Android Services

Introduction

- The android platform supports the 'services' concept we all know from other popular operation systems, as windows. The android platform supports two types of services.
- The first are known as 'local services', and they include those services that belong to specific applications and cannot be accessed from outside the applications they belong to.

Introduction

- The second are known as 'remote services' and they include those services that can be accessed from applications other than those that hold them as well as from those that hold them.
- The 'remote services' are announced to all other applications using the Android Interface Definition Language (AIDL).

Creating Services

❖ We can create a new service by defining a class that extends the `android.app.Service` class,

```
public class MyService extends Service
{
    ...
}
```

and update the application manifest file with its details.

```
<service android:name="TestService1"></service>
```

Make sure to place the <service> element as a child element of <application>.

The `Service` Class

- Over-viewing the public methods the `android.app.Service` class defines will provide us with a better understanding of how does a service work.

```
Application getApplication();
abstract IBinder onBind(Intent intent);
void onConfigurationChanged(Configuration newConfig);
void onCreate();
void onDestroy();
void onLowMemory();
void onRebind(Intent intent);
```

The `Service` Class

```
void onStart(Intent intent, int startId);

boolean onUnbind(Intent intent);

final void setForeground(boolean isForeground);

final void stopSelf();

final void stopSelf(int startId);

final boolean stopSelfResult(int startId);
```

The `Service` Class

- Calling the `getApplication()` method returns the application that owns this service.
- Calling the `onBind()` method returns an interface through which other applications that run on the very same device can interact with the service.

The `Service` Class

- The `onConfigurationChange()` callback method is called when the device configuration changes. The service can use that method to reconfigure itself when the device configuration changes.
- The `onCreate()` is called when the service is created. Once that method was called, the `onStart()` is called. The `onDestroy()` is called when the service ends its life.

Background Tasks

- The main reason for supporting the service concept is to allow the implementation of background tasks. For getting a background task we will usually use a local service.

  An example for a background task can be checking for new emails. Common for email client applications.

Inter Processes Communication

- Services can also serve as a mechanism for implementing communication between separated processes on the same device.

  Implementing a remote service can be useful for handling the communications between activities from various applications that interact with each other. Instead of implementing separated mechanisms for each application all applications shall contact the remote service that shall route each request to the right activity.

Separated Implementation

- Though possible, it isn't a good practice to provide a service that functions both as a local and as a remote service. There are differences in the life cycle of each one of these types of services.

Local Service

- Local services are started when calling the `Context.startService()` method. Once started, they will continue to run on the background until we call the `Context.stopService()` method in order to stop them, or the service chooses to stop itself by calling the `Context.stopSelf()` method.

Local Service

- Local services are useful for those cases in which we need a background task to be executed.

  One example can be the need to access a resource over the network periodically.

  Another example can be performing some sort of a task (periodically as well).

Local Service

```
package com.abelski.samples;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class SimpleBackgroundServiceActivity extends Activity
{
    public TextView tf;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bindBtn = (Button) findViewById(R.id.bt_bind);
```

Local Service

```
        bindBtn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                startService(new Intent(SimpleBackgroundServiceActivity.this,
                    MyBackgroundService.class));
            }
        });
        Button unbindBtn = (Button) findViewById(R.id.bt_unbind);
        unbindBtn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                stopService(new Intent(SimpleBackgroundServiceActivity.this,
                    MyBackgroundService.class));
            }
        });
    }
}
```

Local Service

```
package com.abelski.samples;

import ...

public class MyBackgroundService extends Service
{
    private NotificationManager messagesManager;
    @Override
    public void onCreate()
    {
        super.onCreate();
        messagesManager = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);
    }
    @Override
    public void onStartCommand(Intent intent, int flags, int startId)
    {
        displayMessage("starting Background Service");
        Thread t = new Thread(null,
            new ServiceWorker(),"BackgroundService");
        t.start();
    }
```

Local Service

```
class ServiceWorker implements Runnable
{
    int i=1, sum=0;
    public void run()
    {
        while(i<=1000)
        {
            sum += i;
            try {Thread.sleep(1000);} catch(InterruptedException e){}
            i++;
            System.out.println("i="+i+" sum="+sum);
        }
    }
}

@Override
public void onDestroy()
{
    displayMessage("stopping Background Service");
    super.onDestroy();
}
```

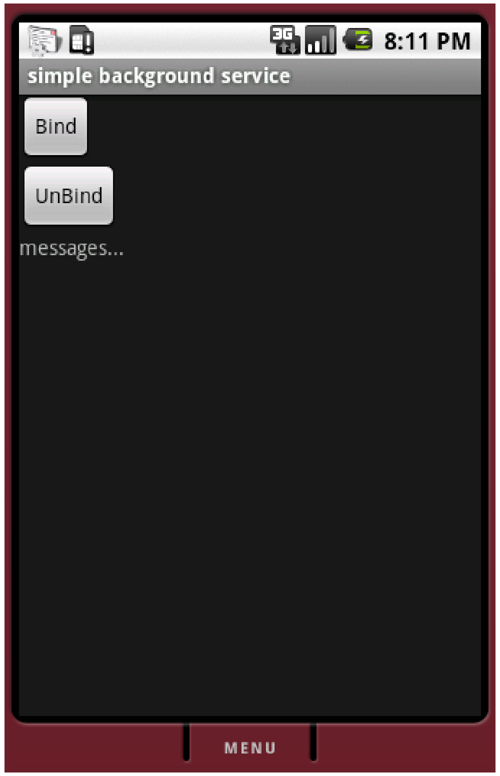
Local Service

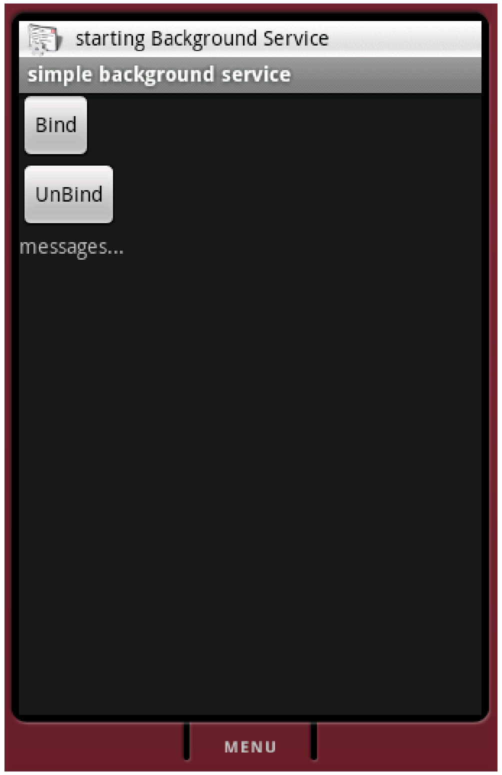
```
    @Override
    public void onStart(Intent intent, int startId)
    {
        super.onStart(intent, startId);
    }

    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }

    private void displayMessage(String message)
    {
        Notification notification = new Notification(R.drawable.icon,
            message,System.currentTimeMillis());
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
            new Intent(this, SimpleBackgroundServiceActivity.class), 0);
        notification.setLatestEventInfo(this,
            "Background Service", message,contentIntent);
        messagesManager.notify(R.string.noto, notification);
    }

}
```

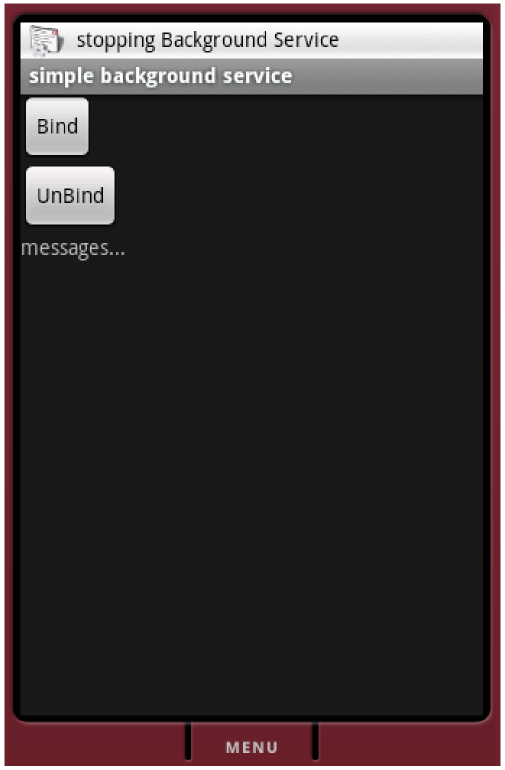
Local Service

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.abelski.samples"
        android:versionCode="1"
        android:versionName="1.0">

    <application android:icon="@drawable/icon" android:label="@string/app_name">

        <activity android:name=".SimpleBackgroundServiceActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

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

    </application>

    <uses-sdk android:minSdkVersion="3" />

</manifest>
```

Local Service




starting Background Service


stopping Background Service

Creating Remote Service

- The purpose of having a remote service is to expose a remotable object for Inter Process Communication (IPC).
- Developing a remote service is similar to developing a local service.
- It is possible to develop a service that functions both as a local one and as a remote one.

Creating Remote Service

- The first step in defining a remote service is creating an `.aidl` file that describes it. The `.aidl` file uses the Java syntax and it should be saved with the `.aidl` extension.
- Adding the `.aidl` file to our eclipse IDE the android eclipse plug-in will call the AIDL compiler to generate Java interface based on that `.aidl` file. The AIDL compiler is automatically called as part of the build process.

Creating Remote Service

- The `.aidl` file includes the definition for the interface `IcurrencyService.` This interface defines the methods and fields available to the clients.
- Based on the `.aidl` file, the aidl compiler creates an interface in the Java programming language.

Creating Remote Service

- That interface includes an inner abstract class named `Stub` that extends `android.os.Binder` and implements our `.aidl` interface. This inner abstract class includes the definition for few additional required methods.

Creating Remote Service

```
package com.abelski.currencyservice;

interface ICurrencyService
{
    double getCurrency(String country);
}
```

This is the .aidl file we define

Creating Remote Service

```
package com.abelski.samples.remoteservice;

public interface ICurrencyService extends android.os.IInterface
{
    public static abstract class Stub extends android.os.Binder
        implements com.abelski.samples.remoteservice.ICurrencyService
    {
        ...
    }
    ...
    public double getCurrency(java.lang.String currencyName) throws
        android.os.RemoteException;
}
```

This is the auto generated interface the aidl compiler generates

Creating Remote Service

- Implementing the service will be by defining a class that extends `android.app.Service` and includes an inner class that extends the `OurInterface.Stub` class and implements the abstract methods that were defined in the `.aidl` file.
- Exposing the service to other clients is done by providing an implementation of the `onBind()` method as well as some configuration code we add into the `AndroidManifest.xml` configuration file.

Creating Remote Service

- The `onBind()` method should return a reference for an `IBinder` object. That object should be of a type that implements the interface that was defined by the AIDL compiler.

Creating Remote Service

```
package com.abelski.currencyservice;

import android.os.RemoteException;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class CurrencyService extends Service
{
    public class CurrencyServiceImpl extends ICurrencyService.Stub
    {
        @Override
        public double getCurrency(String ticker) throws RemoteException
        {
            return 3.86;
        }
    }
```

Creating Remote Service

```
    @Override
    public void onCreate()
    {
        super.onCreate();
    }

    @Override
    public void onDestroy()
    {
        super.onDestroy();
    }

    @Override
    public void onStart(Intent intent, int startId)
    {
        super.onStart(intent, startId);
    }

    @Override
    public IBinder onBind(Intent intent)
    {
        return new CurrencyServiceImpl();
    }
}
```

Creating Remote Service

- In order to expose our service to other clients we should add a service declaration to the `AndroidManifest.xml` file.
- The difference comparing with a local service is the need in having an intent-filter element that exposes the service.

Creating Remote Service

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.abelski.currencyservice" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">

        <service android:name=".CurrencyService">
            <intent-filter>
                <action android:name=
                    "com.abelski.currencyservice.ICurrencyService" />
            </intent-filter>
        </service>

    </application>
    <uses-sdk android:minSdkVersion="7" />

</manifest>
```

Creating Remote Service

- If there are thrown exceptions within the remote service they won't be sent back to the caller.
- When a client calls the remote service the call is executed synchronously. When the remote service requires more than few milliseconds we better call it within a separated thread.

Using Remote Service

- Adding the `.aidl` file to another client application project will indirectly create the same generated interface we have in the remote service.
- We first need to instantiate a `ServiceConnection` object. We should define a class that implements this interface.

Using Remote Service

```
package com.abelski.currencyservice;

interface ICurrencyService
{
    double getCurrency(String country);
}
```

This is the .aidl file we define

Using Remote Service

- Calling `bindService` passing over the accurate required intent will make the remote service available for the client application.
- Calling `unbindService` passing over the reference for the `ServiceConnection` object will tear down the remote service.

Using Remote Service

```
public class MainActivity extends Activity
{
    private ICurrencyService currencyService = null;
    private Button bindBt;
    private Button callBt;
    private Button unbindBt;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bindBt = (Button) findViewById(R.id.bind);
        bindBt.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View view)
            {
                bindService(new Intent(ICurrencyService.class.getName()),
                        serviceConnection, Context.BIND_AUTO_CREATE);
                bindBt.setEnabled(false);
                callBt.setEnabled(true);
                unbindBt.setEnabled(true);
```

Using Remote Service

```
        Toast.makeText(MainActivity.this,
            "Currency Remote Service is Binded",
            Toast.LENGTH_SHORT).show();
    }
});

callBt = (Button) findViewById(R.id.call);
callBt.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View view)
    {
        callService();
    }
});
```

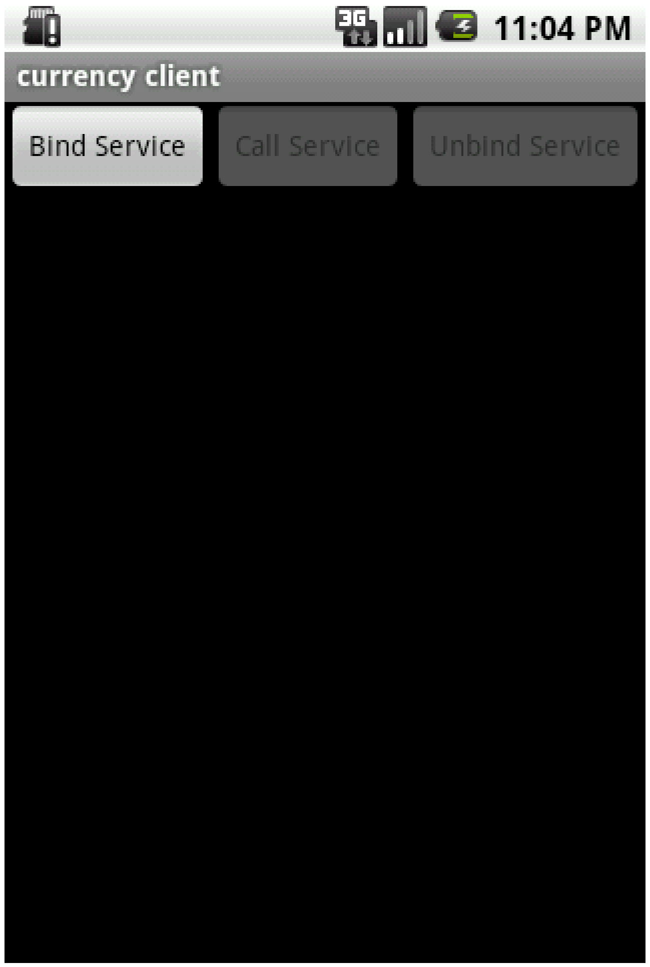
Using Remote Service

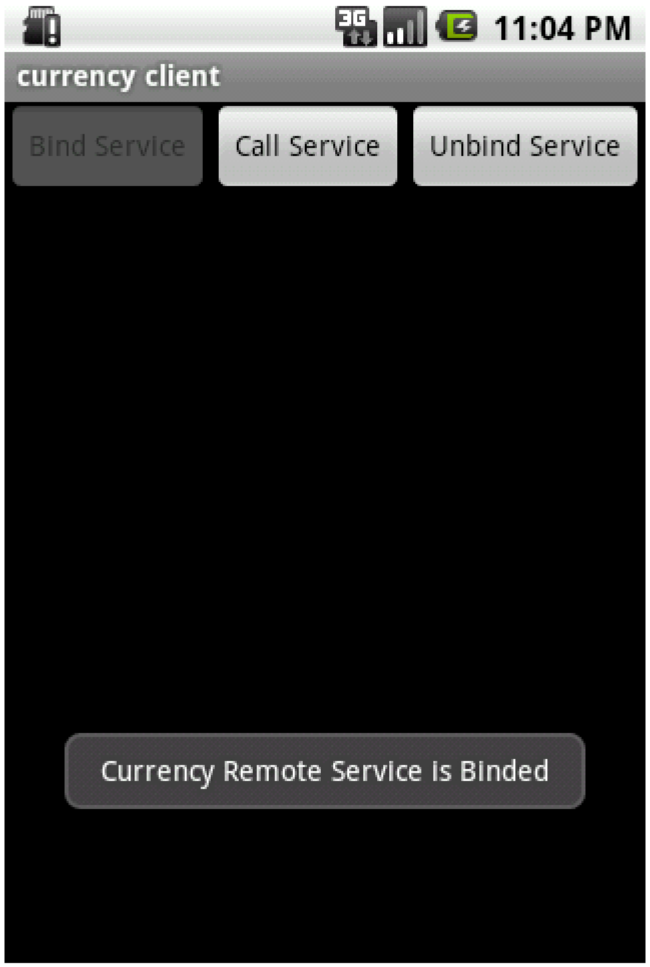
```
        unbindBt = (Button) findViewById(R.id.unbind);
        unbindBt.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View view)
            {
                unbindService(serviceConnection);
                bindBt.setEnabled(true);
                callBt.setEnabled(false);
                unbindBt.setEnabled(false);
                Toast.makeText(MainActivity.this,
                    "Currency Remote Service is Binded",
                    Toast.LENGTH_SHORT).show();
            }
        });
        unbindBt.setEnabled(false);
        callBt.setEnabled(false);
    }
```

Using Remote Service

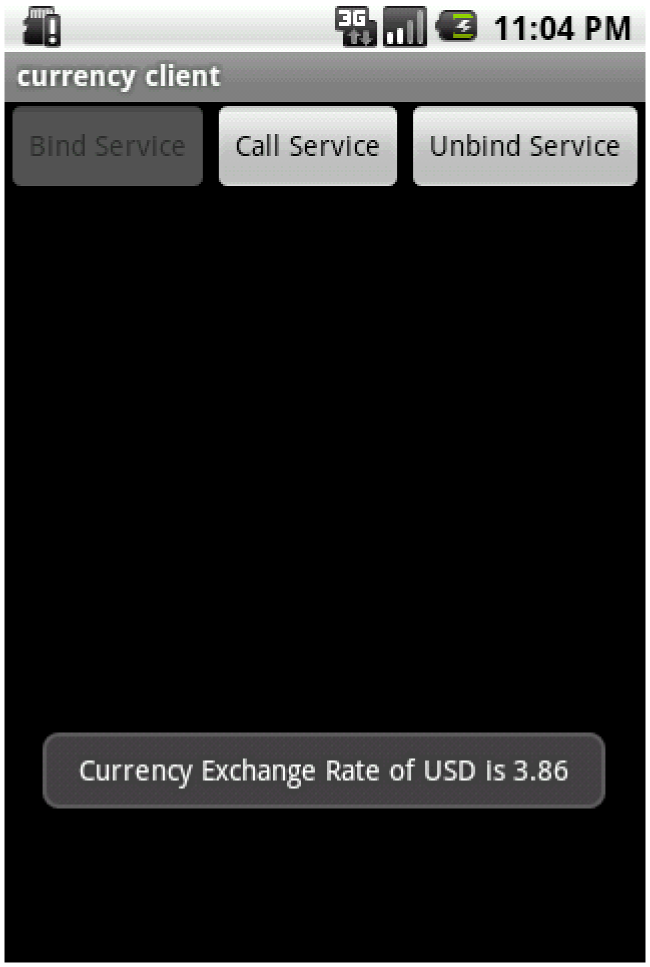
```
    private void callService()
    {
        try
        {
            double result = currencyService.getCurrency("USD");
            Toast.makeText(MainActivity.this, "Currency Exchange Rate of USD is "
                 + result, Toast.LENGTH_SHORT).show();
        }
        catch (RemoteException exception) { }
    }

    private ServiceConnection serviceConnection = new ServiceConnection()
    {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service)
        {
            currencyService = ICurrencyService.Stub.asInterface(service);
        }
        @Override
        public void onServiceDisconnected(ComponentName name)
        {
            currencyService = null;
        }
    };
}
```

Remote Service Code Sample





Currency Remote Service is Binded



Currency Exchange Rate of USD is 3.86