

Яндекс



# Инструменты отладки

Виталий Заянковский, разработчик

# Содержание

- 0 | Введение
- 1 | Отладка кода
- 2 | Покрытие логами
- 3 | Профайлинг

Часть 0

# Введение

# Этапы разработки мобильного приложения

1. Настройка окружения
2. Программирование
3. Сборка и запуск
4. Отладка, профайлинг и тестирование
5. Публикация

# Задачи отладки

1. Повышение стабильности приложения
2. Оптимизация работы приложения

# Оптимизация приложения

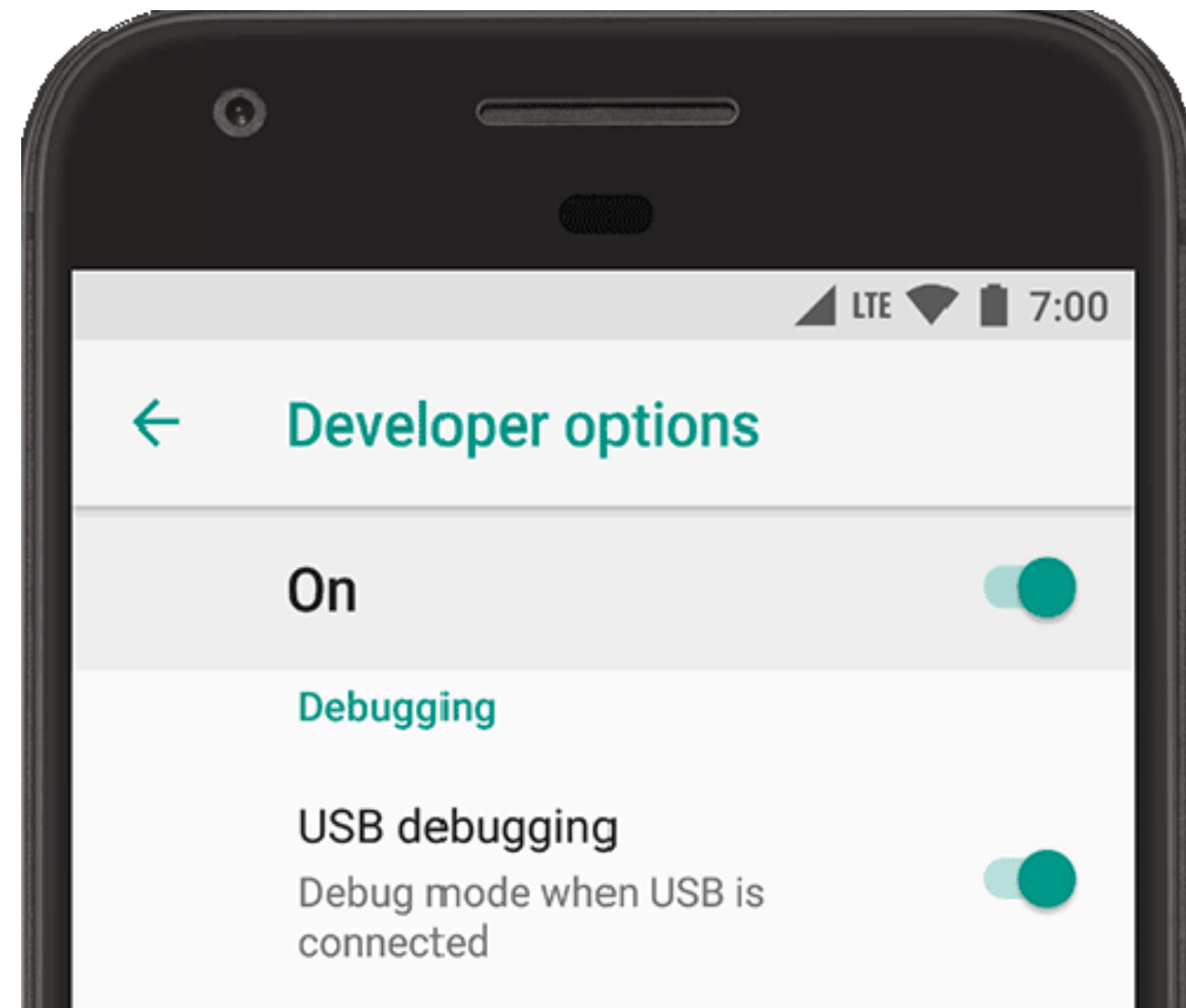
1. Память
2. Производительность
3. Трафик
4. Батарейка

Часть 1

# Отладка кода



# Подготовка к отладке

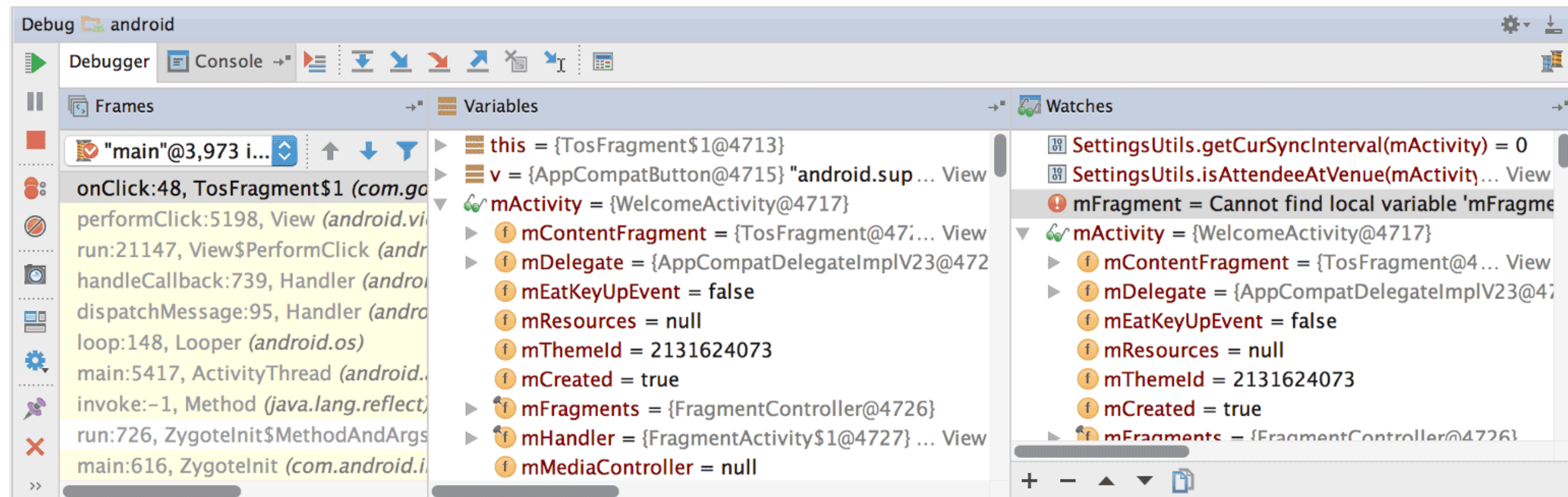


1. Разрешить отладку на устройстве в настройках разработчика (на эмуляторах разрешена по умолчанию).
2. Разрешить отладку для используемого типа сборки (для debug разрешена по умолчанию).

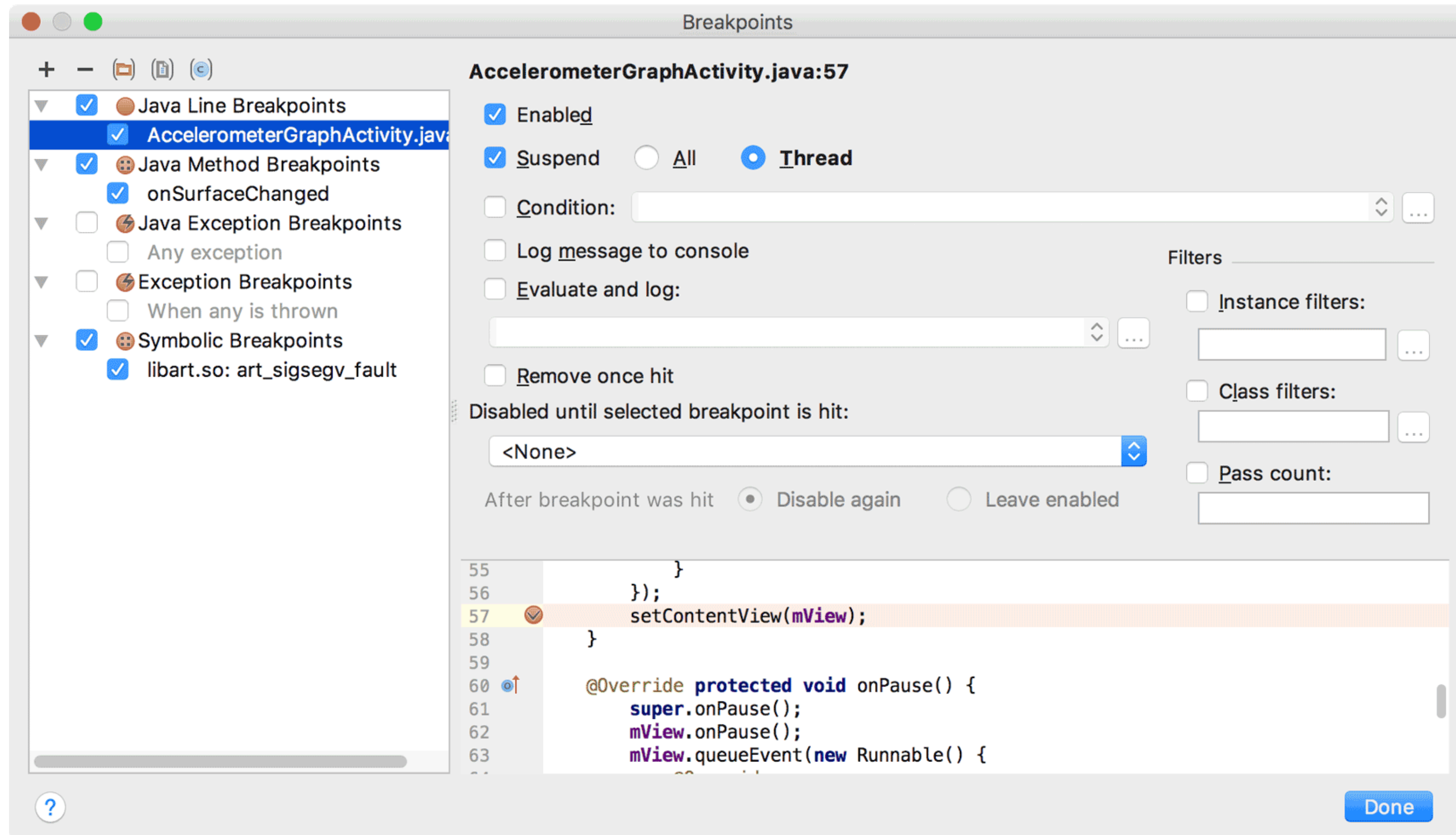
```
android {  
    buildTypes {  
        customDebugType {  
            debuggable true  
            ...  
        }  
    }  
}
```

# Процесс отладки

1. Расставляем брейкпоинты.
2. В тулбаре нажимаем **Debug**, чтобы запустить и отладить приложение.
3. Если приложение уже было запущено, можно использовать **Attach debugger to Android process**.



# Настройка брейкпоинтов



Часть 2

# Покрытие логами

# Написание логов

- > `Log.wtf(String, String)` (assert, What a Terrible Failure)
- > `Log.e(String, String)` (error)
- > `Log.w(String, String)` (warning)
- > `Log.i(String, String)` (information)
- > `Log.d(String, String)` (debug)
- > `Log.v(String, String)` (verbose)

```
private const val TAG = "MyActivity"
```

```
...
```

```
Log.i(TAG, "MyClass.getView() — get item number $position")
```



# Timber

- › <https://github.com/JakeWharton/timber>
- › В качестве тега автоматически используется имя класса.
- › Позволяет задавать разные стратегии логирования. Например, можно настроить, чтобы в Logcat логи писались только в дебажных сборках, а в релизных сборках логи отправлялись в АппМетрику.

# Просмотр логов

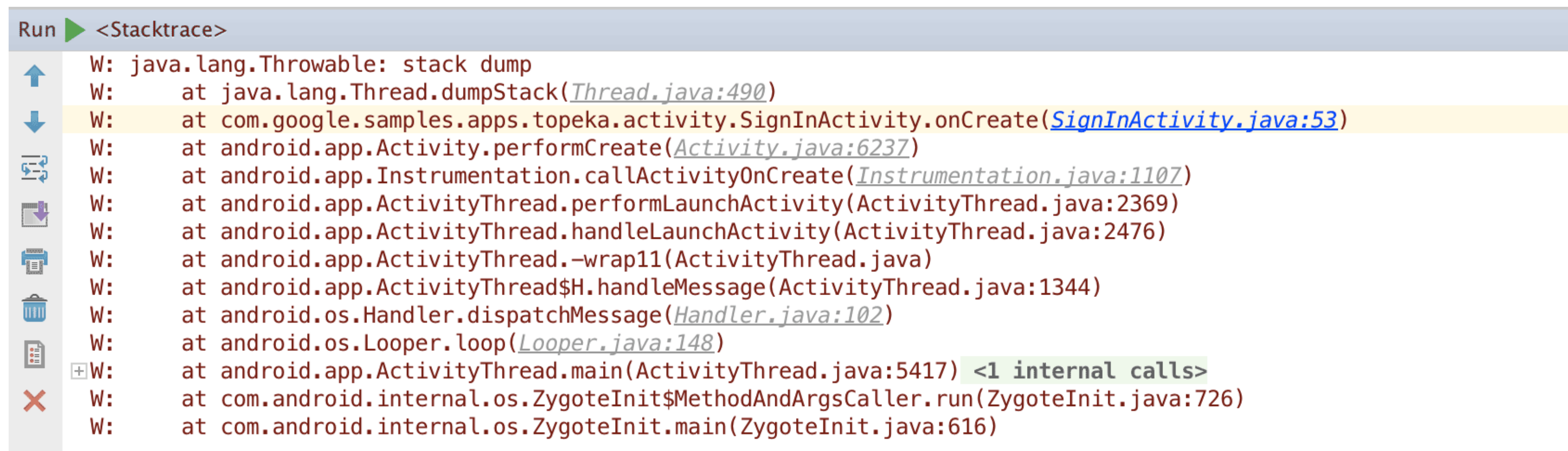
The screenshot shows the Logcat window in an IDE. The title bar is "Logcat". Below the title bar, there are several filters: a device filter set to "Xiaomi Mi A2 Android 9, API 28", an application filter set to "com.yandex.academy.mobdev.pro", a log level filter set to "Verbose", a search bar, a checked "Regex" checkbox, and a "Show only selected application" dropdown. The main area displays a list of log messages. The messages are timestamped and include package names, process IDs, and log levels. The messages are as follows:

```
2020-04-22 12:08:31.054 5636-5636/? I/obdev.profilin: Late-enabling -Xcheck:jni
2020-04-22 12:08:31.447 5636-5636/com.yandex.academy.mobdev.profiling I/Perf: Connecting to perf service.
2020-04-22 12:08:31.518 5636-5636/com.yandex.academy.mobdev.profiling W/obdev.profilin: Accessing hidden method Landroid/graphics/drawable/Drawable;->getOpticalInset()
2020-04-22 12:08:31.518 5636-5636/com.yandex.academy.mobdev.profiling W/obdev.profilin: Accessing hidden field Landroid/graphics/Insets;->left:I (light greylist, Landroid/graphics/Insets;
2020-04-22 12:08:31.518 5636-5636/com.yandex.academy.mobdev.profiling W/obdev.profilin: Accessing hidden field Landroid/graphics/Insets;->right:I (light greylist, Landroid/graphics/Insets;
2020-04-22 12:08:31.518 5636-5636/com.yandex.academy.mobdev.profiling W/obdev.profilin: Accessing hidden field Landroid/graphics/Insets;->top:I (light greylist, Landroid/graphics/Insets;
2020-04-22 12:08:31.518 5636-5636/com.yandex.academy.mobdev.profiling W/obdev.profilin: Accessing hidden field Landroid/graphics/Insets;->bottom:I (light greylist, Landroid/graphics/Insets;
2020-04-22 12:08:31.554 5636-5636/com.yandex.academy.mobdev.profiling W/obdev.profilin: Accessing hidden method Landroid/view/View;->getAccessibilityDelegate()Landroid/view/View$AccessibilityDelegate;
2020-04-22 12:08:31.563 5636-5636/com.yandex.academy.mobdev.profiling W/obdev.profilin: Accessing hidden method Landroid/view/View;->computeFitSystemWindows(Landroid/view/View$FitSystemWindows;
2020-04-22 12:08:31.564 5636-5636/com.yandex.academy.mobdev.profiling W/obdev.profilin: Accessing hidden method Landroid/view/ViewGroup;->makeOptionalFitsSystemWindows()
2020-04-22 12:08:31.723 5636-5636/com.yandex.academy.mobdev.profiling W/obdev.profilin: Accessing hidden method Landroid/widget/TextView;->getTextDirectionHeuristic()
2020-04-22 12:08:31.767 5636-5636/com.yandex.academy.mobdev.profiling D/OpenGLRenderer: Skia GL Pipeline
2020-04-22 12:08:31.838 5636-5684/com.yandex.academy.mobdev.profiling I/Adreno: QUALCOMM build : cf57c9c, I1cb5c4d1cc
    Build Date : 09/23/18
    OpenGL ES Shader Compiler Version: EV031.25.03.01
    Local Branch :
    Remote Branch :
    Remote Branch :
    Reconstruct Branch :
2020-04-22 12:08:31.838 5636-5684/com.yandex.academy.mobdev.profiling I/Adreno: Build Config : S L 6.0.7 AArch64
```

At the bottom of the window, there is a tab bar with icons for "Run", "TODO", "Profiler", "Logcat", "Build", and "Terminal". The "Logcat" tab is currently selected. In the bottom right corner, there is an "Event Log" button.

# Анализ стектрейса

- › Стектрейс будет выведен в лог при падении приложения.
- › Его также можно вывести с помощью вызова `Thread.dumpStack()`
- › Для анализа стороннего стектрейса можно использовать **Analyze Stack Trace** в меню **Analyze**.

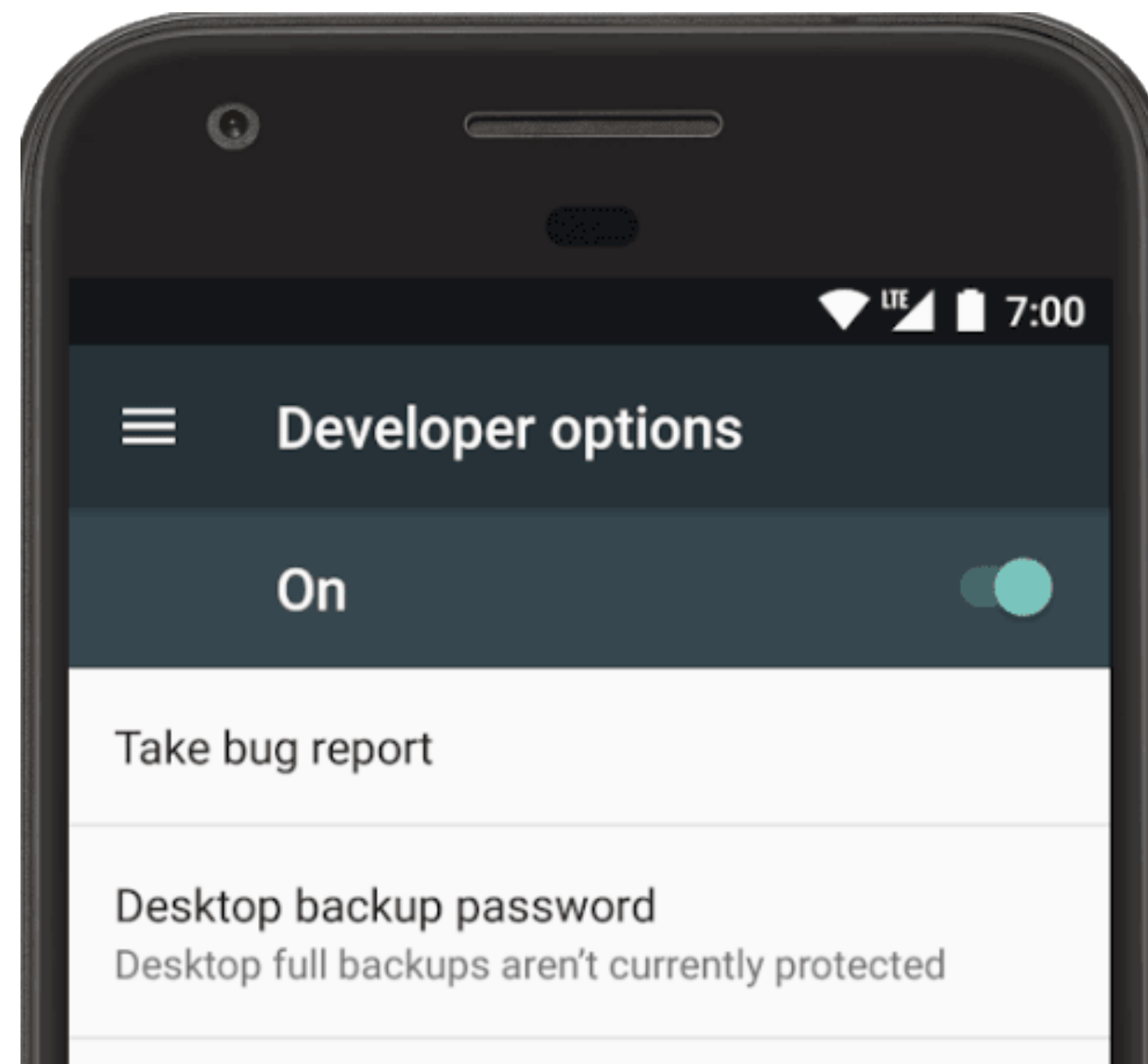


The screenshot shows the 'Run' tab in Android Studio, displaying a stack trace. The title bar of the log window is 'Run <Stacktrace>'. The log contains several lines of output, with the following stack trace entries highlighted in yellow:

```
W: java.lang.Throwable: stack dump
W:   at java.lang.Thread.dumpStack(Thread.java:490)
W:   at com.google.samples.apps.topeka.activity.SignInActivity.onCreate(SignInActivity.java:53)
W:   at android.app.Activity.performCreate(Activity.java:6237)
W:   at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1107)
W:   at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2369)
W:   at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2476)
W:   at android.app.ActivityThread.-wrap11(ActivityThread.java)
W:   at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1344)
W:   at android.os.Handler.dispatchMessage(Handler.java:102)
W:   at android.os.Looper.loop(Looper.java:148)
W:   at android.app.ActivityThread.main(ActivityThread.java:5417) <1 internal calls>
W:   at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:726)
W:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
```



# Генерация багрепорта



- › Содержит разную диагностическую информацию: логи устройства, стектрейсы и пр.
- › Для генерации можно воспользоваться настройками разработчика.
- › Также доступно получение репорта с помощью **adb**:

```
$ adb bugreport E:\Reports\MyBugReports
```

Часть 3

# Профайлинг

# Layout Inspector

- › Позволяет во время исполнения приложения получить информацию о текущем состоянии UI
- › Можно понять, насколько в целом приложение соответствует макету дизайнера, а также посмотреть подробную информацию про каждый элемент
- › Особенно полезно, когда UI строится динамически из кода
- › Находится в меню **Tools**

# Android Profiler

- › Позволяет в реальном времени отслеживать то, как приложение использует процессор, память, сеть и батарею. За каждый из параметров отвечает отдельный профайлер.
- › Есть возможность записать несколько сессий, чтобы их можно было сравнивать между собой. Сессии хранятся до тех пор, пока открыта Android Studio.
- › На устройствах с Android 7.1 и старше некоторая функциональность требует включения **Advanced profiling** на вкладке **Profiling** в меню **Run > Edit Configurations**.

# Профайлинг и отладка собранной APK

1. Для выбора APK используем меню **File > Profile or Debug APK**. Будет создан отдельный проект с её копией.
2. Чтобы можно было профайлить, приложение должно поддерживать отладку.
3. Чтобы можно было отлаживать код, необходимо ещё и прикрепить исходники.

# Профайлинг энергопотребления с помощью Batterystats и Battery Historian

1. Очищаем старые накопленные данные:

```
adb shell dumpsys batterystats --reset
```

2. Отсоединяем телефон и играемся с приложением.

3. Подсоединяем обратно и собираем накопленные данные:

```
adb shell dumpsys batterystats > [path/]batterystats.txt
```

```
adb bugreport > [path/]bugreport.zip (bugreport.txt на Android 6 и старше)
```

4. Заходим на сайт <https://bathist.ef.lc> для визуализации данных.

# Профайлинг отрисовки

- › Для мониторинга скорости отрисовки кадров в настройках разработчика в разделе **Monitoring** можно включить опцию **Profile GPU Rendering**.
- › Чтобы не допустить случаев, когда большие области перерисовываются по несколько раз, стоит воспользоваться опцией **Debug GPU Overdraw** в разделе **Hardware accelerated rendering**.

# Другие полезные настройки разработчика

- › **Select mock location app:** позволяет подменить локацию на устройстве. Предварительно нужно установить приложение, которое будет подменять локацию, например Fake GPS location.
- › **Force RTL layout direction:** используйте, чтобы проверить, как будет выглядеть ваше приложение на языках с обратным направлением письма (справа налево).
- › **Simulate color space:** позволяет посмотреть на ваше приложение глазами людей с особенностями цветовосприятия.
- › **Don't keep activities:** активити уничтожаются сразу после ухода с них. Некоторые пользователи используют эту опцию, чтобы увеличить время автономной работы телефона.



# Просмотр APK на устройстве

1. ManifestViewer: <https://play.google.com/store/apps/details?id=jp.susatthi.ManifestViewer>.
2. Dexplorer: <https://play.google.com/store/apps/details?id=com.dexplorer>.

# StrictMode

- › Позволяет на этапе разработки обнаружить такие ошибки, как обращения к диску на главном потоке, утечки контекста, незакрытые ресурсы и другие.
- › Как и логирование, StrictMode следует включать только в отладочных сборках и обязательно выключать в релизных.

# Device File Explorer

- › Позволяет просматривать, копировать и удалять файлы на устройстве.
- › Полезно для изучения файлов, которые были созданы нашим приложением.
- › Без рута можно смотреть файлы только тех приложений, которые разрешили отладку. Также многие системные директории будут недоступны.

# Hyperion

- › <https://github.com/willowtreeapps/Hyperion-Android>
- › Встраивается в приложение, чтобы его можно было профайлить прямо на устройстве.
- › Много возможностей, среди которых: отображение стектрейса при падении, просмотр информации об элементах UI, мониторинг сетевого трафика и логов Timber, просмотр файлов, баз данных и настроек приложения.

# Полезные ссылки

- › <https://developer.android.com/studio/debug>
- › <https://developer.android.com/studio/profile>



# Спасибо

**Виталий Заянковский**

Разработчик



@zayankovsky