводы по результатам мониторинга:**

Показания MBeans:



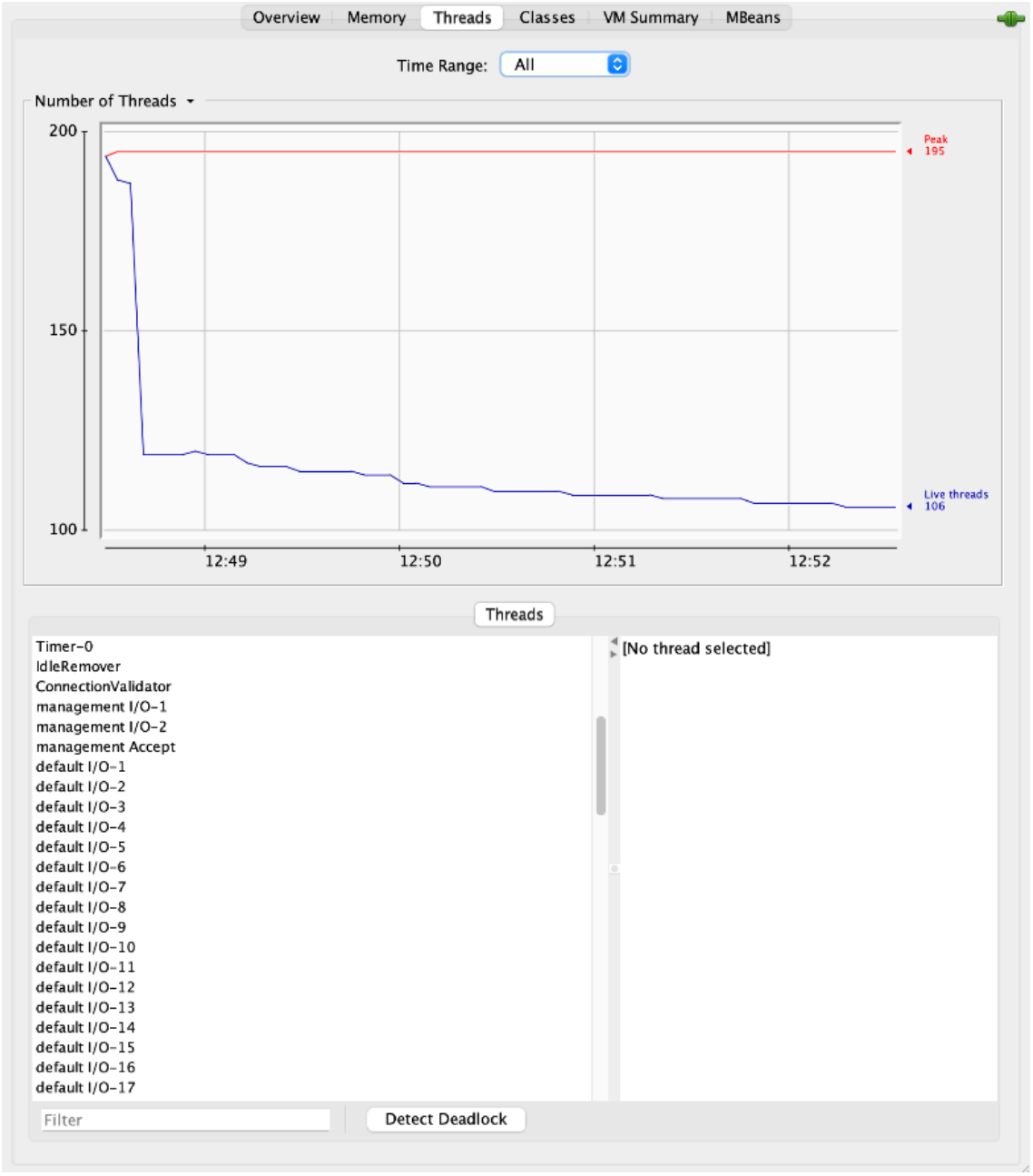
Coordina...	Coordina...	Radius	Hit	Local Time	Script Time
0.3711	-0.4046	1.0	true	2022-05-31 09:48...	0.1537 ms
0.5221	0.3145	1.0	false	2022-05-31 09:49...	0.0223 ms
0.7795	0.1281	1.0	false	2022-05-31 09:49...	0.0289 ms
0.8683	-0.2093	1.0	true	2022-05-31 09:49...	0.0217 ms

Оповещения MBean:

Notification buffer

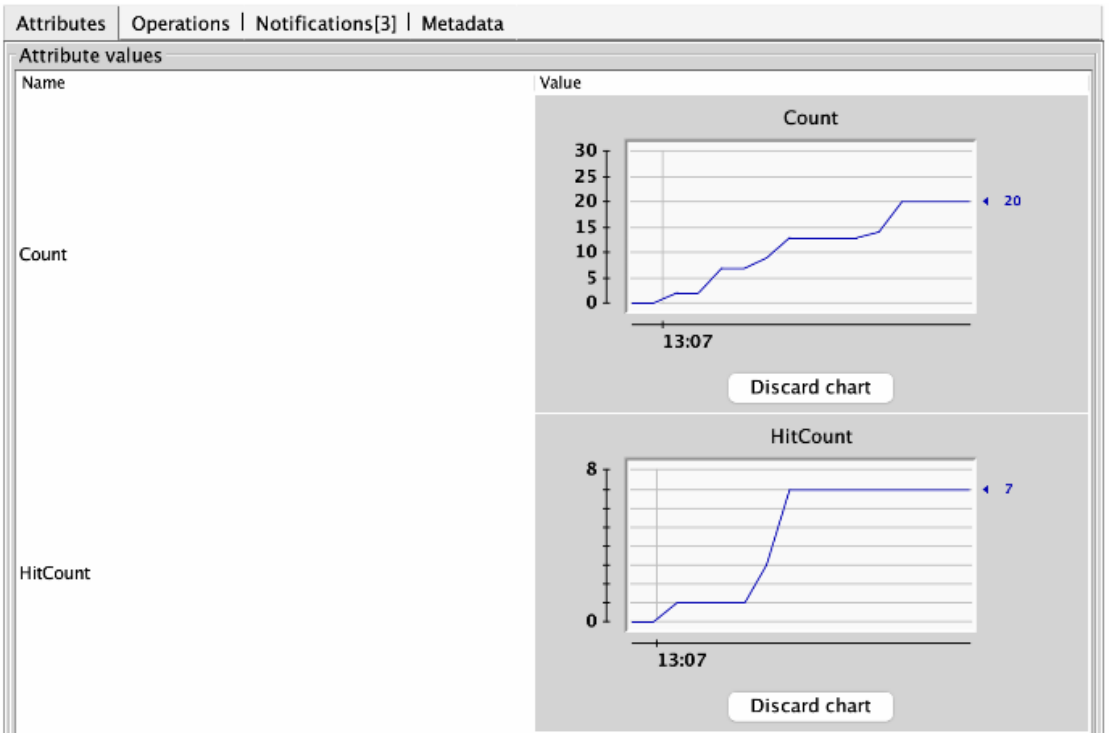
TimeStamp	Type	UserData	SeqNum	Message	Event	Source
12:53:30:999	multiple_of_10		4	The number of points is 50	javax....	nullnu...
12:53:28:513	multiple_of_10		3	The number of points is 40	javax....	nullnu...
12:53:27:046	multiple_of_10		2	The number of points is 30	javax....	nullnu...
12:53:25:647	multiple_of_10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

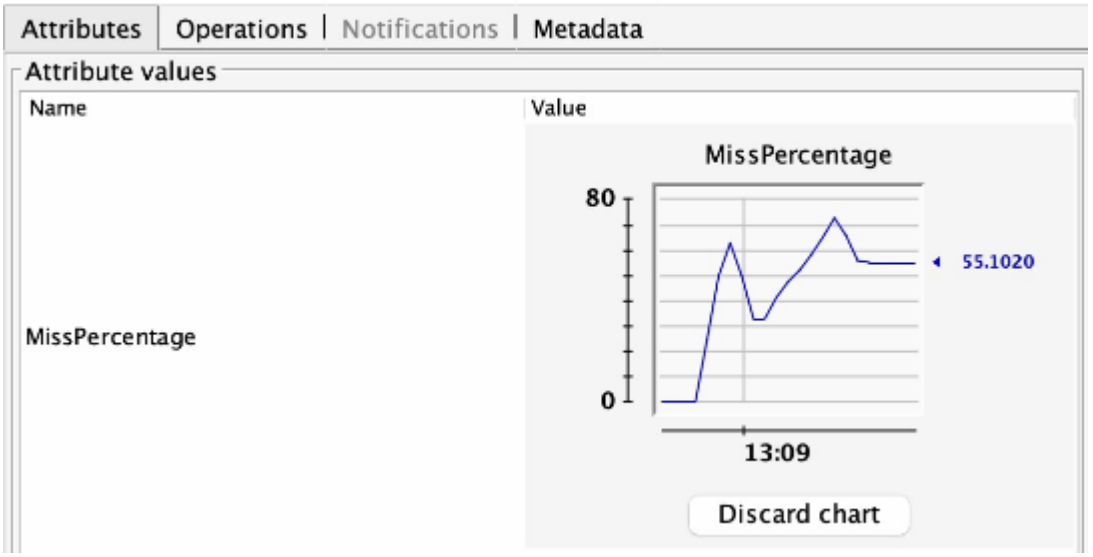
Имена потоков, выполняющихся при запуске программы:



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

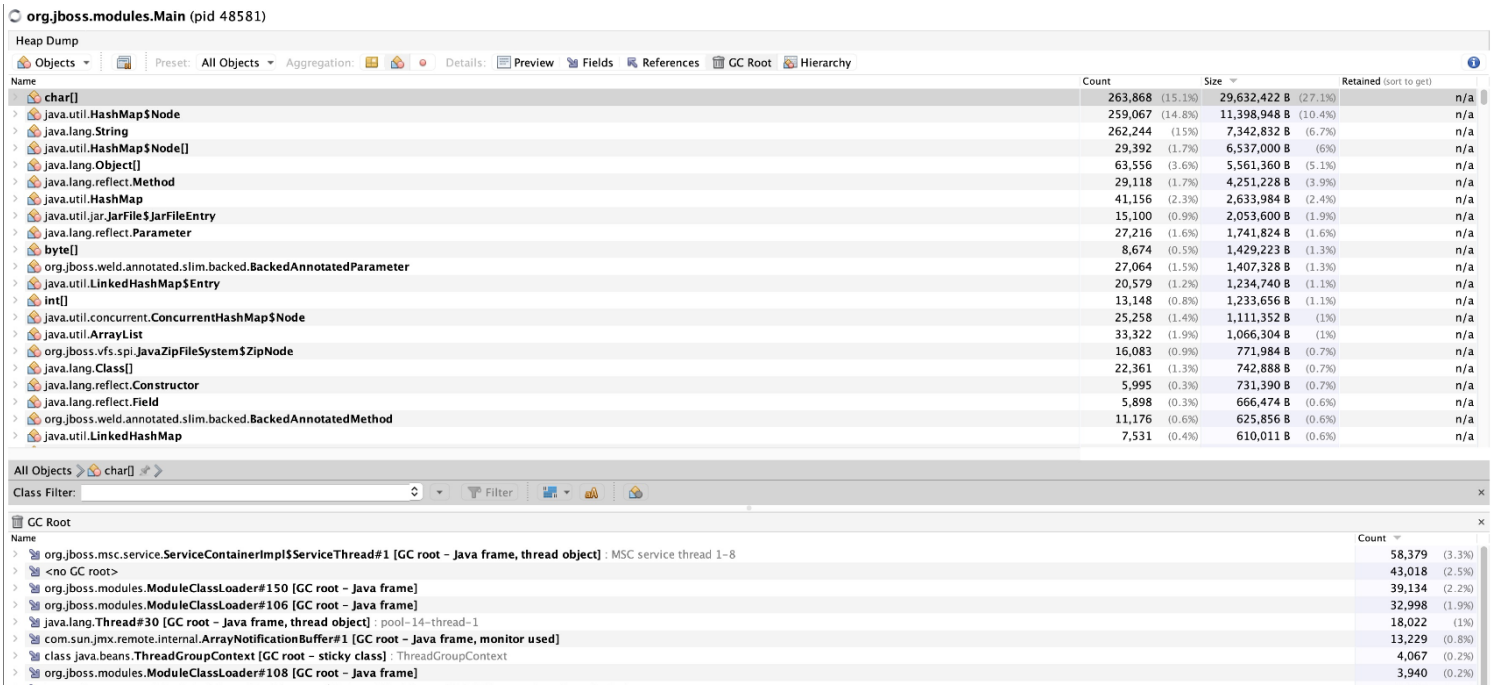
Графики изменения показаний MBeans:





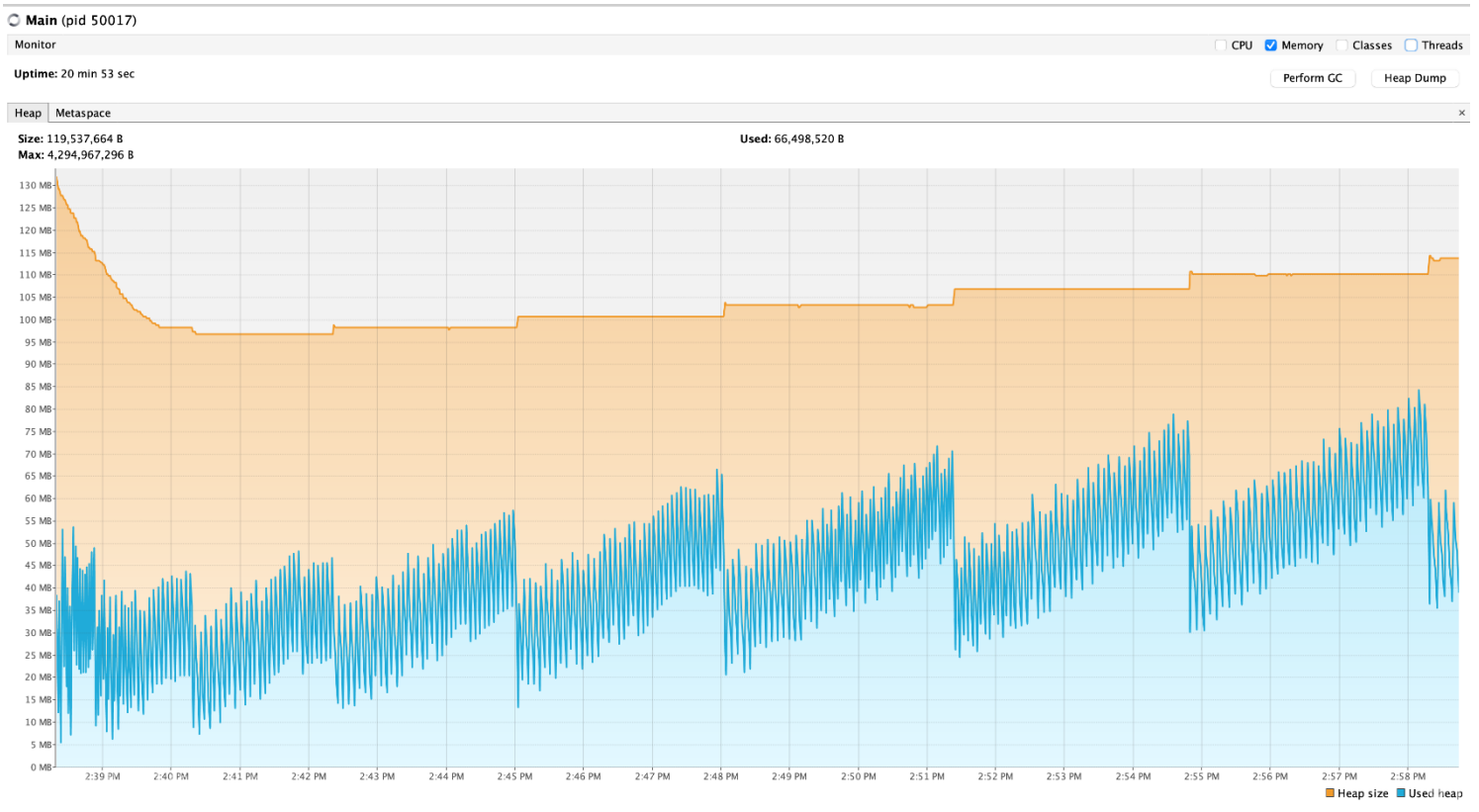
Класс, объекты которого занимают наибольший объем памяти JVM:

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.

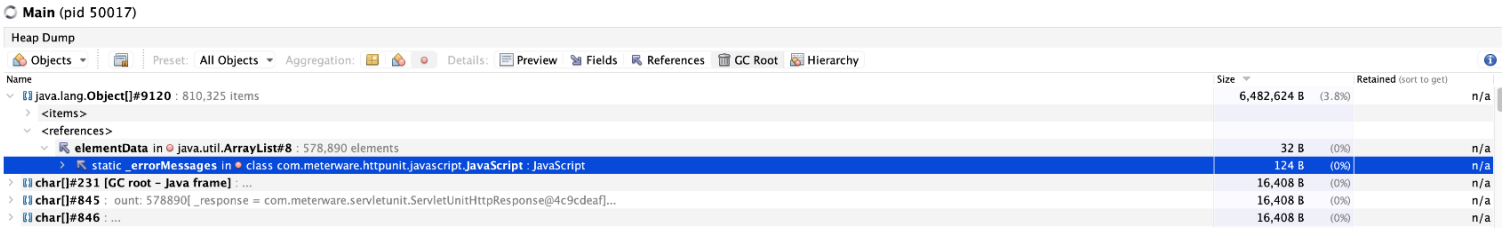


5. Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти:

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.



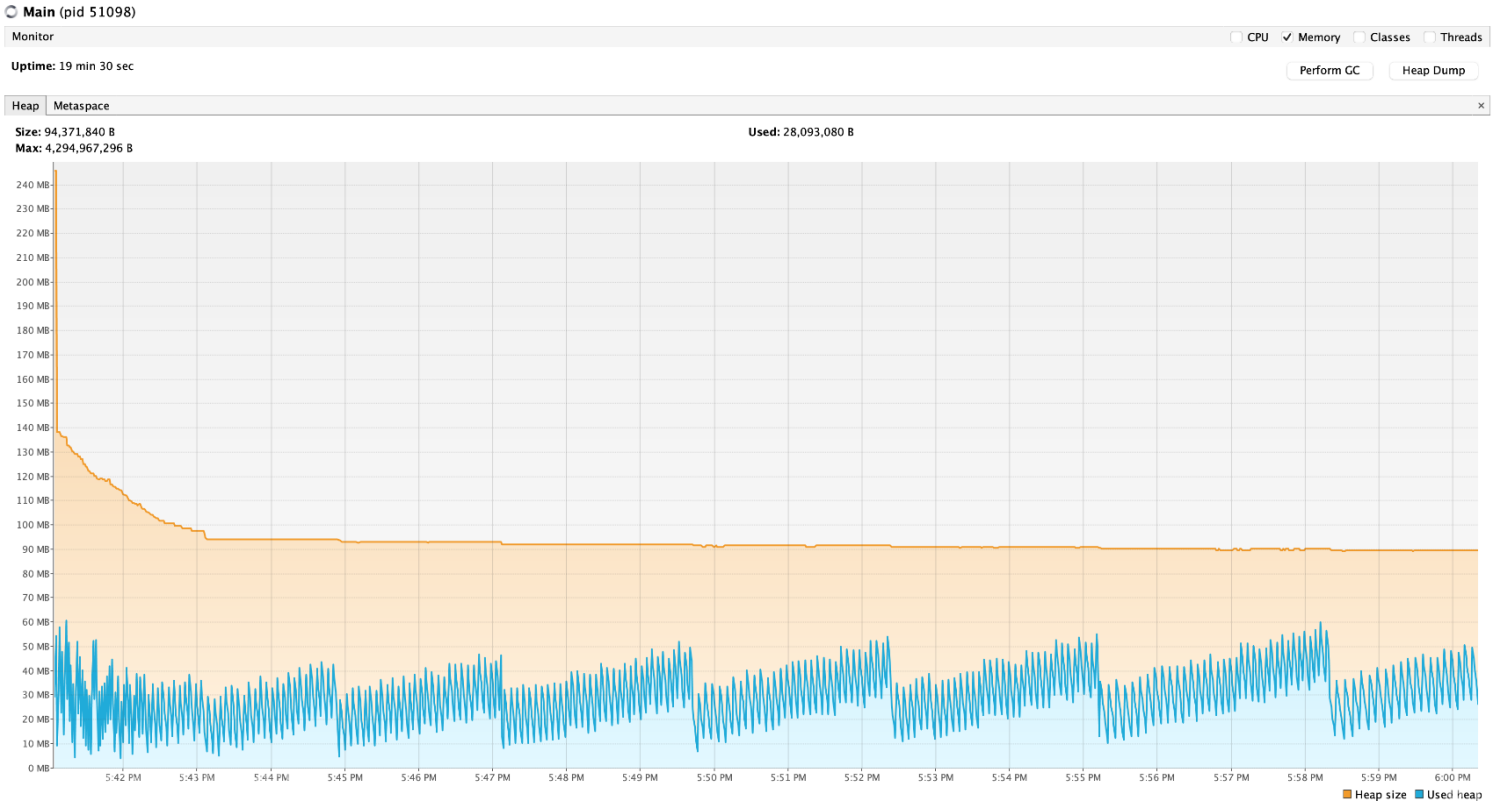
Изучив обращения к этому ArrayList-у, можно увидеть, что элементы в него добавляются, но никогда не очищаются.

```
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 (_errorMessages.size() >= 100) clearErrorMessages();
        _errorMessages.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.