06

Services и ContentProviders

Nikolay Moskvin
moskvin@sibext.com

5 ноября 2013 г.

Содержание

Манифест

sibext company

Можно указать и/или зарегистрировать:

- Название – это Java package для приложения (должно быть уникальным)
- Компоненты приложения – Activity, Service, Broadcast receiver, Content provider
- Разрешения для доступа к защищенной части API и взаимодействия с другими приложениями
- Разрешения для других приложений к компонентом вашего приложения
- Минимальный уровень API
- Компоненты из библиотек, которые будут использоваться в вашем приложении

Манифест

sibext company

```
<manifest>
    <uses-permission />
    <permission />
    ...
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    ...
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
        </activity>
        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>
    </application>
</manifest>
```

Манифест

Подробней о всех тегах `http://developer.android.com/guide/topics/manifest/manifest-intro.html`

- Запрашиваем права на использования интернет соединений и камеры

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

- Запрашиваем feature для чипа камеры

```
<uses-feature android:name="android.hardware.camera" />
```

- `http://developer.android.com/guide/topics/manifest/manifest-intro.html`

Services

Второй по важности компонент после Activity. Позволяет выполнять долгие операции в фоне без пользовательского интерфейса.

- Обработка сетевых транзакций
- Проигрывания музыки
- Выполнения операция с файлами
- Взаимодействия с Contact Provider-ами

Внимание

Запуск любого сервиса происходит в “main” потоке. Сервис не создает отдельного потока для себя.
Если вам требуется делать долгую операцию в сервисе, то воспользуйтесь AsyncTask или HandlerThread

Пример кода для вывода в фон

```
final HandlerThread thread =
    new HandlerThread("ServiceLoop");
thread.start();
```

Services

Имеет две формы взаимодействия

- Started – Запущенный
  - **startService()**
  - запущен на неопределенное время
  - одна операция
  - не возвращает результата
  - **Пример:** Запускаем загрузку файла, после загрузки которого сервис останавливается.
- Bound – Связанный
  - **bindService()**
  - запущен на время, пока “жив” компонент
  - взаимодействует с компонентом (Activity)
  - много компонентов может присоединятся и после отсоединения последнего сервис останавливается.
  - **Пример:** Клиент-Сервер решение.

Внимание

Запуск любого сервиса происходит в “main” потоке. Сервис не создает отдельного потока для себя.

Services

Имеет две формы взаимодействия

- Started – Запущенный
  - **startService()**
  - запущен на неопределенное время
  - одна операция
  - не возвращает результата
  - **Пример:** Запускаем загрузку файла, после загрузки которого сервис останавливается.
- Bound – Связанный
  - **bindService()**
  - запущен на время, пока “жив” компонент
  - взаимодействует с компонентом (Activity)
  - много компонентов может присоединятся и после отсоединения последнего сервис останавливается.
    Позволяет:
    - послать запрос
    - получить результат
    - делать это между двумя процессами (IPC)
  - **Пример:** Клиент-Сервер решение.

Services

Имеет две формы взаимодействия

- Started – Запущенный
  - **startService()**
  - запущен на неопределенное время
  - одна операция
  - не возвращает результата
  - **Пример:** Запускаем загрузку файла, после загрузки которого сервис останавливается.
- Bound – Связанный
  - **bindService()**
  - запущен на время, пока “жив” компонент
  - взаимодействует с компонентом (Activity)
  - много компонентов может присоединятся и после отсоединения последнего сервис останавливается.
  - **Пример:** Клиент-Сервер решение.

Внимание

Запуск любого сервиса происходит в “main” потоке. Сервис не создает отдельного потока для себя.

Services

sibext company

Внимание

Запуск любого сервиса происходит в “main” потоке. Сервис не создает отдельного потока для себя.
Если вам требуется делать долгую операцию в сервисе, то воспользуйтесь AsyncTask или HandlerThread

Пример кода для вывода в фон

```
final HandlerThread thread =
    new HandlerThread("ServiceLoop");
thread.start();
Handler handler = new Handler(thread.getLooper());
handler.post(new Runnable() {
    @Override
    public void run() {
        // Do work with network
        ...
        thread.quit();
    }
});
```

Services

sibext company

Внимание

Запуск любого сервиса происходит в “main” потоке. Сервис не создает отдельного потока для себя.
Если вам требуется делать долгую операцию в сервисе, то воспользуйтесь AsyncTask или HandlerThread

Пример кода для вывода в фон

```
final HandlerThread thread =
    new HandlerThread("ServiceLoop");
thread.start();
Handler handler = new Handler(thread.getLooper());
handler.post(new Runnable() {
    @Override
    public void run() {
        // Do work with network
        ...
        thread.quit();
    }
});
```

Основные методы

sibext company

- **onStartCommand()** – Вызовется системой, когда другой компонент позовет **startService**. Для завершения такого сервиса необходимо вызвать **stopSelf()** или **stopService()**
- **onBind()** – Вызовется системой, когда иной компонент попытается связаться с сервисом с помощью **bindService()**. Если Вам не нужно связывание, просто верните null.
- **onCreate()** – Вызывается первый раз при создании сервиса, если сервер уже запущен, то этот метод не будет вызван.
- **onDestroy()** – Вызывается если сервис уже не нужен и будет в скором времени удален. Используется для очистки внутренних ресурсов.

Запускаем сервис

sibext company

- Определяем методы:

```
public class UpdateService extends Service {
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public int onStartCommand(Intent intent,
            int flags, int startId) {
      // Do something
      stopSelf();
      return super.onStartCommand(intent, flags, startId);
    }
}
```

- Запускаем

```
Intent i = new Intent(this, UpdateService.class);
this.startService(i);
```

- Подробный пример:

http://developer.android.com/guide/components/services.html#ExtendingService

Регистрация сервиса

- Добавляем в манифест тег service с атрибутом **android:name**:

```
<manifest ... >
  ...
  <application ... >
      ...
      <service android:name=".UpdateService"
        android:enabled="true"
        android:exported="false"
        android:process=":split_process"
        />
      ...
  </application>
</manifest>
```

Воскрешение после смерти

sibext company

Метод **onStartCommand()** может возвращает следующие константы:

- **START_NOT_STICKY** – сервис не будет воссоздаваться после смерти
- **START_STICKY** – сервис будет пересоздан, но intent не будет восстановлен
- **START_REDELIVER_INTENT** – полное восстановление, будет вызван **onStartCommand()** с восстановленным intent

Intent Service

Является подходящим решением для простых фоновых операций

`http://developer.android.com/reference/android/app/IntentService.html`

- **Преимущества**
  - Создает новый рабочую нить, которая отделяет сервис от главного потока
  - Создает очередь задач, не нужно беспокоится о синхронизации
  - Останавливает сервис автоматически после обработки всех запросов
  - Переопределен метод **onBind()** возвращает null
  - Переопределен метод **onStartCommand()** в которой реализована очередь и вызов **onHandleIntent()**
- Ограничения

Intent Service

Является подходящим решением для простых фоновых операций
`http://developer.android.com/reference/android/app/IntentService.html`

- Преимущества
- **Ограничения**
  - Нет прямого взаимодействия с Activity. Необходимо пересылать результаты пользовательскому интерфейсу.
  - Рабочий процесс основан на “цепочке”. Если сервис еще работает а Вы послали еще одну операцию, то последняя будет ждать выполнения первой.
  - Любая запущенная операция не может быть прервана.

Intent Service

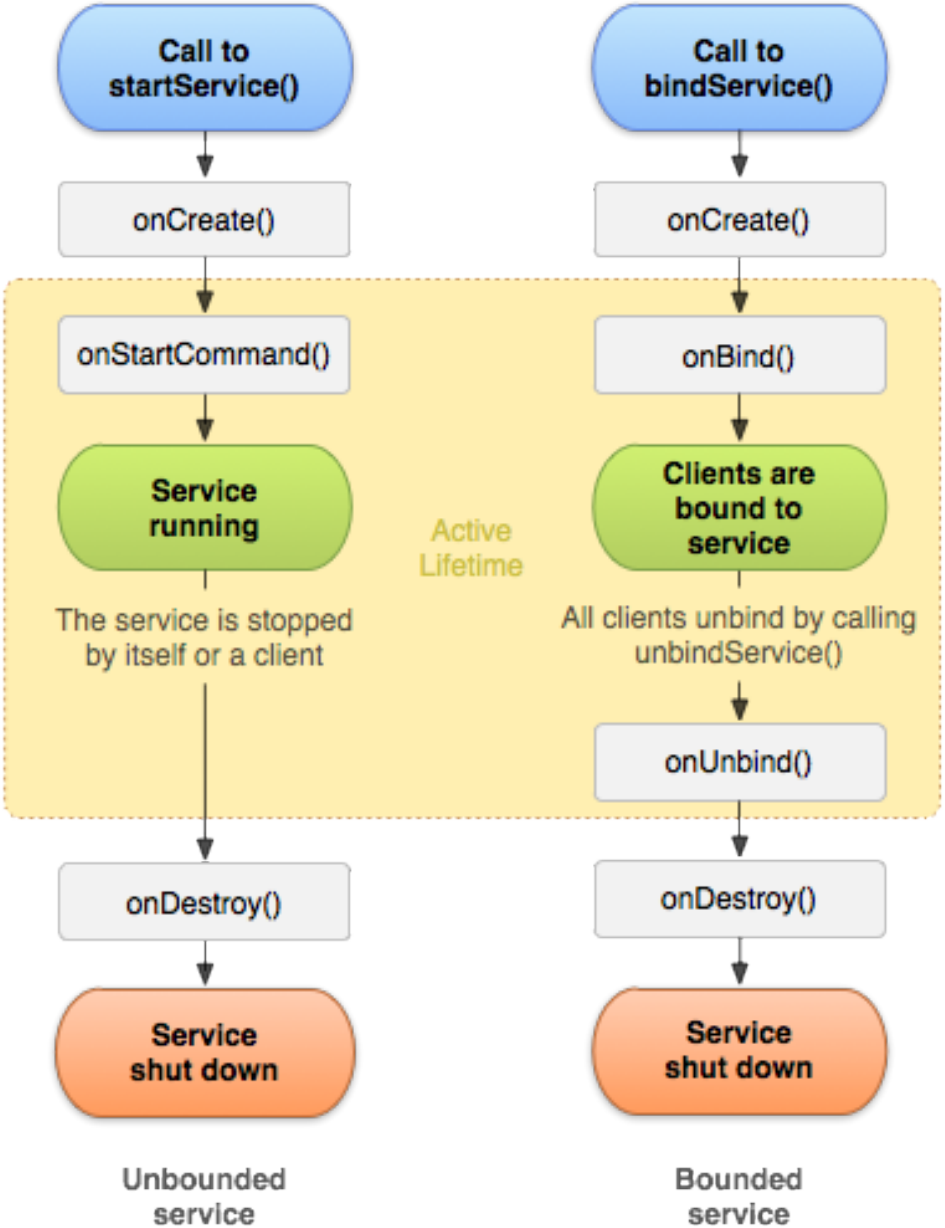
Пример кода для фонового сервиса

- Необходимо наследовать IntentService
- Реализовать метод onHandleIntent()
- Пример:

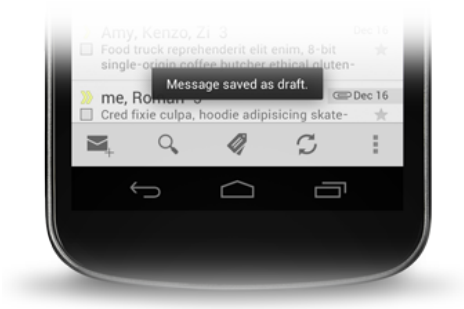
```
public class UsersUpdateService extends IntentService {
  public UsersUpdateService() {
      super("Users␣Update␣Service");
  }

  @Override
    protected void onHandleIntent(Intent intent) {
        // Gets last user id
        int id = intent.getIntExtra("USER_ID", -1);
        ...
        // Do work with network
        ...
    }
}
```

Жизненный цикл



Toasts и Notifications



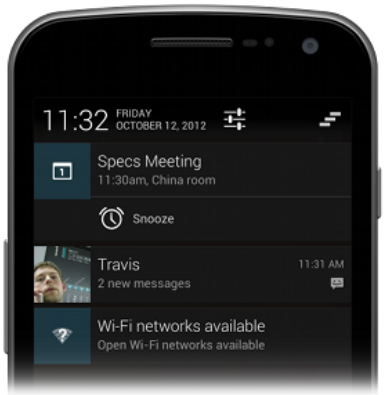
- Toasts – простой способ информировать о происходящем

```
Context context = getApplicationContext();
CharSequence text = "User␣photos␣was␣uploaded!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
toast.show();
```

- Подробности `http://developer.android.com/reference/android/widget/Toast.html`

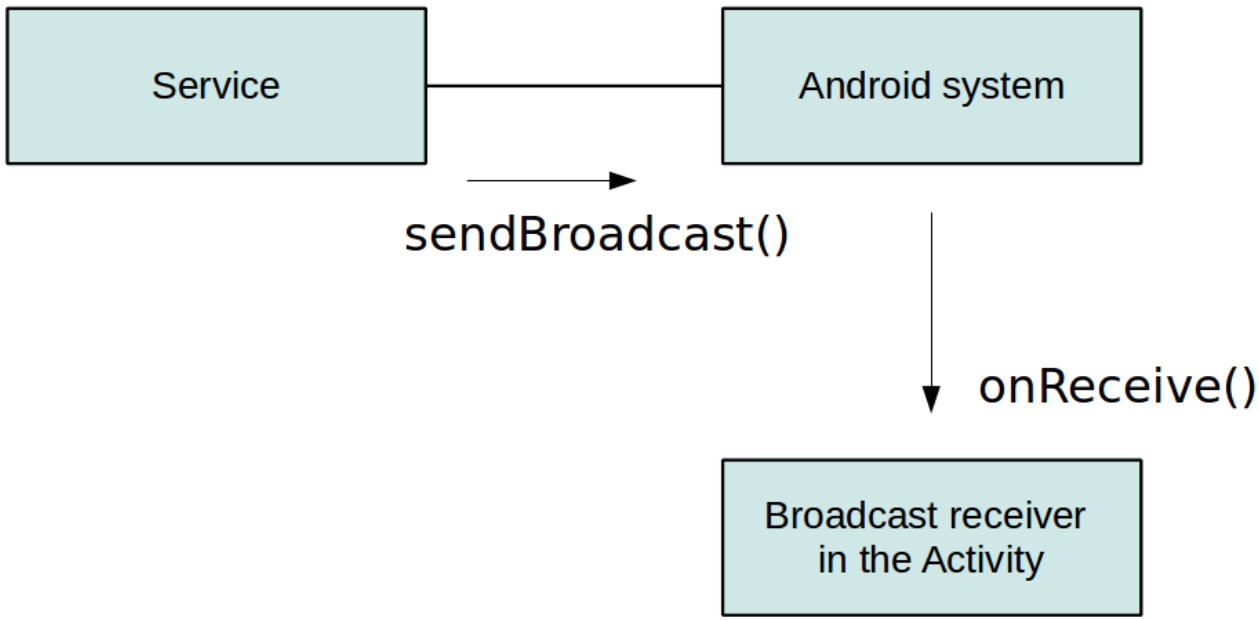
Toasts и Notifications



- Notifications – сообщение пользователю, которое может показываться без обычного пользовательского интерфейса приложения.
- Необходимо сделать:
  - создать нотификацию с помощью строителя
  - указывать источник, который связан с нотификацией
  - у менеджера вызвать метод **notify**
- Пример: `http:`

`//developer.android.com/guide/topics/ui/notifiers/notifications.html#SimpleNotification`

Broadcast receivers



- Используется для посылки сообщение всем приложениям
- **sendBroadcast()** – используется для посылки сообщений
- Можно регистрировать в манифесте или из кода с помощью **registerReceiver()** и **unregisterReceiver()**

Пример использования Broadcast receivers

sibext company

Контролируем в приложении отключение или подключение к сети.

- Регистрации в Манифесте:

```
<receiver
   android:name=".ConnectionStatusChangeReciever"
   android:enabled="true" >
   <intent-filter>
    <action
      android:name="android.net.conn.CONNECTIVITY_CHANGE" />
   </intent-filter>
</receiver>
```

- Определяем класс

```
public class ConnectionStatusChangeReciever extends BroadcastReceiver {

  public static final String ACTION_DISCONNECT = "ACTION_DISCONNECT";

  @Override
  public void onReceive(Context context, Intent intent) {
    if(intent.getAction().equals(ConnectivityManager.CONNECTIVITY_ACTION)) {
        NetworkInfo networkInfo =
          intent.getParcelableExtra(ConnectivityManager.EXTRA_NETWORK_INFO);

        if(networkInfo.getDetailedState() == DetailedState.DISCONNECTED) {
            context.sendBroadcast(new Intent(ACTION_DISCONNECT));
        }
    }
  }
```

Местоположение - это сложно!

sibext company

- Множество источников получения координат. Нужно выбирать в зависимости от **точности**, **скорости** и экономичности (**батарейка**)
  - Global Positioning System
  - Cell Global Identity
  - Wi-Fi
- Пользователь перемещается. Нужно часто вычислять изменения координат.
- Точность может изменяться. Точность не является постоянным свойством для источников. Местоположение полученное 10 сек. назад может быть лучше чем сейчас на этом или другом источнике.

Получение местонахождения

sibext company

- Основывается на callback-ах
- Location Manager – `http://developer.android.com/reference/android/location/LocationManager.html`
- Метод **requestLocationUpdates** имеет параметр Location Listener – `http://developer.android.com/reference/android/location/LocationListener.html`
- Необходимо реализовать метод **onLocationChanged** у которого имеется параметр Location – `http://developer.android.com/reference/android/location/Location.html`

Пример регистрации слушателя

sibext company

- Получаем ссылку на экземпляр класса для менеджера:

```
LocationManager locationManager = (LocationManager)
        this.getSystemService(Context.LOCATION_SERVICE);
```

- Реализуем слушателя координат:

```
LocationListener locationListener = new LocationListener() {
  public void onLocationChanged(Location location) {
    makeUseOfNewLocation(location);
  }
  public void onStatusChanged(String provider,
                    int status, Bundle extras) {}
  public void onProviderEnabled(String provider) {}
  public void onProviderDisabled(String provider) {}
};
```

- Регистрируем нашего слушателя:

```
locationManager.requestLocationUpdates(
            LocationManager.NETWORK_PROVIDER,
            0, //min.time interval between notifications
            0, //min.change in distance between notifications
            locationListener);
```

Регистрация в манифесте

- Если мы хотим слушать координаты от локального GPS чипа, то первым параметром указываем **GPS_PROVIDER**
- Запрашиваем разрешения у пользователя

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

or

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

Перестаем слушать

sibext company

- Можем получить “кое какие” координаты, для того, чтобы не ждать долго актуальных

```
String locationProvider = LocationManager.NETWORK_PROVIDER;
// Or use LocationManager.GPS_PROVIDER

Location lastKnownLocation =
  locationManager.getLastKnownLocation(locationProvider);
```

- Отвязываемся от прослушивания

```
// Remove the listener you previously added
locationManager.removeUpdates(locationListener);
```

В следующей лекции

sibext company

- Научимся работать с фотографиями и акселерометром
- Виджеты для рабочего стола
- Загрузка JSON на основе волейбола