avito.tech Москва 2024

Keep alive Android application

Матвей Плохов Android-разработчик

Плохов Матвей

Android-разработчик

- Работаю в Авито Недвижимость.
- Занимался МТС Лончером.
- Разрабатывал ускорители.
- Рад ответить на вопросы tg: @stringres

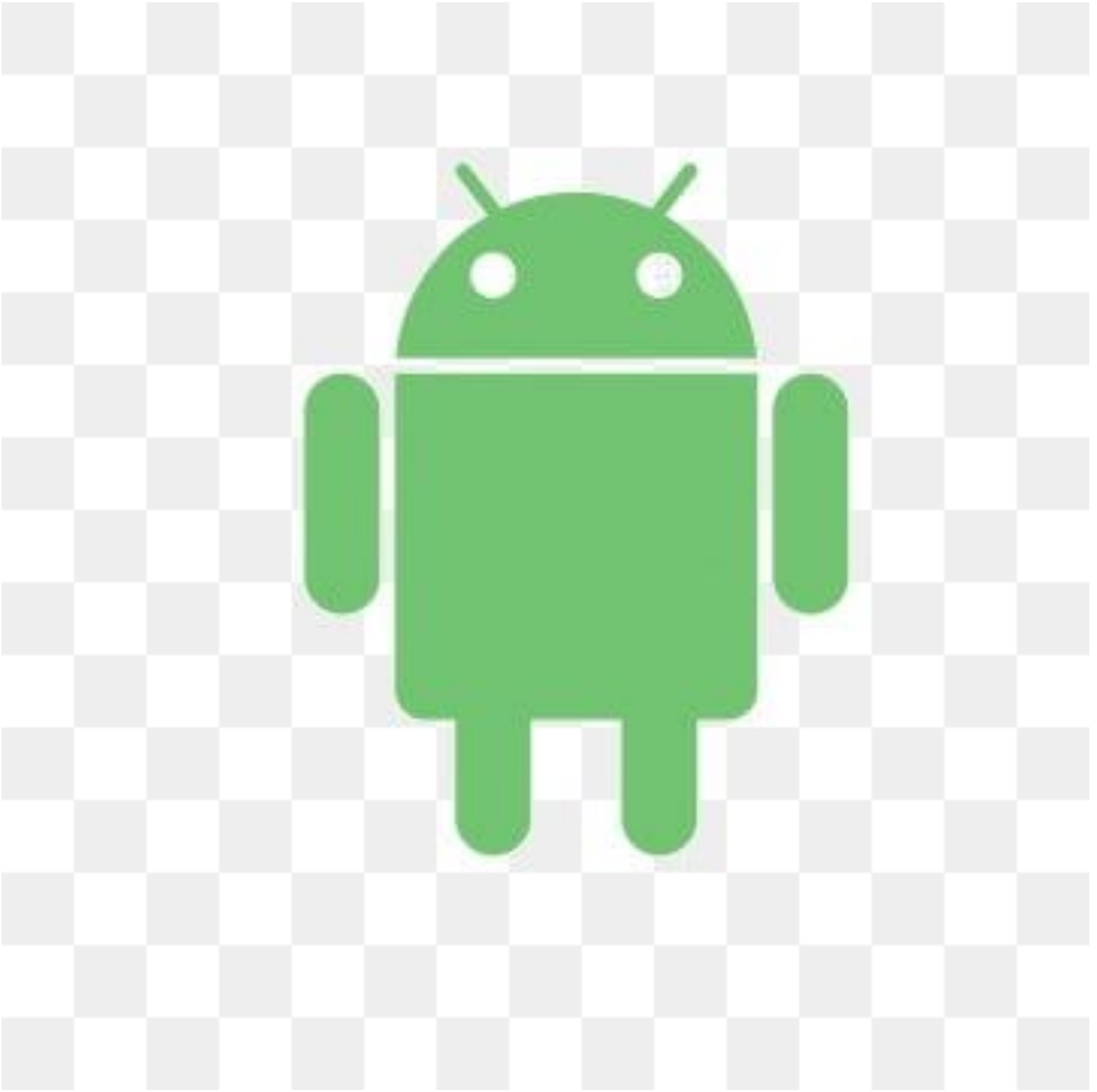
План презентации

Версии Android

Что стоит помнить на протяжении презентации.

На старых версиях Android реализовать бессмертное приложение намного проще, поэтому:

- ≥ Android 7.0
- Проверял на Google Pixel 4a + 13 android.
- Разные устройства — разное поведение.
- 14 Android стал более безопасным.
- Можно комбинировать.



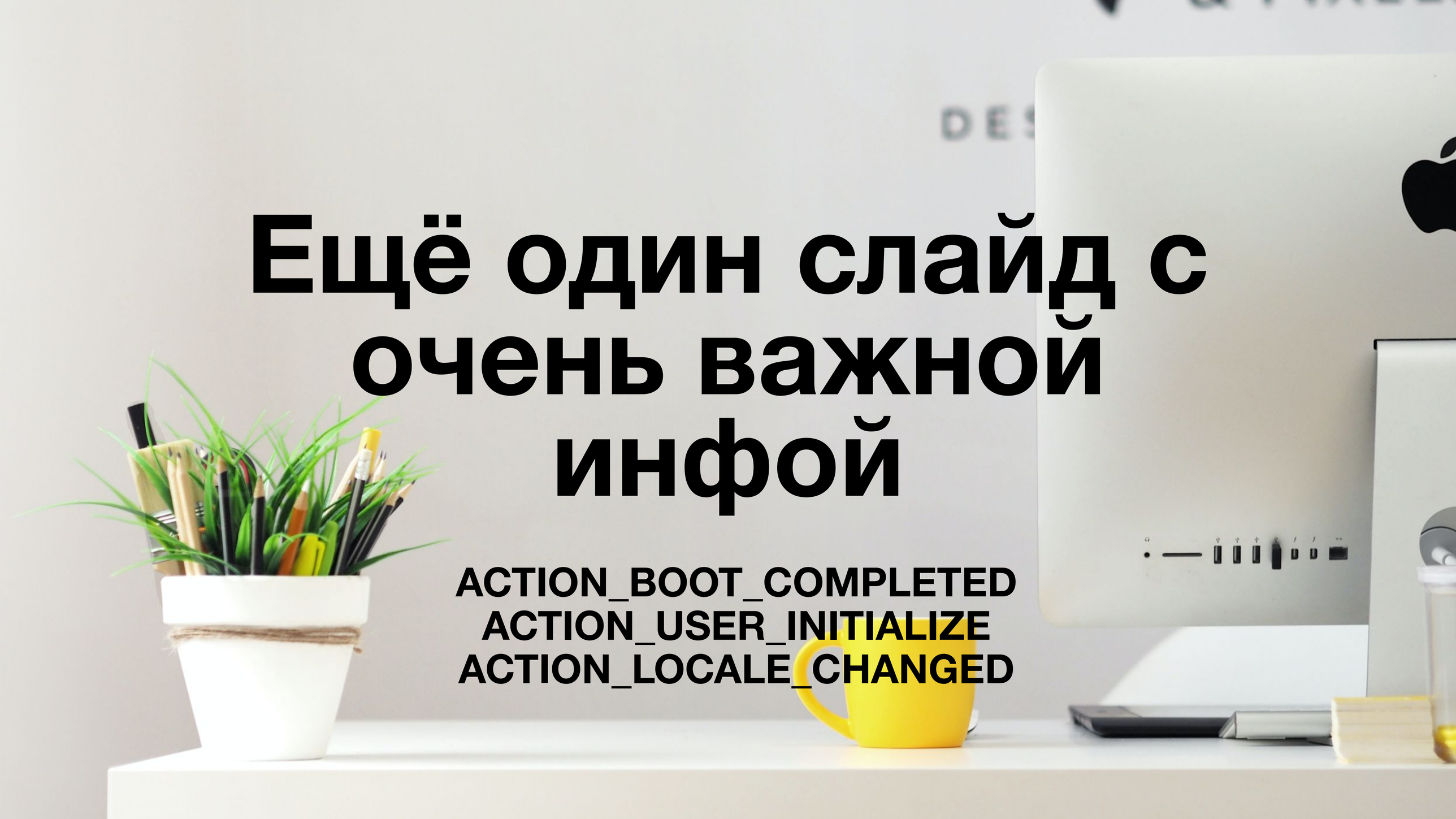
В один прекрасный день

Увеличить работу приложения в фоне
Отправлять уведомления при отсутствии интернета

Запускать процесс в фоне после скачивания
Переживать force stop
Открывать приложение при подключении зарядки

Первый запуск

1 2 3



Ещё один слайд с очень важной инфой

ACTION_BOOT_COMPLETED
ACTION_USER_INITIALIZE
ACTION_LOCALE_CHANGED

Первый запуск

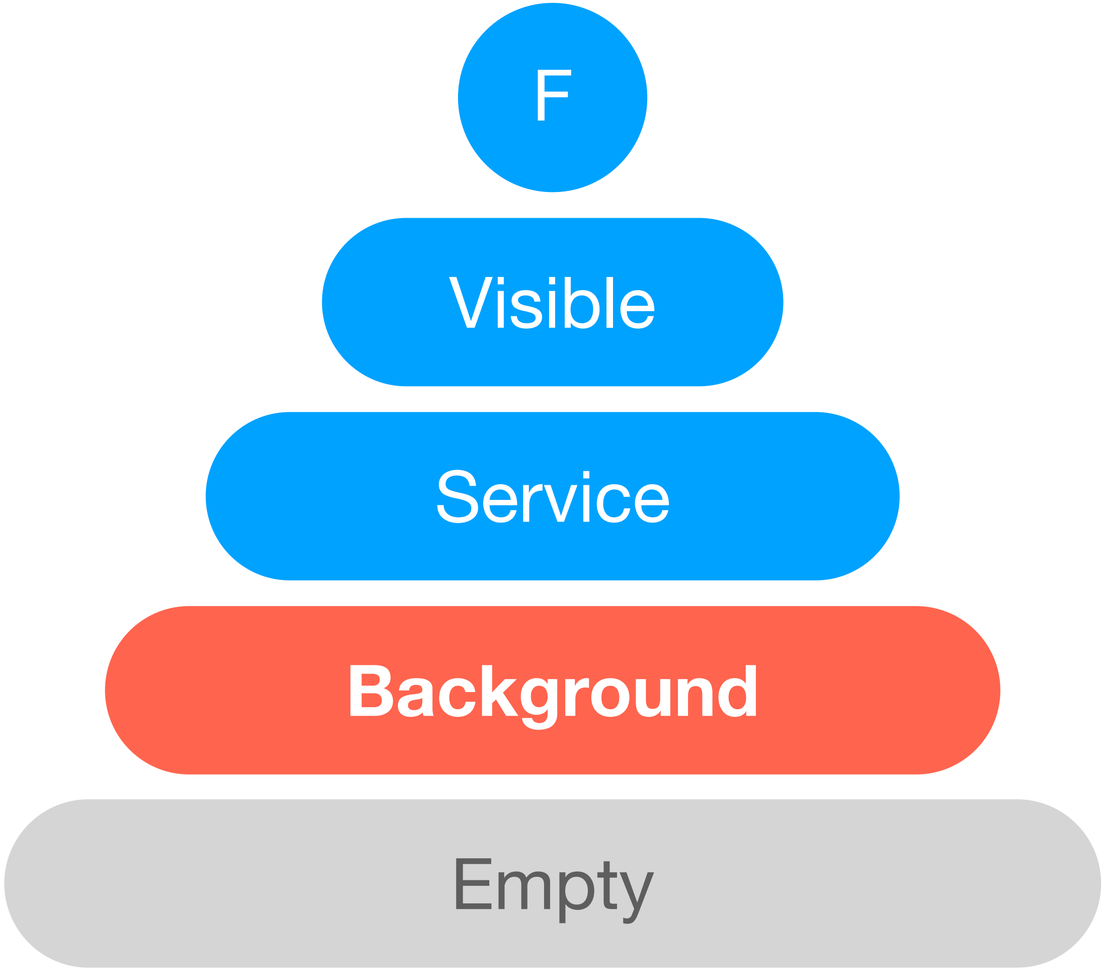
```xml
<provider
    android:name=".ContactDirectoryContentProvider"
    android:authorities="CONTENT_PROVIDER"
    android:enabled="true"
    android:exported="true">

    <meta-data
        android:name="android.content.ContactDirectory"
        android:value="true" />

</provider>
```

Contacts provider

Продолжительность жизни

1 2 3

Background process



Процессы, которые в данный момент не видны пользователю

WorkManager

- Идеально подходит для задач, которые можно отложить, то есть они не должны выполняться в определённое время, но должны выполняться при определённых условиях. Например, когда устройство заряжается или подключено к Wi-Fi.

AlarmManager

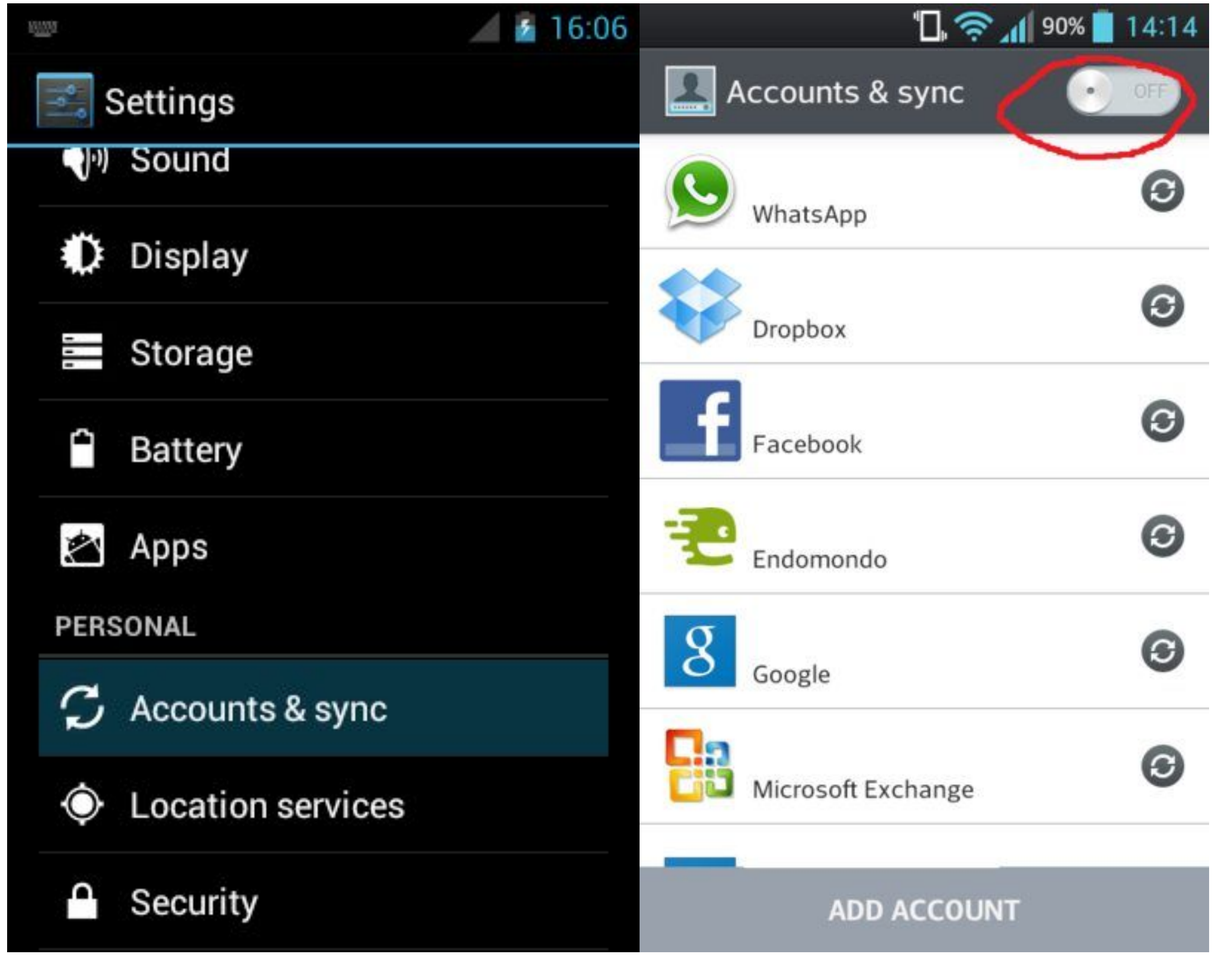
- Идеально подходит для задач, которые должны быть выполнены в точное время или через определенный промежуток времени. Например, приложение для будильника, которое должно включать сигнал в определённое время.

Account synchronization

Роль синхронизации учётной записи

Если данные приложения изменяются, их можно синхронизировать через учётную запись, а затем выполнить операцию синхронизации данных с сервером. При выполнении синхронизации система активирует соответствующий процесс приложения.

- Активация процесса — это лишь побочный эффект синхронизации учётной записи.
- Account synchronization позволяет пережить force stop.
- Почему не AlarmManager и JobScheduler? При force stop они отменяются.



Account synchronization

```
1 <provider
2     android:name=".AccountSyncContentProvider"
3     android:authorities="CONTENT_PROVIDER" />
```

Manifest

Для реализации синхронизации аккаунта нужен sync adapter. Он же требует, чтобы вы объявили тип учётной записи и ContentProvider для связи с ним, но это не означает, что вы должны их использовать. Можно написать заглушку для ContentProvider без реализации методов.

Authorities — это уникальное обозначение, по которому система будет понимать, к какому провайдеру нужно будет обратиться.

Account synchronization

```
1 class ThreadSyncAdapter @JvmOverloads constructor(
2     context: Context,
3     autoInitialize: Boolean,
4     allowParallelSyncs: Boolean = false
5 ) : AbstractThreadedSyncAdapter(context, autoInitialize, allowParallelSyncs) {
6
7     override fun onPerformSync(
8         account: Account,
9         extras: Bundle,
10        authority: String,
11        provider: ContentProviderClient,
12        syncResult: SyncResult
13    ) {
14        // code implementation
15    }
16 }
```

Sync adapter

Синхронизация учётной записи требует настройки класса AbstractThreadedSyncAdapter и поддержания объекта этого класса в сервисе.

Account synchronization

```kotlin
class AccountSyncService : Service() {

    override fun onCreate() {
        synchronized(syncAdapterLock) {
            syncAdapter = syncAdapter ?: ThreadSyncAdapter(
                context = applicationContext,
                autoInitialize = true
            )
        }
    }

    override fun onBind(intent: Intent): IBinder {
        return syncAdapter?.syncAdapterBinder ?: throw IllegalStateException()
    }

    companion object {
        private var syncAdapter: ThreadSyncAdapter? = null
        private val syncAdapterLock = Any()
    }
}
```

Singleton

Account synchronization

```xml
<?xml version="1.0" encoding="utf-8"?>
<sync-adapter xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="ACCOUNT_TYPE"
    android:contentAuthority="CONTENT_PROVIDER"
    android:allowParallelSyncs="false"
    android:supportsUploading="false"
    android:isAlwaysSyncable="true"
    android:userVisible="false" />
```

sync-adapter

Чтобы система знала про SyncAdapter, нам нужно предоставить платформе метаданные, которые описывают компонент и дополнительные флаги.

Account synchronization

```xml
<uses-permission android:name="android.permission.READ_SYNC_SETTINGS" />
<uses-permission android:name="android.permission.WRITE_SYNC_SETTINGS" />
```

```xml
<service
    android:name=".AccountSyncService"
    android:exported="false">
    <intent-filter>
        <action android:name="android.content.SyncAdapter" />
    </intent-filter>
    <meta-data
        android:name="android.content.SyncAdapter"
        android:resource="@xml/sync_adapter" />
</service>
```

Manifest

Account synchronization

```
1 class AccountAuthenticator(
2     context: Context
3 ) : AbstractAccountAuthenticator(context) {
4
5     override fun addAccount(
6         response: AccountAuthenticatorResponse?,
7         accountType: String?,
8         authTokenType: String?,
9         requiredFeatures: Array<out String>?,
10        options: Bundle?
11    ): Bundle {
12        // code implementation
13        return bundleOf()
14    }
15    ...
16 }
```

Account authenticator

Также следует добавить реализацию AbstractAccountAuthenticator, который позволит пользователю самостоятельно добавлять аккаунт.

Account synchronization

```
1 class AccountAuthenticatorService : Service() {
2
3     override fun onCreate() {
4         synchronized(accountAuthenticatorLock) {
5             accountAuthenticator = accountAuthenticator ?: AccountAuthenticator(
6                 context = applicationContext
7             )
8         }
9     }
10
11    override fun onBind(intent: Intent): IBinder {
12        return accountAuthenticator?.iBinder ?: throw IllegalStateException()
13    }
14
15    companion object {
16        private var accountAuthenticator: AccountAuthenticator? = null
17        private val accountAuthenticatorLock = Any()
18    }
19 }
```

Singleton

Account synchronization

```xml
<?xml version="1.0" encoding="utf-8"?>
<account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="ACCOUNT_TYPE"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name" />
```

```xml
<service
    android:name=".AccountAuthenticatorService"
    android:exported="false">
    <intent-filter>
        <action android:name="android.accounts.AccountAuthenticator" />
    </intent-filter>
    <meta-data
        android:name="android.accounts.AccountAuthenticator"
        android:resource="@xml/account_authenticator" />
</service>
```

Manifest

Account synchronization

```kotlin
val accountManager = getSystemService(Context.ACCOUNT_SERVICE) as? AccountManager ?: return
val account = Account("Account name", "ACCOUNT_TYPE")

if (accountManager.getAccountsByType("ACCOUNT_TYPE").isEmpty()) {
    accountManager.addAccountExplicitly(account, null, Bundle())
}

if (!ContentResolver.getMasterSyncAutomatically()) {
    ContentResolver.setMasterSyncAutomatically(true)
}

if (ContentResolver.getIsSyncable(account, "CONTENT_PROVIDER") ≤ 0) {
    ContentResolver.setIsSyncable(account, "CONTENT_PROVIDER", 1)
}

ContentResolver.setSyncAutomatically(account, "CONTENT_PROVIDER", true)
ContentResolver.addPeriodicSync(account, "CONTENT_PROVIDER", Bundle(), 1)

ContentResolver.requestSync(account, "CONTENT_PROVIDER", bundleOf())
```

Create account

Account synchronization

```kotlin
val accountManager = getSystemService(Context.ACCOUNT_SERVICE) as? AccountManager ?: return
val account = Account("Account name", "ACCOUNT_TYPE")

if (accountManager.getAccountsByType("ACCOUNT_TYPE").isEmpty()) {
    accountManager.addAccountExplicitly(account, null, Bundle())
}

if (!ContentResolver.getMasterSyncAutomatically()) {
    ContentResolver.setMasterSyncAutomatically(true)
}

if (ContentResolver.getIsSyncable(account, "CONTENT_PROVIDER") ≤ 0) {
    ContentResolver.setIsSyncable(account, "CONTENT_PROVIDER", 1)
}

ContentResolver.setSyncAutomatically(account, "CONTENT_PROVIDER", true)
ContentResolver.addPeriodicSync(account, "CONTENT_PROVIDER", Bundle(), 1)

ContentResolver.requestSync(account, "CONTENT_PROVIDER", bundleOf())
```

Create account

Account synchronization

```kotlin
val accountManager = getSystemService(Context.ACCOUNT_SERVICE) as? AccountManager ?: return
val account = Account("Account name", "ACCOUNT_TYPE")

if (accountManager.getAccountsByType("ACCOUNT_TYPE").isEmpty()) {
    accountManager.addAccountExplicitly(account, null, Bundle())
}

if (!ContentResolver.getMasterSyncAutomatically()) {
    ContentResolver.setMasterSyncAutomatically(true)
}

if (ContentResolver.getIsSyncable(account, "CONTENT_PROVIDER") ≤ 0) {
    ContentResolver.setIsSyncable(account, "CONTENT_PROVIDER", 1)
}

ContentResolver.setSyncAutomatically(account, "CONTENT_PROVIDER", true)
ContentResolver.addPeriodicSync(account, "CONTENT_PROVIDER", Bundle(), 1)

ContentResolver.requestSync(account, "CONTENT_PROVIDER", bundleOf())
```

Create account

Account synchronization

```kotlin
val accountManager = getSystemService(Context.ACCOUNT_SERVICE) as? AccountManager ?: return
val account = Account("Account name", "ACCOUNT_TYPE")

if (accountManager.getAccountsByType("ACCOUNT_TYPE").isEmpty()) {
    accountManager.addAccountExplicitly(account, null, Bundle())
}

if (!ContentResolver.getMasterSyncAutomatically()) {
    ContentResolver.setMasterSyncAutomatically(true)
}

if (ContentResolver.getIsSyncable(account, "CONTENT_PROVIDER") <= 0) {
    ContentResolver.setIsSyncable(account, "CONTENT_PROVIDER", 1)
}

ContentResolver.setSyncAutomatically(account, "CONTENT_PROVIDER", true)
ContentResolver.addPeriodicSync(account, "CONTENT_PROVIDER", Bundle(), 1)

ContentResolver.requestSync(account, "CONTENT_PROVIDER", bundleOf())
```

Create account

Account synchronization

```kotlin
val accountManager = getSystemService(Context.ACCOUNT_SERVICE) as? AccountManager ?: return
val account = Account("Account name", "ACCOUNT_TYPE")

if (accountManager.getAccountsByType("ACCOUNT_TYPE").isEmpty()) {
    accountManager.addAccountExplicitly(account, null, Bundle())
}

if (!ContentResolver.getMasterSyncAutomatically()) {
    ContentResolver.setMasterSyncAutomatically(true)
}

if (ContentResolver.getIsSyncable(account, "CONTENT_PROVIDER") ≤ 0) {
    ContentResolver.setIsSyncable(account, "CONTENT_PROVIDER", 1)
}

ContentResolver.setSyncAutomatically(account, "CONTENT_PROVIDER", true)
ContentResolver.addPeriodicSync(account, "CONTENT_PROVIDER", Bundle(), 1)

ContentResolver.requestSync(account, "CONTENT_PROVIDER", bundleOf())
```

Create account

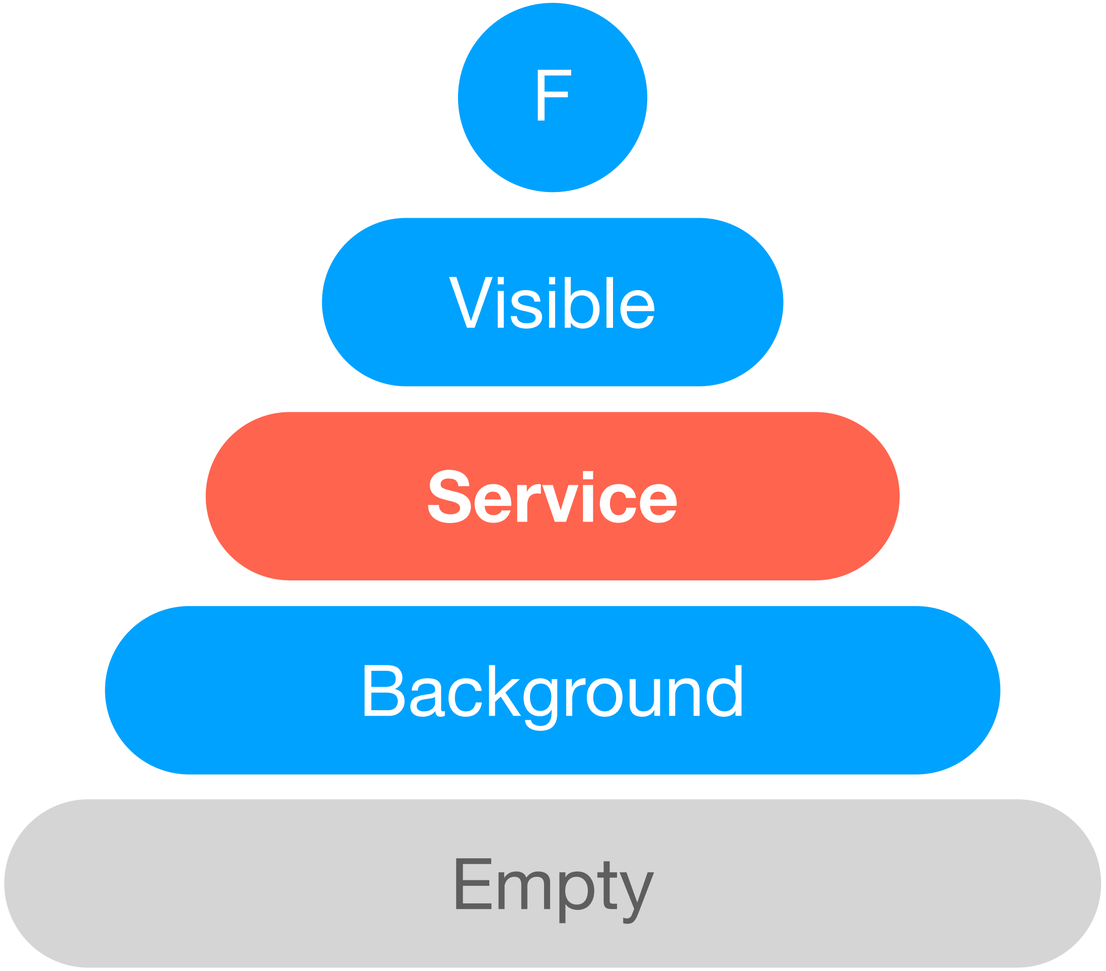
Account synchronization

```kotlin
val accountManager = getSystemService(Context.ACCOUNT_SERVICE) as? AccountManager ?: return
val account = Account("Account name", "ACCOUNT_TYPE")

if (accountManager.getAccountsByType("ACCOUNT_TYPE").isEmpty()) {
    accountManager.addAccountExplicitly(account, null, Bundle())
}

if (!ContentResolver.getMasterSyncAutomatically()) {
    ContentResolver.setMasterSyncAutomatically(true)
}

if (ContentResolver.getIsSyncable(account, "CONTENT_PROVIDER") ≤ 0) {
    ContentResolver.setIsSyncable(account, "CONTENT_PROVIDER", 1)
}

ContentResolver.setSyncAutomatically(account, "CONTENT_PROVIDER", true)
ContentResolver.addPeriodicSync(account, "CONTENT_PROVIDER", Bundle(), 1)

ContentResolver.requestSync(account, "CONTENT_PROVIDER", bundleOf())
```

Create account

Service process

Представляет собой фоновые службы, которые выполняют задачи без непосредственного взаимодействия с пользователем



Sticky services

```
1 class StickService : Service() {
2     override fun onBind(intent: Intent?) = null
3 }
```

onStartCommand START_STICKY

Приложения, которые работают под управлением Android 8.0, не могут использовать startService() в фоновом режиме. Служба, запущенная приложением в фоновом режиме, должна стать службой переднего плана через несколько секунд после запуска.

Silent Music

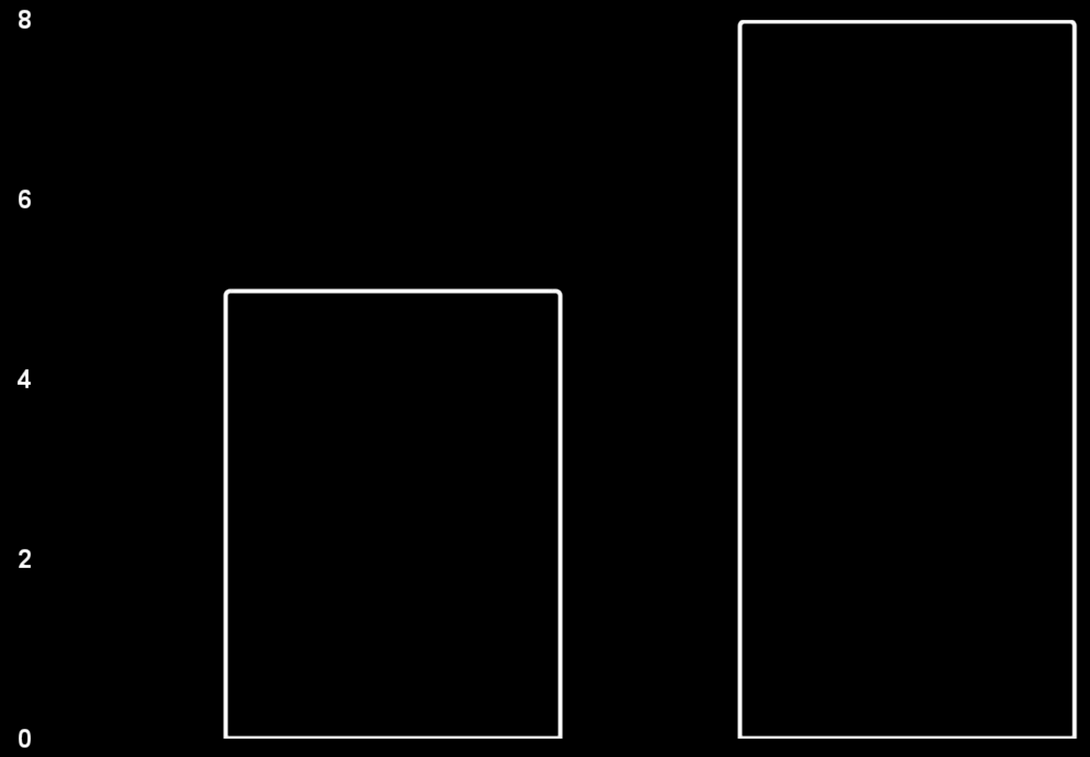
Silent music

```
1 class PlayerMusicService : Service() {
2     ...
3     override fun onCreate() {
4         mediaPlayer = MediaPlayer.create(applicationContext,R.raw.silent_music)
5         mediaPlayer?.isLooping = true
6     }
7
8     override fun onStartCommand(): Int {
9         Thread { startPlayMusic() }.start()
10        return START_STICKY
11    }
12
13    override fun onDestroy() {
14        val intent = Intent(applicationContext,PlayerMusicService::class.java)
15        stopPlayMusic()
16        startService(intent)
17    }
18 }
```

Background music service

START_STICKY будет работать для Android ниже 8.0. Для более новых версий сервис проживет чуть дольше, так как делает некую работу.

Silent music



Не выглядит как что-то стоящее

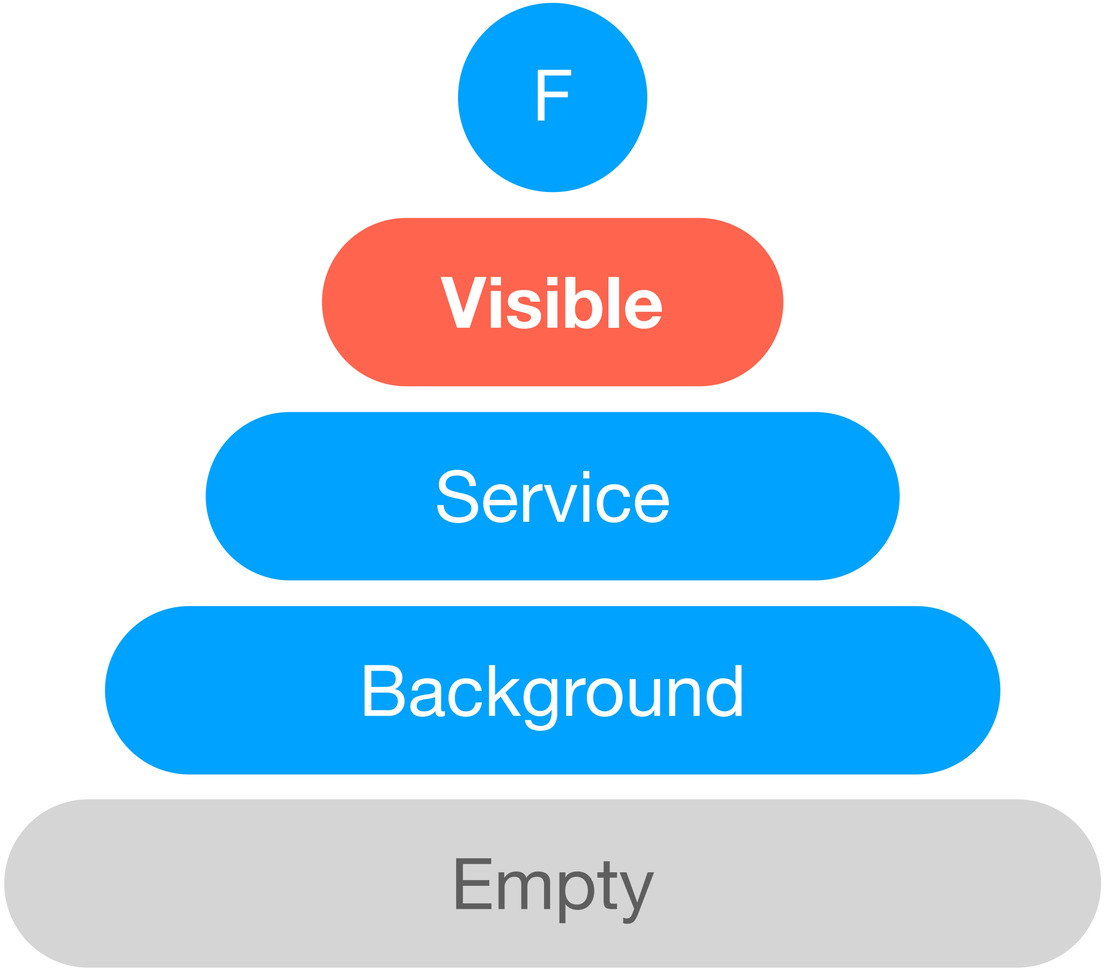
Doze mode

```
1 adb shell dumpsys battery unplug
2 adb shell dumpsys deviceidle force-idle
```

```
1 adb shell input keyevent KEYCODE_WAKEUP
```

Visible process

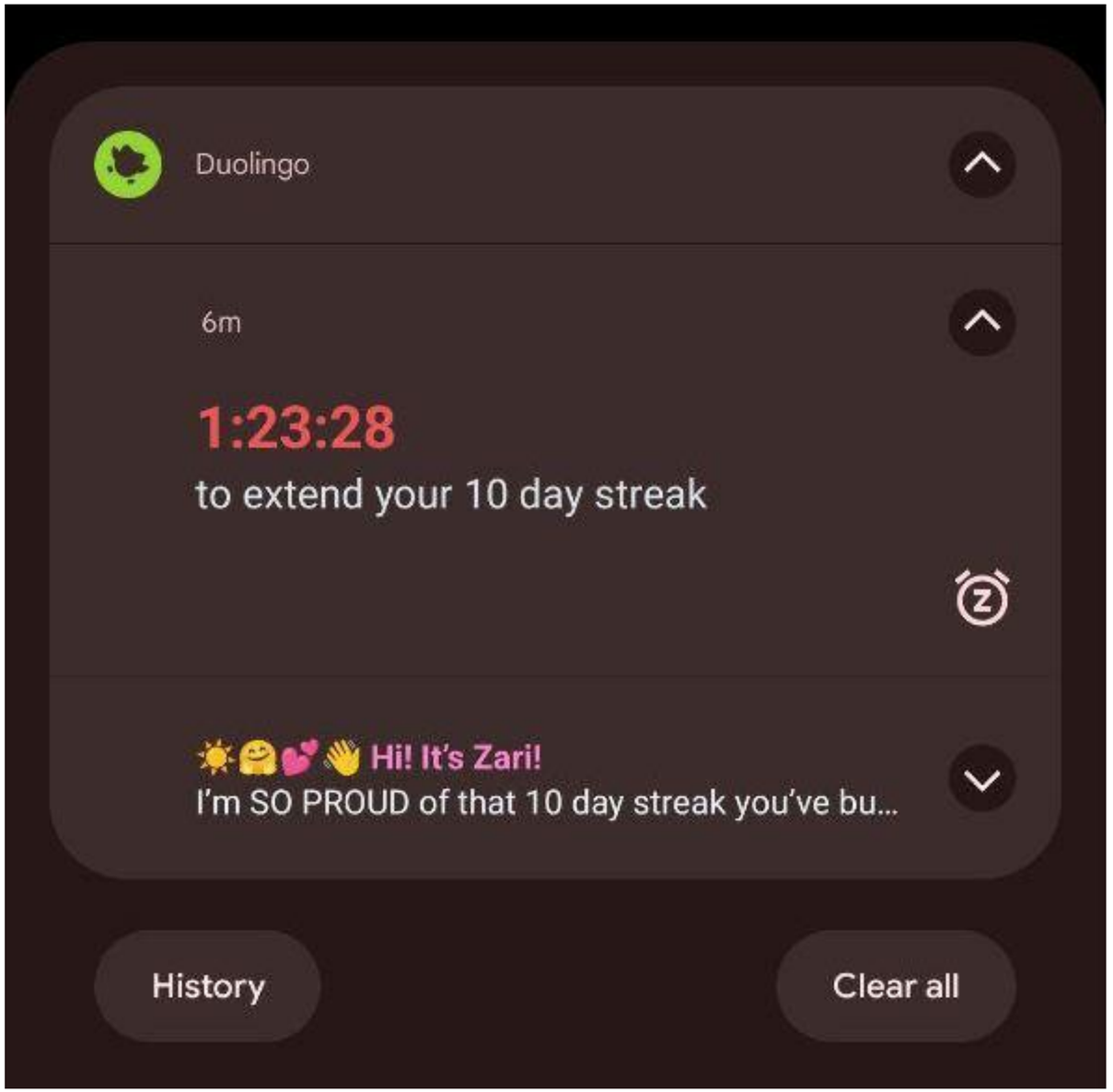
Представляет собой приложения, которые частично видны, но не используются активно



Foreground service

Когда может быть убит

- Extreme Low Memory
- Service Crash
- Doze Mode
- User (android ≥ 13)



Foreground service без уведомления

```kotlin
class ForegroundService : Service() {

    override fun onCreate() {
        super.onCreate()
        if (Build.VERSION.SDK_INT ⩾ Build.VERSION_CODES.O) {
            val builder = NotificationCompat.Builder(this, "service")
            val notification: Notification = builder
                .setOngoing(true)
                .setSmallIcon(R.mipmap.ic_launcher)
                ...
                .build()

            startForeground(10, notification)
        } else {
            startForeground(10, Notification())
            startService(Intent(this, CancelNotificationService::class.java))
        }
    }
}
```

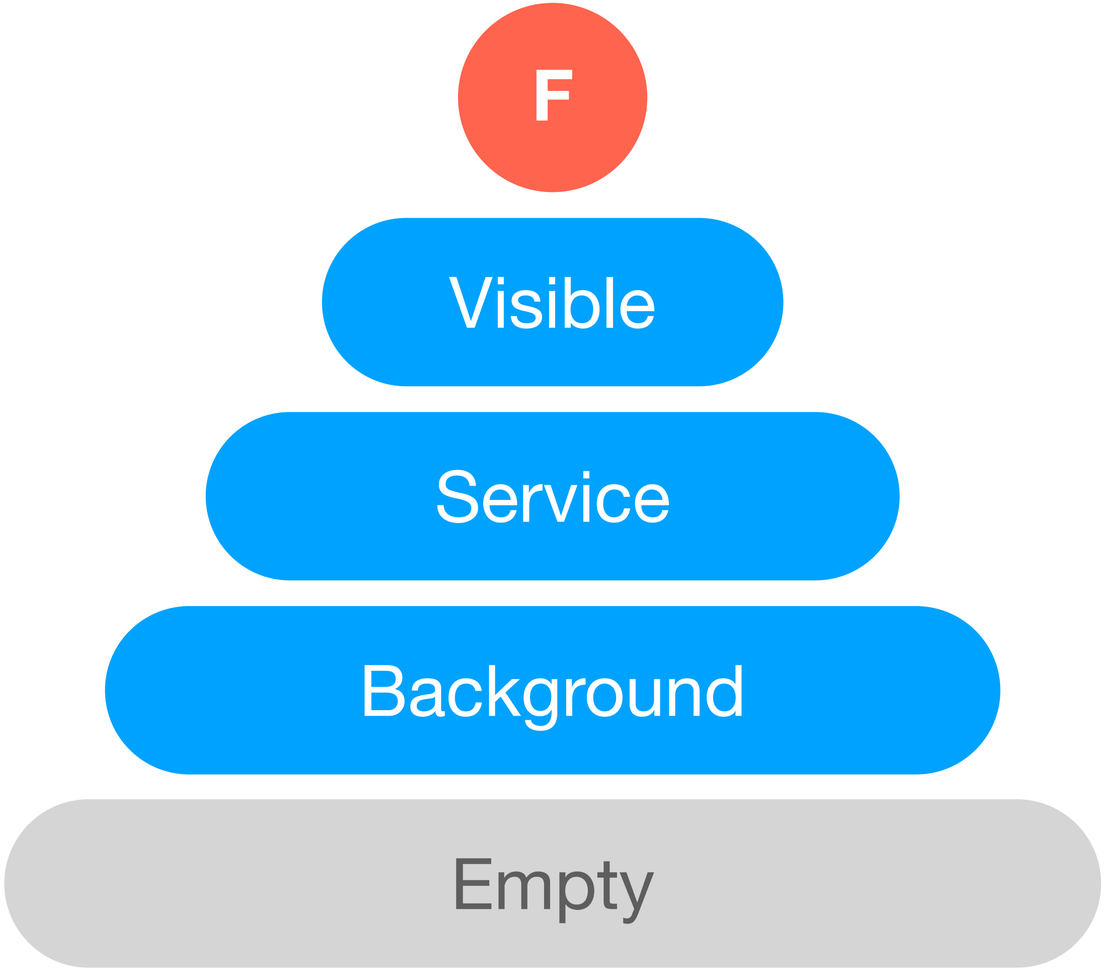
Запуск service
Android < 8.0

Foreground service без уведомления

```
1 class CancelNotificationService : Service() {
2
3     override fun onBind(intent: Intent) = null
4
5     override fun onCreate() {
6         super.onCreate()
7         startForeground(10, Notification())
8         stopSelf()
9     }
10 }
```

Foreground process

Представляет собой приложение, с которым пользователь взаимодействует в данный момент



Start activity

```
1 val intent = Intent(applicationContext, MainActivity::class.java)
2 intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
3 startActivity(intent)
```

А почему бы просто не запускать activity?

One pixel activity

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    window.setGravity(Gravity.LEFT or Gravity.TOP)
    window.attributes.apply {
        width = 1
        height = 1
        x = 0
        y = 0
    }
}
```

Activity размером в один пиксель

One pixel activity

```xml
<style name="OnePixelActivityTheme">
    <item name="android:windowBackground">@null</item>
    <item name="android:windowIsTranslucent">true</item>
</style>
```

```xml
<activity
    android:name=".OnePixelActivity"
    android:excludeFromRecents="true"
    android:taskAffinity="onepixel.afinity"
    android:theme="@style/OnePixelActivityTheme" />
```

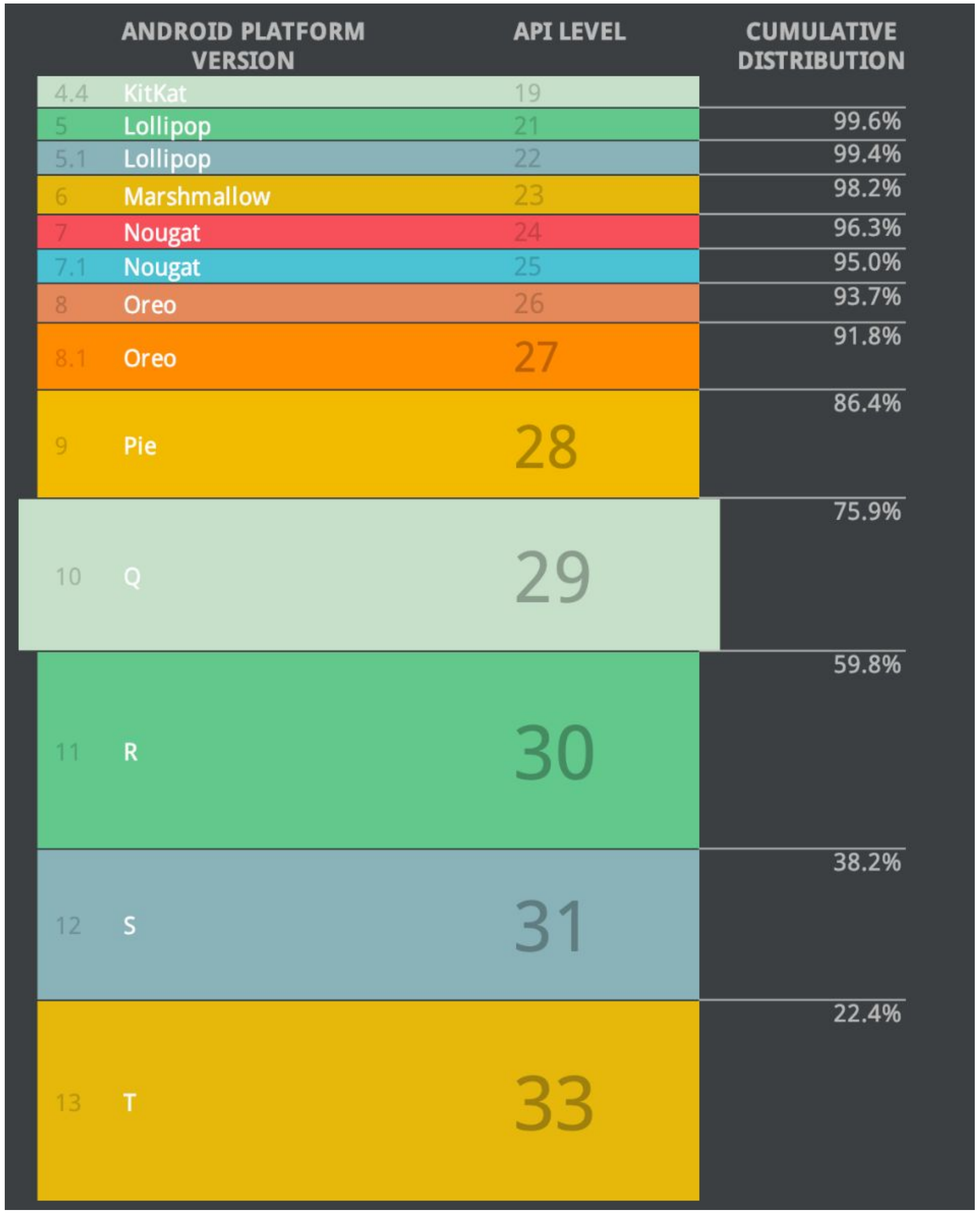
Theme + manifest

One pixel activity

BroadcastReceiver + activity

Чаще всего этот способ используется с событиями ACTION_SCREEN_ON/ACTION_SCREEN_OFF. Это позволяет создать activity, когда пользователь выключил экран и удалить при включении.

- Позволяет увеличить работу приложения в фоне.
- Работает на 24% устройств.
- Скоро выходит 15 Android.



Когда приложения могут запускать activity?

01

Приложение получает PendingIntent, отправленный из другого, видимого приложения.

02

Приложению предоставлено разрешение SYSTEM_ALERT_WINDOW пользователем.

03

Приложение имеет activity в back stack существующей задачи на экране недавних приложений.

04

Приложение имеет видимое window, например, activity на переднем плане.

Start activity

```
boolean shouldAbortBackgroundActivityStart( ... ) {
    final int callingAppId = UserHandle.getAppId(callingUid);
    if (callingUid == Process.ROOT_UID || callingAppId == Process.SYSTEM_UID || callingAppId == Process.NFC_UID) {
        return false;
    }
    if (callingUidHasAnyVisibleWindow || isCallingUidPersistentSystemProcess) {
        return false;
    }
    // don't abort if the callingUid has START_ACTIVITIES_FROM_BACKGROUND permission
    if (mService.checkPermission(START_ACTIVITIES_FROM_BACKGROUND, callingPid, callingUid) == PERMISSION_GRANTED) {
        return false;
    }
    // don't abort if the caller has the same uid as the recents component
    if (mSupervisor.mRecentTasks.isCallerRecents(callingUid)) {
        return false;
    }
    // don't abort if the callingUid is the device owner
    if (mService.isDeviceOwner(callingUid)) {
        return false;
    }
    // don't abort if the callingUid has SYSTEM_ALERT_WINDOW permission
    if (mService.hasSystemAlertWindowPermission(callingUid, callingPid, callingPackage)) {
        return false;
    }
    ...
}
```

Исходный код AOSP

Ограничения фоновых разрешений в AMS.

Start activity

```
1  boolean shouldAbortBackgroundActivityStart( ... ) {
2      final int callingAppId = UserHandle.getAppId(callingUid);
3      if (callingUid == Process.ROOT_UID || callingAppId == Process.SYSTEM_UID || callingAppId == Process.NFC_UID) {
4          return false;
5      }
6      if (callingUidHasAnyVisibleWindow || isCallingUidPersistentSystemProcess) {
7          return false;
8      }
9      // don't abort if the callingUid has START_ACTIVITIES_FROM_BACKGROUND permission
10     if (mService.checkPermission(START_ACTIVITIES_FROM_BACKGROUND, callingPid, callingUid) == PERMISSION_GRANTED) {
11         return false;
12     }
13     // don't abort if the caller has the same uid as the recents component
14     if (mSupervisor.mRecentTasks.isCallerRecents(callingUid)) {
15         return false;
16     }
17     // don't abort if the callingUid is the device owner
18     if (mService.isDeviceOwner(callingUid)) {
19         return false;
20     }
21     // don't abort if the callingUid has SYSTEM_ALERT_WINDOW permission
22     if (mService.hasSystemAlertWindowPermission(callingUid, callingPid, callingPackage)) {
23         return false;
24     }
25      ...
26 }
```

START_ACTIVITIES_FROM_BACKGROUND

Системное разрешение.

Start activity

```
boolean shouldAbortBackgroundActivityStart( ... ) {
    final int callingAppId = UserHandle.getAppId(callingUid);
    if (callingUid == Process.ROOT_UID || callingAppId == Process.SYSTEM_UID || callingAppId == Process.NFC_UID) {
        return false;
    }
    if (callingUidHasAnyVisibleWindow || isCallingUidPersistentSystemProcess) {
        return false;
    }
    // don't abort if the callingUid has START_ACTIVITIES_FROM_BACKGROUND permission
    if (mService.checkPermission(START_ACTIVITIES_FROM_BACKGROUND, callingPid, callingUid) == PERMISSION_GRANTED) {
        return false;
    }
    // don't abort if the caller has the same uid as the recents component
    if (mSupervisor.mRecentTasks.isCallerRecents(callingUid)) {
        return false;
    }
    // don't abort if the callingUid is the device owner
    if (mService.isDeviceOwner(callingUid)) {
        return false;
    }
    // don't abort if the callingUid has SYSTEM_ALERT_WINDOW permission
    if (mService.hasSystemAlertWindowPermission(callingUid, callingPid, callingPackage)) {
        return false;
    }
    ...
}
```

isCallerRecents

Недавние приложения.

Start activity

```
boolean shouldAbortBackgroundActivityStart( ... ) {
    final int callingAppId = UserHandle.getAppId(callingUid);
    if (callingUid == Process.ROOT_UID || callingAppId == Process.SYSTEM_UID || callingAppId == Process.NFC_UID) {
        return false;
    }
    if (callingUidHasAnyVisibleWindow || isCallingUidPersistentSystemProcess) {
        return false;
    }
    // don't abort if the callingUid has START_ACTIVITIES_FROM_BACKGROUND permission
    if (mService.checkPermission(START_ACTIVITIES_FROM_BACKGROUND, callingPid, callingUid) == PERMISSION_GRANTED) {
        return false;
    }
    // don't abort if the caller has the same uid as the recents component
    if (mSupervisor.mRecentTasks.isCallerRecents(callingUid)) {
        return false;
    }
    // don't abort if the callingUid is the device owner
    if (mService.isDeviceOwner(callingUid)) {
        return false;
    }
    // don't abort if the callingUid has SYSTEM_ALERT_WINDOW permission
    if (mService.hasSystemAlertWindowPermission(callingUid, callingPid, callingPackage)) {
        return false;
    }
    ...
}
```

isDeviceOwner

android:sharedUserId="android.uid.system"

Start activity

```
boolean shouldAbortBackgroundActivityStart( ... ) {
    final int callingAppId = UserHandle.getAppId(callingUid);
    if (callingUid == Process.ROOT_UID || callingAppId == Process.SYSTEM_UID || callingAppId == Process.NFC_UID) {
        return false;
    }
    if (callingUidHasAnyVisibleWindow || isCallingUidPersistentSystemProcess) {
        return false;
    }
    // don't abort if the callingUid has START_ACTIVITIES_FROM_BACKGROUND permission
    if (mService.checkPermission(START_ACTIVITIES_FROM_BACKGROUND, callingPid, callingUid) == PERMISSION_GRANTED) {
        return false;
    }
    // don't abort if the caller has the same uid as the recents component
    if (mSupervisor.mRecentTasks.isCallerRecents(callingUid)) {
        return false;
    }
    // don't abort if the callingUid is the device owner
    if (mService.isDeviceOwner(callingUid)) {
        return false;
    }
    // don't abort if the callingUid has SYSTEM_ALERT_WINDOW permission
    if (mService.hasSystemAlertWindowPermission(callingUid, callingPid, callingPackage)) {
        return false;
    }
    ...
}
```

SYSTEM_ALERT_WINDOW

Сложно получить от пользователя.

Рассмотрим цепочку для запуска activity из background

Start activity

```kotlin
fun moveToFront(context: Context) {
    val activityManager = context.getSystemService(Context.ACTIVITY_SERVICE)
    activityManager?.getRunningTasks(50)?.forEach { taskInfo ->
        if (taskInfo.topActivity?.packageName == context.packageName) {
            activityManager.moveTaskToFront(taskInfo.id, 0)
            return
        }
    }
}
```

Back stack activity

Start activity

```
1 fun startActivity(activity: Activity, intent: Intent) {
2     if (!isRunningForeground(activity)) {
3         if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
4             moveToFront(activity)
5             activity.startActivity(intent)
6             activity.moveTaskToBack(true)
7         } else {
8             sendNotificationFullScreen(activity)
9         }
10    } else {
11        activity.startActivity(intent)
12    }
13 }
```

Start activity

```
1  fun startActivity(activity: Activity, intent: Intent) {
2      if (!isRunningForeground(activity)) {
3          if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
4              moveToFront(activity)
5              activity.startActivity(intent)
6              activity.moveTaskToBack(true)
7          } else {
8              sendNotificationFullScreen(activity)
9          }
10     } else {
11         activity.startActivity(intent)
12     }
13 }
```

Start activity

```kotlin
fun startActivity(activity: Activity, intent: Intent) {
    if (!isRunningForeground(activity)) {
        if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
            moveToFront(activity)
            activity.startActivity(intent)
            activity.moveTaskToBack(true)
        } else {
            sendNotificationFullScreen(activity)
        }
    } else {
        activity.startActivity(intent)
    }
}
```

Start activity

```
1 fun startActivity(activity: Activity, intent: Intent) {
2     if (!isRunningForeground(activity)) {
3         if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
4             moveToFront(activity)
5             activity.startActivity(intent)
6             activity.moveTaskToBack(true)
7         } else {
8             sendNotificationFullScreen(activity)
9         }
10    } else {
11        activity.startActivity(intent)
12    }
13 }
```

Start activity

```
1 private fun getChannelNotificationQ(context: Context): Notification {
2     val fullScreenPendingIntent = PendingIntent.getActivity( ... )
3     val notificationBuilder = NotificationCompat.Builder(context, "full_screen")
4         .setOngoing(true)
5         .setSmallIcon(R.mipmap.ic_launcher)
6         .setSound(null)
7         ...
8         .setFullScreenIntent(fullScreenPendingIntent, true)
9
10    return notificationBuilder.build()
11 }
```

Full Screen notification

Start activity

```
1  private int startActivity( ... ) {
2      ...
3      paramInt1 = i;
4      if (i == 0) {
5          paramInt1 = i;
6          if (!ActivityTaskManagerServiceInjector.isAllowedStartActivity( ... ))
7              paramInt1 = 1;
8      }
9      ...
10 }
```

startActivity посредством декомпиляции ROM Xiaomi

Пользователь под ником Дядя Калеб (苍耳叔叔) нашел способ как запускать activity на Xiaomi.

Start activity

```
1 static boolean isAllowedStartActivity( ... , Intent paramIntent,  ... ) {
2       ...
3      if ((paramIntent.getMiuiFlags() & 0×2) ≠ 0) {
4        return true;
5      }
6       ...
7 }
```

MiuiFlags

Xiaomi добавила свои флаги к классу Intent.

Start activity

```
1 public class Intent implements Parcelable, Cloneable {
2     private int mMiuiFlags;
3
4     public Intent addMiuiFlags(int flags) {
5         this.mMiuiFlags |= flags;
6         return this;
7     }
8      ...
9 }
```

Intent

addMiuiFlags можно вызвать через рефлексию.

Start activity

```
1 private fun reflect(intent: Intent) {
2     try {
3         val method = intent.javaClass
4             .getDeclaredMethod("addMiuiFlags", Int::class.javaPrimitiveType)
5         method.invoke(intent, 0×2)
6     } catch (e: Exception) {
7         e.printStackTrace()
8     }
9 }
```

Reflection call


关注

88

39

57

小叶和小莫 LV.3 安卓高级开发工程师 @广州思璞游戏有限公司

目前的话，可以考虑提交一个分屏，然后就可以从后台直接弹出Activity了。

// // public void registerSplit(Context context) { // Log.d("test========", "registerSplit" + "---run ok"); // int i = Build.VERSION.SDK_INT; // if (i < 17) { // return; // } // try { // Object systemService = context.getSystemService(Context.DISPLAY_SERVICE); // if (systemService != null) { // DisplayManager displayManager= (DisplayManager) systemService; // VirtualDisplay virtual_display_other1 = displayManager.createVirtualDisplay("virtual_display_other", 1, 1, 480, null, 11); // Runnable runnable = new Runnable() { // @Override // public void run() { //// try { // new Presentation(context, virtual_display_other1.getDisplay()).show(); //// } catch (Exception unused) { //// } // } // }; // new Handler(Looper.getMainLooper()).post(runnable); // } // } catch (Throwable th) { // th.printStackTrace(); // } // }

收起

5月前 1 评论 ...

用户5853633144764

有没有尝试过，在小米 android P 、Q 上把延迟启动的时间调整为 10s 以上?

3年前 点赞 评论 ...

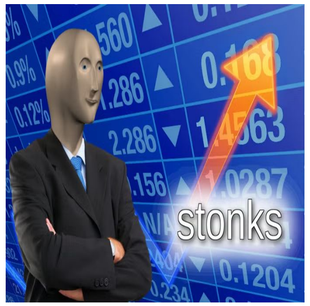
Start activity

```kotlin
fun registerSplit(context: Context) {
    val displayManager = context.getSystemService(Context.DISPLAY_SERVICE)

    if (displayManager ≠ null) {
        val virtualDisplay = displayManager.createVirtualDisplay(
            /* name = */ "virtual_display_name",
            /* width = */ 1,
            /* height = */ 1,
            /* densityDpi = */ 480,
            /* surface = */ null,
            /* flags = */ 11
        )
        Handler(Looper.getMainLooper()).post {
            Presentation(context, virtualDisplay.display).show()
        }
    }
}
```

Split screen

VIRTUAL_DISPLAY_FLAG_PUBLIC or VIRTUAL_DISPLAY_FLAG_PRESENTATION or VIRTUAL_DISPLAY_FLAG_OWN_CONTENT_ONLY = 11

Удалось воспроизвести



- [ ] Pixel 4a Android 13
- [ ] Samsung A32 Android 12
- [ ] Pocophone f1 Android 10

Не получилось воспроизвести

- [ ] Nothing phone Android 14
- [ ] HUAWEI Mate 50 Android 12
- [ ] Redmi Note 11 Android 13

Пережить force stop?

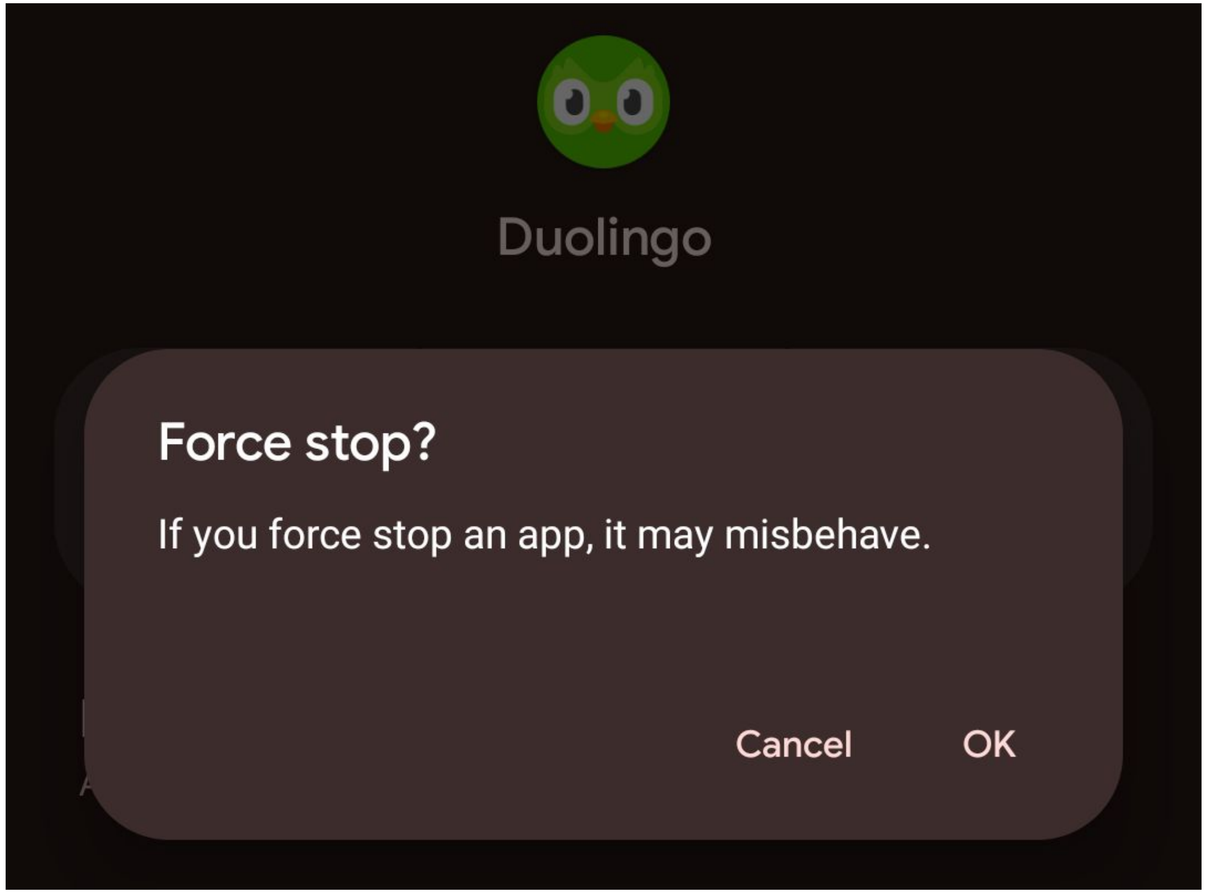
1 2 3

Пережить force stop?

Как работает force stop

Поскольку мы хотим, чтобы приложение жило вечно, нам нужно узнать, как оно умирает. Вообще говоря, у системы есть два способа убить процессы, оба из которых предоставляются через ActivityManagerService:

- killBackgroundProcesses,
- forceStopPackage.



Пережить force stop?

```
1  public void forceStopPackage(final String packageName, int userId) {
2      ...
3      for (int user : users) {
4          int pkgUid = pm.getPackageUid(packageName, MATCH_DEBUG_TRIAGED_MISSING, user);
5
6          if (pkgUid == -1) {
7              Slog.w(TAG, "Invalid packageName: " + packageName);
8              continue;
9          }
10         if (mUserController.isUserRunning(user, 0)) {
11             // Завершить процесс на основе UID и имени пакета
12             forceStopPackageLocked(packageName, pkgUid, "from pid " + callingPid);
13         }
14     }
15     ...
16 }
```

Method forceStopPackage

Система использует uid для принудительной остановки процесса, поэтому независимо от того, являетесь ли вы родным процессом или Java-процессом, принудительная остановка сработает для всех.

Пережить force stop?

```
1 final boolean forceStopPackageLocked(String packageName, int appId,
2         boolean callerWillRestart, boolean purgeCache, boolean doit,
3         boolean evenPersistent, boolean uninstalling, int userId, String reason) {
4     ...
5     boolean didSomething = mProcessList.killPackageProcessesLocked(packageName, appId, userId,
6             ProcessList.INVALID_ADJ, callerWillRestart, true /* allowRestart */, doit,
7             evenPersistent, true /* setRemoved */,
8             packageName == null ? ("stop user " + userId) : ("stop " + packageName));
9
10    didSomething |=
11            mAtmInternal.onForceStopPackage(packageName, doit, evenPersistent, userId);
12
13    // Очищаем services
14    // Очищаем broadcast receivers
15    // Очистка providers
16    // Очищаем другое
17
18    return didSomething;
19 }
```

Method forceStopPackageLocked

Пережить force stop?

```
1  void kill(String reason, boolean noisy) {
2      if (!killedByAm) {
3          ...
4          if (pid > 0) {
5              Process.killProcessQuiet(pid);
6              ProcessList.killProcessGroup(uid, pid);
7          } else {
8              pendingStart = false;
9          }
10         if (!mPersistent) {
11             killed = true;
12             killedByAm = true;
13         }
14         ...
15     }
16 }
```

Method kill

Пережить force stop?

```
1 void kill(String reason, boolean noisy) {
2     if (!killedByAm) {
3         ...
4         if (pid > 0) {
5             Process.killProcessQuiet(pid);
6             ProcessList.killProcessGroup(uid, pid);
7         } else {
8             pendingStart = false;
9         }
10        if (!mPersistent) {
11            killed = true;
12            killedByAm = true;
13        }
14        ...
15    }
16 }
```

Method kill

Название картинки на весь слайд

```
1 static int KillProcessGroup(uid_t uid, int initialPid, int signal, int retries) {
2     int retry = retries;
3     int processes;
4     while ((processes = DoKillProcessGroupOnce(cgroup, uid, initialPid, signal)) > 0) {
5         LOG(VERBOSE) << "Killed " << processes << " processes for processgroup " << initialPid;
6         if (retry > 0) {
7             std::this_thread::sleep_for(5ms);
8             --retry;
9         } else {
10            break;
11        }
12    }
13 }
```

40

Method killProcessGroup

Tiann and library Leoric

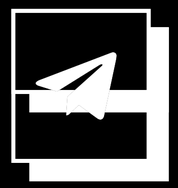
Dual-Process Activation на максималках

Это то же самое, что я говорил про Dual-Process Activation, но работает на блокировке файлов. Этот способ намного быстрее любых других, связанных с запуском foreground сервисов, и написан на C++.

- К сожалению, в 14 Android этот способ не будет работать.
- Это pet-project и сам автор говорит, что он далёк от идеала: «Пожалуйста, не думайте, что вы получите вечную жизнь, получив прямой доступ к коду».

Матвей Плохов

Android developer



@stringres

Мысль, которой хотите поделиться на прощание, мотивирующая цитата, призыв подписаться на вас

Хочу сказать спасибо маме, папе, Google, StackOverflow, ChatGpt, GitHub, csdn, juejin, Александру Соколинскому и Татьяне Рябовой.