Services in Android

Android Services:

Background Running Components Without Direct Interaction with User

A.R. Kazemi

Android Service

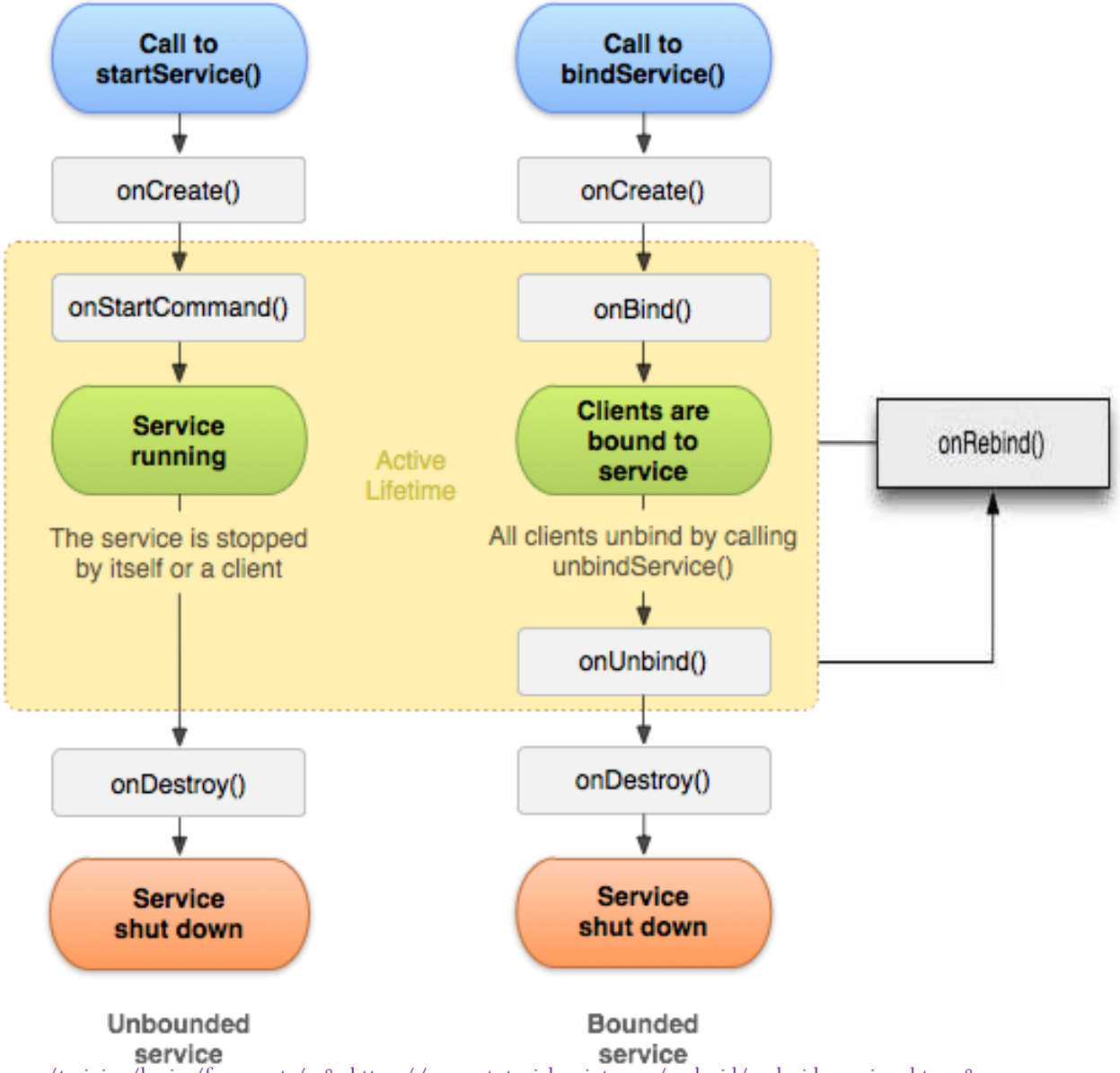
- One of 4 Android basic app components that runs in background
  - Android components
    - Activities
    - **Services**
    - Broadcast Receivers
    - Content Providers
- Performs long running operations
- Does not need direct interaction with user
- Works even if the application is destroyed

Service Running States

- Started Service
  - A Service which is started via an application component such as an Activity e.g. by calling :
    - **startService(Intent intent, int flags, int startId)**
  - After start, the service may (but not always) continue to run for ever
    - Eeven after death of the component which started it

- Bound Service
  - A Service is in Bound state if an application component such as an Activity binds to it by calling :
    - **bindService(Intent intent, ServiceConnection conn, int flags)**
  - It provides a client/server interface which allows other component to interact with service
  - This interface is accessible even between different processes via IPC Messaging.

Service States



Service Categories/Types

- **Service Categories:**
  - **System Services**
    - Provided by Android
    - Accessible via getSystemService(SERVICE_NAME_CONSTANT)
    - Some service name constants are defined in the **Context** class
    - Example
      - getSystemService(Context. SENSOR_SERVICE)
      - getSystemService(Context. ALARM_MANAGER)
  - **Custom Services**
    - Defined by programmers to run their ongoing tasks

- **Service Types:**
  - **Foreground Services**: are noticeable by user e.g. a service which plays sounds/media, they must display a status bar icon
  - **Background Services:** perform operations not directly noticeable by the user e.g a service which syncs data with server or cleans the app storage
  - **Bound Services:** a service which at least one app component binds to it by calling bindService(Intent intent)

Base Service Class(es)

- There are two basic Service classes to be extended to define Custom Services

  - **Service**
    - General purpose service which may handle multiple requests simultaneously
    - Programmer should consider the synchronization and multi-threading on demand.
    - **import android.app.Service;**

  - **IntentService (a sub-class of Service):**
    - A simpler service which must handle just a single request at a time and uses just one pre-defined worker thread on background.
    - Requests delivered to onStartCommad() are Queued and sent one by one to **onHandleIntent(Intent intent)** method.
    - The service is stopped automatically after that all start requests are handled.
    - **import android.app.IntentService;**

Writing a Service (Using Service)

- Write a class which extends Android class **Service**:
  - **import android.app.Service;**
  - **class MyService extends Service { … }**

- Override some or all of following methods:
  - void onCreate()
  - int onStartCommand(Intent intent, int flag, int startId)
    - Called by system when a component stars a service by calling startService(Intent intent)
  - IBinder onBind(Intent intent)
    - Called by system when a component calls bindService(Intent intent) to bind to it.
    - A bind interface object or null must be returned
  - boolean onUnBind(Intent intent)
    - Called by system when all the clients have disconnected from one of service published interfaces
  - void onReBind(Intent intent)
    - Called by system when a new client wants to bind to service after all previous ones have disconnected
  - void onDestroy()
    - Called by the system when the service is no longer used

Writing a Service (Using IntentService)

- Write a class which extends Android class **IntentService**:
  - **import android.app.IntentService;**
  - **class MyService extends IntentService { … }**
- Override following method
  - void onHandleIntent(Intent intent)

- Notes:
  - Although all methods inherited from **Service** can be overridden here but they have proper default implementations.
  - For most purposes just overriding onHandleIntent() is sufficient.
  - It has a default **onStartCommand(Intent intent)** implementation which puts the received intent to a Queue to be passed to **onHandleIntent()** one by one.
  - If you override **onCreate()**, **onStartCommand()**, or **onDestroy()** be sure to **call the super implementation** so that the IntentService can handle the life of its working thread.
  - There is no need to stop it, it stops automatically after all started tasks are handled by onHandleIntent().

Writing a Service (Java/Manifest)

```
<service
    android:name="MyService"
    android:icon="@drawable/icon"
    android:label="@string/service_name"
    >
</service>
```

JAVA

```
public class MyService extends Service {

  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
      //TODO do something useful
      return Service.START_NOT_STICKY;
  }

  @Override
  public IBinder onBind(Intent intent) {
    //TODO for communication return IBinder implementation
    return null;
  }
}
```

Writing a Service, (more details)

```
import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;

public class HelloService extends Service {

   /** indicates how to behave if the service is killed */
   int mStartMode;

   /** interface for clients that bind */
   IBinder mBinder;

   /** indicates whether onRebind should be used */
   boolean mAllowRebind;

   /** Called when the service is being created. */
   @Override
   public void onCreate() {

   }

   /** The service is starting, due to a call to startService() */
   @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
      return mStartMode;
   }
   /** A client is binding to the service with bindService() */
   @Override
   public IBinder onBind(Intent intent) {
      return mBinder;
   }

   /** Called when all clients have unbound with unbindService()*/
   @Override
   public boolean onUnbind(Intent intent) {
      return mAllowRebind;
   }

   /** Called when a client is binding to the service
       with bindService()*/
   @Override
   public void onRebind(Intent intent) {

   }

   /** Called when The service is no longer used and is
       being destroyed */
   @Override
   public void onDestroy() {

   }
}
```

Writing an IntentService

```
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.
      try {
          Thread.sleep(5000);
      } catch (InterruptedException e) {
          // Restore interrupt status.
          Thread.currentThread().interrupt();
      }
  }
}
```

Writing a Service (to be used as Bound Service)

```
public class MyService extends Service {
    /** Define a binder class used as service interface */
    public class MyBinder extends Binder {
        // add methods for client to interact with server
    }

    // Binder object given to clients by calling startBind()
    private final IBinder mBinder = new MyBinder();
    @Override
    public IBinder onBind(Intent intent) {
        // init resources for the given client intent
        return mBinder;
    }
    // Override other Service Methods
}
```

Declaring a Service in the Manifest

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

- There are a few attributes for <service> element
  - Such as:
    - permissions needed to start the declared service
    - The process in which the service is started, …
- See the <service> element for more options
  - https://developer.android.com/guide/topics/manifest/service-element.html
- The only **required** attribute is **android:name**

Starting a Service (using startService)

```
// use this to start and trigger a service
Intent i= new Intent(context, MyService.class);
// potentially add data to the intent
i.putExtra("KEY1", "Value to be used by the service");
context.startService(i);
```

- Important:
  - For **Security** reason always use an **explicit intent** to start your services.

- The startService() effects
  - If the service is not running yet, then Android System:
    - Creates a **new** instance of the service class
    - Calls **onCreate()** method on the service object to initialize
    - Calls **onStartCommand(Intent intent, …)** on the service object by passing the intent passed by the service caller to startService()
  - If the service is already running, then Android System:
    - Just calls **onStartCommand()** on the existing service again

Starting a Service (using bindService)

```
class BindingActivity extends Activity{
    private MyService.MyBinder mBinder = null;
    private ServiceConnection mConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName className,
            IBinder serviceBinder) {
            // Now bound to MyService, cast the IBinder to access service
            mBinder = (MyService.MyBinder) serviceBinder;;
        }
        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            mBinder = null;
        }
    };
    protected void onStart(){
        super.onStart();
        Intent intent = new Intent(this, MyService.class);
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
    }
```

Stopping a Service

- The service stops itself by calling
  - **stopSelf()**, permanently stops
  - **stopSelf(int startId)**, stops if the startId matches the last start id
  - must called when the services intended job is complete

- Another app component stops the service by calling:
  - **stopService(Intent intent)**
  - one call to it suffices even if you have called startService(…) multiple times from other app components.

- Bounded Services may be stopped when Binding comonent(s) do unbind.
  - **unbindService(ServiceConnection conn)**
  - conn is the same ServiceConnection callback object which is passed to
    - bindService(Intent, ServiceConnection, int)

Stopping a Service (started by bindService)

```
class BindingActivity extends Activity{
    private MyService.MyBinder mBinder = null;
    private ServiceConnection mConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName className,
            IBinder serviceBinder) {
            // Now bound to MyService, cast the IBinder to access service
            mBinder = (MyService.MyBinder) serviceBinder;;
        }
        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            mBinder = null;
        }
    };
    protected void onStart(){ ... }
    protected void onStop(){
        super.onStop();
        unbindService(mConnection);
    }
```