

Многопоточность и сетевое взаимодействие

Айдаров Асхар



Не забудьте
отметиться
на портале



Содержание

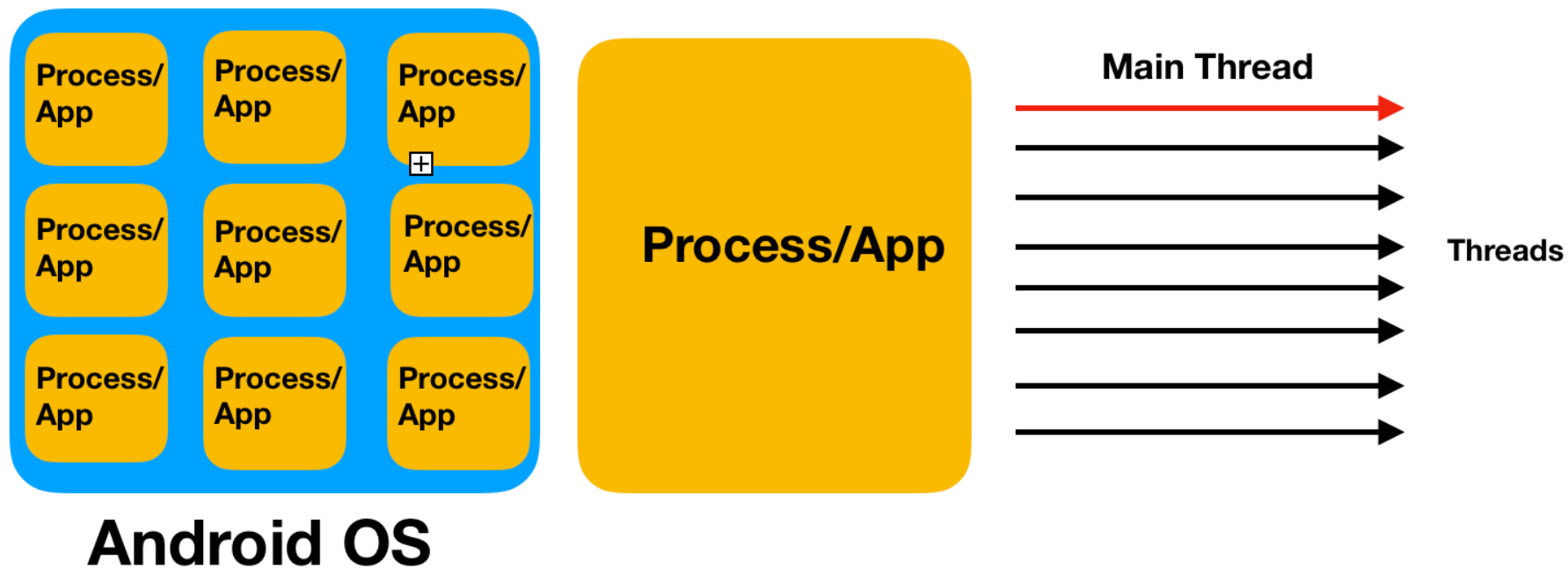
Многопоточность в Android: устройство и особенности

Взаимодействие с сетью: клиенты и сериализация данных

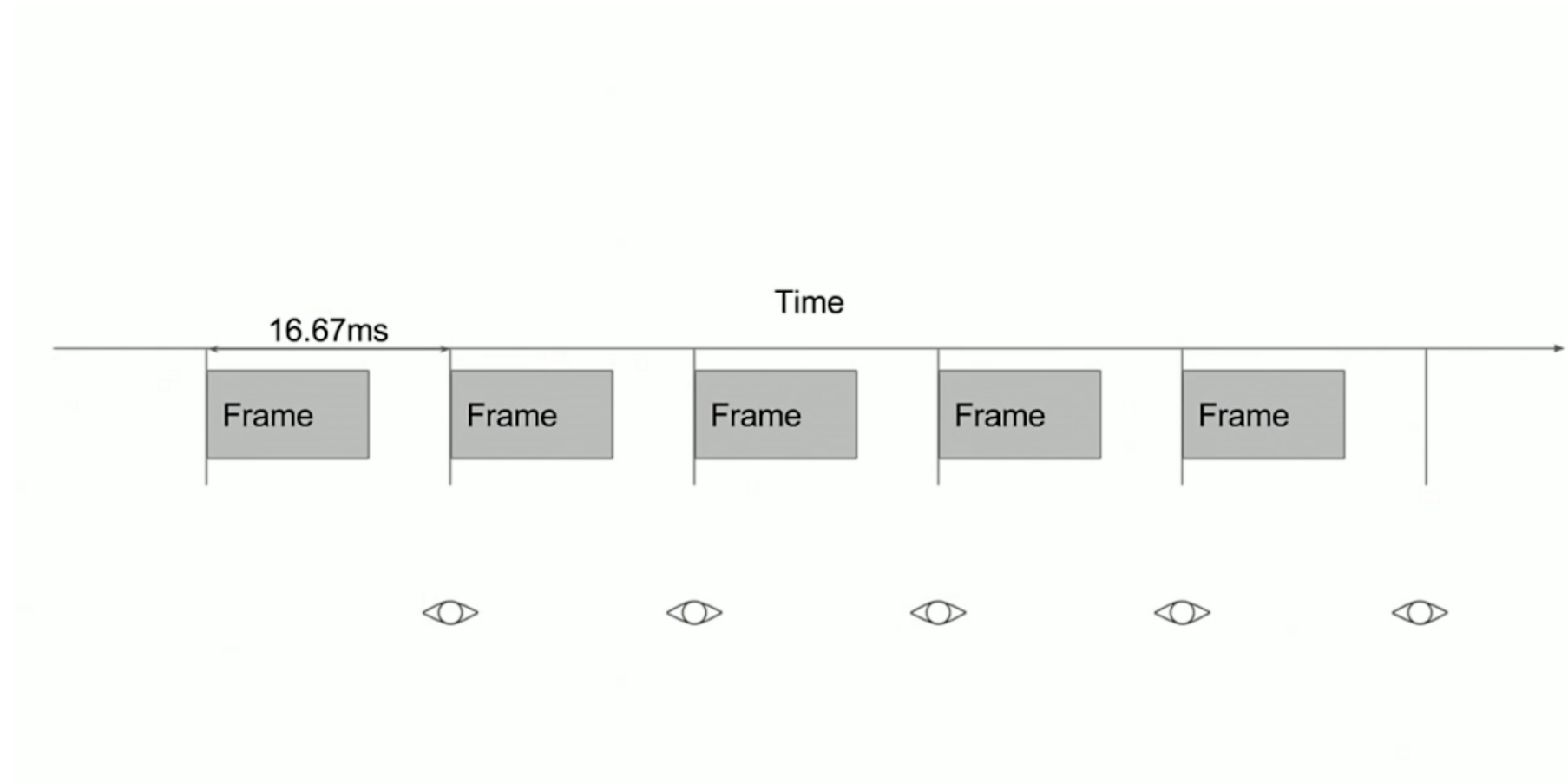
Многопоточность в Android



Процессы и потоки

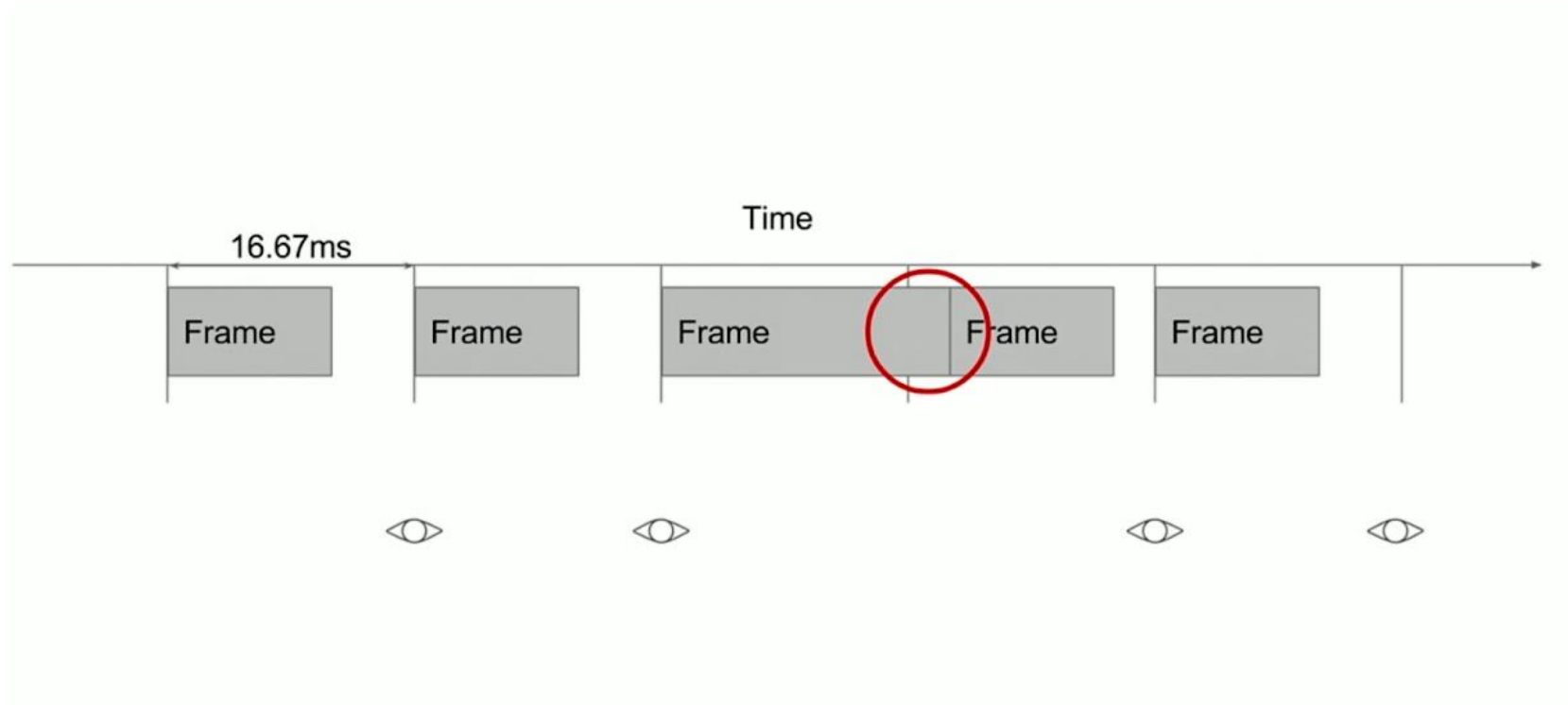


Отрисовка



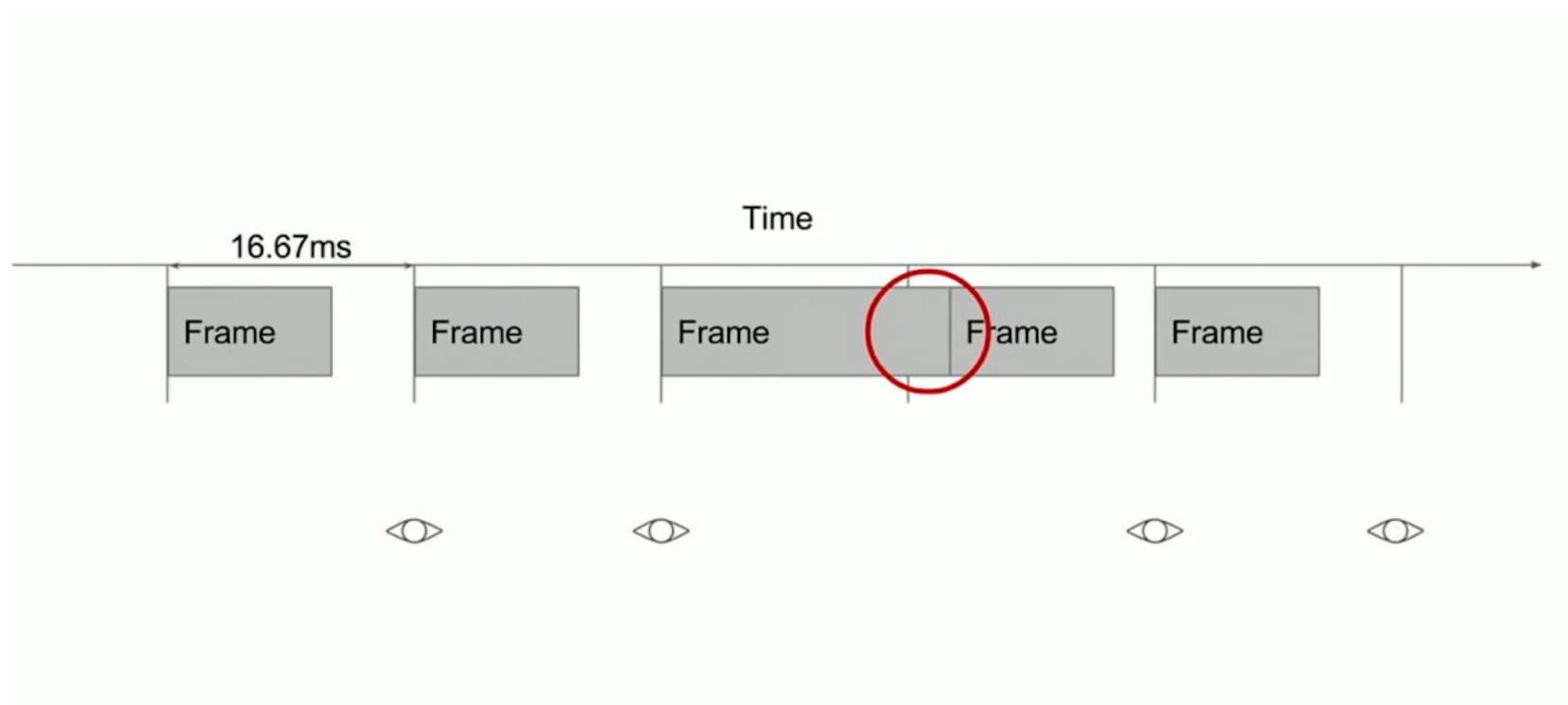
$60\text{fps} \rightarrow 1000\text{ms}/60\text{f} \rightarrow \sim 16.67\text{ms}$

Отрисовка. Фриз



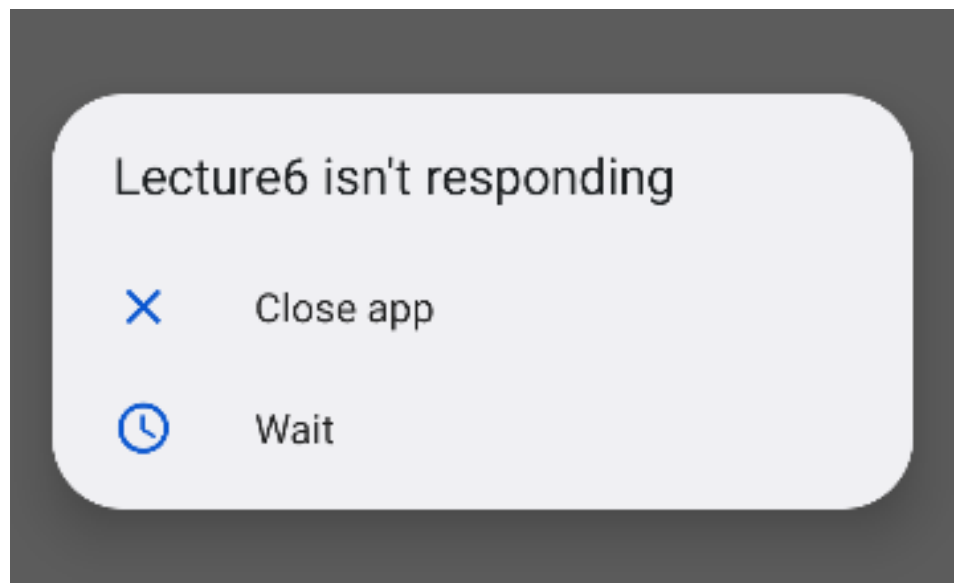
60fps -> $1000\text{ms}/60\text{f}$ -> $\sim 16.67\text{ms}$

Отрисовка. Фриз



60fps -> $1000\text{ms}/60\text{f}$ -> $\sim 16.67\text{ms}$

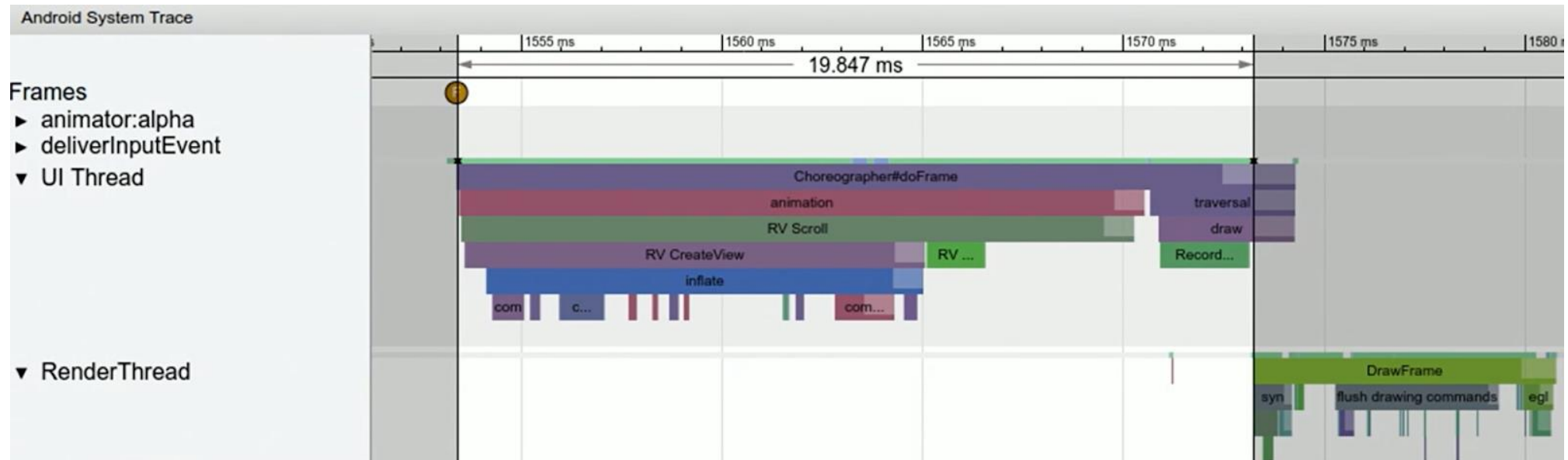
ANR. Приложение не отвечает ~5 секунд



Detect problems. Logs

```
I/Choreographer: Skipped 53 frames!  The application may be doing too much work on its main thread.  
• I/Choreographer: Skipped 138 frames!  The application may be doing too much work on its main thread.  
• I/Choreographer: Skipped 58 frames!  The application may be doing too much work on its main thread.  
• I/Choreographer: Skipped 310 frames!  The application may be doing too much work on its main thread.  
• I/Choreographer: Skipped 31 frames!  The application may be doing too much work on its main thread.
```

Detect problems. Profiler



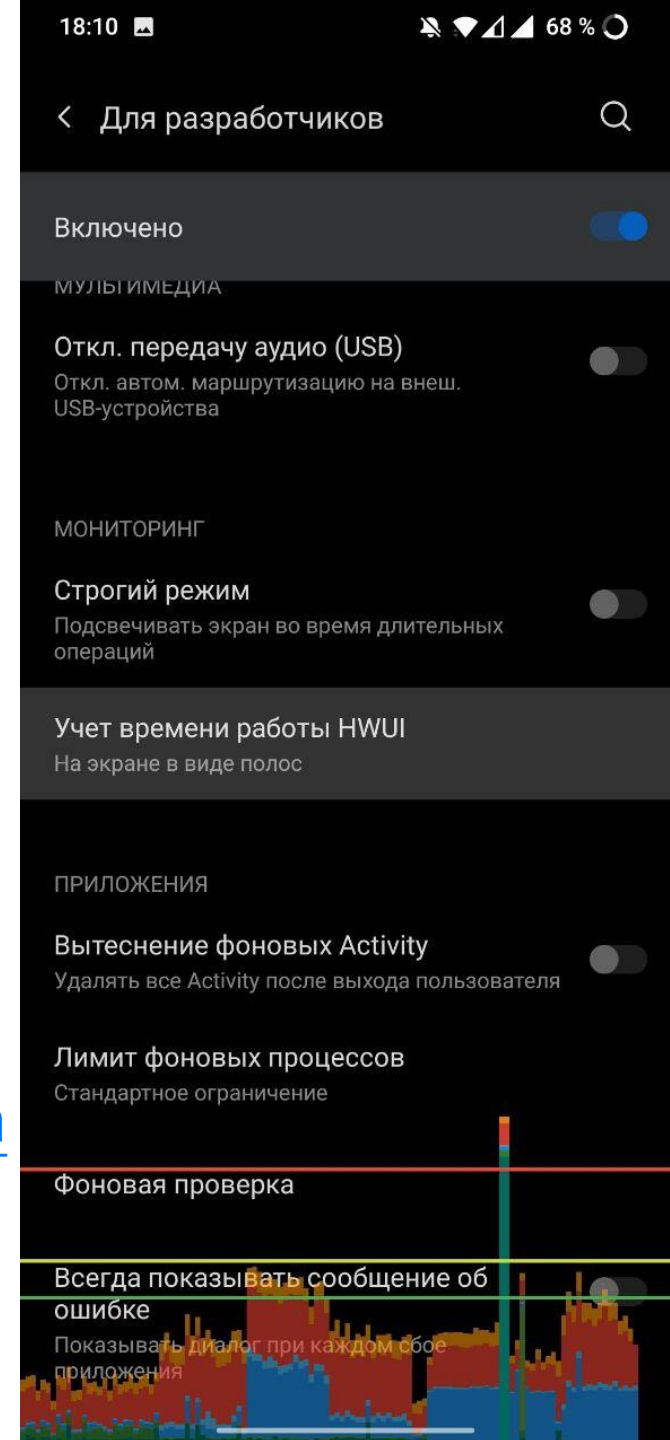
Detect problems. HWUI Rendering

Вертикальные линии: дают понять сколько времени заняла отрисовка кадра

Горизонтальные:

- Зеленая линия – 16мс
- Желтая – 21мс
- Красная – 31мс

<https://developer.android.com/topic/performance/rendering/inspect-gpu-rendering>



Strict Mode

```
if (BuildConfig.DEBUG) {  
    StrictMode.setThreadPolicy(  
        ThreadPolicy.Builder()  
            .detectDiskReads()  
            .detectDiskWrites()  
            .detectNetwork() // or .detectAll() for all detectable problems  
            .penaltyDeath()  
            .penaltyDialog()  
            .build()  
    )  
}
```

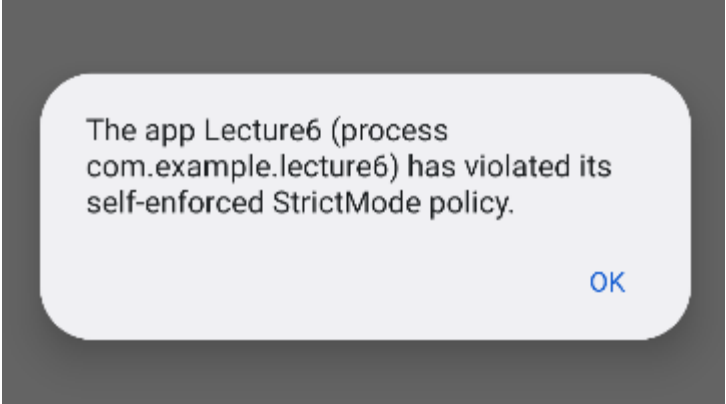
Strict Mode. Write on Main Thread

```
getSharedPreferences( name: "Name", MODE_PRIVATE)  
    .edit()  
    .commit()
```

Strict Mode. Write on Main Thread

```
getSharedPreferences( name: "Name", MODE_PRIVATE)  
    .edit()  
    .commit()
```

Strict Mode. Write on Main Thread

A screenshot of an Android toast message. The message is white with rounded corners and a thin grey border, set against a dark grey background. The text inside the toast reads: "The app Lecture6 (process com.example.lecture6) has violated its self-enforced StrictMode policy." In the bottom right corner of the toast, there is a blue "OK" button.

The app Lecture6 (process com.example.lecture6) has violated its self-enforced StrictMode policy.

OK

```
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:1005)
Caused by: java.lang.RuntimeException: StrictMode ThreadPolicy violation
    at android.os.StrictMode$AndroidBlockGuardPolicy.onThreadPolicyViolation(StrictMode.java:1876)
    at android.os.StrictMode$AndroidBlockGuardPolicy.handleViolationWithTimingAttempt(StrictMode.java:1733)
    at android.os.StrictMode$AndroidBlockGuardPolicy.startHandlingViolationException(StrictMode.java:1704)
    at android.os.StrictMode$AndroidBlockGuardPolicy.onReadFromDisk(StrictMode.java:1659)
    at libcore.io.BlockGuardOs.access(BlockGuardOs.java:74)
    at libcore.io.ForwardingOs.access(ForwardingOs.java:131)
    at android.app.ActivityThread$AndroidOs.access(ActivityThread.java:7719)
    at java.io.UnixFileSystem.checkAccess(UnixFileSystem.java:281)
    at java.io.File.exists(File.java:813)
    at android.app.ContextImpl.getDataDir(ContextImpl.java:2887)
    at android.app.ContextImpl.getPreferencesDir(ContextImpl.java:688)
    at android.app.ContextImpl.getSharedPreferencesPath(ContextImpl.java:915)
    at android.app.ContextImpl.getSharedPreferences(ContextImpl.java:537)
    at android.content.ContextWrapper.getSharedPreferences(ContextWrapper.java:196)
```


Network. Main Thread

AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```

Код:

```
findViewById<TextView>(R.id.sample_text_view).text = loadText()
```

loadData – загружает строку из сети

Нельзя выполнять запросы в сеть из главного потока

```
Process: com.example.lecture06, PID: 24973
java.lang.RuntimeException: Unable to start activity ComponentInfo{com.example.lecture06/com.example.lecture06.MainActivity}: android.os.NetworkOnMainThreadException
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3639)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3796)
    at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:103)
    at android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:135)
    at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2214)
    at android.os.Handler.dispatchMessage(Handler.java:106)
    at android.os.Looper.loopOnce(Looper.java:201)
    at android.os.Looper.loop(Looper.java:288)
    at android.app.ActivityThread.main(ActivityThread.java:7842) <1 internal line>
```

Исправляем

```
val sampleTextView = findViewById<TextView>(R.id.sample_text_view)

Thread {
    sampleTextView.text = loadText()
}.start()
}
```

Нельзя менять UI не из главного потока

```
android.os.RuntimeException: Only the original thread that created a view hierarchy can touch its views.  
Process: com.example.lecture6, PID: 22661  
at android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.  
    at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:9312)  
    at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:1772)  
    at android.view.View.requestLayout(View.java:25697)  
    at android.view.View.requestLayout(View.java:25697)  
    at android.view.View.requestLayout(View.java:25697)  
    at android.view.View.requestLayout(View.java:25697)  
    at android.view.View.requestLayout(View.java:25697)  
    at android.widget.TextView.checkForRelayout(TextView.java:9821)  
    at android.widget.TextView.setText(TextView.java:6398)  
    at android.widget.TextView.setText(TextView.java:6227)  
    at android.widget.TextView.setText(TextView.java:6179)  
    at com.example.lecture6.MainActivity.onCreate$lambda-0(MainActivity.kt:18)
```

ActivityThread.class

```
final Looper mLooper = Looper.myLooper(); //Бесконечные цикл
final H mH = new H(); // наследник Handler

public static void main(String[] args) {
    // some code

    Looper.prepareMainLooper();

    Looper.loop();

    throw new RuntimeException("Main thread loop unexpectedly exited");
}
```

Handler. Looper. Message

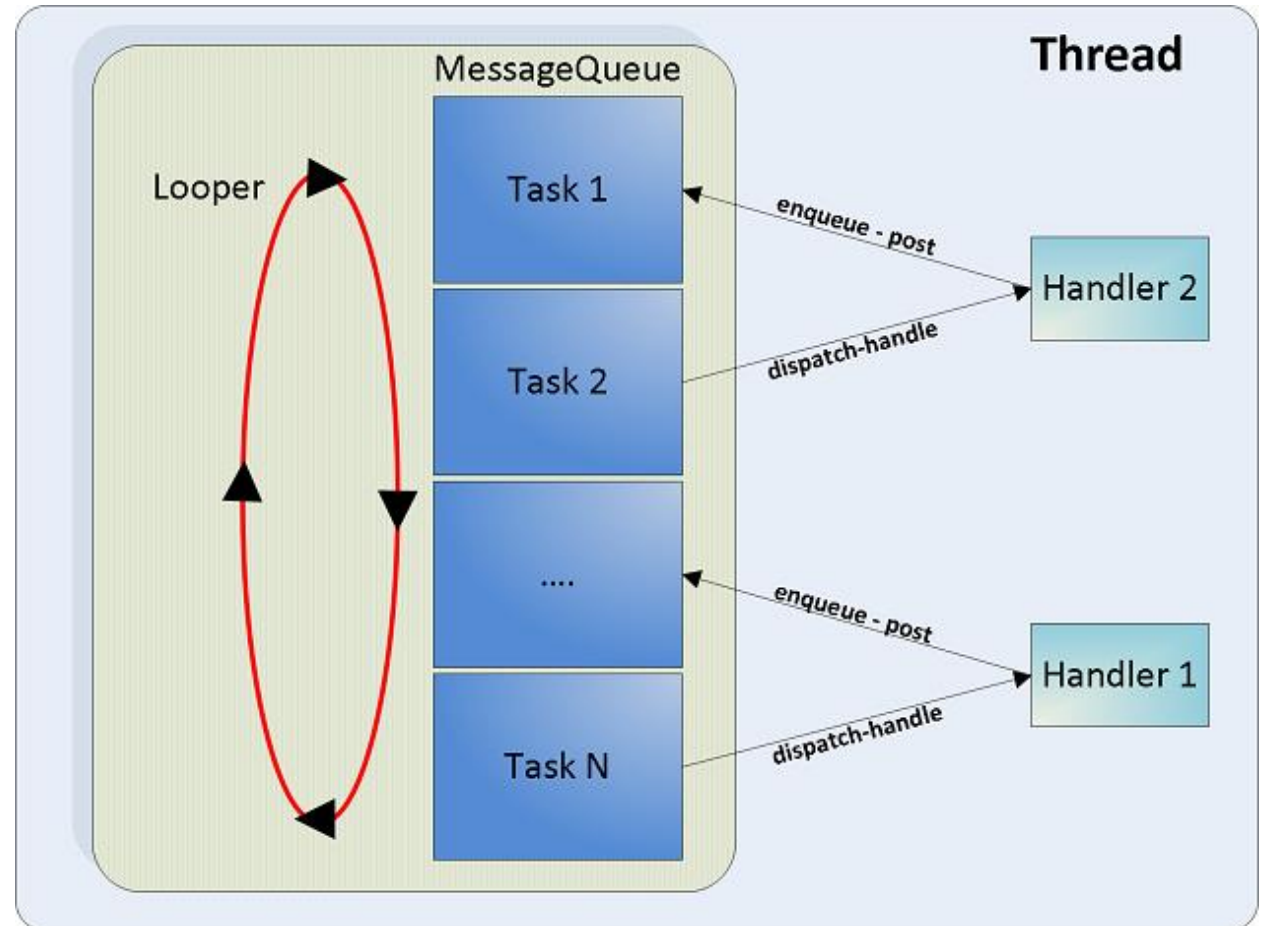
Looper - это бесконечный цикл обработки сообщений, которые находятся в MessageQueue.

Экземпляр Looper для каждого потока он свой.

MessageQueue - очередь сообщений.

Message - экземпляр сообщения.

Handler - обработчик сообщений.



Handler. Looper. Message

```
val sampleThread = SampleThread()
sampleThread.start()

val handler = sampleThread.handler
handler?.sendMessage(handler.obtainMessage(OPERATION_ONE))
```

```
class SampleThread : Thread() {

    var handler: Handler? = null

    override fun run() {
        Looper.prepare()

        handler = object : Handler(Looper.myLooper()!!) {
            override fun handleMessage(msg: Message) {
                when (msg.what) {
                    OPERATION_ONE -> {
                        looper.quitSafely()
                    }
                    OPERATION_TWO -> {
                        // ignore
                    }
                }
            }
        }

        Looper.loop()

        Log.d( tag: "SampleThread", msg: "quit from loop")
    }

    companion object {
        const val OPERATION_ONE = 1
        const val OPERATION_TWO = 2
    }
}
```

Handler. Looper. Message

```
val thread = HandlerThread( name: "SampleThread")
thread.start()

val handler = object : Handler(thread.looper) {
    override fun handleMessage(msg: Message) {
        when (msg.what) {
            OPERATION_ONE -> {
                looper.quitSafely()
            }
            OPERATION_TWO -> {
                // ignore
            }
        }
    }
}

handler.sendMessage(handler.obtainMessage(OPERATION_ONE))
```


Отправка кода для выполнения на UI потоке

```
val sampleTextView = findViewById<TextView>(R.id.sample_text_view)

Thread {
    val loadedText = loadText()
    this@MainActivity.runOnUiThread { sampleTextView.text = loadedText }
    sampleTextView.post { sampleTextView.text = loadedText }
    Handler(Looper.getMainLooper()).post { sampleTextView.text = loadedText }
}.start()
```

Не делайте так!

Есть альтернативы



RxJava



Coroutines

Dispatcher.Main internal

```
internal class AndroidDispatcherFactory : MainDispatcherFactory {  
  
    override fun createDispatcher(allFactories: List<MainDispatcherFactory>): MainCoroutineDispatcher {  
        val mainLooper = Looper.getMainLooper() ?: throw IllegalStateException("The main looper is not available")  
        return HandlerContext(mainLooper.asHandler(async = true))  
    }  
  
    override fun hintOnError(): String = "For tests Dispatchers.setMain from kotlinx-coroutines-test module can be used"  
  
    override val loadPriority: Int  
        get() = Int.MAX_VALUE / 2  
}
```

Итог

- В Android есть главный поток, который отвечает за отрисовку, обработку событий и изменения UI.
- UI можно менять только из главного потока.
- В главном потоке нельзя выполнять тяжелые операции: ходить в сеть, записывать и читать с постоянного хранилища и выполняться тяжелые вычисления. Контролировать это можно с помощью StrictMode.
- Для операции выше использовать coroutines с необходимыми Dispatcher. Для работы с сетью и хранилищем - Dispatcher.IO, для вычислений Dispatcher.Default
- Для выполнения coroutines на главном потоке, использовать Dispatcher.Main

Взаимодействие с сетью



OkHttp

- Interceptors
- Кэширование
- Работа cookie
- Таймауты
- SSL Pinning
- GZIP сжатие
- Пул соединений
- WebSocket
- и тд.

<https://github.com/square/okhttp/tree/master/samples/guide/src/main/java/okhttp3/recipes/kt>

OkHttp. Пример

- execute – выполнить запрос синхронного
- enqueue – выполнить запрос асинхронно

```
val client = OkHttpClient.Builder().build()

val httpUrl = "http://publicobject.com/helloworld.txt".toHttpUrl()
    .newBuilder()
    .addQueryParameter(name = "simple", value = "query")
    .build()

val request = Request.Builder()
    .url(url = "http://publicobject.com/helloworld.txt")
    .addHeader(name = "Accept", value = "application/json")
    .addHeader(name = "Content-Type", value = "application/json; charset=utf-8")
    .build()

client.newCall(request = request).execute().use { response ->
    response.body!!.charStream().forEachLine(::println)
}

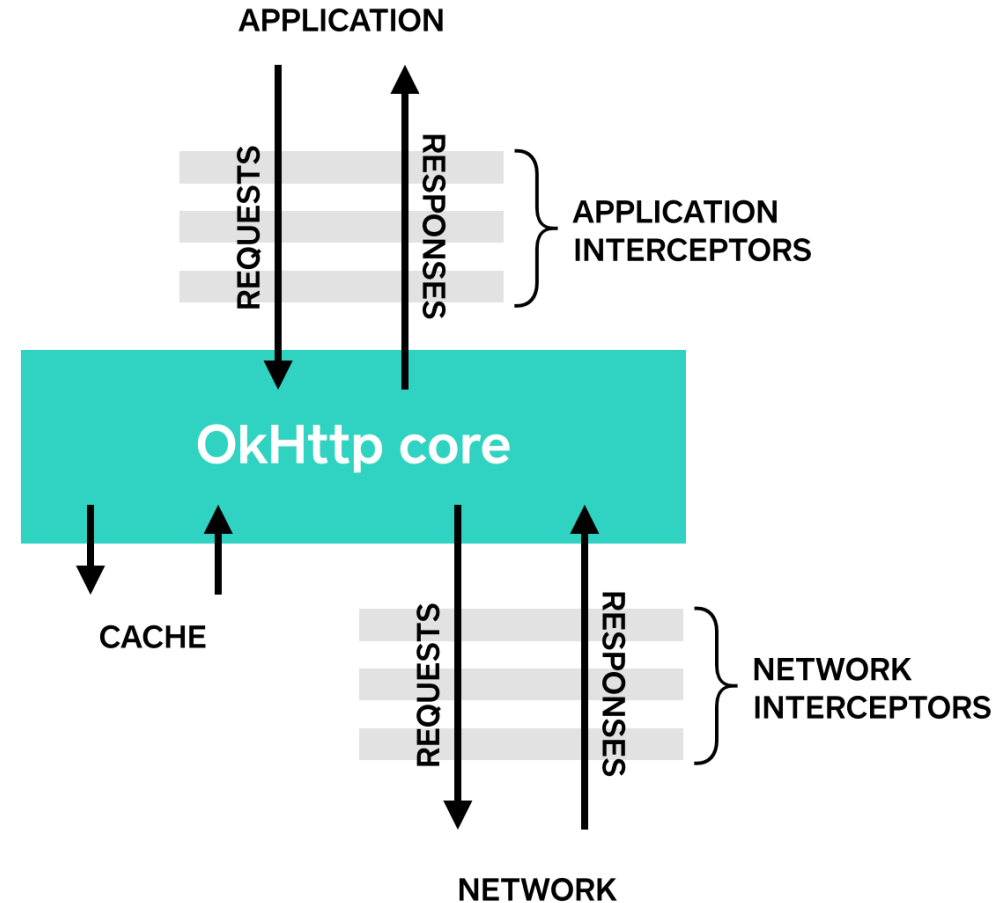
client.newCall(request = request).enqueue(responseCallback = object : Callback {
    override fun onFailure(call: Call, e: IOException) {...}

    override fun onResponse(call: Call, response: Response) {...}
})
```

```
implementation platform('com.squareup.okhttp3:okhttp-bom:4.9.3')
implementation 'com.squareup.okhttp3:okhttp'
```

OkHttp. Interceptors

- Application – interceptor между вашим кодом и библиотекой
- Network – interceptor между библиотекой и сеть. Может перехватывать трафик(redirect и тд)



OkHttp Interceptors Examples

```
class SimpleBearerInterceptor(private val accessToken: String) : Interceptor {  
    override fun intercept(chain: Interceptor.Chain): Response =  
        chain.proceed(  
            chain.request()  
                .newBuilder()  
                .addHeader(name: "Authorization", value: "Bearer $accessToken")  
                .build()  
        )  
}
```

```
class CacheControlInterceptor : Interceptor {  
    override fun intercept(chain: Interceptor.Chain): Response {  
        val response = chain.proceed(chain.request())  
        return response.newBuilder()  
            .header(name: "Cache-Control", value: "max-age=100")  
            .build();  
    }  
}
```

```
val loggingInterceptor = HttpLoggingInterceptor().apply {  
    level = HttpLoggingInterceptor.Level.BODY  
}
```

```
val client = OkHttpClient.Builder()  
    .addInterceptor(interceptor = loggingInterceptor)  
    .addInterceptor(interceptor = SimpleBearerInterceptor(accessToken = "code"))  
    .addNetworkInterceptor(interceptor = loggingInterceptor)  
    .addNetworkInterceptor(CacheControlInterceptor())  
    .build()
```

Еще проще? Retrofit

- REST клиент: обертка над Okhttp3
- Простое описание запросов
- Конвертеры для сериализации/десериализации
- Адаптеров для изменения типа ответа: адаптер для RxJava2 – ответы в виде стримов.
- Поддержка coroutines из коробки



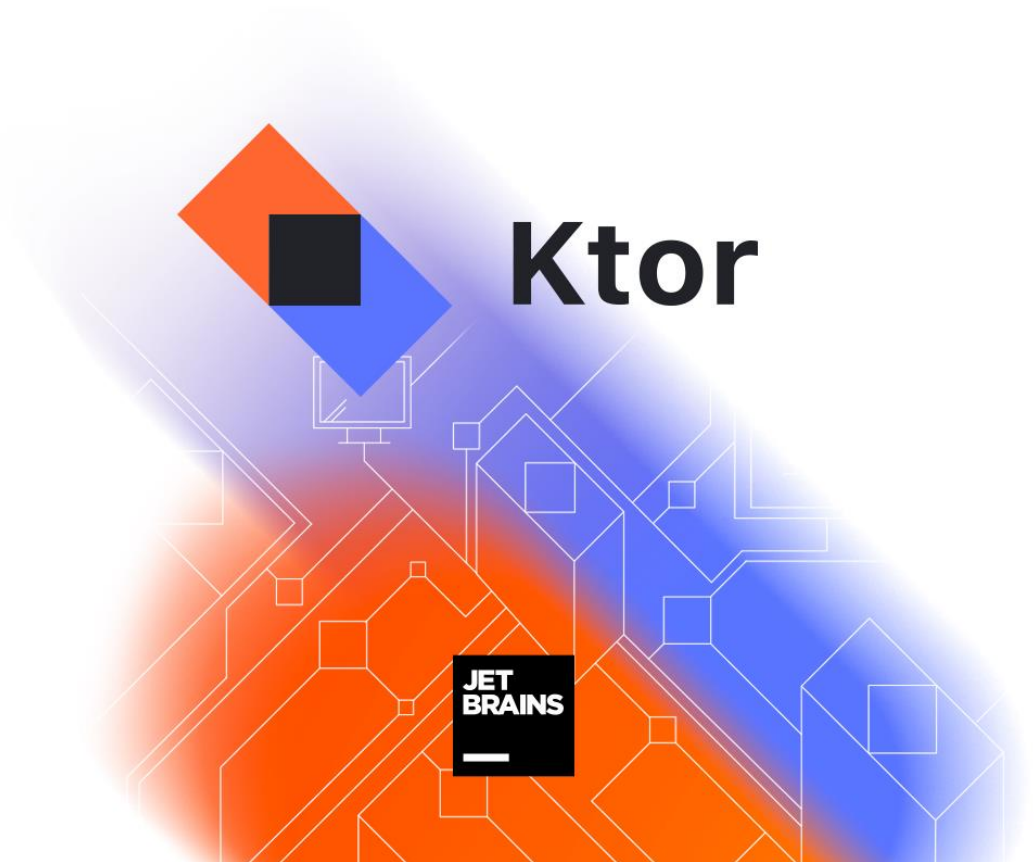
Retrofit

```
interface UserApi {  
  
    @Headers( ...value: "Cache-Control: max-age=640000")  
    @GET( value: "/users/{id}")  
    suspend fun getUser(@Path( value: "id") userId: Int): ResponseBody  
}
```

```
val client = OkHttpClient.Builder()  
    .addInterceptor(interceptor = loggingInterceptor)  
    .addNetworkInterceptor(interceptor = loggingInterceptor)  
    .build()  
  
val retrofit = Retrofit.Builder()  
    .client(client)  
    .baseUrl( baseUrl: "https://jsonplaceholder.typicode.com")  
    .build()  
  
val userApi = retrofit.create(UserApi::class.java)  
userApi.getUser(userId = 1).charStream().forEachLine(::println)
```

Есть хорошие альтернативы

- Мультиплатформенная библиотека
- Может быть оберткой над многими клиентами
- Активно расширяется функционал



JSON

- Kotlin Serialization (хорошо дружит с ktor. работает не только с JSON)
- Gson
- Moshi
- Jackson
- и тд

JSON

- Kotlin Serialization (хорошо дружит с ktor. работает не только с JSON)
- Gson
- Moshi
- Jackson
- и тд

Moshi + Retrofit. 1

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'  
implementation 'com.squareup.moshi:moshi-kotlin:1.13.0'
```

```
data class UserData(  
    @Json(name = "id") val id: Int,  
    @Json(name = "email") val email: String,  
    @Json(name = "name") val name: String,  
    @Json(name = "username") val username: String,  
    @Json(name = "phone") val phoneNumber: String,  
)
```

Moshi + Retrofit. 2

```
interface UserApi {  
  
    @Headers( ...value: "Cache-Control: max-age=640000")  
    @GET( value: "/users/{id}")  
    suspend fun getUser(@Path( value: "id") userId: Int): Response<UserData>  
}
```

```
val moshi = Moshi.Builder()  
    .addLast(KotlinJsonAdapterFactory())  
    .build()  
  
val retrofit = Retrofit.Builder()  
    .client(client)  
    .baseUrl( baseUrl: "https://jsonplaceholder.typicode.com")  
    .addConverterFactory(MoshiConverterFactory.create(moshi))  
    .build()  
  
val userApi = retrofit.create(UserApi::class.java)  
userApi.getUser(userId = 1).body()?.let(::println)
```


Moshi. CodeGen 1

```
plugins {  
    id 'com.google.devtools.ksp' version '1.6.10-1.0.4'  
    ...  
}
```

```
dependencies {  
    ksp 'com.squareup.moshi:moshi-kotlin-codegen:1.13.0'  
    ...  
}
```

Moshi. CodeGen 2

```
@JsonClass(generateAdapter = true)
data class UserData(
    @Json(name = "id") val id: Int,
    @Json(name = "email") val email: String,
    @Json(name = "name") val name: String,
    @Json(name = "username") val username: String,
    @Json(name = "phone") val phoneNumber: String,
)
```

```
public class UserDataJsonAdapter(
    moshi: Moshi
) : JsonAdapter<UserData>() {
    private val options: JsonReader.Options = JsonReader.Options.of("id", "email", "name", "username",
        "phone")

    private val intAdapter: JsonAdapter<Int> = moshi.adapter(Int::class.java, emptySet(), "id")
```

Moshi. CodeGen 3

```
val retrofit = Retrofit.Builder()
    .client(client)
    .baseUrl(baseUrl: "https://jsonplaceholder.typicode.com")
    .addConverterFactory(MoshiConverterFactory.create())
    .build()

val userApi = retrofit.create(UserApi::class.java)

val adapter = Moshi.Builder().build().adapter(UserData::class.java)
println(adapter.toJson(userApi.getUser(userId = 1)))
```

Теперь к
практике



Не забудьте
отметиться
на портале



Спасибо
за внимание!