Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Гамаонов Алан Батразович

Энергопрофилирование многопоточных Android приложений

Курсовая работа

Научный руководитель :
ст. преп. С. Ю. Сартасов

Санкт-Петербург, 2020 г.

Содержание

Введение

Смартфоны, планшеты и иные мобильные устройства стали повсеместным явлением в современном мире. Уже в 2016 году число пользователей смартфонов составляло 2.1 миллиарда людей, а к 2019 возрастет до 2.5 миллиардов[?]. Но мобильные устройства требуют энергоресурсов — чем сложнее устроены приложения, тем быстрее разряжается батарея, и время работы смартфона сокращается. Есть несколько подходов к решению этой проблемы, и среди них:

- Физическое увеличение батареи
- Увеличение энергоэффективности работы компонент девайсов (процессора, работы с памятью, периферийных устройств)
- Разработка приложений, потребляющих как можно меньше энергии

На разряд также влияют такие факторы, как параллельное выполнение программ или приложений, внешние переменные и износ батареи ввиду срока жизни устройства. В последнее время рынок электронных устройств развивался, стали распространены литиумные батареи с большей вместимостью, появилась новая архитектура CPU (big.LITTLE2), что положительно сказалось на времени жизни современных девайсов. Однако было выявлено, что улучшения батареи и технологий в области аппаратных средств имеют предел[?]. Поэтому разработчикам приходится уделять пристальное внимание улучшению работы самих приложений. Но как узнать, сработала оптимизация или нет, стал ли продукт более энергоэффективным? Для этого разработчики используют профилирование — сбор характеристик

работы приложения. Но измерить потребление ресурсов в течение какого-то времени и связать с протекающими в это время операциями — не самая тривиальная задача. Стоит также учитывать, что на сложность профилирования оказывает влияние и многопоточность — разделение задач, реализуемых приложением, на цельные блоки, которые могут одновременно исполняться и работать с информацией. Поэтому было решено реализовать утилиту, которая поможет отследить энергоэффективность программного кода, имеющего возможность многопоточного исполнения.

1 Цели и задачи

Целью данной работы является разработка средств визуализации энергопотребления для плагина среды разработки Android Studio. Для достижения этой цели были поставлены следующие задачи:

- Рассмотреть существующие инструменты визуализации данных в виде графиков
- Продумать формат представления данных
- Визуализировать энергопотребление
- Интегрироваться с другими модулями плагина

2 Обзор предметной области

2.1 Анализ существующих библиотек визуализации данных

Для визуализации данных были рассмотрены следующие библиотеки:

- XChart[1]
- JFreeChart[2]
- Charts4j[3]
- Java Chart Construction Kit[4]
- JChart2D[5]

Сравнение проходило по следующим критериям:

1. Наличие возможности отображения данных в виде необходимых типов графиков, а именно: Bar chart, Pie chart, Line chart
2. Достаточная степень кастомизации элементов графика
3. Возможность комбинирования различных типов графиков на одной координатной плоскости
4. Поддержка со стороны разработчиков

Библиотека	Кр. 1	Кр. 2	Кр. 3	Кр. 4
XChart	+	+	+	+
JFreeChart	+	+	+	+
Charts4j	+	-	-	-
JCCKit	+	-	-	-
JChart2D	+	+	+	-

Таблица 1: Сравнение библиотек

После анализа была выбрана библиотека XChart ввиду следующих факторов:

- XChart написана с использованием Java Swing, что упрощает интеграцию в плагин, так как ту же библиотеку использует Intellij Platform
- Открытый исходный код, активное сообщество и постоянная поддержка со стороны разработчиков
- Широкие возможности кастомизации и встроенная возможность сохранить график в форматах PNG, JPG, BMP и GIF

2.2 Выделение необходимой функциональности

На текущем этапе разработки было решено реализовать следующую функциональность:

- Отображение общей информации об энергопотреблении тестов в виде Bar Chart (столбчатой диаграммы)
- Отображение информации об энергопотреблении методов внутри теста в виде Pie Chart (круговой диаграммы) с указанием доли от общего потребления энергии в процентах

3 Инструменты

В данной секции описаны программные инструменты, которые использовались в процессе работы.

3.1 Kotlin

Kotlin — статически-типизированный язык программирования, разрабатываемый компанией JetBrains. Kotlin направлен на JVM, но также может быть скомпилирован в JavaScript или машинный код. Kotlin был выбран как основный язык фреймворка, т.к. Google объявили Android разработку
Kotlin-ориентированной, что подразумевает поддержку нововведений Kotlin'ом приоритетнее Java[6]. Кроме того, Kotlin DSL является заменой Groovy как скриптового языка программирования в системе сборки Gradle.

3.2 Swing

Swing - одна из наиболее известных библиотек для разработки графических пользовательских интерфейсов на языке Java, предоставляющая все необходимые компоненты (такие как кнопки, текстовые поля, панели и многие другие) для работы с оными. Визуализация в Intellij Platform основана именно на данной библиотеке.

3.3 XChart

XChart[1] - легковесная и удобная библиотека для представления данных в виде графиков и диаграмм. Как было сказано выше, она позволяет отображать графики в различных стилях, задавать темы оформления, сохранять их в различных форматах, а также динамически изменять.

4 Реализация

4.1 Данные

Для представления данных был создан класс EnergyConsumption, фактичесеки являющийся структурой с двумя полями: consumer и energy. Consumer содержит имя теста или метода, а energy потреблённую энергию в миллиджоулях.

Данные могут быть двух видов: общая информация обо всех тестах или подробная информация о методах одного определённого теста. Данные о методах также могу содержать информацию о процессе и потоке, в котором исполнялся метод, эта информация отображена в имени метода. Таким образом модуль визуализации получает либо список из EnergyConsumtion (в первом случае), либо список из EnergyConsumtion и название текущего теста (во втором случае).

4.2 Архитектура

Было решено реализовать собственный swing-компонент для отображения на нём графиков. Для этого был создан класс ChartPanel, являющийся наследником класса JPanel из библиотеки Swing. ChartPanel содержит единственный публичный метод showChart(2), принимающий данные в одном из видов, и строящий по этим данным соответствующий график.

Также были созданы две формы:

- TestsProfilingResultContentView для отображения общей информации о тестах
- TestProfilingResultDetailsContentView для отображения детальной информации о методах конкретного теста

На каждую форму был помещён компонент, связанный с экземпляром ChartPanel, который и отображает графики. Кроме того, TestsProfilingResultContentView содержит таблицу, отображающую все данные. TestProfilingResultDetailsContentView также содержит подобную таблицу, но позволяет сортировать методы по процессам и потокам. Чтобы отобразить информацию о конкретном методе, достаточно выбрать соответствующее поле таблицы, после чего данные автоматически будут отображены в виде графиков.

4.3 Диаграммы и графики

Как уже было сказано, чтобы отобразить данные на ChartPanel в виде графика, необходимо передать в showChart(2) список EnergyConsumtion и имя потребителя. Так как при отображении общей информации о тестах второй аргумент остаётся пустым (используется стандартное значение), по его наличию можно судить о том, в каком виде необходимо визуализировать данные. В частности, при отсутствии второго аргумента, строится столбчатая диаграмма, а при его наличии - круговая.

Являясь наследником класса JPanel, ChartPanel имеет возможность отображать различные swing-компоненты. Вследствие этого, showChart(2) вызывает метод setupNewChart(3), принимающий три аргумента (список EnergyConsumption, имя потребителя и тип графика) и возвращающий JPanel. После получения панели с готовым графиком, она выводится на основную панель с заранее заданными параметрами, необходимыми для корректного отображения окна "Navitas Profiler".

Предлагаю разобрать детали реализации отображения данных в виде столбчатой и круговой диаграммы.

4.3.1 Детальная информация об энергопотреблении

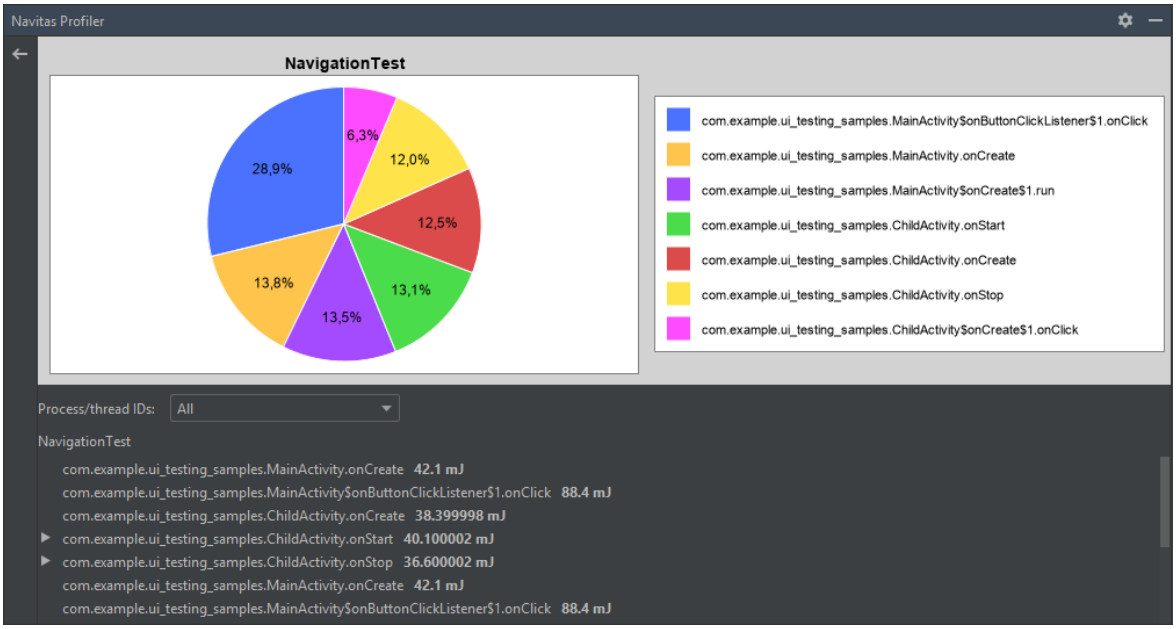

Рис. 1: Пример подробного отчёта о конкретном тесте

При отображении детальной информации существует проблема множественного вызова одного и того же метода, вследствие чего каждый такой вызов будет отображаться в виде отдельной величины. Для решения этой проблемы было решено слить множественные вызовы одного метода в один. В таком случае мы, фактически, не теряем информацию о том, в каком процессе или потоке был вызван метод, так как с помощью таблицы можно отсортировать данные по нужному фильтру.

После предварительной обработки данные сортируются по убыванию и на их основе строится круговая диаграмма, заголовком которой выступает название отображаемого теста/метода.

4.3.2 Общая информация об энергопотреблении

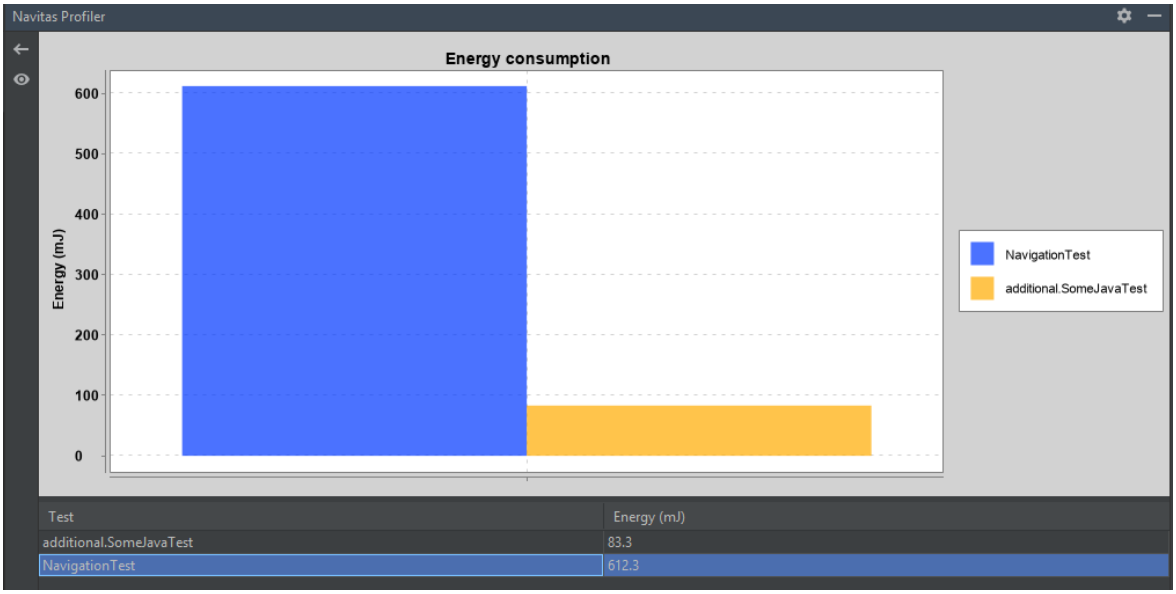

Рис. 2: Пример отчёта об энергопотреблении всех тестов

При отображении общей информации о тесте на координатной плоскости по вертикальной оси отображается объём потреблённой энергии в миллиджоулях, а по горизонтальной расположены потребители (тесты) в порядке убывания.

Обозначения на горизонтальной оси (названия тестов) скрываются, так как их длина слишком велика, что приводит к их некорректному отображению.

5 Проверка работоспособности

Проверка корректности работы была произведена на специально созданном приложении. В частности, данные об энергопотреблении были получены с помощью замеров на реальном устройстве (энергопрофилирование - задача других членов команды).

Заключение

В результате работы были выполнены следующие задачи:

- Произведен анализ существующих библиотек для отображения данных в виде графиков
- Продуман формат данных, позволяющий обобщить полученную информацию
- Реализована визуализация данных в виде столбчатой и круговой диаграмм
- Модуль интегрирован с основным плагином

У данной работы имеются дальнейшие перспективы развития: добавление новых видов отчётов (к примеру, трасс энергопотребления), а также возможности сравнить результаты текущего запуска с предыдущими.

Список литературы

[1] Knowm. XChart. https://github.com/knowm/XChart. Дата обращения 1.06.2020.

[2] JFree. JFreeChart. https://github.com/jfree/jfreechart. Дата обращения 1.06.2020.

[3] Julien Chastang. Charts4j. https://github.com/julienchastang/charts4j. Дата обращения 1.06.2020.

[4] Fjelmer. Java Chart Construction Kit. http://jcckit.sourceforge.net/index.html. Дата обращения 1.06.2020.

[5] Achimwestermann. JChart2D. https://sourceforge.net/projects/jchart2d/. Дата обращения 1.06.2020.

[6] Google I/O 2019. Empowering developers to build the best experiences on Android + Play. https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html. Дата обращения 20.12.2019.