Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

**Лабораторная работа №4**

по дисциплине **«Основы программной инженерии»**

Вариант: **1580**

**Преподаватель**:   
Цопа Евгений Алексеевич

**Выполнил**: Барсуков Максим

**Группа**: Р3215

Санкт-Петербург, 2024

Оглавление

[1. Задание 3](#_Toc168467499)

[2. Выполнение 4](#_Toc168467500)

[1. Исходный код разработанных MBean 4](#_Toc168467501)

[2. JConsole 8](#_Toc168467502)

[3. VisualVM 12](#_Toc168467503)

[4. Исследование программы на утечки памяти 17](#_Toc168467504)

[3. Вывод 23](#_Toc168467505)

# Задание

Вариант: **1580**

|  |
| --- |
|  |

# Выполнение

<https://github.com/maxbarsukov-itmo/mispi-4>



## 1. Исходный код разработанных MBean

* web.beans. HitRatioMBean.java



* web.beans. AttemptStatsMBean.java



* web.beans.HitRatio.java



* web.beans.HitRatio.java



* web.utils.MBeanRegistry.java



## 2. JConsole

Показания MBean-классов, разработанных в ходе выполнения задания 1:

* **Метаданные** Mbean’ов:

|  |
| --- |
|  |
|  |

* **Показания** Mbean’ов:

|  |
| --- |
| C:\Users\maxim\AppData\Local\Microsoft\Windows\INetCache\Content.Word\msg812743785-189402.jpg |
|  |

* **VM Summary** и время, прошедшее с момента запуска виртуальной машины:

|  |
| --- |
|  |

* **Графики**:

|  |
| --- |
|  |
|  |

* **Уведомления**:

|  |
| --- |
|  |
|  |

* **Выводы по результатам мониторинга**:

В ходе мониторинга с использованием утилиты JConsole было

определено, что:

* Узнать время работы JVM можно с помощью Uptime в VM Stats.
* MBean HitRatio и AttemptStats были успешно разработаны и

зарегистрированы. В случае трех последовательных промахов

MBean AttemptStats отправляет уведомления, которые были получены с помощью JConsole. Таким образом, было зарегистрировано и

протестировано получение уведомлений от MBean, что позволяет

оперативно реагировать на события, что является важной частью

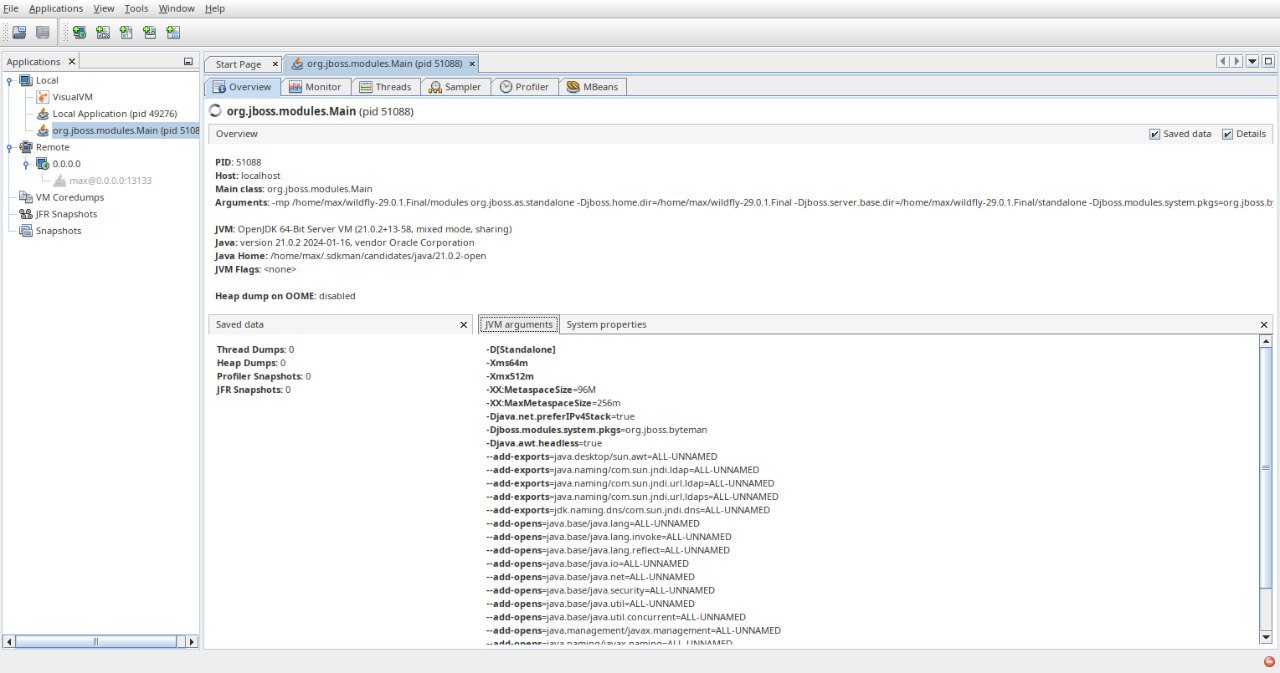
мониторинга. Аналогично с HitRatio MBean.

## 3. VisualVM

* **Снять график изменения показаний MBean-классов**:

|  |
| --- |
|  |
|  |

* **JVM Overview**:



* **Мониторинг**:

|  |
| --- |
|  |

* Определить **имя класса, объекты которого занимают наибольший объём** памяти JVM; **определить пользовательский класс**, в экземплярах которого находятся эти объекты.

|  |
| --- |
| C:\Users\maxim\AppData\Local\Microsoft\Windows\INetCache\Content.Word\msg812743785-189416.jpg |
|  |

* **Выводы по результатам мониторинга и профилирования**:

**Изменения показаний** MBean-классов с течением времени:

● HitRatio MBean: На графике HitRatio видны изменения количества точек в базе данных приложения, количество попаданий и процент попаданий. График показывает динамику добавления попавших и не попавших точек. Видно, что процент "попаданий" варьируется с течением времени, что отражает взаимодействие пользователя с системой (создание точек на графике).

● AttemptStats MBean: График AttemptStats показывает общее число установленных пользователем точек, а также число точек, не попадающих в область и также меняется со временем.

Определение **имени класса**, объекты которого занимают **наибольший объём** памяти JVM; определение пользовательского класса, в экземплярах которого находятся эти объекты.

● На основе профилирования памяти видно, что больше всего памяти занимают значения коллекции TreeMap, на втором месте итераторы – всего объекты класса TreeMap занимают ~45%.

● Больше всего памяти из пользовательских классов занимают объекты Attempt – 84% от пользовательских классов. Однако в масштабе всего приложения эти объекты занимают примерно 0.2%.

## 4. Исследование программы на утечки памяти

**Избыточный вызов метода**

В процессе анализа производительности приложения 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 ненужным задержкам и снижению общей производительности системы.

|  |
| --- |
|  |

Метод не несёт никакой функциональной нагрузки и является избыточным – поэтому его удаление не повлияет на корректность работы приложения, но повысит производительность за счет устранения задержек.

|  |
| --- |
|  |

**Утечка**

Установим максимальный размер кучи в 30Мб с помощью -Xmx30m и запустим программу и посмотрим, как она себя ведет, в VisualVM.

1. Мониторинг, видна очевидная переработка GC:

|  |
| --- |
| C:\Users\maxim\AppData\Local\Microsoft\Windows\INetCache\Content.Word\msg812743785-189555.jpg |

2. Программа падает через ~5 минут, независимо от выделенной памяти для кучи:

|  |
| --- |
| C:\Users\maxim\AppData\Local\Microsoft\Windows\INetCache\Content.Word\msg812743785-189557.jpg |

3. С помощью Heap Dump найдем объекты, занимающие большую часть памяти.

|  |
| --- |
|  |
|  |
|  |

Из графиков использования памяти видно, что

* При работе приложения на каждый запрос создаются экземпляры строк и массивы byte[], затем используются по всему приложению и остаются использованными, но не очищенными в памяти. Хоть GC и отрабатывает корректно, он тратит слишком много ресурсов на очищение памяти, поэтому чтобы убрать излишнюю нагрузку надо устранить утечку;
* Размер кучи постоянно увеличивается, что свидетельствует о проблемах с использованием памяти в программе;
* Через некоторое количество времени получаем ошибку OutOfMemoryError.

Немного исследовав кучу, находим повторяющиеся строки:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Объекты \_errorMessages хранятся в ArrayList

Найдем строчку с добавлением объектов в этот список:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описаниеВ результате получается накопление \_errorMessages в списке, за счет чего и получается переполнение памяти. В программе есть функция для очистки \_errorMessage, однако мы можем увидеть, что на самом деле она не используется

Изображение выглядит как текст, Шрифт, рукописный текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Решением будет очистка списка с \_errorMessage после выполнения очередного запроса.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Запустим программу. Теперь изменения памяти во времени менее пилообразные, GC работает более оптимально.

|  |
| --- |
|  |

Программа работает стабильно и не падает с OutOfMemoryException.

|  |
| --- |
| C:\Users\maxim\AppData\Local\Microsoft\Windows\INetCache\Content.Word\msg812743785-189571.jpg |

# Вывод

Во время выполнения лабораторной работы я познакомился с практикой написания MBeans в веб-приложениях, были изучены утилиты для мониторинга и профилирования работы программы JConsole и VisualVM, а также был получен опыт по полученным данным определять утечки памяти и устранять их.