**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

**Факультет программной инженерии и компьютерной техники**

**Отчет**

**по лабораторной работе №4**

**по дисциплине**

**«Основы программной инженерии»**

**Вариант 322**

**Выполнили: Краков Кирилл Константинович**

**Кадилов Михаил Владимирович**

**Группа: P3231**

**Преподаватель: Кулинич Ярослав Вадимович**

**Санкт-Петербург**

**2024**

Оглавление

[Текст задания 3](#_Toc167518557)

[Исходный код разработанных MBean-классов и сопутствующих классов 4](#_Toc167518558)

[Скриншоты программы JConcole со снятыми показаниями, выводы по результатам мониторинга. 6](#_Toc167518559)

[Скриншоты программы VisualVM со снятыми показаниями, выводы по результатам профилирования. 8](#_Toc167518560)

[Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти 10](#_Toc167518561)

[Вывод 16](#_Toc167518562)

# Текст задания

Вариант № 322

1. Для своей программы из [лабораторной работы #3](https://se.ifmo.ru/courses/web#lab3) 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 процентное отношение "попаданий" 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](https://se.ifmo.ru/documents/10180/189115/HttpUnit.tar.gz/7bf1032e-d16e-be85-c71b-dbe73c0178ba?t=1651168887037&download=true). 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.

**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.

# Исходный код разработанных MBean-классов и сопутствующих классов

PointCounterMBean:

package beans.mbeans;  
  
public interface PointCounterMBean {  
 int getTotalPoints();  
 int getMissedPoints();  
 void incrementTotalPoints();  
 void incrementMissedPoints();  
 void resetConsecutiveMisses();  
 void resetAndInitializeCounts();  
}

PointCounter:

package beans.mbeans;  
  
import beans.Point;  
import beans.ResultTable;  
  
import javax.management.Notification;  
import javax.management.NotificationBroadcasterSupport;  
import java.util.List;  
import java.util.concurrent.atomic.AtomicInteger;  
  
public class PointCounter extends NotificationBroadcasterSupport implements PointCounterMBean{  
 private AtomicInteger totalPoints = new AtomicInteger(0);  
 private AtomicInteger missedPoints = new AtomicInteger(0);  
 private int consecutiveMisses = 0;  
 private long sequenceNumber = 1;  
 private ResultTable resultTable;  
  
 public PointCounter(ResultTable resultTable) {  
 this.resultTable = resultTable;  
 initializeCounts();  
 }  
  
 private void initializeCounts() {  
 totalPoints.set(0);  
 missedPoints.set(0);  
 List<Point> results = resultTable.getResults();  
 for (Point point : results) {  
 totalPoints.incrementAndGet();  
 if (!point.isHit()) {  
 missedPoints.incrementAndGet();  
 }  
 }  
 }  
  
 @Override  
 public int getTotalPoints() {  
 return totalPoints.get();  
 }  
  
 @Override  
 public int getMissedPoints() {  
 return missedPoints.get();  
 }  
  
 @Override  
 public void incrementTotalPoints() {  
 totalPoints.incrementAndGet();  
 }  
  
 @Override  
 public void incrementMissedPoints() {  
 missedPoints.incrementAndGet();  
 consecutiveMisses++;  
 if (consecutiveMisses == 3) {  
 Notification n = new Notification("ConsecutiveMisses", this, sequenceNumber++,  
 System.*currentTimeMillis*(), "the user made 3 misses in a row");  
 sendNotification(n);  
 consecutiveMisses = 0;  
 }  
 }  
  
 @Override  
 public void resetConsecutiveMisses() {  
 consecutiveMisses = 0;  
 }  
  
 @Override  
 public void resetAndInitializeCounts() {  
 totalPoints.set(0);  
 missedPoints.set(0);  
 consecutiveMisses = 0;  
 initializeCounts();  
 }  
}

HitPercentageMBean:

package beans.mbeans;  
  
public interface HitPercentageMBean {  
 double getHitPercentage();  
}

HitPercentage:

package beans.mbeans;  
  
public class HitPercentage implements HitPercentageMBean{  
 private PointCounterMBean pointCounter;  
  
 public HitPercentage(PointCounterMBean pointCounter) {  
 this.pointCounter = pointCounter;  
 }  
  
 @Override  
 public double getHitPercentage() {  
 int totalPoints = pointCounter.getTotalPoints();  
 int hittedPoints = pointCounter.getTotalPoints() - pointCounter.getMissedPoints();  
 if (totalPoints == 0) {  
 return 0;  
 }  
 return (double) hittedPoints / totalPoints \* 100;  
 }  
}

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

Снятые показания разработанных MBean-классов:

Изображение выглядит как текст, снимок экрана, программное обеспечение, диаграмма

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, диаграмма

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Сообщения о промахах:

Изображение выглядит как текст, линия, снимок экрана, число

Автоматически созданное описание

Имена всех потоков, выполняющихся при запуске программы:

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Выводы по результатам мониторинга:

MBeans PointCounter и HitPercentage были успешно разработаны и зарегистрированы в классе ResultTable c помощью MBeanServer. В случае 3 промахов подряд PointCounter отправляет уведомление (Notification), которое получается с помощью JConsole.

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

Снятый график изменения показаний разработанных MBean-классов с течением времени:

Изображение выглядит как текст, снимок экрана, программное обеспечение, диаграмма

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Определение имени класса, объекты которого занимают наибольший объём памяти JVM: (получили класс byte[])

Изображение выглядит как текст, снимок экрана, число, линия

Автоматически созданное описание

Определение пользовательского класса, в экземплярах которого находятся объекты класса byte[]: (получили класс beans.PointTable):

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Выводы по результатам мониторинга и профилирования:

На графике TotalPoints видны изменения количества точек в базе данных приложения. График показывает динамику добавления и удаления точек. Также на графике виден момент очистки базы данных. Аналогично, на графике MissedPoints видны изменения числа точек, не попадающих в область.

График HitPercentage показывает процентное соотношение "попаданий" к общему числу кликов. Видно, что процент "попаданий" варьируется с течением времени, что отражает взаимодействие пользователя с системой (создание точек на графике).

На основании профилирования Heap Dump, классом, объекты которого занимают наибольший объём памяти JVM, оказался класс byte[]. Пользовательским классом, в экземплярах которого находятся эти объекты, оказался класс beans.Point.

# Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти

Избыточный вызов метода java.lang.Thread.sleep(200):

При анализе производительности программы с помощью профилировщика Intellij Idea была обнаружена проблема связанная с избыточным вызовом метода java.lang.Thread.sleep(200). Данный метод вызывает приостановку выполнения потока на 200 миллисекунд, что приводит к задержкам при работе программы (main процесс большую часть времени находится в ожидании) и снижению общей производительности (всего 179 counts за 38 секунд, 22.8% работы CPU направлены на выполнение java.lang.Thread.run()).Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Метод не несёт никакой функциональной нагрузки и является избыточным. Его удаление не повлияло на корректность работы приложения, при этом main процесс стал занят большую часть времени, а также удалось добиться серьёзного повышения производительности за счёт устранения задержек (39453 counts за 38 секунд, 99,1 % работы CPU направлен на выполнение Main.main(String [])).

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Утечка памяти:

Для начала установим ограничение максимального размера кучи на 30 Мб (Для этого в конфигурацию запуска добавим опцию (-Xmx30m) для VM).

Через несколько минут после запуска программы выбрасывается исключение:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

При анализе графика использования памяти в куче видно, что размер кучи постоянно увеличивается, несмотря на работу garbage collector’a.

Изображение выглядит как текст, График, линия, снимок экрана

Автоматически созданное описание

Следовательно, можно сделать вывод, что утечка памяти действительно существует.

Локализуем проблему. Для этого проанализируем кучу и найдем классы объектов, которые занимают много памяти:

Изображение выглядит как текст, снимок экрана, линия, число

Автоматически созданное описание

Больше всего у нас объектов типа byte[] (массивов байт), а также объектов String, которые включают в себя объекты типа byte[]. При анализе Heap Dump наибольший размер в памяти занял объект java.lang.Object[]#7895 – arrayList под названием \_errorMessages, в котором содержится очень большое количество элементов типа String с информацией об ошибках.

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, линия, число

Автоматически созданное описание

Найдём исходный класс по его расположению (com.meterware.httpunit.javascript.JavaScript) и посмотрим, что в нём находится.



Добавление элементов в данный список происходит в методе класса JavaSript, а вот метод очищения списка нигде применяется:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Метод clearErrorMessages из класса JavaScript используется в классе com.meterware.httpunit.javascript. JavaScriptEngineFactory:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Метод getErrorMessages из JavaScriptEngineFactory используется в классе com.meterware.httpunit.HttpUnitOptions:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

А вот метод clearScriptErrorMessages из HttpUnitOptions не используется нигде:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Таким образом, элементы накапливаются в списке\_errorMessages при этом нигде не очищаются, что приводит к утечке памяти в связи с отсутствием очистки.

Устраним данную проблему, добавив очистку списка \_errorMessages после выполнения каждого запроса (методом clearScriptErrorMessages) в главном цикле программы:

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Шрифт

Автоматически созданное описание

Проверим устранение утечки памяти. Для этого снова запустим программу и проведём анализ графика изменения использования памяти в куче. В результате наблюдаем, что теперь не расходуется больше 30 Мб кучи, размер кучи постоянно не растёт и сборщик мусора работает в нормальном режиме:

Изображение выглядит как текст, снимок экрана, График, Шрифт

Автоматически созданное описание

Следовательно, утечка памяти была успешно устранена.

# Вывод

В ходе выполнения данной лабораторной работ мы научились создавать MBeans, анализировать программу в процессе ее выполнения, а также получать информацию о ресурсах, которые она использует. Примерами таких ресурсов могут являться время выполнения и используемая память. Для этого нам пришлось научиться работать с VisualVM, а также встроенным профилировщиком Intellij Idea. Помимо этого мы научились выводить сообщения в JConsole.