МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

Кафедра компьютерной математики и программирования

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | Кочин Д. А. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7 |
| «Исследование производительности программного продукта с помощью профайлера» |
| по курсу: УПРАВЛЕНИЕ КАЧЕСТВОМ ПО |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4831 |  |  |  | К. А. Корнющенков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2020

**Цель работы**

В рамках лабораторной работы необходимо произвести профилирование приложения.

**Задание на лабораторную работу**

- Взять задание из лабораторной работы номер 2. Модифицировать полученный код чтобы разработанную функцию можно было выполнять большое количество раз (например, 10000).

- Подключиться к исполняемому коду профилировщиком (или использовать встроенный в IDE) и проанализировать – какой фрагмент кода занял больше всего процессорного времени.

- Предложить способы оптимизации

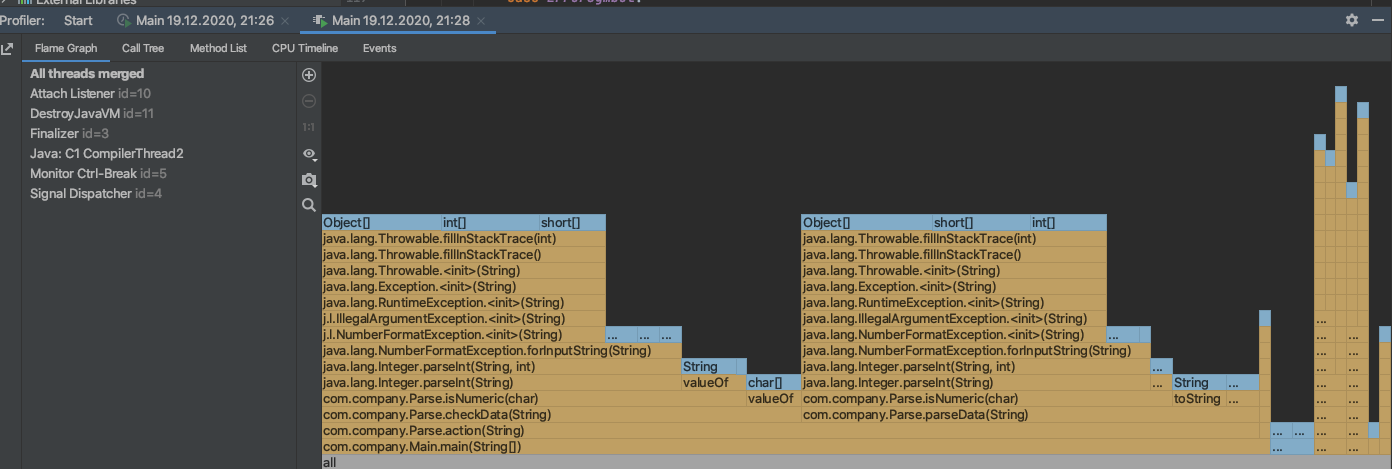


Рис 1 – Запись профайлера 1

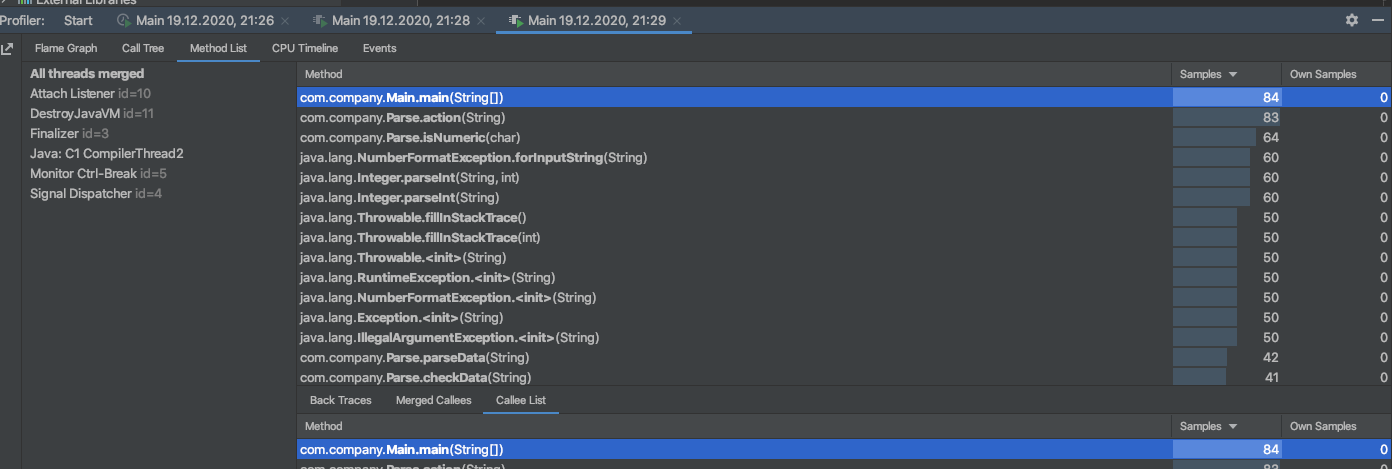


Рис 2 – Запись профайлера 2

**Анализ полученных данных**

Исходя из записей профайлера можно сделать вывод что основная нагрузка приходит на метод action класса Parse. Для увеличения производительности можно разделить метод на отдельные на 2 блока: 1 – срабатывает при правильно введенной строки, а 2 – отлавливает и выводит нужную ошибку

**Выводы**

В ходе лабораторной работы были проанализированны функции при помощи профилировщика. При помощи профилировщика были найдены узкие места программы и были предложены способы увилечения производительности и оптимизации кода.

**Приложение 1. Код программы**

package com.company;  
  
import java.util.LinkedList;  
import java.util.Scanner;  
  
class Pair<T,U> {  
 public final T t;  
 public final U u;  
  
 public Pair(T t, U u) {  
 this.t= t;  
 this.u= u;  
 }  
}  
  
enum ErrorEnum {  
 *Good*,  
 *ErrorSymbol*,  
 *ErrorFirstOrLastItemIsNotNumber*}  
  
class Parse{  
 public static boolean isNumeric(char str) {  
 try {  
 Integer.*parseInt*(String.*valueOf*(str));  
 return true;  
 } catch(NumberFormatException e){  
 return false;  
 }  
 }  
  
 public static ErrorEnum checkData(String data){  
 for (int i=0;i<data.length();i++){  
 if ((i == 0 || i == data.length() - 1) && !*isNumeric*(data.charAt(i))) {  
 return ErrorEnum.*ErrorFirstOrLastItemIsNotNumber*;  
 }  
 if (*isNumeric*(data.charAt(i)) | String.*valueOf*(data.charAt(i)).equals("+")  
 | String.*valueOf*(data.charAt(i)).equals("\*") | String.*valueOf*(data.charAt(i)).equals("/") |  
 String.*valueOf*(data.charAt(i)).equals("-")){  
 continue;  
 } else {  
 return ErrorEnum.*ErrorSymbol*;  
 }  
 }  
 return ErrorEnum.*Good*;  
 }  
  
 public static Pair<LinkedList<Integer>, LinkedList<String>> parseData(String data){  
 LinkedList<Integer> number = new LinkedList<Integer>();  
 LinkedList<String> action = new LinkedList<String>();  
 String timeNumber = "";  
 for (int i=0;i<data.length();i++){  
 if (!*isNumeric*(data.charAt(i))){  
 action.add(String.*valueOf*(data.charAt(i)));  
 if (timeNumber != "") {  
 number.add(Integer.*valueOf*(timeNumber));  
 timeNumber = "";  
 }  
 } else{  
 timeNumber += String.*valueOf*(data.charAt(i));  
 }  
 }  
 if (timeNumber != "") {  
 number.add(Integer.*valueOf*(timeNumber));  
 timeNumber = "";  
 }  
 return new Pair(number,action);  
 }  
  
 public static Pair<LinkedList<Integer>, LinkedList<String>> hardAction(LinkedList<Integer> number, LinkedList<String> action){  
 Integer i = 0;  
 Boolean check = true;  
 while (check){  
 if (action.get(i).equals("\*")) {  
 number.set(i,number.get(i)\*number.get(i+1));  
 number.remove(i+1);  
 action.remove(i+1-1);  
 i = 0;  
 }else if (action.get(i).equals("/")) {  
 number.set(i,number.get(i)/number.get(i+1));  
 number.remove(i+1);  
 action.remove(i+1-1);  
 i = 0;  
 }  
 i += 1;  
 if (action.size() <= i){  
 break;  
 }  
 }  
 return new Pair(number,action);  
 }  
  
 public static String simpleAction(LinkedList<Integer> number, LinkedList<String> action){  
 Integer i = 0;  
  
 while (!action.isEmpty()){  
 if (action.get(i).equals("+")) {  
 number.set(i,number.get(i)+number.get(i+1));  
 number.remove(i+1);  
 action.remove(i+1-1);  
 i = 0;  
 }else if (action.get(i).equals("-")) {  
 number.set(i,number.get(i)-number.get(i+1));  
 number.remove(i+1);  
 action.remove(i+1-1);  
 i = 0;  
 }  
 }  
 return String.*valueOf*(number.get(0));  
 }  
  
 public static String action(String data){  
 switch (*checkData*(data)){  
 case *Good*: {  
 Pair<LinkedList<Integer>, LinkedList<String>> parse = *parseData*(data);  
 parse = *hardAction*(parse.t, parse.u);  
 return *simpleAction*(parse.t, parse.u);  
 }  
 case *ErrorSymbol*:  
 return "Error - symbol";  
 case *ErrorFirstOrLastItemIsNotNumber*:  
 return "Error - first or last item is not number";  
 default:  
 return null;  
 }  
 }  
}  
  
public class Main {  
  
 public static void main(String[] args) {  
 for (int i=0;i<10000;i++) {  
 System.*out*.println(Parse.*action*("5+6\*2-3"));  
 }  
 }  
}