Один из важнейших компонентов Android — **Service**.

Понятие Service

**Сервис** — это компонент, который может выполняться "в фоне", то есть независимо от того, находится ли сейчас приложение на экране.

Сервисы делятся на два типа:

* **Запущенный**. Сервис, который был запущен другим компонентом приложения. Такой сервис в общем случае работает независимо от других компонентов и, как правило, живёт дольше них. Служба может остановить сама себя, либо может быть уничтожена системой, если приложение будет закрыто окончательно (кроме одного исключения, о котором позже), либо если системе будет мало памяти.
* **Привязанный**. Сервис, к которому привязан другой компонент приложения (например, Activity). После привязки компонент может обмениваться данными с сервисом. Как только все компоненты отвяжутся от сервиса, он будет остановлен.

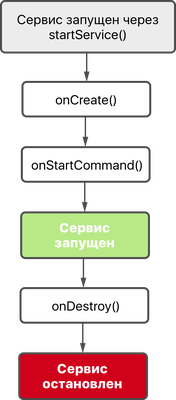
Оба этих типа могут использоваться одновременно.

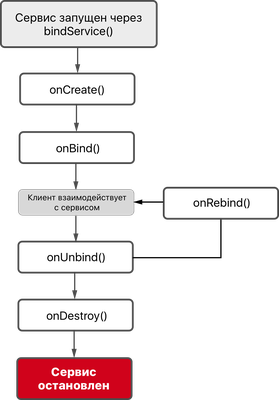
**Обратите внимание:** несмотря на то, что сервисы работают независимо от остальных компонентов, они выполняют всю работу на **основном** потоке. Поэтому если вам нужно выполнить "тяжелую" задачу, например, скачать что-то из сети, придётся создать фоновый поток.

Что ж, давайте попробуем создать сервис на практике и посмотрим, как это работает.

Жизненный цикл сервиса

Точно так же, как и любой другой компонент Android-приложения, Service имеет свой жизненный цикл. Он несколько отличается у запущенного и привязанного сервисов:

[](https://skillberg.com/media/uploads/2017/11/17/cf287795ffac477fb1a9cbf8841e9849.png)Жизненный цикл запущенного сервиса

[](https://skillberg.com/media/uploads/2017/11/17/b76a40b7d85840d1a83c9f0a158633ab.png)Жизненный цикл привязанного сервиса

* onCreate() — вызывается, когда сервис создаётся
* onStartCommand() — вызывается у запущенного сервиса, когда он был запущен другим компонентом. Может быть вызван несколько раз за время жизни сервиса.
* onDestroy() — вызывается, когда сервис уничтожается. Так же, как и в случае с Activity, вызов этого метода не гарантируется.
* onBind() — вызывается, когда компонент привязывается к сервису.
* onUnbind() — вызывается, когда компонент отвязывается от сервиса.
* onRebind() — вызывается, когда компонент привязывается к сервису после того, как был вызван onUnbind().

Запущенный сервис

Создайте новый класс, наследующийся от Service:

public class WeatherService extends Service {

@Nullable

@Override

public IBinder onBind(Intent intent) {

return null;

}

}

Как и любой компонент приложения, сервис должен быть зарегистрирован в манифесте внутри тега <application>:

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

Тег <service> поддерживает следующие атрибуты:

* android:description — описание для пользователя.
* android:directBootAware — поддерживает ли сервис **Direct Boot** (защищённый режим; вам вряд ли потребуется). По умолчанию false.
* android:enabled — включен ли сервис. По умолчанию true.
* android:exported — могут ли другие приложения взаимодействовать с сервисом.
* android:icon — иконка сервиса.
* android:isolatedProcess — должен ли сервис быть запущен в отдельном изолированном процессе. По умолчанию false.
* android:label — имя сервиса для пользователя.
* android:name — имя класса, который содержит реализацию сервиса.
* android:permission — разрешение, которое должно быть у приложения для запуска сервиса или привязки к нему.
* android:process — имя процесса, в котором будет работать сервис. По умолчанию используется процесс приложения.

Итак, мы создали сервис. Теперь переопределим метод onStartCommand():

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

Log.i(TAG, "onStartCommand");

return START\_STICKY;

}

Что за START\_STICKY?

В методе onStartCommand() мы должны вернуть константу, говорящую системе, что делать, если сервис будет убит. Варианты таковы:

* START\_STICKY — сервис будет перезапущен системой. При этом первоначальный Intent не будет передан, вместо него будет null.
* START\_REDELIVER\_INTENT — сервис будет перезапущен, при этом в onStartCommand() будет передан первоначальный Intent.
* START\_NOT\_STICKY — сервис не будет автоматически перезапущен.

**Обратите внимание:** если сервис будет убит из-за необработанного исключения, он будет пересоздан лишь несколько раз. Если он продолжит "падать", система больше не перезапустит его.

Теперь запустим сервис. В конце MainActivity.onCreate() добавьте следующие строки:

Intent intent = new Intent(this, WeatherService.class);

startService(intent);

Запустите приложение. В логах вы увидите:

11-16 02:08:11.673 4210-4210/? I/WeatherService: onStartCommand

Ок, оно работает.

Давайте добавим немного функциональности в наш сервис. Пускай он считает от 0 до переданного числа, увеличивая счётчик раз в секунду.

Добавьте следующие поля в WeatherService:

public static final String EXTRA\_COUNT\_TO = "count\_to";

private int countTo;

private int currentNumber = 0;

Добавьте метод:

private void startCount() {

final Handler handler = new Handler();

handler.post(new Runnable() {

@Override

public void run() {

currentNumber++;

Log.i(TAG, "Current number: " + currentNumber);

if (currentNumber < countTo) {

handler.postDelayed(this, 1000);

}

}

});

}

И измените метод onStartCommand() следующим образом:

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

countTo = intent.getIntExtra(EXTRA\_COUNT\_TO, 0);

startCount();

return START\_REDELIVER\_INTENT;

}

В MainActivity измените код запуска сервиса:

Intent intent = new Intent(this, WeatherService.class);

intent.putExtra(WeatherService.EXTRA\_COUNT\_TO, 5);

startService(intent);

Теперь запустите приложение и посмотрите в логи:

11-16 02:15:06.942 4768-4768/com.skillberg.weather2 I/WeatherService: Current number: 1

11-16 02:15:07.944 4768-4768/com.skillberg.weather2 I/WeatherService: Current number: 2

11-16 02:15:08.951 4768-4768/com.skillberg.weather2 I/WeatherService: Current number: 3

11-16 02:15:09.962 4768-4768/com.skillberg.weather2 I/WeatherService: Current number: 4

11-16 02:15:10.974 4768-4768/com.skillberg.weather2 I/WeatherService: Current number: 5

А теперь попробуйте убить приложение, посмотрите в логи ещё раз, и с сожалением наблюдайте за тем, как сервис начал считать заново — приложение было убито, и система перезапустила сервис :(

Как сделать так, чтобы сервис был максимально "живучим", и не умирал вместе с приложением? Нужно создать так называемый **Foreground Service**.

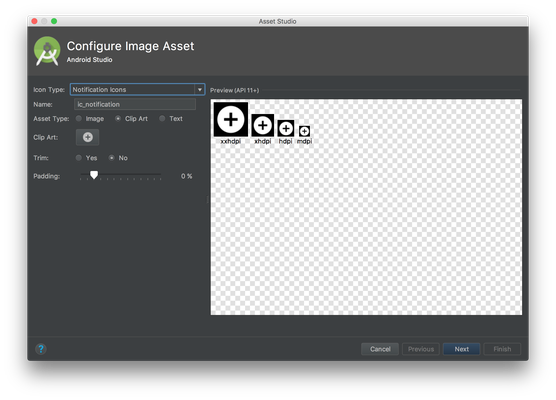
**Foreground Service** отличается от обычного тем, что у него минимум шансов быть убитым, а визуально он отличается тем, что создаёт уведомление.

Давайте попробуем создать уведомление, в котором к тому же будем показывать текущее значение currentNumber.

Уведомления

В первую очередь, попробуем создать обычное уведомление.

Для уведомления нам потребуется иконка. Создадим её так же, как и создавали иконки ранее, но в типе выберите **Notification Icons**:

[](https://skillberg.com/media/uploads/2017/11/17/3e19be18a9d94a73a506d9bbbf228b9e.png)Создаём иконку для уведомлений

Создайте в сервисе новый метод:

private void showNotification() {

}

В нём создадим Builder для уведомлений:

NotificationCompat.Builder builder = new NotificationCompat.Builder(this);

Зададим иконку, заголовок и текст уведомления:

builder.setSmallIcon(R.drawable.ic\_notification);

builder.setContentTitle("Content title");

builder.setContentText("Content text");

Теперь нужно передать Intent, который будет запускать MainActivity при клике на уведомления:

Intent mainIntent = new Intent(this, MainActivity.class);

Однако, есть проблема: по сути, этот Intent будет запущен другим приложением, поэтому мы не сможем передать его напрямую. Чтобы это заработало, нам нужен класс PendingIntent, он позволяет передать Intent в другое приложение:

PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,

mainIntent, PendingIntent.FLAG\_UPDATE\_CURRENT);

Теперь зададим этот PendingIntent:

builder.setContentIntent(pendingIntent);

И, наконец, создадим уведомление:

Notification notification = builder.build();

За показ уведомлений отвечает класс NotificationManager. Получим его:

NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION\_SERVICE);

Ну и покажем уведомление:

notificationManager.notify(0, notification);

Первый параметр — идентификатор уведомления, он должен быть уникален в рамках одного приложения. Поскольку у нас одно единственное уведомление, мы можем использовать константу.

Теперь вызовите метод showNotification() в onStartCommand():

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

countTo = intent.getIntExtra(EXTRA\_COUNT\_TO, 0);

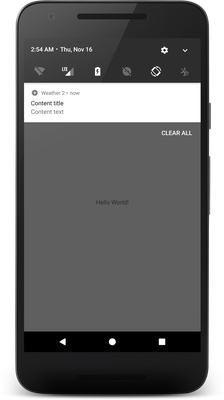
showNotification();

startCount();

return START\_REDELIVER\_INTENT;

}

Запустите приложение, и посмотрите, что получилось:

[](https://skillberg.com/media/uploads/2017/11/17/0d740b26cda44c5a91983c42e11ce8a8.png)Уведомление в Android

Foreground Service

Теперь мы имеем представление о том, как показывать уведомления. Пора создать **Foreground Service**!

Переименуйте метод showNotification() в createNotification() и измените его следующим образом:

private Notification createNotification() {

NotificationCompat.Builder builder = new NotificationCompat.Builder(this);

builder.setSmallIcon(R.drawable.ic\_notification);

builder.setContentTitle("Counter is running");

builder.setContentText("Current number: " + currentNumber);

builder.setOngoing(true);

Intent mainIntent = new Intent(this, MainActivity.class);

PendingIntent pendingIntent = PendingIntent.getActivity(

this,

0,

mainIntent,

PendingIntent.FLAG\_UPDATE\_CURRENT

);

builder.setContentIntent(pendingIntent);

return builder.build();

}

Как видите, мы добавили currentNumber в текст уведомления, и теперь мы не показываем уведомление через NotificationManager — просто вернём уведомление из этого метода.

Переопределите метод onCreate():

@Override

public void onCreate() {

super.onCreate();

Notification notification = createNotification();

startForeground(1, notification);

}

Когда мы вызываем startForeground(), система самостоятельно показывает уведомление. Первый параметр — идентификатор для уведомления. Идентификатор **ни в коем случае** не должен быть равен 0!

В метод startCount() добавим код, который будет обновлять уведомление:

private void startCount() {

final Handler handler = new Handler();

handler.post(new Runnable() {

@Override

public void run() {

currentNumber++;

Log.i(TAG, "Current number: " + currentNumber);

Notification notification = createNotification();

startForeground(1, notification);

if (currentNumber < countTo) {

handler.postDelayed(this, 1000);

}

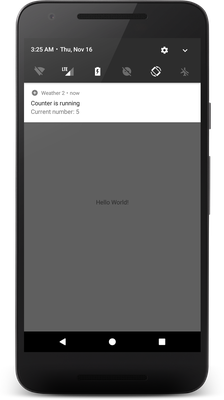
}

});

}

Мы просто пересоздаём уведомление и снова показываем его, используя тот же самый идентификатор уведомления.

Запустите приложение, и вы увидите, как счётчик обновляется!

[](https://skillberg.com/media/uploads/2017/11/17/d24d13a4da6143d8b4a8bb1e05f59778.png)Уведомление из Foreground Service

Кроме того, если вы убьёте приложение, ничего не изменится — сервис продолжит считать!

Осталось сделать кое-что ещё — останавливать сервис после того, как он закончит считать.

Вообще, за остановку сервиса отвечает метод stopSelf() — этого достаточно.

Если же вы хотите просто сделать **Foreground Service** обычным, достаточно вызвать метод stopForeground(), передав ему true в качестве параметра. Если ему передаётся true, то уведомление будет удалено, иначе — нет.

private void startCount() {

final Handler handler = new Handler();

handler.post(new Runnable() {

@Override

public void run() {

currentNumber++;

Log.i(TAG, "Current number: " + currentNumber);

Notification notification = createNotification();

startForeground(1, notification);

if (currentNumber < countTo) {

handler.postDelayed(this, 1000);

} else {

stopForeground(true);

}

}

});

}

Теперь, когда сервис закончит считать, он перестанет быть сервисом "переднего плана". Система же сама решит, когда его убить.

Привязанный сервис

А что, если мы хотим передать какой-то сигнал из Activity в сервис? Мы, конечно, можем использовать Intent, но как быть, если нам нужно сделать это в то время, когда сервис уже запущен? Нам поможет привязка к сервису!

Давайте модифицируем наш сервис таким образом, чтобы можно было выполнить привязку к нему, и передать ему какой-то сигнал по нажатию на кнопку в MainActivity.

После привязки к сервису в Activity передаётся специальный объект Binder, с помощью которого и происходит коммуникация. Мы будем передавать объект типа Message. Обработкой таких объектов занимается Handler.

Создадим в сервисе класс, наследующийся от Handler:

private class SignalHandler extends Handler {

@Override

public void handleMessage(Message msg) {

Log.i(TAG, "Got signal");

}

}

Для отправки и приёма сообщений нам также понадобится объект Messenger:

private final Messenger messenger = new Messenger(new SignalHandler());

Из метода onBind() вернём Binder из Messenger:

@Nullable

@Override

public IBinder onBind(Intent intent) {

return messenger.getBinder();

}

В MainActivity также создадим Messenger:

@Nullable

private Messenger messenger;

Через него мы будем отправлять сообщения в сервис.

Теперь в MainActivity создадим ServiceConnection. У него есть два метода:

* onServiceConnected() — вызывается, когда установлено соединение с сервисом.
* onServiceDisconnected() — вызывается, когда соединение с сервисом разорвано.

lang=java private final ServiceConnection serviceConnection = new ServiceConnection() { @Override public void onServiceConnected(ComponentName name, IBinder service) { Log.i(TAG, "Service connected!");

}

@Override

public void onServiceDisconnected(ComponentName name) {

Log.i(TAG, "Service disconnected!");

messenger = null;

}

};

Когда сервис подключается, создаём Messenger, когда отключается — обнуляем.

Мы будем подключаться к сервису в методе onStart(), а отключаться в onStop():

@Override

protected void onStart() {

super.onStart();

bindService(new Intent(this, WeatherService.class),

serviceConnection,

Context.BIND\_AUTO\_CREATE);

}

@Override

protected void onStop() {

if (messenger != null) {

unbindService(serviceConnection);

}

super.onStop();

}

Метод bindService() осуществляет привязку к сервису. Он принимает следующие параметры:

* intent для идентификации сервиса.
* serviceConnection, методы которого будут вызываться при привязке и отвязке от сервиса.
* flags — флаги для подключения.

Какими могут быть флаги:

* BIND\_AUTO\_CREATE — запустит сервис автоматически, если он ещё не был запущен.
* BIND\_DEBUG\_UNBIND — специальный флаг для отладки метода unbindService().
* BIND\_EXTERNAL\_SERVICE — флаг для привязки к сервису другого приложения.
* BIND\_IMPORTANT — говорит о том, что сервис очень важен, и при привязке станет сервисом **переднего плана** (**foreground**).
* BIND\_NOT\_FOREGROUND — сервис будет оставаться в **background** при привязке.
* BIND\_WAIVE\_PRIORITY — привязка не будет влиять на приоритет по CPU/памяти (т.е. не будет делать сервис foreground или background).

Теперь измените лэйаут MainActivity на такой:

<?xml version="1.0" encoding="utf-8"?>

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.skillberg.weather2.MainActivity">

<Button

android:id="@+id/send\_btn"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:text="@string/send\_to\_service" />

</FrameLayout>

И в конце MainActivity.onCreate() добавьте следующий код:

findViewById(R.id.send\_btn).setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

if (messenger != null) {

Message message = new Message();

try {

messenger.send(message);

} catch (RemoteException e) {

Log.e(TAG, "Failed to send message: " + e.getMessage());

}

}

}

});

Запустите приложение и нажмите на кнопку **Send to Service**:

11-16 22:54:05.783 4117-4117/com.skillberg.weather2 I/WeatherService: Got signal

Отлично, сервис получил сигнал!

На этом всё, а в следующем уроке мы начнём получать погоду в нашем сервисе!

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/WeatherAndroid2/tree/lesson-34)