

Фоновая работа, уведомления



Не забудьте
отметиться
на портале



Содержание

Service

Notifications

Firebase Cloud Messaging

WorkManager

System Restriction

Service



Основные компоненты

Activity – окно приложения, который видит и с которым взаимодействует пользователь

Service – выполнение долгих задач, предоставление функциональности другим приложениям

ContentProvider – предоставление доступа к данным приложения

Broadcast Receiver – приемник широковещательным сообщений

Service: варианты использования

Service: варианты использования

- Проигрывание медиа

Service: варианты использования

- Проигрывание медиа
- VPN

Service: варианты использования

- Проигрывание медиа
- VPN
- Межпроцессное взаимодействие

Service: варианты использования

- Проигрывание медиа
- VPN
- Межпроцессное взаимодействие
- Видео- и аудио-звонки

Service: варианты использования

- Проигрывание медиа
- VPN
- Межпроцессное взаимодействие
- Видео- и аудио-звонки
- Другие долгие задачи с уведомлением о работе для пользователей

Service – типы

- По способу использования
 - Started
 - Bounded
- По взаимодействию с пользователем
 - Background
 - Foreground

Service: Started Background

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application>
        <activity/>

        <service android:name=".SimpleService" />

    </application>

</manifest>
```

Service: Started Background

```
class SimpleService : Service() {  
    override fun onCreate() { super.onCreate() }  
    override fun onDestroy() { super.onDestroy() }  
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {  
        stopSelf() // stopSelf(startId)  
  
        val data = intent?.data  
  
        when (flags) {  
            START_FLAG_REDELIVERY -> {}  
            START_FLAG_RETRY -> {}  
        }  
        return START_STICKY // START_REDELIVER_INTENT | START_NOT_STICKY  
    }  
    override fun onBind(intent: Intent?): IBinder? = null  
}
```

Service: Started Background

```
class SimpleService : Service() {  
    override fun onCreate() { super.onCreate() }  
    override fun onDestroy() { super.onDestroy() }  
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {  
        stopSelf() // stopSelf(startId)  
  
        val data = intent?.data  
  
        when (flags) {  
            START_FLAG_REDELIVERY -> {}  
            START_FLAG_RETRY -> {}  
        }  
        return START_STICKY // START_REDELIVER_INTENT | START_NOT_STICKY  
    }  
    override fun onBind(intent: Intent?): IBinder? = null  
}
```

Service: Started Background

```
class SimpleActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?, persistentState: PersistableBundle?) {  
        startService(Intent(this, SimpleService::class.java))  
        stopService(Intent(this, SimpleService::class.java))  
        super.onCreate(savedInstanceState, persistentState)  
    }  
}
```

```
class SimpleService : Service() {  
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {  
        stopSelf() // stopSelf(startId)  
  
        val data = intent?.data  
  
        when (flags) {  
            START_FLAG_REDELIVERY -> {}  
            START_FLAG_RETRY -> {}  
        }  
        return START_STICKY // START_REDELIVER_INTENT | START_NOT_STICKY  
    }  
}
```

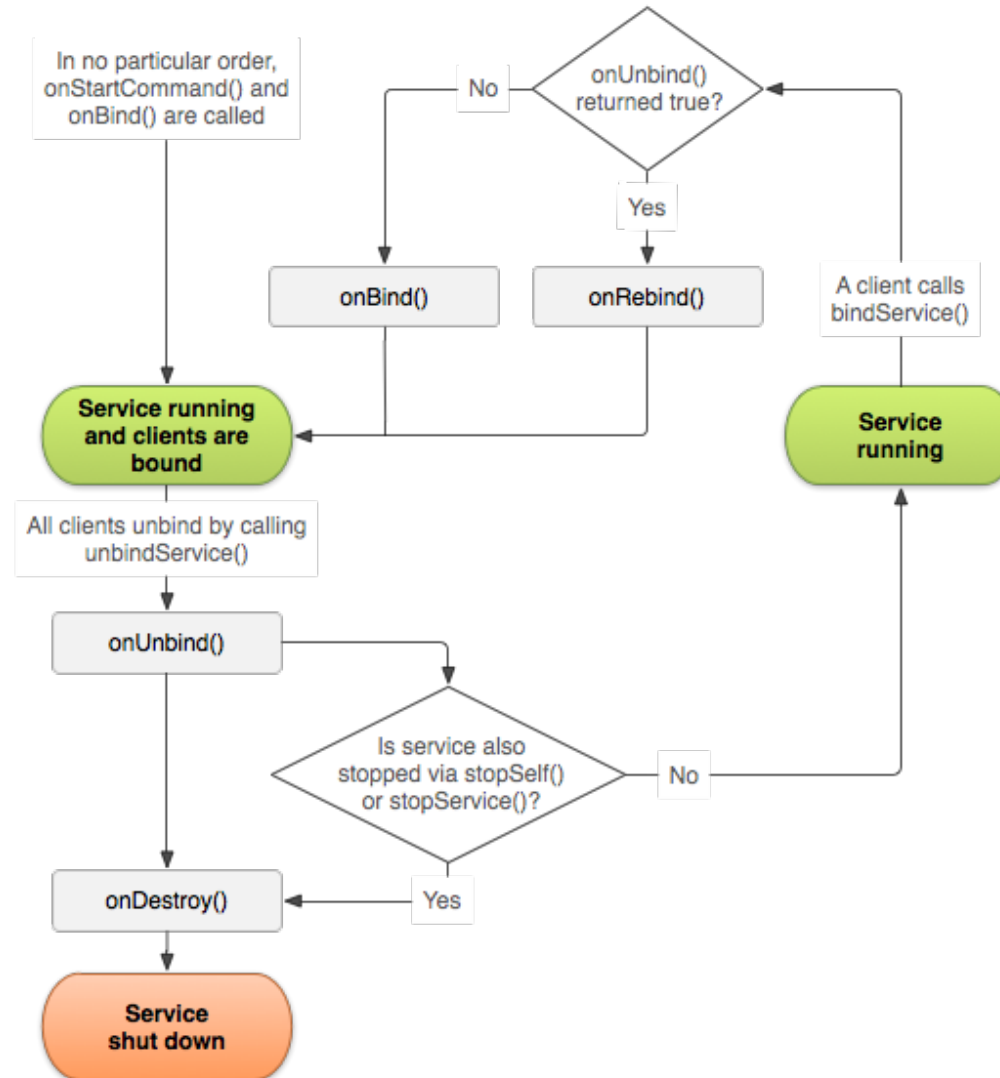

Service: Bounded Background

```
class BindService : Service() {  
  
    private val binder = LocalBinder()  
  
    override fun onUnbind(intent: Intent?): Boolean = super.onUnbind(intent)  
  
    override fun onRebind(intent: Intent?) = super.onRebind(intent)  
  
    override fun onBind(intent: Intent?): IBinder = binder  
  
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int = START_STICKY  
  
    inner class LocalBinder : Binder() {  
        val service: BindService  
        get() = this@BindService  
    }  
}
```

Service: Bounded Background

```
class BindActivity : ComponentActivity() {  
  
    private var service: BindService? = null  
  
    private val serviceConnection = object : ServiceConnection {  
        override fun onServiceConnected(name: ComponentName?, binder: IBinder?) {  
            service = (binder as? BindService.LocalBinder)?.service  
        }  
  
        override fun onServiceDisconnected(name: ComponentName?) {  
            service = null  
        }  
    }  
  
    override fun onStart() {  
        val intent = Intent(this, BindService::class.java)  
        // startService(intent)  
        bindService(intent, serviceConnection, BIND_AUTO_CREATE)  
        super.onStart()  
    }  
  
    override fun onStop() {  
        unbindService(serviceConnection)  
        super.onStop()  
    }  
}
```

Service: Bounded + Started



Service: Foreground

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <application>

        <activity/>

        <service android:name=".ForegroundService" android:foregroundServiceType="camera|microphone"/>

    </application>

</manifest>
```

Service: Foreground

```
class ForegroundService : Service() {
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
            //      startForeground(1, createNotification(), FOREGROUND_SERVICE_TYPE_MANIFEST)
            //      startForeground(1, createNotification(), FOREGROUND_SERVICE_TYPE_LOCATION)
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
                startForeground(
                    1,
                    createNotification(),
                    FOREGROUND_SERVICE_TYPE_CAMERA or FOREGROUND_SERVICE_TYPE_MICROPHONE
                )
            }
        } else {
            startForeground(1, createNotification())
        }
        return super.onStartCommand(intent, flags, startId)
    }
    private fun createNotification(): Notification {...}

    override fun onBind(intent: Intent?): IBinder? = null
}
```

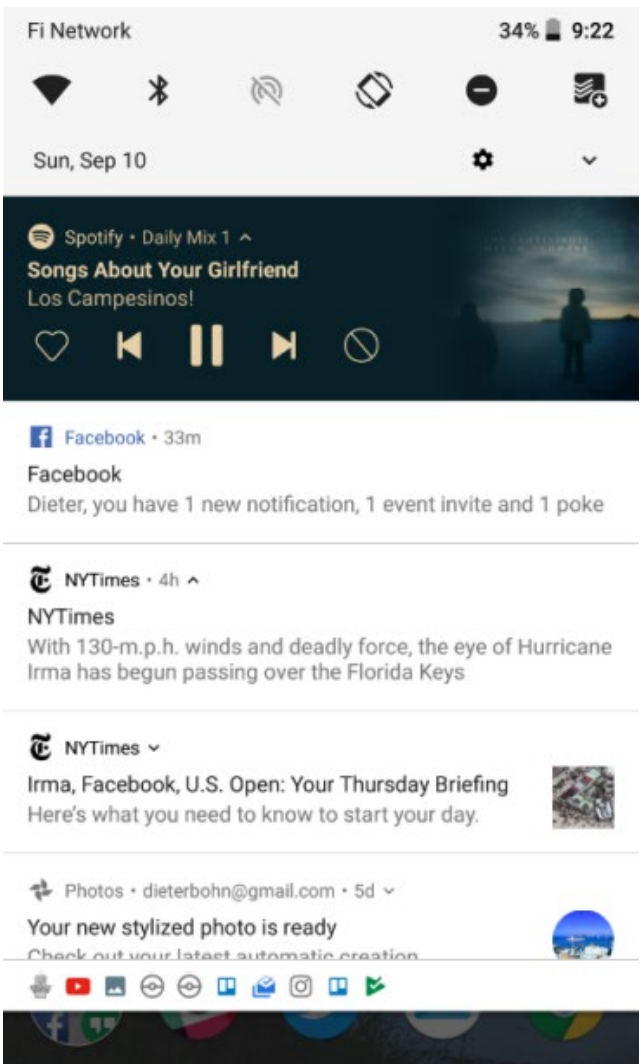
Service: Foreground

```
class ForegroundActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        ContextCompat.startForegroundService(this, Intent(this, ForegroundService::class.java))  
    }  
}
```

Уведомления



Уведомления



Уведомления

```
class ForegroundService : Service() {
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
            //      startForeground(1, createNotification(), FOREGROUND_SERVICE_TYPE_MANIFEST)
            //      startForeground(1, createNotification(), FOREGROUND_SERVICE_TYPE_LOCATION)
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
                startForeground(
                    1,
                    createNotification(),
                    FOREGROUND_SERVICE_TYPE_CAMERA or FOREGROUND_SERVICE_TYPE_MICROPHONE
                )
            }
        } else {
            startForeground(1, createNotification())
        }
        return super.onStartCommand(intent, flags, startId)
    }
    private fun createNotification(): Notification {...}

    override fun onBind(intent: Intent?): IBinder? = null
}
```

Уведомления

- Notification – описание самого уведомления

Уведомления

- Notification – описание самого уведомления
- NotificationChannel – описание канала, в который входит уведомления. Позволяет описать общие настройки и отображается в настройках, чтобы пользователь мог отключить уведомления по каналу.

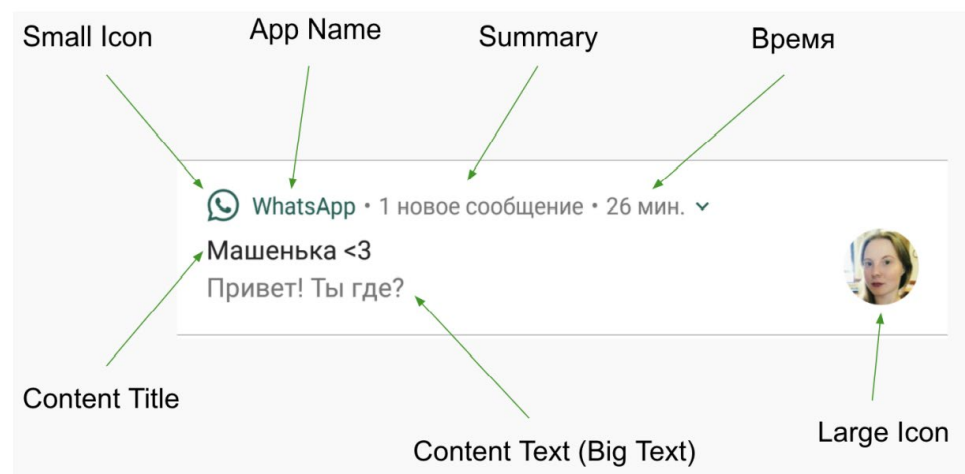
Уведомления

- Notification – описание самого уведомления
- NotificationChannel – описание канала, в который входит уведомления. Позволяет описать общие настройки и отображается в настройках, чтобы пользователь мог отключить уведомления по каналу.
- NotificationGroup – группировка каналов на экране настроек

Уведомления

- Notification – описание самого уведомления
- NotificationChannel – описание канала, в который входит уведомления. Позволяет описать общие настройки и отображается в настройках, чтобы пользователь мог отключить уведомления по каналу.
- NotificationGroup – группировка каналов на экране настроек
- NotificationManager – отвечает за показ и удаление уведомлений, а так же создание каналов и групп

Уведомления. Разбор



Уведомления. Стил

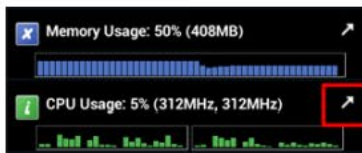
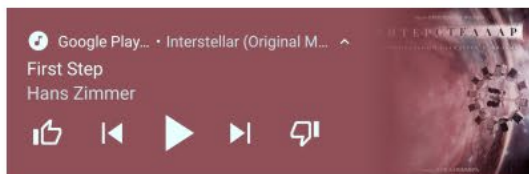
BigTextStyle

 NYTimes • 4h ^

NYTimes

With 130-m.p.h. winds and deadly force, the eye of Hurricane Irma has begun passing over the Florida Keys

MediaStyle



CustomView

InboxStyle

 Gmail • bogdan247@gmail.com • 11m v

scottadamgordon Samsung's DJ Koh: here's why the Note...

scottadamgordon IFA preview update [News]

Bloomberg Technology No more whole paychecks

BigPictureStyle

 Скриншот ^

Скриншот сохранен

Нажмите, чтобы просмотреть

Закат: 16:05

Влажность: 90%

ТЕМП. ОСАДКИ ВЕТЕР ВЛАЖН.



ОТПРАВИТЬ УДАЛИТЬ

Уведомления. Важность

User-visible importance level	Importance (Android 8.0 and higher)	Priority (Android 7.1 and lower)
Urgent Makes a sound and appears as a heads-up notification	<code>IMPORTANCE_HIGH</code>	<code>PRIORITY_HIGH</code> or <code>PRIORITY_MAX</code>
High Makes a sound	<code>IMPORTANCE_DEFAULT</code>	<code>PRIORITY_DEFAULT</code>
Medium No sound	<code>IMPORTANCE_LOW</code>	<code>PRIORITY_LOW</code>
Low No sound and does not appear in the status bar	<code>IMPORTANCE_MIN</code>	<code>PRIORITY_MIN</code>

Уведомления

```
private fun createChannel() {  
    val myGroup = NotificationChannelGroupCompat.Builder(groupId)  
        .setName("My group")  
        .setDescription("My group description")  
        .build()  
  
    val myChannel = NotificationChannelCompat.Builder(  
        /* id = */ channelId,  
        /* importance = */ NotificationManager.IMPORTANCE_HIGH  
    )  
        .setName("My channel name")  
        .setDescription("My channel description")  
        .setGroup(groupId)  
        // .setSound()  
        // .setVibrationPattern()  
        // .setLightsEnabled()  
        // .setLightColor()  
        .build()  
  
    manager.createNotificationChannelGroup(myGroup)  
    manager.createNotificationChannel(myChannel)  
}
```

Уведомления

```
val manager = NotificationManagerCompat.from(this)
```

```
private fun createNotification(): Notification {
```

```
    val channelId = createChannel(manager)
```

```
    return NotificationCompat.Builder(this, channelId)
```

```
        .setTitle("Title")
```

```
        .setContentText("Text")
```

```
        .setContentIntent(
```

```
            PendingIntent.getActivity(
```

```
                this,
```

```
                1,
```

```
                Intent(this, MainActivity::class.java),
```

```
                FLAG_IMMUTABLE or FLAG_UPDATE_CURRENT
```

```
            )
```

```
        )
```

```
        .setWhen(System.currentTimeMillis()).setShowWhen(true)
```

```
        .setStyle(NotificationCompat.BigTextStyle().bigText("long long..."))
```

```
        .setPriority(NotificationCompat.PRIORITY_HIGH) // only for Android 7 and lower
```

```
        .setOngoing(true)
```

```
        .setSmallIcon(R.drawable.ic_launcher_foreground)
```

```
        .addAction(NotificationCompat.Action(null, "Hi", null))
```

```
        // .setLargeIcon()
```

```
        // .setCustomContentView()
```

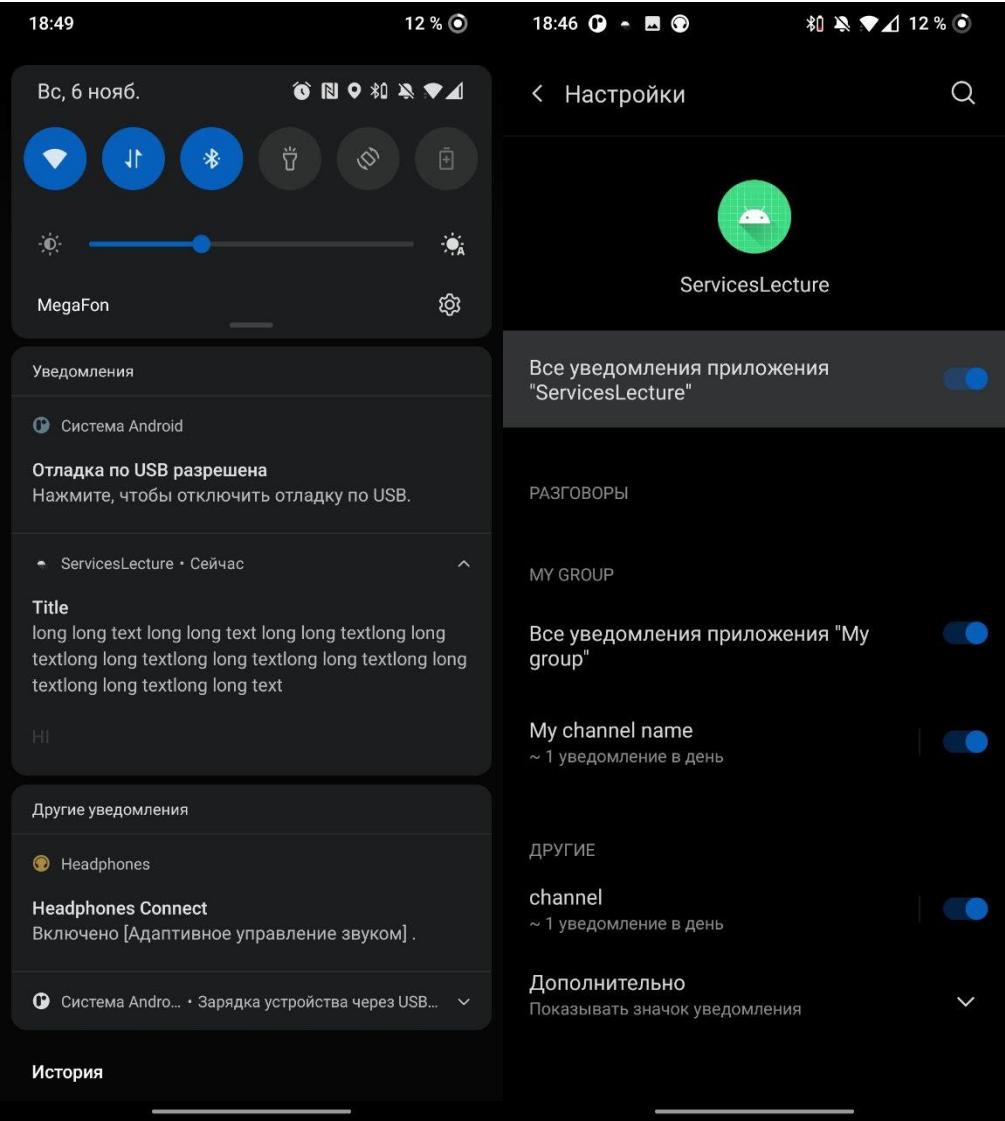
```
        // .setDeleteIntent()
```

```
        // .setGroup("")
```

```
        .build()
```

```
    }
```

Уведомления

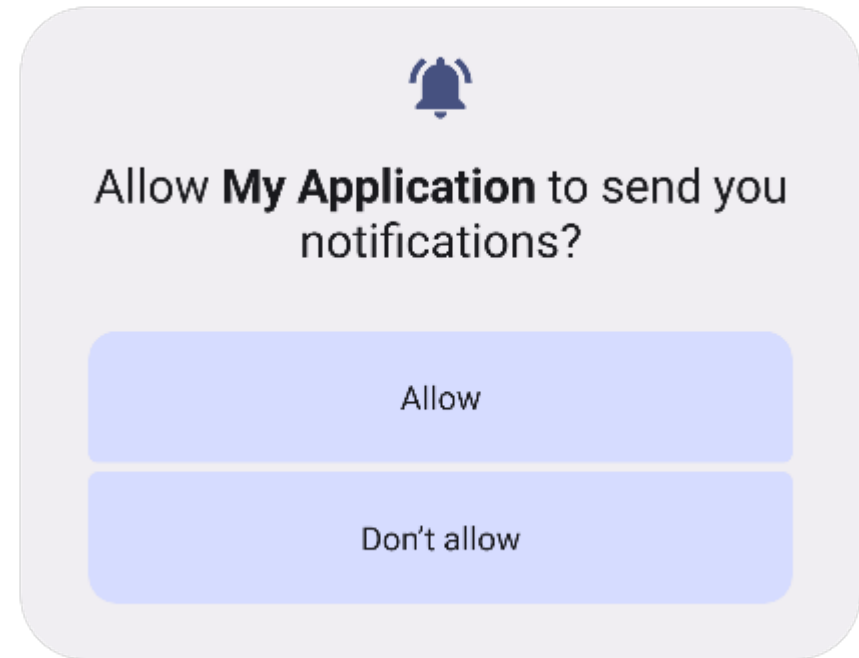


Разрешение на уведомления Android 13 (API 33)

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
  xmlns:tools="http://schemas.android.com/tools">  
  
  <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>  
  
</manifest>
```

Без разрешения уведомление будет отображено если это уведомление от:

- Foreground Service
- Музыкального плеера



Уведомления. Лайфхак

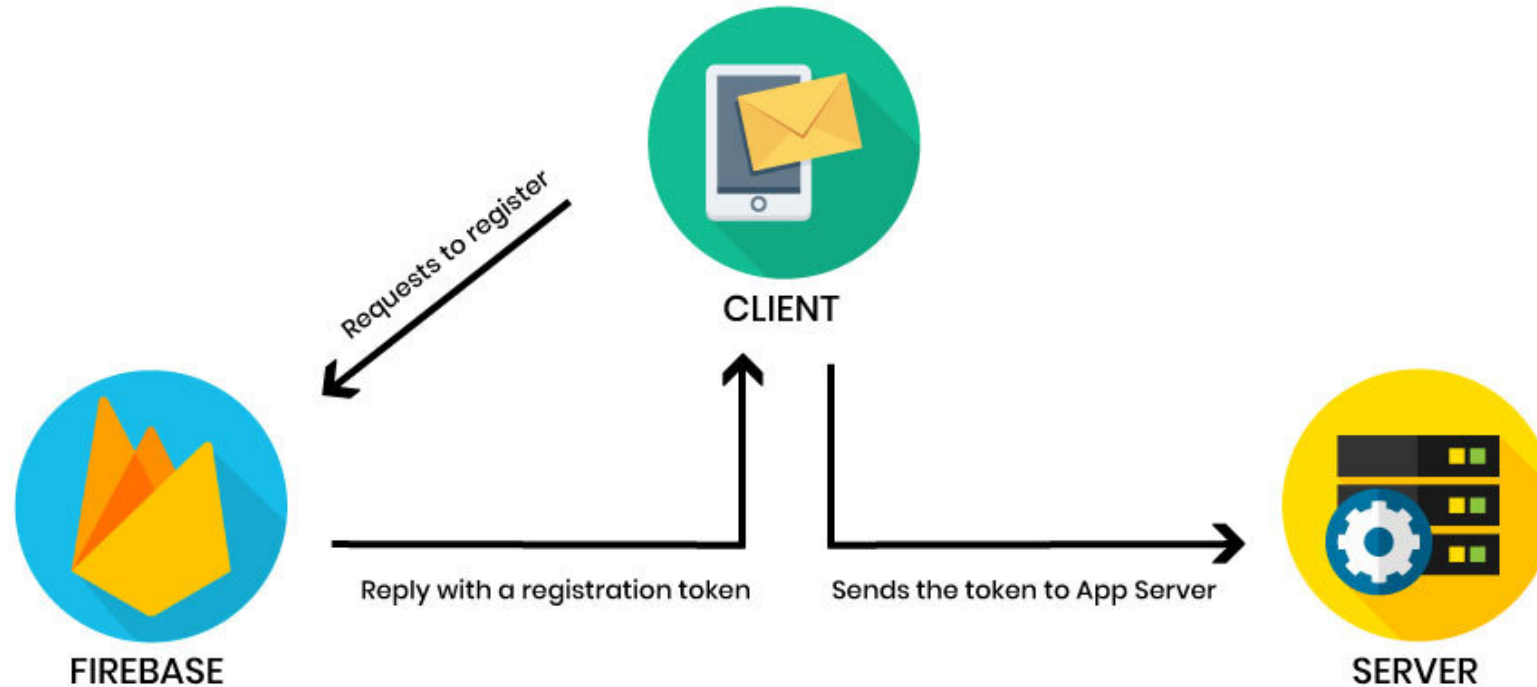
- Добавляется через виджет настроек на главном экране
- При удержании даты уведомления раскрывается полное описание



FCM




Firebase Cloud Messaging




Firebase Assistant


Assistant: Firebase

 **Firebase**


Firebase gives you the tools and infrastructure from Google to help you develop, grow and earn money from your app. [Learn more](#)

▶  **Analytics**


Measure user activity and engagement with free, easy, and unlimited analytics. [More info](#)

▶  **Authentication**


Sign in and manage users with ease using popular login providers like Google, Facebook, and others. You can even use a custom authentication system. [More info](#)

▶  **Realtime Database**


Store and sync data with this cloud-hosted NoSQL database. Data is synced across all clients in realtime and remains available when your app goes offline. [More info](#)

▶  **Cloud Firestore**


Store and sync your app data with this flexible, scalable NoSQL cloud-hosted database. [More info](#).

▶  **Cloud Storage for Firebase**


Store and retrieve large files like images, audio, and video without writing server-side code. [More info](#)

▶  **Cloud Functions for Firebase**


Automatically run backend code in response to events triggered by Firebase features and HTTPS requests. [More info](#)

▶  **Firebase ML**


Firebase ML is a mobile SDK that brings Google's machine learning expertise to Android and iOS apps in a powerful yet easy-to-use package. [More info](#)

▶  **Crashlytics**

Get clear, actionable insight into app issues that erode your app quality. [More info](#)

▶  **Performance Monitoring**

Gain insight into the performance characteristics of your app. [More info](#)

▶  **Test Lab**

Assistant

Device Manager

Gradle

Notifications

Device File Explorer

Emulator

Assistant: Firebase

← **Firebase** > Cloud Messaging

Set up Firebase Cloud Messaging [KOTLIN]

Firebase Cloud Messaging lets you receive and send messages from your app to the server and other clients. This how to set up FCM and enable your app to receive notifications.

[Launch in browser](#)

1

Connect your app to Firebase

✓ Connected

2

Add FCM to your app

✓ Dependencies set up correctly

NOTE: After adding the SDK, here are some other helpful configurations to consider:

- Do you want an easier way to manage library versions?
You can use the [Firebase Android BoM](#) to manage your Firebase library versions and ensure that you using compatible library versions.

3

Access the device registration token

On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you single devices, or create device groups, you'll need to access this token.

You can access the token's value by creating a new class which extends [FirebaseInstanceIdService](#) . In that cl within [onTokenRefresh](#) , and log the value as shown:

```
override fun onNewToken(token: String) {  
    Log.d(TAG, "Refreshed token: $token")  
}
```


Firebase Cloud Messaging

```
<service
  android:name=".FirebaseSimpleService"
  android:exported="true">
  <intent-filter>
    <action android:name="com.google.firebase.MESSAGING_EVENT" />
  </intent-filter>
</service>
```

implementation 'com.google.firebase:firebase-messaging-ktx:23.1.0'

Firebase Cloud Messaging

```
class FirebaseSimpleService : FirebaseMessagingService() {  
  
    override fun onNewToken(token: String) { sendTokenToServer(token) }  
  
    override fun onMessageReceived(message: RemoteMessage) { showNotification(message) }  
}  
  
// try retrieve token if exist  
suspend fun getToken(): String? = FirebaseMessaging.getInstance().token.await()
```

WorkManager



WorkManager

Библиотека для выполнения асинхронных задач в фоне.

- Учитывает состояние системы - наличие сети или зарядки – и позволяет настраивать ограничения для запуска задач
- Выполнение одноразовых и периодических задач
- Гарантирует выполнение задачи даже после перезагрузки устройства
- Позволяет планировать порядок выполнения задач и следить за их статусом
- Имеет настройку для повтора задач при неуспешном выполнении
- Экономит системных ресурсов и заряд устройства

WorkManager

- **Worker** – содержит код задачи, которая должна выполняться

WorkManager

- **Worker** – содержит код задачи, которая должна выполниться
- **WorkerRequest** – описание того при каких условиях должна запуститься задача, как часто должна выполняться и какие данные должен получить Worker

WorkManager

- **Worker** – содержит код задачи, которая должна выполниться
- **WorkerRequest** – описание того при каких условиях должна запуститься задача, как часто должна выполняться и какие данные должен получить Worker
- **Constraints** – ограничения, которые передаются в WorkerReques

WorkManager

- **Worker** – содержит код задачи, которая должна выполняться
- **WorkerRequest** – описание того при каких условиях должна запуститься задача, как часто должна выполняться и какие данные должен получить Worker
- **Constraints** – ограничения, которые передаются в WorkerRequest
- **WorkManager** – управляет запуском задач, порядком их выполнения и позволяет получать информацию о статусе выполнения

WorkManager: типы задач

По скорости запуска выполнения задачи:

- Немедленные
- Обычные
- Отложенные

По скорости выполнения:

- Быстрые: необязательно показывать уведомление
- Долгие: необходимо показать уведомление, чтобы система не убила выполнение

По типу планирования:

- Одноразовые
- Периодические

WorkManager. Создание и запуск задачи

```
class SimpleWorker(appContext: Context, params: WorkerParameters) : CoroutineWorker(appContext, params) {  
  
    // if expedited  
    override suspend fun getForegroundInfo(): ForegroundInfo {  
        return ForegroundInfo(/* notificationId = */ 1, /* notification = */ createNotification(applicationContext, "BaseWorker", "Im so important"))  
    }  
  
    override suspend fun doWork(): Result {  
        val msg = inputData.getString(MSG_KEY).orEmpty()  
        return Result.success()  
    }  
  
    companion object {  
        private const val MSG_KEY = "MSG_KEY"  
  
        fun getWorkData(msg: String): Data {  
            return workDataOf(  
                MSG_KEY to msg,  
            )  
        }  
    }  
}
```

WorkManager. Создание и запуск задачи

```
val constraints = Constraints.Builder()  
    .setRequiredNetworkType(NetworkType.CONNECTED)  
    .build()
```

```
PeriodicWorkRequestBuilder<SimpleWorker>(MIN_PERIODIC_INTERVAL_MILLIS, TimeUnit.MILLISECONDS)  
val request = OneTimeWorkRequestBuilder<SimpleWorker>()  
    .setInitialDelay(5, TimeUnit.SECONDS)  
    .setExpedited(OutOfQuotaPolicy.RUN_AS_NON_EXPEDITED_WORK_REQUEST)  
    .setConstraints(constraints)  
    .addTag("simple")  
    .build()
```

```
workManager.enqueue(request)  
workManager.getWorkInfosByTagLiveData("simple").observe(this) {}
```

WorkManager. Создание и запуск задачи

```
val constraints = Constraints.Builder()  
    .setRequiredNetworkType(NetworkType.CONNECTED)  
    .build()
```

```
PeriodicWorkRequestBuilder<SimpleWorker>(MIN_PERIODIC_INTERVAL_MILLIS, TimeUnit.MILLISECONDS)  
val request = OneTimeWorkRequestBuilder<SimpleWorker>()  
    .setInitialDelay(5, TimeUnit.SECONDS)  
    .setExpedited(OutOfQuotaPolicy.RUN_AS_NON_EXPEDITED_WORK_REQUEST)  
    .setConstraints(constraints)  
    .addTag("simple")  
    .build()
```

```
workManager.enqueue(request)  
workManager.getWorkInfosByTagLiveData("simple").observe(this) {}
```

WorkManager. Создание и запуск задачи

```
val constraints = Constraints.Builder()  
    .setRequiredNetworkType(NetworkType.CONNECTED)  
    .build()
```

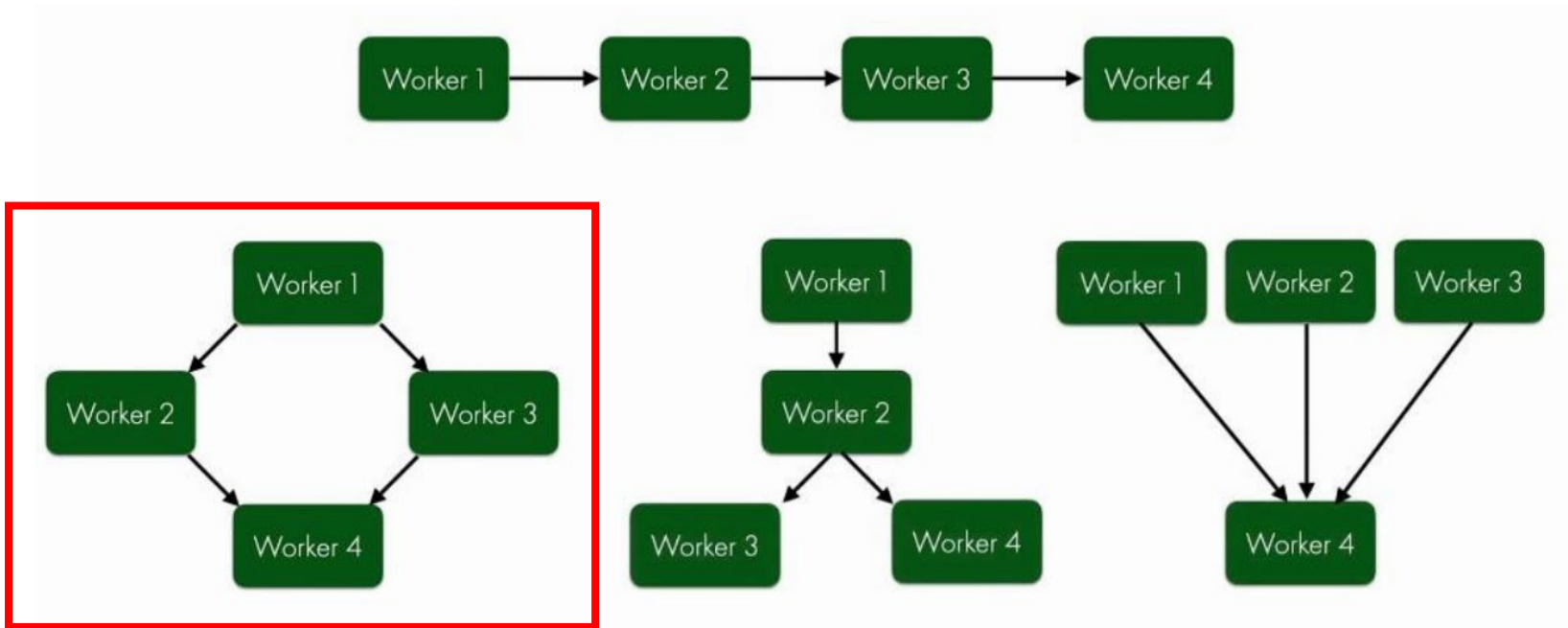
```
PeriodicWorkRequestBuilder<SimpleWorker>(MIN_PERIODIC_INTERVAL_MILLIS, TimeUnit.MILLISECONDS)  
val request = OneTimeWorkRequestBuilder<SimpleWorker>()  
    .setInitialDelay(5, TimeUnit.SECONDS)  
    .setExpedited(OutOfQuotaPolicy.RUN_AS_NON_EXPEDITED_WORK_REQUEST)  
    .setConstraints(constraints)  
    .addTag("simple")  
    .build()
```

```
workManager.enqueue(request)  
workManager.getWorkInfosByTagLiveData("simple").observe(this) {}
```

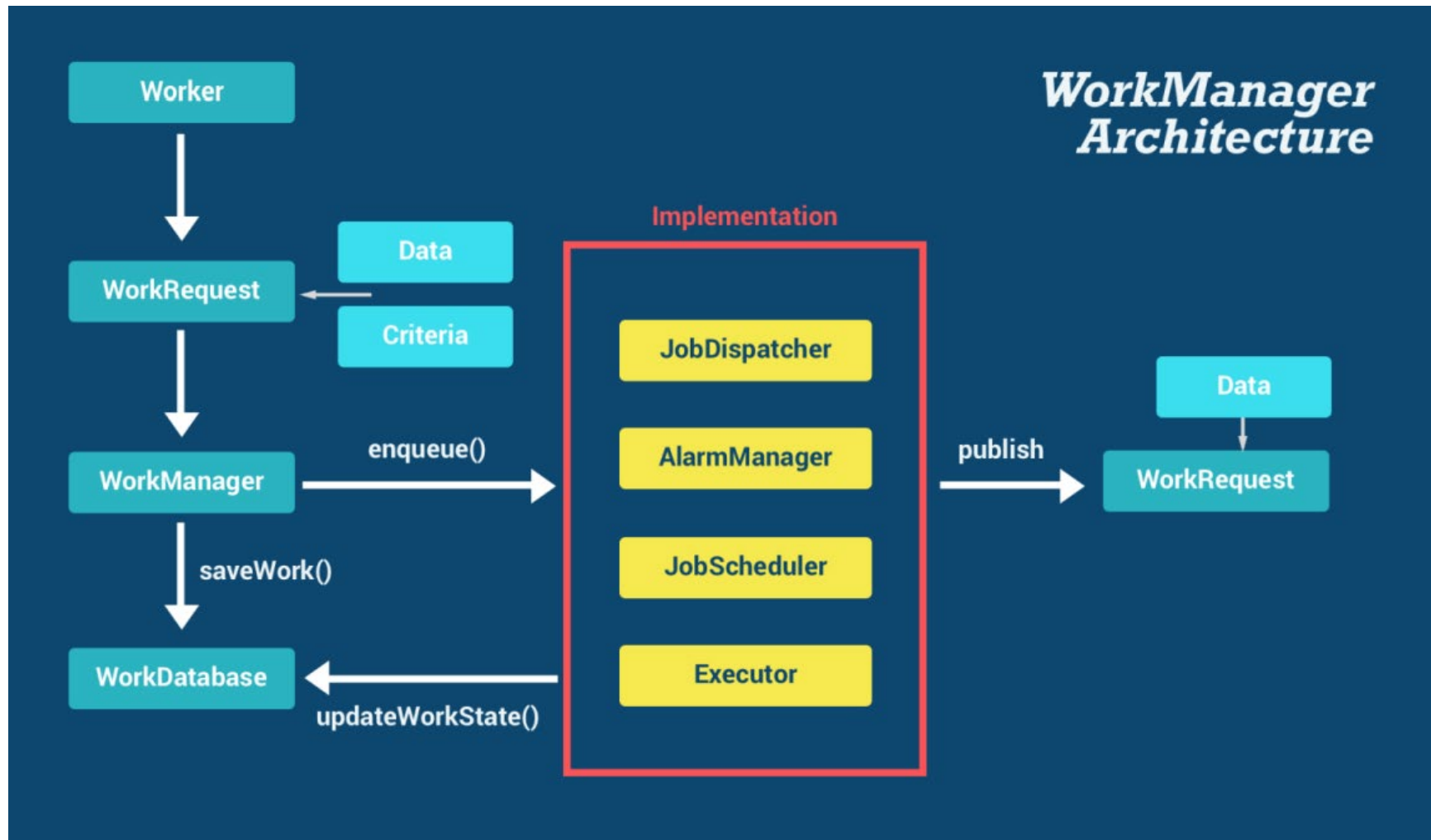
WorkManager. Порядок выполнения задач

workManager

```
.beginWith(request1)  
.then(listOf(request2, request3))  
.then(request4)  
.enqueue()
```



WorkManager



WorkManager. Foreground

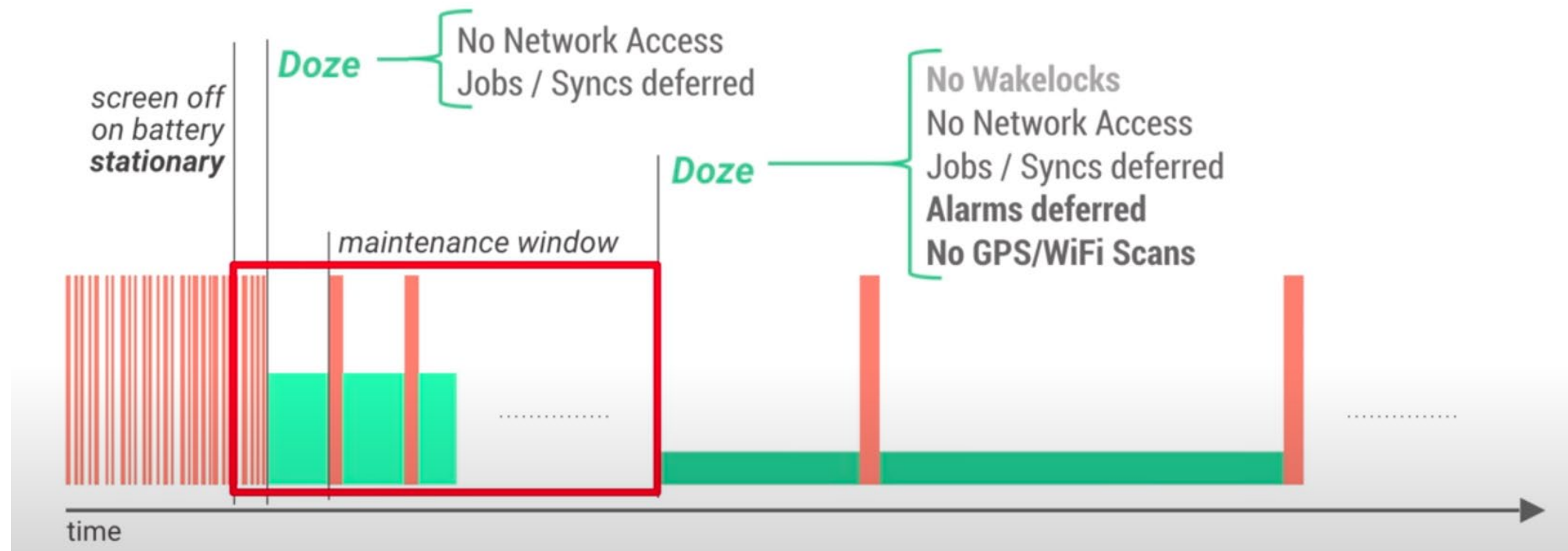
```
<service
    android:name="androidx.work.impl.foreground.SystemForegroundService"
    android:foregroundServiceType="microphone"
    tools:node="merge" />
```

```
class SimpleWorker {
    ...
    override suspend fun doWork(): Result {
        setForegroundAsync(
            ForegroundInfo(
                /* notificationId = */ 1,
                /* notification = */ createNotification(applicationContext, "BaseWorker", msg),
                FOREGROUND_SERVICE_TYPE_MANIFEST // FOREGROUND_SERVICE_TYPE_MICROPHONE
            )
        )
        return Result.success()
    }
    ...
}
```


Ограничения



Doze Mode - режим «спячки». Android 6 (API 23)



App Standby Buckets. Android 9 (API 28)

Ограничения не вводятся если:

- Пользователь запускает приложение сам
- Имеются процессы, выполняемые в foreground
- Приложение имеет уведомления, ожидающих обработки

App Standby Buckets. Android 9 (API 28)



TARE Android 13 (API 33)

TARE - The Android Resource Economy

Работает на системе кредитов, которые выдаются приложениям для выполнения задач и количество которых зависит от заряда устройства.

Приложение тратит кредиты на выполнение задач.

18:05



Настройки

JobScheduler

Ограничение потребления

Начальное ограничение потребления
29000 А

Максимальное ограничение потребления
250000 А

Баланс

Максимальный баланс при полной зарядке
60000 А

Минимальный баланс при полном заряде
(для исключенных приложений)
15000 А

Минимальный баланс при полном заряде
(для консольных системных приложений)
7500 А

Минимальный баланс при полном заряде
(для остальных приложений)
2000 А

Действия (затраты на выполнение)

Запускается задание (макс. приоритет)
3 А

Выполняется задание (макс. приоритет)
2 А

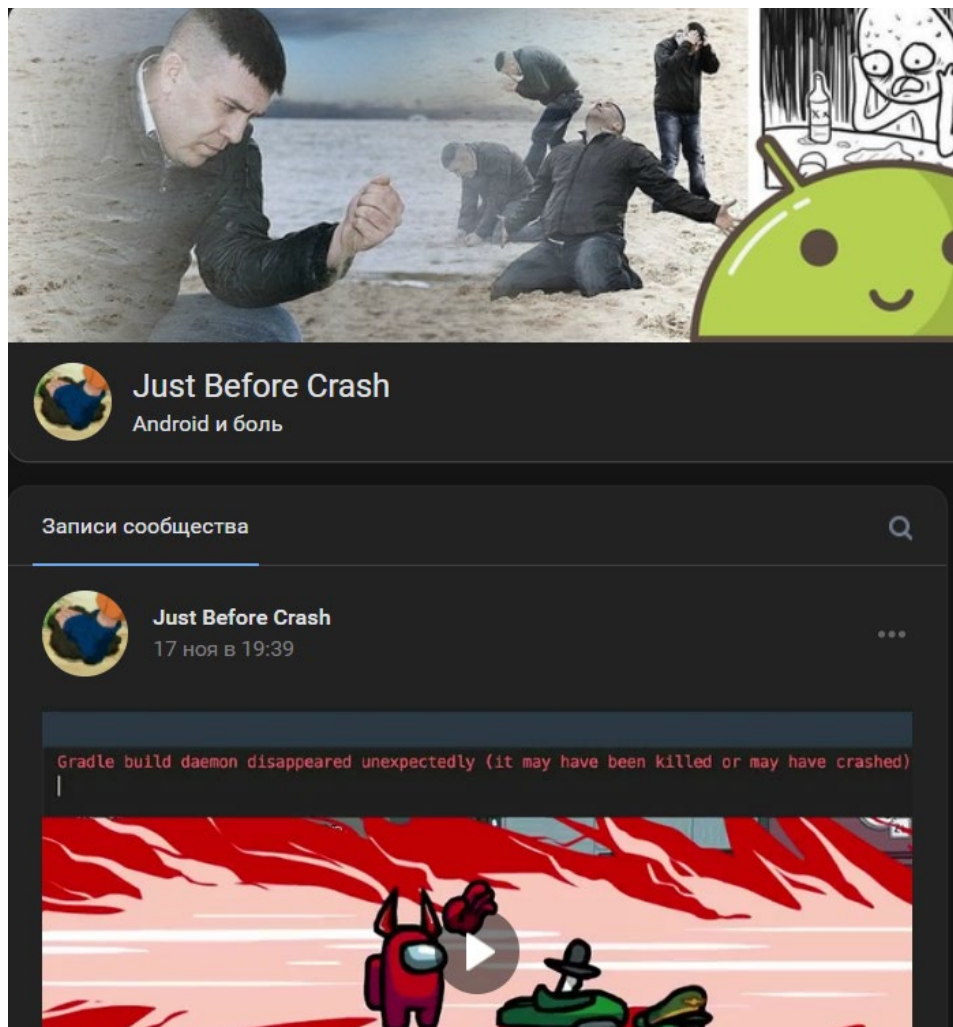
	Network	Job	Alarm	FCM
Doze	до окна обслуживания	до окна обслуживания	до окна обслуживания или до 9 минут	High: без ограничений Normal: до ближайшего окна обслуживания
Active	без ограничений	без ограничений	без ограничений	без ограничений
Working Set	без ограничений	до 2 часов	до 6 минут	без ограничений
<u>Frequent</u>	без ограничений	до 8 часов	до 30 минут	High: 10 в день
Rare	до 24 часов	до 24 часов	до 2 часов	High: 5 в день

Что и когда использовать?



Практика

Реклама



vk.com/love.android



Не забудьте
отметиться
на портале



Спасибо
за внимание!