

4.2.1 Service初涉

分类 [Android 基础入门教程](#)

Android 基础入门教程(Q群号 : 153836263)

本节引言

好的，我们在前三节中对Android中的Activity进行了研究学习，相信大家获益良多吧！本节开始我们继续来学习Android中的第二个组件：Service(服务)，好，废话不多说，开始本节内容！

1.线程的相关概念

在开始学习Service之前我们先来了解下线程的一些概念！

1) 相关概念：

程序：为了完成特定任务，用某种语言编写的一组指令集合(一组**静态代码**)

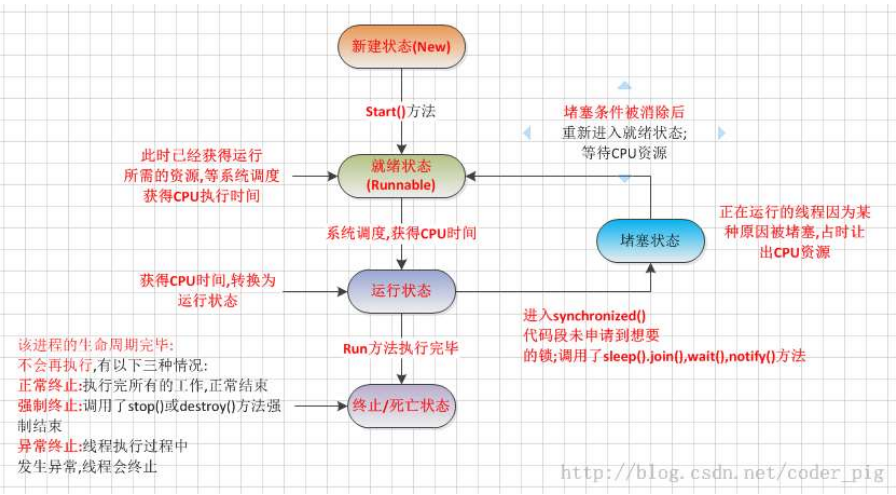
进程：**运行中的程序**，系统调度与资源分配的一个**独立单位**，操作系统会为每个进程分配一段内存空间！程序的依次动态执行，经历代码的加载，执行，执行完毕的完整过程！

线程：比进程更小的执行单元，每个进程可能有多条线程，**线程**需要放在一个**进程**中才能执行，**线程**由**程序**负责管理，而**进程**则由**系统**进行调度！

多线程的理解：**并行**执行多个条指令，将**CPU时间片**按照调度算法分配给各个**线程**，实际上是**分时**执行的，只是这个切换的时间很短，用户感觉到“同时”而已！

2) 线程的生命周期：

- 1.0 Android基础入门教程
- 1.0.1 2015年最新Android基...
- 1.1 背景相关与系统架构分析
- 1.2 开发环境搭建
- 1.2.1 使用Eclipse + ADT + S...
- 1.2.2 使用Android Studio开...
- 1.3 SDK更新不了问题解决
- 1.4 Genymotion模拟器安装
- 1.5.1 Git使用教程之本地仓...
- 1.5.2 Git之使用GitHub搭建...
- 1.6 .9(九妹)图片怎么玩
- 1.7 界面原型设计
- 1.8 工程相关解析(各种文件...
- 1.9 Android程序签名打包
- 1.11 反编译APK获取代码&...
- 2.1 View与ViewGroup的概念
- 2.2.1 LinearLayout(线性布局)
- 2.2.2 RelativeLayout(相对布...
- 2.2.3 TableLayout(表格布局)
- 2.2.4 FrameLayout(帧布局)
- 2.2.5 GridLayout(网格布局)
- 2.2.6 AbsoluteLayout(绝对...
- 2.3.1 TextView(文本框)详解
- 2.3.2 EditText(输入框)详解
- 2.3.3 Button(按钮)与ImageB...
- 2.3.4 ImageView(图像视图)
- 2.3.5.RadioButton(单选按钮...
- 2.3.6 开关按钮ToggleButton...
- 2.3.7 ProgressBar(进度条)
- 2.3.8 SeekBar(拖动条)
- 2.3.9 RatingBar(星级评分条)
- 2.4.1 ScrollView(滚动条)



3) 创建线程的三种方式：

- 1. 继承Thread类
- 2. 实现Runnable接口
- 3. 实现Callable接口 如果：使用的是2创建的线程的话，可以直接这样启动：

```
new Thread(myThread).start();
```

当更多的时候我们喜欢使用匿名类，即下面这种写法：

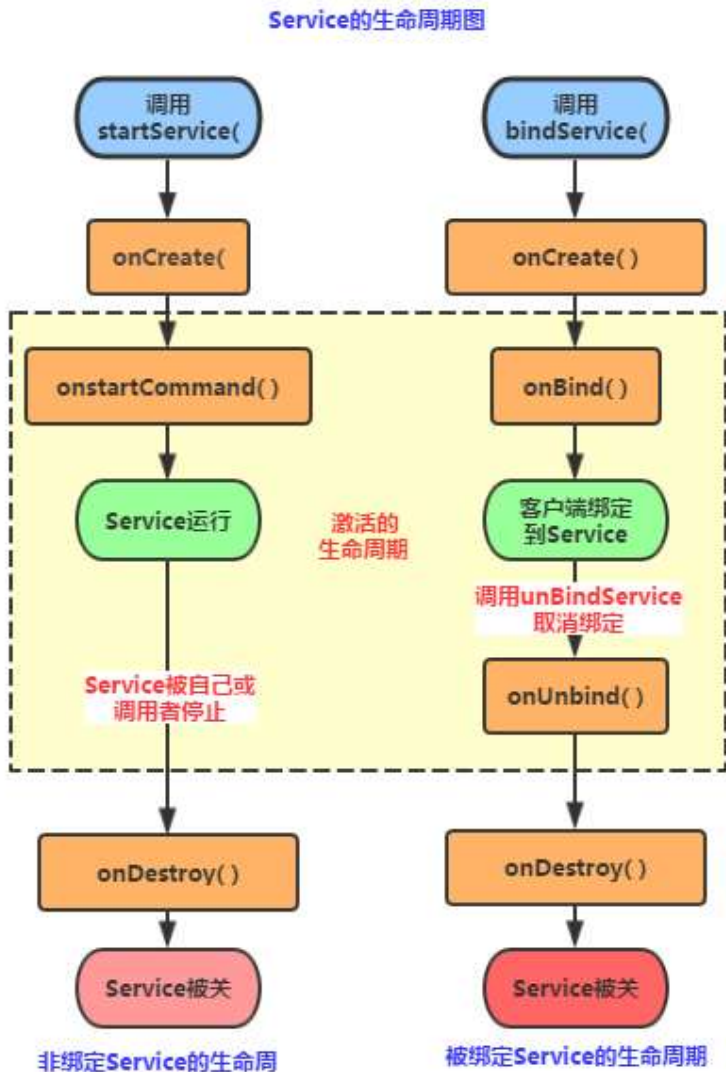
```
new Thread(new Runnable() {  
    public void run();  
}).start();
```

2.Service与Thread线程的区别

其实他们两者并没有太大的关系，不过有很多朋友经常把这两个混淆了！ Thread是线程，程序执行的最小单元，分配CPU的基本单位！而Service则是Android提供一个允许长时间留驻后台的一个组件，最常见的用法就是做轮询操作！或者想在后台做一些事情，比如后台下载更新！记得别把这两个概念混淆！

3.Service的生命周期图

- 2.4.2 Date & Time组件(上)
- 2.4.3 Date & Time组件(下)
- 2.4.4 Adapter基础讲解
- 2.4.5 ListView简单实用
- 2.4.6 BaseAdapter优化
- 2.4.7 ListView的焦点问题
- 2.4.8 ListView之checkbox错...
- 2.4.9 ListView的数据更新问题
- 2.5.0 构建一个可复用的自定...
- 2.5.1 ListView Item多布局的...
- 2.5.2 GridView(网格视图)的...
- 2.5.3 Spinner(列表选项框)...
- 2.5.4 AutoCompleteTextVie...
- 2.5.5 ExpandableListView(...
- 2.5.6 ViewPager(翻转视图)...
- 2.5.7 Toast(吐司)的基本使用
- 2.5.8 Notification(状态栏通...
- 2.5.9 AlertDialog(对话框)详解
- 2.6.0 其他几种常用对话框基...
- 2.6.1 PopupWindow(悬浮框...
- 2.6.2 菜单(Menu)
- 2.6.3 ViewPager的简单使用
- 2.6.4 DrawerLayout(官方侧...
- 3.1.1 基于监听的事件处理机制
- 3.2 基于回调的事件处理机制
- 3.3 Handler消息传递机制浅析
- 3.4 TouchListener PK OnTo...
- 3.5 监听EditText的内容变化
- 3.6 响应系统设置的事件(Co...
- 3.7 AsyncTask异步任务
- 3.8 Gestures(手势)
- 4.1.1 Activity初学乍练
- 4.1.2 Activity初窥门径
- 4.1.3 Activity登堂入室
- 4.2.1 Service初涉
- 4.2.2 Service进阶
- 4.2.3 Service精通
- 4.3.1 BroadcastReceiver牛...
- 4.3.2 BroadcastReceiver庖...
- 4.4.1 ContentProvider初探



4.生命周期解析

好的，从上图的生命周期，我们可以知道，Android中使用Service的方式有两种：

1) **StartService()启动Service**

2) **BindService()启动Service**

PS:还有一种，就是启动Service后，绑定Service！

1) 相关方法详解：

onCreate()：当Service第一次被创建后立即回调该方法，该方法在整个生命周期中只会调用一次！

onDestory()：当Service被关闭时会回调该方法，该方法只会回调一次！

onStartCommand(intent,flag,startId)：早期版本是onStart(intent,startId)，当客户端调用startService(Intent)方法时会回调，可多次调用StartService方法，但不会再创建新的Service对象，而是继续复用前面产生的Service对象，但会继续回调onStartCommand()方法！

IBinder onOnbind(intent)：该方法是Service都必须实现的方法，该方法会返回一个IBinder对象，app通过该对象与Service组件进行通信！

onUnbind(intent)：当该Service上绑定的所有客户端都断开时会回调该方

4.4.2 ContentProvider再探...

4.5.1 Intent的基本使用

4.5.2 Intent之复杂数据的传递

5.1 Fragment基本概述

5.2.1 Fragment实例精讲一...

5.2.2 Fragment实例精讲一...

5.2.3 Fragment实例精讲一...

5.2.4 Fragment实例精讲一...

5.2.5 Fragment实例精讲一...

6.1 数据存储与访问之——文...

6.2 数据存储与访问之——S...

6.3.1 数据存储与访问之——...

6.3.2 数据存储与访问之——...

7.1.1 Android网络编程要学...

7.1.2 Android Http请求头与...

7.1.3 Android HTTP请求方...

7.1.4 Android HTTP请求方...

7.2.1 Android XML数据解析

7.2.2 Android JSON数据解析

7.3.1 Android 文件上传

7.3.2 Android 文件下载 (1)

7.3.3 Android 文件下载 (2)

7.4 Android 调用 Webservice

7.5.1 WebView(网页视图)基...

7.5.2 WebView和JavaScip...

7.5.3 Android 4.4后WebVie...

7.5.4 WebView文件下载

7.5.5 WebView缓存问题

7.5.6 WebView处理网页返...

7.6.1 Socket学习网络基础准备

7.6.2 基于TCP协议的Socket...

7.6.3 基于TCP协议的Socket...

7.6.4 基于UDP协议的Socke...

8.1.1 Android中的13种Draw...

8.1.2 Android中的13种Draw...

8.1.3 Android中的13种Draw...

8.2.1 Bitmap(位图)全解析 P...

8.2.2 Bitmap引起的OOM问题

8.3.1 三个绘图工具类详解

8.3.2 绘图类实战示例

法！

2) StartService启动Service

- ①首次启动会创建一个Service实例,依次调用onCreate()和onStartCommand()方法,此时Service 进入运行状态,如果再次调用StartService启动Service,将不会再创建新的Service对象, 系统会直接复用前面创建的Service对象,调用它的onStartCommand()方法！
- ②但这样的Service与它的调用者无必然的联系,就是说当调用者结束了自己的生命周期, 但是只要不调用stopService,那么Service还是会继续运行的!
- ③无论启动了多少次Service,只需调用一次StopService即可停掉Service

3) BindService启动Service

- ①当首次使用bindService绑定一个Service时,系统会实例化一个Service实例,并调用其onCreate()和onBind()方法,然后调用者就可以通过IBinder和Service进行交互了,此后如果再次使用bindService绑定Service,系统不会创建新的Service实例,也不会再调用onBind()方法,只会直接把IBinder对象传递给其他后来增加的客户端!
- ②如果我们解除与服务的绑定,只需调用unbindService(),此时onUnbind和onDestory方法将会被调用!这是一个客户端的情况,假如是多个客户端绑定同一个Service的话,情况如下 当一个客户完成和service之间的互动后, 它调用 unbindService() 方法来解除绑定。当所有的客户端都和service解除绑定后, 系统会销毁service。(除非service也被startService()方法开启)
- ③另外,和上面那张情况不同,bindService模式下的Service是与调用者相互关联的,可以理解为 "一条绳子上的蚂蚱",要死一起死,在bindService后,一旦调用者销毁,那么Service也立即终止!
通过BindService调用Service时调用的Context的bindService的解析
bindService(Intent service,ServiceConnection conn,int flags)
service:通过该intent指定要启动的Service
conn:ServiceConnection对象,用户监听访问者与Service间的连接情况, 连接成功回调该对象中的onServiceConnected(ComponentName,IBinder)方法; 如果Service所在的宿主由于异常终止或者其他原因终止,导致Service与访问者间断开 连接时调用**onServiceDisconnected**(CompanentName)方法,主动通过**unBindService()** 方法断开并不会调用上述方法!
flags:指定绑定是否自动创建Service(如果Service还未创建), 参数可以是0(不自动创建),BIND_AUTO_CREATE(自动创建)

4) StartService启动Service后bindService绑定

如果Service已经由某个客户端通过StartService()启动,接下来由其他客户端 再调用bindService() 绑定到该Service后调用unbindService()解除绑定最后在 调用bindService()绑定到Service的话,此时所触发的生命周期方法如下:
onCreate()->onStartCommand()->onBind()->onUnbind()-

- 8.3.3 Paint API之—— Mask...
- 8.3.4 Paint API之—— Xferm...
- 8.3.5 Paint API之—— Xferm...
- 8.3.6 Paint API之—— Xferm...
- 8.3.7 Paint API之—— Xferm...
- 8.3.8 Paint API之—— Xferm...
- 8.3.9 Paint API之—— Color...
- 8.3.10 Paint API之—— Colo...
- 8.3.11 Paint API之—— Colo...
- 8.3.12 Paint API之—— Path...
- 8.3.13 Paint API之—— Sha...
- 8.3.14 Paint几个枚举/常量值...
- 8.3.15 Paint API之——Type...
- 8.3.16 Canvas API详解(Part 1)
- 8.3.17 Canvas API详解(Part...
- 8.3.18 Canvas API详解(Part...
- 8.4.1 Android动画合集之帧...
- 8.4.2 Android动画合集之补...
- 8.4.3 Android动画合集之属...
- 8.4.4 Android动画合集之属...
- 9.1 使用SoundPool播放音...
- 9.2 MediaPlayer播放音频与...
- 9.3 使用Camera拍照
- 9.4 使用MediaRecord录音
- 10.1 TelephonyManager(电...
- 10.2 SmsManager(短信管理...
- 10.3 AudioManager(音频管...
- 10.4 Vibrator(振动器)
- 10.5 AlarmManager(闹钟服务)
- 10.6 PowerManager(电源服...
- 10.7 WindowManager(窗口...
- 10.8 LayoutInflater(布局服务)
- 10.9 WallpaperManager(壁...
- 10.10 传感器专题(1)——相...
- 10.11 传感器专题(2)——方...
- 10.12 传感器专题(3)——加...
- 10.12 传感器专题(4)——其...
- 10.14 Android GPS初涉
- 11.0 《2015最新Android基...

>onRebind()

PS:前提是:onUnbind()方法返回true!!! 这里或许部分读者有疑惑了,调用了unbindService后Service不是应该调用 onDestory()方法么!其实这是因为这个Service是由我们的StartService来启动的,所以你调用onUnbind()方法取消绑定,Service也是不会终止的!

得出的结论:假如我们使用bindService来绑定一个启动的Service,注意是已经启动的Service!!! 系统只是将Service的内部IBinder对象传递给Activity,并不会将Service的生命周期 与Activity绑定,因此调用unBindService()方法取消绑定时,Service也不会被销毁!

5.生命周期验证

接下来我们写代码来验证下生命周期:

1) 验证StartService启动Service的调用顺序

首先我们自定义一个Service,重写相关的方法,用户在logcat上打印验证:

TestService1.java

```
public class TestService1 extends Service {
    private final String TAG = "TestService1";
    //必须要实现的方法
    @Override
    public IBinder onBind(Intent intent) {
        Log.i(TAG, "onBind方法被调用!");
        return null;
    }

    //Service被创建时调用
    @Override
    public void onCreate() {
        Log.i(TAG, "onCreate方法被调用!");
        super.onCreate();
    }

    //Service被启动时调用
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i(TAG, "onStartCommand方法被调用!");
        return super.onStartCommand(intent, flags, startId);
    }

    //Service被关闭之前回调
    @Override
    public void onDestroy() {
        Log.i(TAG, "onDestory方法被调用!");
        super.onDestroy();
    }
}
```

AndroidManifest.xml完成Service注册

```
<!-- 配置Service组件,同时配置一个action -->
<service android:name=".TestService1">
    <intent-filter>
        <action android:name="com.jay.example.service.T
EST_SERVICE1"/>
    </intent-filter>
</service>
```

紧接着是简单的布局文件,两个按钮,再最后是MainActivity的编写,在按钮的点击事件中分别 调用startService()和stopService()!

```
public class MainActivity extends Activity {

    private Button start;
    private Button stop;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        start = (Button) findViewById(R.id.btnstart);
        stop = (Button) findViewById(R.id.btnstop);
        //创建启动Service的Intent,以及Intent属性
        final Intent intent = new Intent();
        intent.setAction("com.jay.example.service.TEST_SERVICE1
");

        //为两个按钮设置点击事件,分别是启动与停止service
        start.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                startService(intent);
            }
        });

        stop.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                stopService(intent);
            }
        });
    }
}
```

运行截图：



点击开始服务:

Application	Tag	Text
com.jay.example...	TestService1	onCreate方法被调用!
com.jay.example...	TestService1	onStartCommand方法被调用!

吃饱饭没事做,点多几下:

com.jay.example...	TestService1	onCreate方法被调用!
com.jay.example...	TestService1	onStartCommand方法被调用!
com.jay.example...	TestService1	onStartCommand方法被调用!
com.jay.example...	TestService1	onStartCommand方法被调用!

最后点击停止服务:

com.jay.example...	TestService1	onCreate方法被调用!
com.jay.example...	TestService1	onStartCommand方法被调用!
com.jay.example...	TestService1	onStartCommand方法被调用!
com.jay.example...	TestService1	onStartCommand方法被调用!
com.jay.example...	TestService1	onDestroy方法被调用!

结果分析 :

从上面的运行结果我们可以验证我们生命周期图中解释的内容: 我们发现 onBind()方法并没有被调用,另外多次点击启动Service,只会重复地调用 onStartCommand 方法!无论我们启动多少次Service,一个stopService就会停止Service!

2) 验证BindService启动Service的顺序:

在开始讲写代码之前,我们先要来了解一些东西先: 首先是第一个大图下面给出的Context的bindService方法 :

ServiceConnection对象:监听访问者与Service间的连接情况,如果成功连接,回调 onServiceConnected(),如果异常终止或者其他原因终止导致Service与访问者断开 连接则回调onServiceDisconnected方法,调用unBindService()不会调用该方法!

onServiceConnected方法中有一个IBinder对象,该对象即可实现与被绑定 Service 之间的通信!我们再开发Service类时,默认需要实现IBinder onBind()方法,该方法返回的 IBinder对象会传到ServiceConnection对象中的 onServiceConnected的参数,我们就可以 在这里通过这个IBinder与Service进行通信!

总结 :

Step 1:在自定义的Service中继承Binder,实现自己的IBinder对象

Step 2:通过onBind()方法返回自己的IBinder对象

Step 3:在绑定该Service的类中定义一个ServiceConnection对象,重写两个方法, onServiceConnected和onDisconnected! 然后直接读取IBinder传递过来的参数即可!

那么好了,接下来就是写代码验证了,这里的话我们定义一个用来计时的Service, 然后来演示BindService的用法以及方法调用流程!代码比较简单,不解释了!

TestService2.java:

```
public class TestService2 extends Service {
    private final String TAG = "TestService2";
    private int count;
    private boolean quit;

    //定义onBinder方法所返回的对象
    private MyBinder binder = new MyBinder();
    public class MyBinder extends Binder
    {
        public int getCount()
        {
            return count;
        }
    }

    //必须实现的方法,绑定改Service时回调该方法
    @Override
    public IBinder onBind(Intent intent) {
        Log.i(TAG, "onBind方法被调用!");
        return binder;
    }

    //Service被创建时回调
    @Override
    public void onCreate() {
        super.onCreate();
        Log.i(TAG, "onCreate方法被调用!");
        //创建一个线程动态地修改count的值
        new Thread()
        {
            public void run()
            {
                while(!quit)
                {
                    try
                    {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {e.printStackTrace();
Trace();}

                    count++;
                }
            }
        };
    }
}
```



```

        }.start();

    }

    //Service断开连接时回调
    @Override
    public boolean onUnbind(Intent intent) {
        Log.i(TAG, "onUnbind方法被调用!");
        return true;
    }

    //Service被关闭前回调
    @Override
    public void onDestroy() {
        super.onDestroy();
        this.quit = true;
        Log.i(TAG, "onDestroyed方法被调用!");
    }

    @Override
    public void onRebind(Intent intent) {
        Log.i(TAG, "onRebind方法被调用!");
        super.onRebind(intent);
    }
}

```

在AndroidManifest.xml中对Service组件进行注册:

```

<service android:name=".TestService2" android:exported="false">

    <intent-filter>
        <action android:name="com.jay.example.service.TEST_
SERVICE2"/>
    </intent-filter>
</service>

```

MainActivity.java:

```

public class MainActivity extends Activity {

    private Button btnbind;
    private Button btncancel;
    private Button btnstatus;

    //保持所启动的Service的IBinder对象,同时定义一个ServiceConnectio
n对象
    TestService2.MyBinder binder;
    private ServiceConnection conn = new ServiceConnection() {

        //Activity与Service断开连接时回调该方法
        @Override

```

```

        public void onServiceDisconnected(ComponentName name) {

            System.out.println("-----Service DisConnected-----
--");
        }

        //Activity与服务连接成功时回调该方法
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {

            System.out.println("-----Service Connected-----"
);

            binder = (TestService2.MyBinder) service;
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnbind = (Button) findViewById(R.id.btnbind);
        btncancel = (Button) findViewById(R.id.btncancel);
        btnstatus = (Button) findViewById(R.id.btnstatus);
        final Intent intent = new Intent();
        intent.setAction("com.jay.example.service.TEST_SERVICE2
");

        btnbind.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                //绑定service
                bindService(intent, conn, Service.BIND_AUTO_CREATE);
            }
        });

        btncancel.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                //解除service绑定
                unbindService(conn);
            }
        });

        btnstatus.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(), "Service的count的值为:"
                    + binder.getCount(), Toast.LENGTH_SHORT
                ).show();
            }
        });
    }

```

}



反馈

运行截图：



点击锁定Service:

```

EGL emulation  eglSurfaceAttrib not implemented
TestService2   onCreate方法被调用!
TestService2   onBind方法被调用!
System.out     -----Service Connected-----

```

继续点击锁定:没有任何变化

```

EGL emulation  eglSurfaceAttrib not implemented
TestService2   onCreate方法被调用!
TestService2   onBind方法被调用!
System.out     -----Service Connected-----

```

获取当前Service的状态:



解除绑定:

```

TestService2   onCreate方法被调用!
TestService2   onBind方法被调用!
System.out     -----Service Connected-----
TestService2   onUnbind方法被调用!
TestService2   onDestroy方法被调用!

```

如果我们再绑定后直接关掉Activity的话会报错, 然后会自动调用onUnbind和onDestory方法!

```

Activity com.jay.example.servicetestdemo2.MainActivity has leaked S
erviceConnection com.jay.example.servicetestdemo2.MainActivity$1@53
235ae4 that was originally bound here
android.app.ServiceConnectionLeaked: Activity com.jay.example.servi
cetestdemo2.MainActivity has leaked ServiceConnection com.jay.examp
le.servicetestdemo2.MainActivity$1@53235ae4 that was originally bou
nd here

```

...

```
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:560)
at dalvik.system.NativeStart.main(Native Method)
onUnbind方法被调用!
onDestroyed方法被调用!
```

从上面的运行结果验证了生命周期图中的:

使用BindService绑定Service,依次调用onCreate(),onBind()方法, 我们可以在onBind()方法中返回自定义的IBinder对象;再接着调用的是ServiceConnection的onServiceConnected()方法该方法中可以获得IBinder对象,从而进行相关操作;当Service解除绑定后会自动调用onUnbind和onDestroyed方法,当然绑定多客户端情况需要解除所有的绑定才会调用onDestoryed方法进行销毁哦！

← 4.1.3 Activity登堂入室

4.2.2 Service进阶 →



在线实例

- HTML 实例
- CSS 实例
- JavaScript 实例
- Ajax 实例
- jQuery 实例
- XML 实例
- Java 实例

字符集&工具

- HTML 字符集设置
- HTML ASCII 字符集
- HTML ISO-8859-1
- HTML 实体符号
- HTML 拾色器
- JSON 格式化工具

最新更新

- Swift 正式开源
- PHP 7 正式发布
- Shell 编程快速入门
- Shell 文件包含
- Shell 输入/输出...
- Shell printf 命令
- Shell 基本运算符

站点信息

- 意见反馈
- 免责声明
- 关于我们
- 文章归档

关注微信



Copyright © 2013-2015 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1