# Службы Андроид

# Содержание

- Определения
- Многопоточность и службы
- Соотношения методов службы
- Объявление службы в манифесте
- Методы обратного вызова
- Выполнение службы на переднем плане
- Управление жизненным циклом службы

# Службы определение

Service является компонентом приложения, который может **выполнять длительные операции в фоновом режиме и не содержит пользовательского интерфейса**.

Другой компонент приложения может запустить службу, которая продолжит работу в фоновом режиме даже в том случае, когда пользователь перейдет в другое приложение.

Кроме того, компонент может привязаться к службе для взаимодействия с ней и даже выполнять межпроцессное взаимодействие (IPC).

Служба может **обрабатывать сетевые транзакции, воспроизводить музыку, выполнять ввод-вывод** файла или взаимодействовать с поставщиком контента.

Фактически служба может принимать **две формы.**

# 1. Запущенная

Служба является «запущенной», когда компонент приложения (например, операция) запускает ее вызовом startService().

**После запуска** служба может работать **в фоновом режиме** в течение неограниченного времени, **даже если уничтожен компонент, который ее запустил**.

Обычно запущенная служба выполняет одну операцию и не возвращает результатов вызывающему компоненту.

Например, она может загружать или выгружать файл по сети.

Когда операция выполнена, служба **должна остановиться самостоятельно.**

# 2. Привязанная

Служба является «привязанной», когда компонент приложения привязывается к ней вызовом bindService().

Привязанная служба предлагает интерфейс клиент-сервер, который позволяет компонентам взаимодействовать со службой, отправлять запросы, получать результаты и даже делать это между разными процессами посредством межпроцессного взаимодействия (IPC).

Привязанная служба работает только пока к ней привязан другой компонент приложения.

К службе могут быть привязаны несколько компонентов одновременно, но **когда все они отменяют привязку, служба уничтожается.**

# Служба может работать обеими способами

— она может быть запущенной (и работать в течение неограниченного времени) и допускать привязку.

Это зависит от реализации пары методов обратного вызова: onStartCommand() позволяет компонентам запускать службу, а onBind() позволяет выполнять привязку.

Можно объявить закрытую службу в файле манифеста и заблокировать доступ к ней из других приложений

# Многопоточность и службы

Служба работает в основном потоке ведущего процесса — служба **не** создает своего потока и **не** выполняется в отдельном процессе (если вы не указали иное).

Это означает, что если ваша служба собирается выполнять любую работу с высокой нагрузкой ЦП или блокирующие операции (например, воспроизведение MP3 или сетевые операции), вы должны создать в службе новый поток для выполнения этой работы.

Используя отдельный поток, вы снижаете риск возникновения ошибок «Приложение не отвечает», и основной поток приложения может отрабатывать взаимодействие пользователя с вашими операциями

Если вы используете службу, она выполняется в основном потоке вашего приложения по умолчанию, поэтому вы должны создать **новый поток в службе, если она выполняет интенсивные или блокирующие операции.**

# Что лучше использовать — службу или поток?

Служба — это просто **компонент,** который может **выполняться в фоновом режиме**, даже когда пользователь не взаимодействует с приложением.

Следовательно, вы должны создавать службу только в том случае, если вам нужно именно это.

Если вам требуется выполнить работу за пределами основного потока, но только в то время, когда пользователь взаимодействует с приложением, то вам, вероятно, следует создать новый поток, а не службу.

Например, если вы хотите воспроизводить определенную музыку, но только во время работы операции, вы можете создать поток в onCreate(), запустить его выполнение в методе onStart(), а затем остановить его в методе onStop().

Также рассмотрите возможность использования класса AsyncTask или HandlerThread вместо обычного класса Thread

# Создать службу

Чтобы создать службу, необходимо создать подкласс класса Service (или одного из существующих его подклассов).

В реализации необходимо переопределить некоторые методы обратного вызова, которые обрабатывают ключевые моменты жизненного цикла службы и при необходимости предоставляют механизм привязывания компонентов.

Наиболее важные методы обратного вызова, которые необходимо переопределить:

# onStartCommand()

Система вызывает этот метод, когда другой компонент, например, операция, запрашивает запуск этой службы, вызывая startService().

После выполнения этого метода служба запускается и может в течение неограниченного времени **работать в фоновом режиме**.

Если вы реализуете такой метод, вы обязаны остановить службу посредством вызова stopSelf() или stopService(). (Если требуется только обеспечить привязку, реализовывать этот метод не обязательно).

# onBind()

Система вызывает этот метод, когда другой компонент хочет выполнить привязку к службе (например, для выполнения удаленного вызова процедуры) путем вызова bindService().

В реализации этого метода вы должны обеспечить интерфейс, который клиенты используют для взаимодействия со службой, возвращая IBinder.

Всегда необходимо реализовывать этот метод, но если вы не хотите разрешать привязку, необходимо возвращать значение null.

# onCreate()&onDestroy()

onCreate()

Система вызывает этот метод при первом создании службы для выполнения однократных процедур настройки (перед вызовом onStartCommand() или onBind()).

Если служба уже запущена, этот метод не вызывается.

onDestroy()

Система вызывает этот метод, когда служба более не используется и выполняется ее уничтожение.

Ваша служба должна реализовать это для очистки ресурсов, таких как потоки, зарегистрированные приемники, ресиверы и т. д.

Это последний вызов, который получает служба.

# Соотношения методов службы

Если компонент запускает службу посредством вызова startService() (что приводит к вызову onStartCommand()), то служба продолжает работу, пока она не остановится самостоятельно с помощью stopSelf() или другой компонент не остановит ее посредством вызова stopService().

Если компонент вызывает bindService() для создания службы (и onStartCommand() *не* вызывается), то служба работает, пока к ней привязан компонент. Как только выполняется отмена привязки службы ко всем клиентам, система уничтожает службу.

# Уничтожение службы

Система Android будет принудительно останавливать службу только в том случае, когда не хватает памяти, и необходимо восстановить системные для операции, которая отображается на переднем плане.

Если служба привязана к операции, которая отображается на переднем плане, менее вероятно, что она будет уничтожена, и если служба объявлена для выполнения на переднем плане (как обсуждалось выше), она почти никогда не будет уничтожаться.

В противном случае, если служба была запущена и является длительной, система со временем будет опускать ее положение в списке фоновых задач, и служба станет очень чувствительной к уничтожению — если ваша служба запущена, вы должны предусмотреть изящную обработку ее перезапуска системой.

Если система уничтожает вашу службу, она перезапускает ее, как только снова появляется доступ к ресурсам (хотя это также зависит от значения, возвращаемого методом onStartCommand(), как обсуждается ниже).

# Объявление службы в манифесте

- Все службы, как и операции (и другие компоненты), должны быть объявлены в файле манифеста вашего приложения.
- Чтобы объявить службу, добавьте элемент <service> , в качестве дочернегоэлемента <application> . Например:
- <manifest ... >

```
  ...
  <application ... >
     <service android:name=".ExampleService" />
     ...
  </application>
</manifest>
```

# Синтаксис объявления service в манифесте

```
<service android:description="string resource"
         android:directBootAware=["true" | "false"]
         android:enabled=["true" | "false"]
         android:exported=["true" | "false"]
         android:foregroundServiceType=["connectedDevice" | "dataSync" |
                                        "location" | "mediaPlayback" | "mediaProjection" |
                                        "phoneCall"]
         android:icon="drawable resource"
         android:isolatedProcess=["true" | "false"]
         android:label="string resource"
         android:name="string"
         android:permission="string"
         android:process="string" >
    . . .
</service>
```

Имеются другие атрибуты, которые можно включить в элемент <service> для задания свойств, например, необходимых для запуска разрешений, и процесса, в котором должна выполняться служба.

Атрибут android:name является единственным обязательным атрибутом — он указывает имя класса для службы.

# Обеспечение безопасности

Для обеспечения безопасности приложения **всегда используйте явное намерение при запуске или привязке Service** и не объявляйте фильтров намерений для службы.

Если вам важно допустить некоторую неопределенность в отношении того, какая служба запускается, вы можете предоставить фильтры намерений для ваших служб и исключить имя компонента из Intent, но затем вы должны установить пакет для намерения с помощью setPackage(), который обеспечивает достаточное устранение неоднозначности для целевой службы.

Дополнительно **можно обеспечить доступность вашей службы только для вашего приложения**, включив атрибут android:exported и установив для него значение "false". Это не позволяет другим приложениям запускать вашу службу даже при использовании явного намерения.

# Создание запущенной службы

Запущенная служба — это служба, которую запускает другой компонент вызовом startService(), что приводит к вызову метода onStartCommand() службы.

При запуске служба обладает сроком жизни, не зависящим от запустившего ее компонента, и может работать в фоновом режиме в течение неограниченного времени, даже если уничтожен компонент, который ее запустил.

Поэтому после выполнения своей работы служба должна остановиться самостоятельно посредством вызова метода stopSelf(), либо ее может остановить другой компонент посредством вызова методаstopService().

# Activity может запустить службу методом startService()

Компонент приложения, например, операция, может запустить службу, вызвав метод startService() и передав объект Intent, который указывает службу и любые данные, которые служба должна использовать.

Служба получает этот объект Intent в методе onStartCommand().

**Пример**

Предположим, что операции требуется сохранить некоторые данные в сетевой базе данных.

Операция может запустить службу и предоставить ей данные для сохранения, передав намерение в метод startService().

Служба получает намерение в методе onStartCommand(), подключается к Интернету и выполняет транзакцию с базой данных.

Когда транзакция выполнена, служба останавливается самостоятельно и уничтожается.

Имеется два класса, которые вы можете наследовать для создания запущенной службы:

- 1. Service

Это базовый класс для всех служб.

Когда вы наследуете этот класс, важно создать новый поток, в котором будет выполняться вся работа службы, поскольку по умолчанию служба использует основной поток вашего приложения, что может замедлить любую операцию, которую выполняет ваше приложение.

# 2. IntentService

- Это подкласс класса Service, который использует рабочий поток для обработки всех запросов запуска поочередно.
- Это оптимальный вариант, если вам не требуется, чтобы ваша служба обрабатывала несколько запросов одновременно.
- Достаточно реализовать метод onHandleIntent(), который получает намерение для каждого запроса запуска, позволяя выполнять фоновую работу.

# Наследование класса IntentService

Так как большинству запущенных приложений не требуется обрабатывать несколько запросов одновременно, (что может быть действительно опасным сценарием), вероятно будет лучше, если вы реализуете свою службу с помощью класса IntentService.

Класс IntentService делает следующее:

- Создает рабочий поток по умолчанию, который выполняет все намерения, доставленные в метод onStartCommand(), отдельно от основного потока вашего приложения.
- Создает рабочую очередь, которая передает намерения по одному в вашу реализацию метода onHandleIntent(), поэтому вы не должны беспокоиться относительно многопоточности.
- Останавливает службу после обработки всех запросов запуска, поэтому вам никогда не требуется вызывать stopSelf().
- Предоставляет реализацию метода onBind() по умолчанию, которая возвращает null.
- Предоставляет реализацию метода onStartCommand() по умолчанию, которая отправляет намерение в рабочую очередь и затем в вашу реализацию onHandleIntent().

# Пример реализации класса IntentService:

Достаточно реализовать метод onHandleIntent() для выполнения работы, предоставленной клиентом. (Хотя, кроме того, вы должны предоставить маленький конструктор для службы).

```
public class HelloIntentService extends IntentService {
    /**
     * A constructor is required, and must call the super IntentService(String)
     * constructor with a name for the worker thread.
     */
    public HelloIntentService() {
        super("HelloIntentService");  }
    /**
     * The IntentService calls this method from the default worker thread with
     * the intent that started the service. When this method returns, IntentService
     * stops the service, as appropriate.
     */
    @Override
    protected void onHandleIntent(Intent intent) {
        // Normally we would do some work here, like download a file.
        // For our sample, we just sleep for 5 seconds.
        long endTime = System.currentTimeMillis() + 5*1000;
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(endTime - System.currentTimeMillis());            } catch (Exception e) {            }        }    }
}}
```

Это все, что нужно: конструктор и реализация класса onHandleIntent().

# Методы обратного вызова

Если вы решили переопределить также и другие методы обратного вызова, такие как onCreate(), onStartCommand() или onDestroy(), обязательно вызовите реализацию суперкласса, чтобы класс IntentService мог правильно обрабатывать жизненный цикл рабочего потока.

Например, метод onStartCommand() должен возвращать реализацию по умолчанию (которая доставляет намерение в onHandleIntent()):

```
@Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();
        return super.onStartCommand(intent,flags,startId);
    }
```

Помимо onHandleIntent(), единственный метод, из которого вам не требуется вызывать суперкласс, это метод onBind() (но его нужно реализовывать только в случае, если ваша служба допускает привязку).

# Наследование класса Service

Использование класса IntentService значительно упрощает реализацию запущенной службы.

Однако, если необходимо, чтобы ваша служба поддерживала многопоточность (вместо обработки запросов запуска через рабочую очередь), можно наследовать класс Service для обработки каждого намерения.

В качестве примера приведена следующая реализация класса Service, которая выполняет ту же работу, как и пример выше, использующий класс IntentService.

То есть для каждого запроса запуска он использует рабочий поток для выполнения задания и обрабатывает запросы по одному.

# public class HelloService extends Service

```
public class HelloService extends Service {
        private Looper mServiceLooper;
        private ServiceHandler mServiceHandler;

        // Handler that receives messages from the thread

        private final class ServiceHandler extends Handler {
           public ServiceHandler(Looper looper) {
              super(looper);      }

           @Override
           public void handleMessage(Message msg) {
              // Normally we would do some work here, like download a file.
              // For our sample, we just sleep for 5 seconds.
              long endTime = System.currentTimeMillis() + 5*1000;
              while (System.currentTimeMillis() < endTime) {
                 synchronized (this) {
                    try {
                       wait(endTime - System.currentTimeMillis());
                    } catch (Exception e) {               }           }        }
              // Stop the service using the startId, so that we don't stop
              // the service in the middle of handling another job

              stopSelf(msg.arg1);      }  }
```

public final class Looper extends Object

java.lang.Object

# public void onCreate()

```
@Override
    public void onCreate() {
      // Start up the thread running the service.  Note that we create a
      // separate thread because the service normally runs in the process's
      // main thread, which we don't want to block.  We also make it
      // background priority so CPU-intensive work will not disrupt our UI.

      HandlerThread thread = new HandlerThread("ServiceStartArguments",
            Process.THREAD_PRIORITY_BACKGROUND);
      thread.start();

      // Get the HandlerThread's Looper and use it for our Handler

      mServiceLooper = thread.getLooper();
      mServiceHandler = new ServiceHandler(mServiceLooper);
    }
```

# onStartCommand ……

```
@Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();

        // For each start request, send a message to start a job and deliver the
        // start ID so we know which request we're stopping when we finish the job
        Message msg = mServiceHandler.obtainMessage();
        msg.arg1 = startId;
        mServiceHandler.sendMessage(msg);

        // If we get killed, after returning from here, restart
        return START_STICKY;  }

    @Override
    public IBinder onBind(Intent intent) {
        // We don't provide binding, so return null
        return null;  }

    @Override
    public void onDestroy() {
      Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();  }}
```

Всплывающее уведомление (Toast Notification)

# Комментарий

Однако, так как вы обрабатываете каждый вызов onStartCommand() самостоятельно, вы можете выполнять несколько запросов одновременно.

Данный код выполняет не совсем эту работу, но при необходимости вы можете создавать новые потоки для каждого запроса и сразу запускать их (а не ожидать завершения предыдущего запроса).

Обратите внимание, что метод onStartCommand() должен возвращать целое число. Это целое число описывает, как система должна продолжать выполнение службы в случае, когда система уничтожила ее (как описано выше, реализация по умолчанию для класса IntentService обрабатывает эту ситуацию, хотя вы изменить ход реализации). Значение, возвращаемое методом onStartCommand(), должно быть одной из следующих констант:

START_NOT_STICKY START_STICKY START_REDELIVER_INTENT

# Запуск службы

Можно запустить службу из операции или другого компонента приложения, передав объект Intent(указывающий службу, которую требуется запустить) в startService().

Система Android вызывает метод onStartCommand() службы и передает ей Intent. (Ни в коем случае не следует вызывать метод onStartCommand() напрямую).

Например, операция может запустить службу из примера в предыдущем разделе (HelloSevice), используя явное намерение с помощью startService():

```
Intent intent = new Intent(this, HelloService.class);
    startService(intent);
```

Метод startService() возвращается немедленно, и система Android вызывает метод службы onStartCommand(). Если служба еще не выполняется, система сначала вызывает onCreate(), а затем onStartCommand().

# Остановка службы

Запущенная служба должна управлять своим жизненным циклом.

То есть, система не останавливает и не уничтожает службу, если не требуется восстановить память системы, и служба продолжает работу после возвращения из метода onStartCommand().

Поэтому служба должна останавливаться самостоятельно посредством вызова метода stopSelf(), либо другой компонент может остановить ее посредством вызова метода stopService().

Получив запрос на остановку посредством stopSelf() или stopService(), система как можно скорее уничтожает службу .

# Создание привязанной службы

Привязанная служба — это служба, которая допускает привязку к ней компонентов приложения посредством вызова bindService() для создания долговременного соединения (и обычно не позволяет компонентам *запускать* ее посредством вызова startService()).

Вы должны создать привязанную службу, когда вы хотите взаимодействовать со службой из операций и других компонентов вашего приложения или показывать некоторые функции вашего приложения другим приложениям посредством межпроцессного взаимодействия (IPC).

Чтобы создать привязанную службу, необходимо реализовать метод обратного вызова onBind() для возвращения объекта IBinder, который определяет интерфейс взаимодействия со службой.

После этого другие компоненты приложения могут вызвать метод bindService() для извлечения интерфейса и начать вызывать методы службы.

Служба существует только для обслуживания привязанного к ней компонента приложения, поэтому, когда нет компонентов, привязанных к службе, система уничтожает ее (вам *не* требуется останавливать привязанную службу, как это требуется для службы, запущенной посредством onStartCommand()).

# Отправка уведомлений пользователю

После запуска служба может уведомлять пользователя о событиях, используя Всплывающие уведомления или Уведомления в строке состояния.

Всплывающее уведомление — это сообщение, кратковременно появляющееся на поверхности текущего окна, тогда как уведомление в строке состояния — это значок в строке состояния с сообщением, который пользователь может выбрать, чтобы выполнить действие (такое как запуск операции).

Обычно уведомление в строке состояния является самым удобным решением, когда завершается какая-то фоновая работа (например, завершена загрузка файла), и пользователь может действовать.

Когда пользователь выбирает уведомление в расширенном виде, уведомление может запустить операцию (например, для просмотра загруженного файла).

# Запуск службы на переднем плане

Служба переднего плана — это служба, о которой пользователь активно осведомлен, и поэтому она не является кандидатом для удаления системой в случае нехватки памяти.

Служба переднего плана должна выводить уведомление в строку состояния, которая находится под заголовком «Постоянные». Это означает, что уведомление не может быть удалено, пока служба не будет остановлена или удалена с переднего плана.

Например, музыкальный проигрыватель, который воспроизводит музыку из службы, должен быть настроен на работу на переднем плане, так как пользователь точно знает о его работе. Уведомление в строке состояния может показывать текущее произведение и позволять пользователю запускать операцию для взаимодействия с музыкальным проигрывателем.

# Выполнение службы на переднем плане

Для запроса на выполнение вашей службы на переднем плане вызовите метод startForeground(). Этот метод имеет два параметра: целое число, которое однозначно идентифицирует уведомление и объект Notification для строки состояния. Например:

```
Notification notification = new Notification(R.drawable.icon,
    getText(R.string.ticker_text), System.currentTimeMillis());
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
notification.setLatestEventInfo(this, getText(R.string.notification_title),
        getText(R.string.notification_message), pendingIntent);
startForeground(ONGOING_NOTIFICATION_ID, notification);
```

- Чтобы удалить службу с переднего плана, вызовите stopForeground(). Этот метод содержит логическое значение, указывающее, следует ли также удалять уведомление в строке состояния. Этот метод *не*останавливает службу. Однако, если вы останавливаете службу, работающую на переднем плане, уведомление также удаляется.

# Управление жизненным циклом службы

Жизненный цикл службы от создания до уничтожения может следовать двум разным путям:

Запущенная служба

Служба создается, когда другой компонент вызывает метод startService(). Затем служба работает в течение неограниченного времени и должна остановиться самостоятельно посредством вызова метода stopSelf(). Другой компонент также может остановить службу посредством вызова метода stopService(). Когда служба останавливается, система уничтожает ее.

Привязанная служба

Служба создается, когда другой компонент (клиент) вызывает метод bindService(). Затем клиент взаимодействует со службой через интерфейс IBinder. Клиент может закрыть соединение посредством вызова метода unbindService(). К одной службе могут быть привязано несколько клиентов, и когда все они отменяют привязку, система уничтожает службу. (Служба *не* должна останавливаться самостоятельно.)

# Реализация обратных вызовов жизненного цикла

Подобно операции, служба содержит методы обратного вызова жизненного цикла, которые можно реализовать для контроля изменений состояния службы и выполнения работы в соответствующие моменты времени.

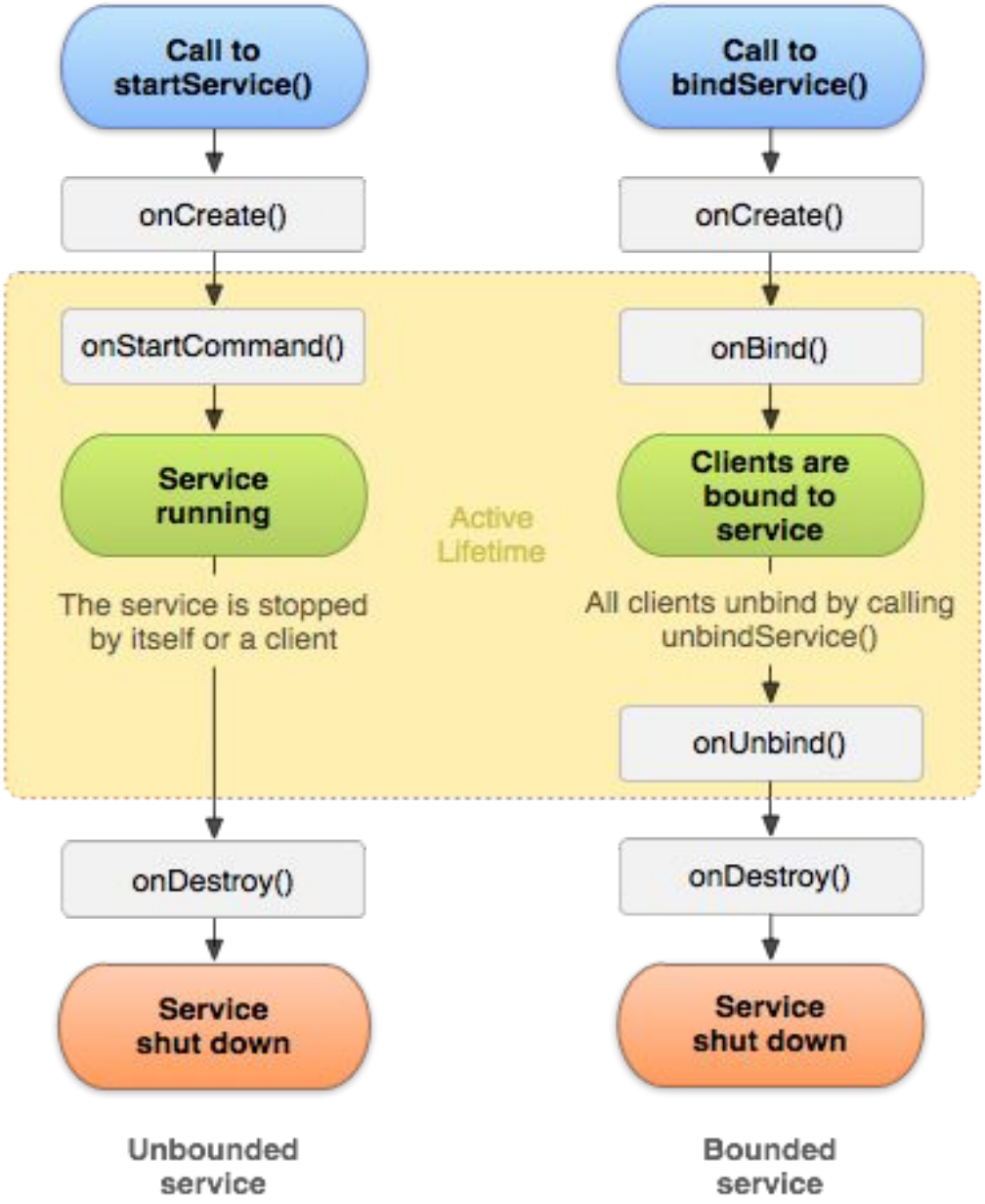
Указанная ниже базовая служба показывает каждый из методов жизненного цикла.

```
public class ExampleService extends Service {
        int mStartMode;       // indicates how to behave if the service is killed
        IBinder mBinder;      // interface for clients that bind
        boolean mAllowRebind; // indicates whether onRebind should be used

        @Override
        public void onCreate() {
            // The service is being created
        }
        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
            // The service is starting, due to a call to startService()
            return mStartMode;
        }
```

# Реализация обратных вызовов жизненного цикла

```
@Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
    }
    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
    }
}
```


Call to startService()
onCreate()
onStartCommand()
Service running
The service is stopped by itself or a client
onDestroy()
Service shut down
Unbounded service
Active Lifetime
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service

# Два вложенных цикла в жизненном цикле службы:

- **Весь жизненный цикл** службы происходит между вызовом метода onCreate() и возвратом из метода onDestroy(). Подобно операции, служба выполняет начальную настройку в методеonCreate() и освобождает все оставшиеся ресурсы в методе onDestroy(). Например, служба воспроизведения музыки может создать поток для воспроизведения музыки в методе onCreate(), затем остановить поток в методе onDestroy().
- Методы onCreate() и onDestroy() вызываются для всех служб, независимо от метода создания:startService() или bindService()

# Активный жизненный цикл

- **Активный жизненный цикл** службы начинается с вызова метода onStartCommand() или onBind(). Каждый метод направляется намерением Intent, которое было передано методу startService() или bindService(), соответственно.
- Если служба запущена, активный жизненный цикл заканчивается одновременно с окончанием всего жизненного цикла (служба активна даже после возврата из метода onStartCommand()).
- Если служба является привязанной, активный жизненный цикл заканчивается, когда возвращается метод onUnbind().

# Литература

- https://developer.android.com/guide/components/services.html