МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федерально автономное бюджетное образовательное учреждение высшего образования

«Севастопольский государственный университет»

кафедра Информационных систем

Куркчи Ариф Эрнестович

Институт информационных технологий и управления в технических системах

курс 3 группа ИС/б-31-о

09.03.02 Информационные системы и технологии (уровень бакалавриата)

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Тестирование программного обеспечения»

на тему «Исследование способов профилирования программного обеспечения»

Отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

(дата)

Руководитель практикума

ст. преподаватель   В. А. Строганов

(должность) (подпись) (инициалы, фамилия)

Севастополь 2016

1. ЦЕЛЬ РАБОТЫ

Исследовать критические по времени выполнения участки программного кода и возможности их устранения. Приобрести практические навыки анализа программ с помощью профайлера JProfiler.

1. ПОСТАНОВКА ЗАДАЧИ

Провести профилирование программного кода полного цикла работы программы с использованием теста ConsumerTest, написанного в лабораторной работе №3.

1. ХОД РАБОТЫ

Запустив профайлер JProfiler, предварительно зациклив выполнение теста ConsumerTest.fullCycle. Построив граф вызовов (рисунок 1) с корнем в самом затратном методе программы – вызове метода Stock.addItem и раскрыв ветви становится понятно, что наиболее узкой точкой программы является использование класса StringBuilder. В Java этот класс используется для абсолютно любой конкатенации строк, включая бинарный оператор «+», который транслируется в использование StringBuilder. Как видно по предыдущим узлам этот класс используется из метода Report.addLine. Этот метод производит добавление строки в отчёт, конкатинируя элементы строки отчёта.

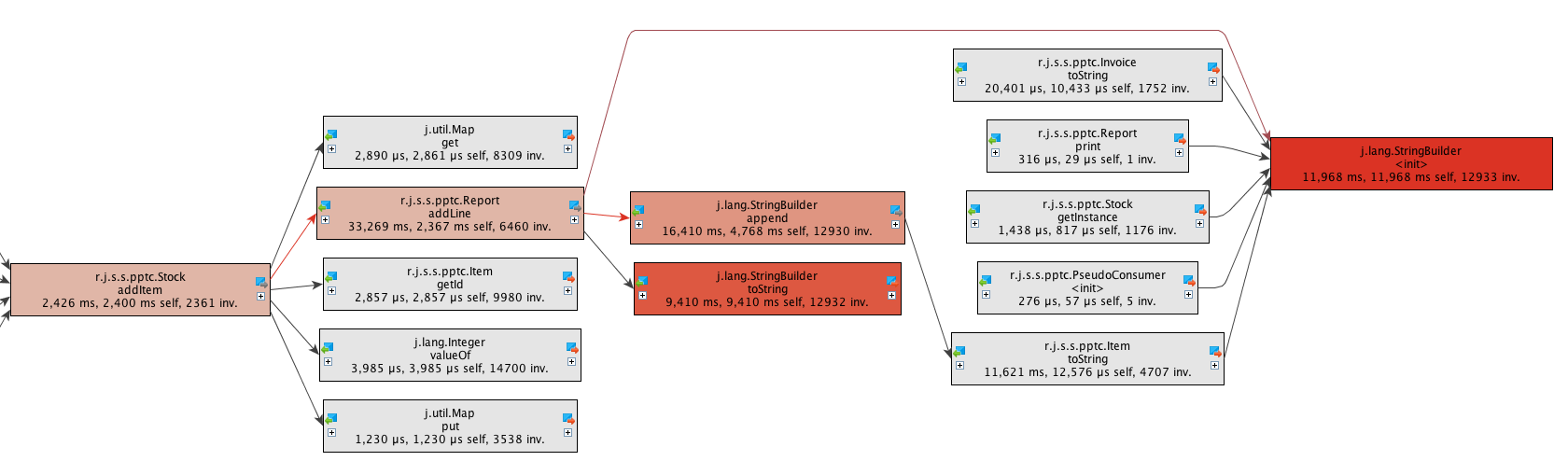


Рисунок 1 – Изначальный граф вызовов

|  |  |
| --- | --- |
| Исходный класс Report | |
| |  |  | | --- | --- | | 1 | package ru.justnero.study.sevsu.pptc; | | 2 |  | | 3 | import *java.util.ArrayList*; | | 4 | import *java.util.List*; | | 5 |  | | 6 | public class Report { | | 7 |  | | 8 | public final *String* name; | | 9 | public final *List<String>* lines; | | 10 |  | | 11 | public Report(*String* *name*) { | | 12 | this.name = name; | | 13 | this.lines = new *ArrayList<>*(); | | 14 | } | | 15 |  | | 16 | public *Report* addLine(final *Object*... *elements*) { | | 17 | *StringBuilder* sb = new *StringBuilder*(); | | 18 | for(*Object* el : elements) { | | 19 | sb.append(el); | | 20 | } | | 21 | lines.add(sb.toString()); | | 22 | return this; | | 23 | } | | 24 |  | | 25 | public *int* getSize() { | | 26 | return lines.size(); | | 27 | } | | 28 |  | | 29 | public *void* print() { | | 30 | *System*.out.println("\t\t"+name); | | 31 | lines.forEach(*System*.out::println); | | 32 | } | | 33 |  | | 34 | } | |  |

Перепишем класс Report для устранения использования StringBuilder. Для этого заменим список строк списком элементов типа Object, а класс реализующий список выберем LinkedList, так как у него наименьшее время добавления элемента. При вызове метода addLine будем передавать все его параметры в список через метод add, не забыв добавить после них символ перевода строки «\n». Метод print также перепишем, просто заменив println, на print, так как теперь переводы строк мы выписываем сами. Взглянув теперь на граф вызовов (рисунок 2) увидим, что метод addLine более не вызывает ни одного метода класса StringBuilder, а время работы самого метода сократилось с 2 миллисекунд до 83 микросекунд.

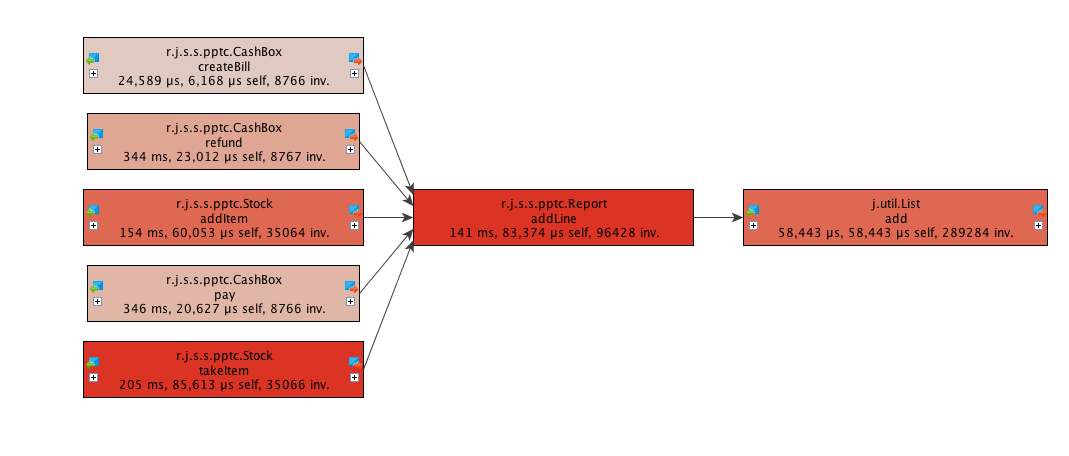


Рисунок 2 – Итоговый граф вызовов

|  |  |
| --- | --- |
| Изменённый класс Report | |
| |  |  | | --- | --- | | 1 | package ru.justnero.study.sevsu.pptc; | | 2 |  | | 3 | import *java.util.LinkedList*; | | 4 | import *java.util.List*; | | 5 |  | | 6 | public class Report { | | 7 |  | | 8 | public final *String* name; | | 9 | public final *List<Object>* elements; | | 10 |  | | 11 | public Report(*String* *name*) { | | 12 | this.name = name; | | 13 | this.elements = new *LinkedList<>*(); | | 14 | } | | 15 |  | | 16 | public *Report* addLine(final *Object*... *parts*) { | | 17 | for (*Object* part : parts) { | | 18 | elements.add(part); | | 19 | } | | 20 | elements.add("\n"); | | 21 | return this; | | 22 | } | | 23 |  | | 24 | public *int* getSize() { | | 25 | return elements.size(); | | 26 | } | | 27 |  | | 28 | public *void* print() { | | 29 | *System*.out.println("\t\t" + name); | | 30 | elements.forEach(*System*.out::print); | | 31 | } | | 32 |  | | 33 | } | |  |

ВЫВОДЫ

В ходе данной лабораторной работы были изучены подходы профилированию программного обеспечения. Выявлено узкое место выполнения программы в использовании конкатенации строк при формировании отчётов. Устранение узкого места позволило сократить время добавления строки отчёта в более чем 20 раз.