抖音面试题

笔试部分

（面试官也不知道为啥hr给了我一套社招的题）

1.中间透明、四周蒙层遮蔽的效果是如何实现的

2.关于promise的一段代码的执行结果是什么

3.输出结果是什么？this.name中的this指谁？

window.name = "byteDance";

function A() {

this.name = 123;

}

A.prototype.getA = function() {

console.log(this.name);

return this.name + 1;

}

let a = new A();

let funA = a.getA;

funA();

4.给一个字符串，找出出现次数最多的字符和对应的次数。（编程）

5.最大子序和，时间复杂度O（n）

面试环节：

js

1.xss攻击是什么样的形式？如何防范？vue其实已经帮我们做了防范，v-html="xxx",xxx是什么

2.vue是单向绑定的，是如何实现数据改变、相应文档进行更改的？

3.user-scalable=no 表示什么？在IOS上即时设为no也是不生效的，如何解决？如果监听touch事件，callback里面写什么？直接return会有问题。

4.echarts是基于什么的？canvas的大小会随着窗口大小发生改变么？

5.写一个函数，实现深拷贝，要考虑所有数据类型

6.以下两段程序分别输出什么

var script=document.createElement('script');

script.src="xxx/xxx/React.js"

document.appendChild(script);

console.log(window.React);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var script=document.createElement('script');

script.text="alert(from script)";

console.log("before script");

document.appendChild(script);

console.log("after script");

7.项目使用的什么框架？vue的线上线下环境部署使用的什么技术？

8.preflighted Requests 预检请求了解么

html&css

1.让你实现一个弹窗，你会怎么设计文档结构？会考虑哪些问题？半透明背景的设置rgba和opacity有什么区别？这两个有一个用在这里会有问题；点击穿透怎么解决（是阻止默认事件还是阻止冒泡事件？）

2.在页面加载时考虑过哪些优化

3.图片懒加载

4.商品评价的时候，默认5星，评分3分、3.5分，怎么实现？评分3.6、3.7呢

http

1.与缓存相关的header有哪几个？分别是什么内容？（last-Modified之类的）如果是last-modified，服务器查询了下，没有更新数据，会给客户端返回哪个码？

2.http是基于什么协议的？TCP三次握手与四次挥手的过程。

3.dns解析的过程

4.什么情况下属于跨域？跨域的解决方式（jsonp了解下）

5.说一下cookie与session。cookie添加到了http请求的哪里，设置什么字段就可以携带cookie?

6.从用户点击注册开始，都发生了什么

J2SE篇幅

1.Java里头的多态是什么意思？

其实回答这个问题很简单，大家可以直接列举出形成多态具备的条件就可以了

首先必须要有继承或者实现

其次是要有方法的重写

最后一个是父类引用指向子类对象

大多数情况下回答这三个就ok了，再具体点直接给面试官举个例子：

动物类 Animal   狗Dog extends Animal   猫Cat extends Animal   --->>继承

Animal有个方法 吃得方法eat()   狗不满意这个父类的吃法，自己重写了eat() 猫也不满意这个父类的吃饭，自己也重写了eat() --->>重写

最后 Animal animal = new Dog(); --->>父类引用指向子类对象

       animal.eat();调用这个方法实际执行的是狗的吃的方法，这就是多态，实际运行时动态绑定。

2.GC的原理

当堆内存里的对象没有引用指向了，GC就来了。一句话言简意赅

3.抽象类跟接口的区别

抽象类可以有非抽象的方法，接口只有方法的定义没有实现，实现接口必须重写接口所有定义的方法，接口里头变量是static final的常量

4.StringBuffer跟StringBuider的区别

StringBuider非线程安全，执行速度快，单线程用这个类

5.sleep(), wait()的区别

sleep不释放同步锁，自动唤醒，需要try catch,  wait释放同步锁，需要notify来唤醒

sleep是线程的方法    wait是Object的方法

6.递归执行路径有几条。

主要是考察你对java线程的执行路径，其实递归并不是开启线程，虽然是重复调用方法本身，其实执行路径就一条。

7.LinkedList跟ArrayList的区别

一个链表结构，一个数组结构，LinkedList查找慢，插入快，ArrayList查找快，插入慢。

Android篇幅

1.手写一个线程安全的单例

public class SigleInstance {

private static SigleInstance instance;

public static SigleInstance getInstance() {

        if (instance == null) {

            syncronized(SigleInstance.class) {

                    if (instance == null) {

                            instance = new SingleInstance();

                    }

            }

       }

        return instance;

}

}

这个算是目前来说最有的单例，最好是能手写出来，面试官主要是考你为什么同步关键字要写在方法里头，写在那块，其实你只要答到点上就ok了，主要原因就是不用频繁加锁，因为频繁加锁会消耗性能，当instance new出来之后，可以不用在方法外加锁就直接返回实例，减少了一次加锁的操作。

2.Bitmap图像模式有哪几种，给出一张1080 \* 1920的，ARGB 8888格式的占用内存是多大

ARGB 8888一个像素占用4个字节，一个字节8位，1080 \* 1920 \* 4 \* 8

3.静态方法跟静态代码块有什么区别。

静态方法属于类，用类名.就能直接调用，静态代码块是类加载的时候就会调用的

4.Android进程有哪几种

前台进程，可见进程，服务进程，后台进程，空进程

5.Runnable , Callable, Feature三个的区别

Runnable 没有返回值，Callable有返回值，Feature一般配合Callable来使用，用于检测任务的状态，具有有是否执行完成，还可以做取消任务操作，再可以稍微撤下AsyncTask源码，因为有返回值，其实底层也是用Callable跟Feature来实现的， 显得你会懂一些源码，会是加分项

6.一个应用允许多个进程吗？

允许，最直接的直接说在功能清单文件里配置Service为一个独立进程，android：process就能配置

7.多进程会引起一些什么样的异常呢？

（1)静态成员和单例模式完全shixiao

  (2)线程同步机制完全失效

  (3)SharedPreferences的可靠性下降

  (4)Application会多次创建

8.HashMap的底层数据结构

  数组加链表

9.Serializable和Parcelable的区别

Serializable是java中序列化接口，使用开销很大，序列化和反序列化过程需要大量的I/0操作，Parcelable是Android中序列化的方式，用起来虽然麻烦一些，效率很高，Parcelable主要用在内存序列化上，如果说要序列化到存储设备中，建议大家使用Serializable,具体通过Object流来做序列化操作。

1.什么是接口？Android中有哪些机制是基于接口编程？

接口就是一些方法特征的集合，用interface修饰。

面向接口编程：http://blog.csdn.net/wangjinyu501/article/details/74355351

2.说明Activity和Service的生命周期？

3.Intent flag有哪些？作用是什么？

自己看（25种） ：https://www.jianshu.com/p/08177910b0a2

4.Android的多线程模型有哪几种？

Android提供了四种常用的操作多线程的方式，分别是：

1. Handler+Thread

2. AsyncTask

3. ThreadPoolExecutor

4. IntentService

详细介绍：https://www.jianshu.com/p/2b634a7c49ec

5.如何规避oom?

1.使用更加轻量的数据结构

2.避免在Android里面使用Enum

3.减小Bitmap对象的内存占用

4.使用更小的图片

5.复用系统自带的资源

6.注意在ListView/GridView等出现大量重复子组件的视图里面对ConvertView的复用

7.Bitmap对象的复用

8.避免在onDraw方法里面执行对象的创建

9.避免对象的内存泄露(重点)

10.考虑使用Application Context而不是Activity Context

11.注意WebView的泄漏(重点)

12.资源文件需要选择合适的文件夹进行存放

13.谨慎使用static对象(重点)

14.特别留意单例对象中不合理的持有

15.珍惜Services资源

16.谨慎使用“抽象”编程

17.谨慎使用依赖注入框架

18..谨慎使用多进程

19.Handler的使用（重点）

20.强软弱虚引用的应用（重点）

22.主线程操作UI，子线程操作数据（必填）

原因地址：http://blog.csdn.net/ljx19900116/article/details/50037627

6.数据存储有哪些方式？

1.sharedpreferences 2.file 3.Sqlite 4.ContentProvide 5.网络存储

7.如何做多设备支持？

个人理解为屏幕适配，不知道出题者的意图

8.Android的布局方式有哪些？分别是如何实现OnDraw的？

LinearLayout，RelativeLayout，TableLayout，FrameLayout，AbsoluteLayout，GridLayout

9.ScrollView是否可以和listView混合使用？如何可以，说明混合使用的方式，如果不行，说明原因。

可以，计算整个ListView的高度，填充数据后重新设置ListView高度，重写onMeasure和onInterceptTouchEvent方法

10.在创建fragment时如何传递初始化参数？

Fragment初始化一定要提供默认构造函数。不能用构造函数传递参数！不要写带参数的构造函数。在Fragment里添加获取Fragment的newInstance函数，以后获取Fragment就使用这个函数，不要使用构造函数新建Fragment！使用setArgument和getArgument传递参数

---------------------

作者：playboy1314

来源：CSDN

原文：https://blog.csdn.net/qq\_32865887/article/details/79398095

版权声明：本文为博主原创文章，转载请附上博文链接！

1、四大组件是什么？

Activity【活动】：用于表现功能。

Service【服务】：后台运行服务，不提供界面呈现。

BroadcastReceiver【广播接收器】：用来接收广播。

Content Provider【内容提供商】：支持在多个应用中存储和读取数据，相当于数据库。

2、四个组件的生命周期？

Activity生命周期图及 Fragment生命周期图

Service的生命周期：首先Service有两种启动方式，而在这两种启动方式下，它的生命周期不同。

通过startService()方法启动的服务

      初始化结束后系统会调用 void onStart(Intent intent) 方法，用于处理传递给startService()的Intent对象。如音乐服务会打开Intent 来探明将要播放哪首音乐，并开始播放。注意：多次调用startService()方法会多次触发onStart()方法。

通过bindService ()方法启动的服务

      初始化结束后系统会调用 IBinder onBind(Intent intent) 方法，用来绑定传递给bindService 的Intent 的对象。注意：多次调用bindService()时，如果该服务已启动则不会再触发此方法。

3、Activity的四种启动模式对比？

Standard:标准的启动模式，如果需要启动一个activity就会创建该activity的实例。也是activity的默认启动模式。

SingeTop:如果启动的activity已经位于栈顶，那么就不会重新创建一个新的activity实例。而是复用位于栈顶的activity实例对象。如果不位于栈顶仍旧会重新创建activity的实例对象。

SingleTask:设置了singleTask启动模式的activity在启动时，如果位于activity栈中，就会复用该activity，这样的话，在该实例之上的所有activity都依次进行出栈操作，即执行对应的onDestroy()方法，直到当前要启动的activity位于栈顶。一般应用在网页的图集，一键退出当前的应用程序。

singleInstance:如果使用singleInstance启动模式的activity在启动的时候会复用已经存在的activity实例。不管这个activity的实例是位于哪一个应用当中，都会共享已经启动的activity的实例对象。使用了singlestance的启动模式的activity会单独的开启一个共享栈，这个栈中只存在当前的activity实例对象。

4、Activity在有Dialog时按Home键的生命周期？

当我们的Activity上弹出Dialog对话框时，程序的生命周期依然是onCreate() ---> onStart() ---> onResume()，在弹出Dialog的时候并没有onPause()和onStop()方法。而在此时我们按下Home键，才会继续执行onPause()和onStop()方法。这说明对话框并没有使Activity进入后台，而是在点击了Home键后Activity才进入后台工作。

原因就是，其实Dialog是Activity的一个组件，此时Activity并不是不可见，而是被Dialog组件覆盖了其他的组件，此时我们无法对其他组件进行操作而已。

5、两个Activity 之间跳转时必然会执行的是哪几个方法？

首先定义两个Activity，分别为A和B。

当我们在A中激活B时，A调用onPause()方法，此时B出现在屏幕时，B调用onCreate()、onStart()、onResume()。

这个时候B【B不是一个透明的窗体或对话框的形式】已经覆盖了A的窗体，A会调用onStop()方法。

6、 前台切换到后台，然后再回到前台，Activity生命周期回调方法。弹出Dialog，生命值周期回调方法？

首先定义两个Activity，分别为A和B。

完整顺序为：A调用onCreate()方法 ---> onStart()方法 ---> onResume()方法。当A启动B时，A调用onPause()方法，然后调用新的Activity B，此时调用onCreate()方法 ---> onStart()方法 ---> onResume()方法将新Activity激活。之后A再调用onStop()方法。当A再次回到前台时，B调用onPause()方法，A调用onRestart()方法 ---> onStart()方法 ---> onResume()方法，最后调用B的onStop()方法 ---> onDestory()方法。

弹出Dialog时，调用onCreate()方法 ---> onStart()方法 ---> onResume()方法。

7、fragment各种情况下的生命周期？

 由于Fragment的生命周期与Activity的生命周期有着牵扯，所以把两者的图放到一起作为对比理解。

 接下来就不同情况下的Fragment生命周期做一简单介绍：

Fragment在Activity中replace

新替换的Activity：onAttach() ---> onCreate() ---> onCreatView() ---> onViewCreated ---> onActivityCreated() ---> onStart --->onResume()

被替换的Activity：onPause() ---> onStop() ---> onDestoryView() ---> onDestory() ---> onDetach()

Fragment在Activity中replace，并addToBackStack

新替换的Fragment（没有在BackStack中）：onAttach > onCreate > onCreateView > onViewCreated > onActivityCreated > onStart > onResume

新替换的Fragment（已经在BackStack中）：onCreateView > onViewCreated > onActivityCreated > onStart > onResume

被替换的Fragment：onPause > onStop > onDestroyView

Fragment在ViewPager中切换

我们称切换前的的Fragment称为PreviousFragment，简称PF；切换后的Fragment称为NextFragment，简称NF；其他Fragment称为OtherFragment，简称OF。

（在ViewPager中setUserVisibleHint能反映出Fragment是否被切换到后台或前台，所以在这里也当作生命周期）

如果相关的Fragment没有被加载过：

NF： setUserVisibleHint(false)【用户不可见】 > onAttach > onCreate > setUserVisibleHint(true)【用户可见】 > onCreateView > onViewCreated > onActivityCreated > onStart > onResume

OF跟NF相邻： setUserVisibleHint(false) > onAttach > onCreate > onCreateView > onViewCreated > onActivityCreated > onStart > onResume

如果相关的Fragment已经被加载过：

NF跟PF相邻  ：setUserVisibleHint(true)

NF跟PF不相邻：setUserVisibleHint(true) > onCreateView > onViewCreated > onActivityCreated > onStart > onResume

PF跟NF相邻  ：setUserVisibleHint(false)

PF跟NF不相邻：setUserVisibleHint(false) > onPause > onStop > onDestroyView

OF跟PF相邻：onPause > onStop > onDestroyView

OF跟NF相邻：onCreateView > onViewCreated > onActivityCreated > onStart > onResume

OF夹在PF和NF中间：不调用任何生命周期方法

NF跟PF相邻  ：setUserVisibleHint(true)

NF跟PF不相邻：setUserVisibleHint(true) > onCreateView > onViewCreated > onActivityCreated > onStart > onResume

PF跟NF相邻  ：setUserVisibleHint(false)

PF跟NF不相邻：setUserVisibleHint(false) > onPause > onStop > onDestroyView

OF跟PF相邻：onPause > onStop > onDestroyView

OF跟NF相邻：onCreateView > onViewCreated > onActivityCreated > onStart > onResume

OF夹在PF和NF中间：不调用任何生命周期方法

如果重写了FragmentPagerAdapter的DestroyItem方法，并且相关的Fragment已经加载过：

相互切换时只会调用setUserVisibleHint

Fragment进入了运行状态：

Fragment在进入运行状态时，以下四个生命周期会随它所属的Activity一起被调用：

onPause() ---> onStop() ---> onStart() ---> onResume()

关于Fragment的onActivityResult方法：

使用Fragment的startActivity方法时，FragmentActivity的onActivityResult方法会回调相应的Fragment的onActivityResult方法，所以在重写FragmentActivity的onActivityResult方法时，注意调用super.onActivityResult。

8、 如何实现Fragment的滑动？

将Fragment与viewpager绑定，通过viewpager中的touch事件，会进行move事件的滑动处理。

Fragment布局

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="@android:color/holo\_red\_light">

<TextView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:text="Fragment One" />

</LinearLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="@android:color/holo\_red\_light">

<TextView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:text="Fragment Two" />

</LinearLayout>

Fragment代码：

public class FragmentOne extends Fragment {

@Nullable

@Override

public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, Bundle savedInstanceState) {

return inflater.inflate(R.layout.fragment\_one, container, false);

}

}

public class FragmentTwo extends Fragment {

@Nullable

@Override

public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, Bundle savedInstanceState) {

return inflater.inflate(R.layout.fragment\_Two, container, false);

}

}

viewpager布局：

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.example.spreadtrumshitaoli.fragmentscroll.MainActivity">

<android.support.v4.view.ViewPager

android:id="@+id/view\_pager"

android:layout\_height="match\_parent"

android:layout\_width="match\_parent"/>

</android.support.constraint.ConstraintLayout>

MainActivity代码：

public class MainActivity extends AppCompatActivity {

private FragmentOne fragmentOne;

private FragmentTwo fragmentTwo;

private ViewPager viewPager;

private ArrayList<Fragment> mFragmentList = new ArrayList <Fragment>();

private FragmentPagerAdapter fragmentPagerAdapter;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

init();

}

private void init() {

viewPager = (ViewPager) findViewById(R.id.view\_pager);

fragmentOne = new FragmentOne();

fragmentTwo = new FragmentTwo();

mFragmentList.add(fragmentOne);

mFragmentList.add(fragmentTwo);

//将adapter和fragment绑定在一起。

fragmentPagerAdapter = new FragmentPagerAdapter(getSupportFragmentManager()) {

@Override

public Fragment getItem(int i) {

return mFragmentList != null ? mFragmentList.get(i) : null;

}

@Override

public int getCount() {

return mFragmentList != null ? mFragmentList.size() : 0;

}

};

viewPager.setAdapter(fragmentPagerAdapter);

viewPager.setOnPageChangeListener(new ViewPager.OnPageChangeListener() {

@Override

public void onPageScrolled(int i, float v, int i1) {

}

@Override

public void onPageSelected(int i) {

//TODO:

}

@Override

public void onPageScrollStateChanged(int i) {

}

});

}

}

在这段代码中，我们

首先fragment以及viewpager都实例化；

再将fragment添加到泛型arraylist里；

最后将带有fragment的arraylist和adapter绑定。

9、fragment之间传递数据的方式？

方法一：

1、在MainFragment中设置一个setData()方法，在方法中设置更改按钮名称;

//MainFragment.java文件中

public void setData(String string) {

bt\_main.setText(string);

}

2、在MenuFragment中的ListView条目点击事件中通过标签获取到MainFragment，并调用对应的setData()方法，将数据设置进去，从而达到数据传递的目的。

lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

MainFragment mainFragment =

(MainFragment) getActivity()

.getSupportFragmentManager()

.findFragmentByTag("mainFragment");

mainFragment.setData(mDatas.get(position));

}

});

只需上面区区两步即可达到数据传递的目的。

方法二：

采取接口回调的方式进行数据传递。

step1: 在Menuragment中创建一个接口以及接口对应的set方法：

//MenuFragment.java文件中

public interface OnDataTransmissionListener {

public void dataTransmission(String data);

}

public void setOnDataTransmissionListener(OnDataTransmissionListener mListener) {

this.mListener = mListener;

}

step2: 在MenuFragment中的ListView条目点击事件中进行接口进行接口回调

//MenuFragment.java文件中

lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

/\*\*

\* 方法二：采取接口回调的方式进行数据传递

\*/

if (mListener != null) {

mListener.dataTransmission(mDatas.get(position));

}

}

});

step3: 在MainActivity中根据menuFragment获取到接口的set方法，在这个方法中进行进行数据传递，具体如下：

//在MainActivity.java中

menuFragment.setOnDataTransmissionListener(new MenuFragment.OnDataTransmissionListener() {

@Override

public void dataTransmission(String data) {

mainFragment.setData(data); //注：对应的mainFragment此时应该要用final进行修饰

}

});

通过上面的三步也可以轻松做到Fragment数据之间的传递。

方法三：

使用三方开源框架：EventBus

那么问题来了：EventBus是个啥东西？？？

简单来说，EventBus是一款针对Android优化的发布/订阅（publish/subscribe）事件总线。主要功能是替代Intent,Handler,BroadCast在Fragment，Activity，Service，线程之间传递消息。简化了应用程序内各组件间、组件与后台线程间的通信。优点是开销小，代码更优雅，以及将发送者和接收者解耦。比如请求网络，等网络返回时通过Handler或Broadcast通知UI，两个Fragment之间需要通过Listener通信，这些需求都可以通过EventBus实现。

下面我们就用EventBus来实现以下Fragment之间的数据传递：

step1：引入EventBus

compile 'org.greenrobot:eventbus:3.0.0'

step2：注册事件接收者

这里MainFragment是要接收MenuFragment发送来的数据，所以我们在MainFragment中的onCreateView()方法中进行注册：

EventBus.getDefault().register(this);

step3：发送事件

注：发送事件之前其实还有一步定义事件类型，这里我们传递的数据只有一个类型，所以这一步取消了。

MenuFragment发送数据给MainFragment，所以我们在MenuFragment中将要传递的数据进行发送事件操作：

lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

EventBus.getDefault().post(mDatas.get(position));

}

});

step4：接收消息并处理

在MainFragment中我们接收来自MenuFragment传递过来的数据,并进行对应的处理（注：EventBus 3.0版本这一步必须要写注解@Subscribe (与2.4版本有所区别)）：

@Subscribe

public void onEvent(String data) {

bt\_main.setText(data);

}

通过上面这一步即可完成数据之间的传递，需要注意的是在销毁的时候我们要注销事件接收。

step5：注销事件接收

//MainFragment.java中

@Override

public void onDestroy() {

super.onDestroy();

EventBus.getDefault().unregister(this);

}

以上五步完成了Fragment之间的数据传递，看似比上面两个方法要复杂的多，但当我们涉及到复杂的Fragment之间数据传递（例如Fragment中嵌套多层Fragment）时，就会体会到EventBus的爽快之处~~~这里不进行赘述了。

10、Activity 怎么和Service 绑定？

这需要实现service中的onBind()函数以返回service实例给activity

1、创建service类和activity类。

2、在service类中定义一个内部类继承自Binder()类：

public class MyBinder extends Binder{

        public Service1 getService(){

            return Service1.this;

        }

    }

实例化onBind()方法：

private final IBinder binder = new MyyBinder();

@Override

public IBinder onBind(Intent intent){

        Log.i(LOG，"onBind......");

        return binder;

}

3、在activity中完成绑定

Intent intent = new Intent(Activity1.this,Activity2.class);

          bindService(intent,conn,Context.BIND\_AUTO\_CREATE);

bindService的第二个参数是一个ServiceConnection类型的参数。service和其他组件之间的连接都表示为一个ServiceConnection，要想将service和其他组件进行绑定，就需要实现一个新的ServiceConnection。

public ServiceConnection conn= new ServiceConnection() {

        @Override

        public void onServiceDisconnected(ComponentName name) {

//当连接意外断开时调用

            Log.i(LOG, "onServiceDisconnected>>>>>>>>");

            myservice = null;

        }

        @Override

        public void onServiceConnected(ComponentName name, IBinder service) {

//当建立连接时调用

            Log.i(LOG, "onServiceConnected>>>>>>>>");

            myservice = ((Service1.MyBinder)service).getService();

        }

    };

bindService的第三个参数是一个flag。

   可以使用的flag有：

BIND\_AUTO\_CREATE：绑定完成后就启动目标service

BIND\_DEBUG\_UNBIND：这只在debug时使用，跟unbind有关。

BIND\_NOT\_FOREGROUND：确保被绑定的service永远不会有运行于前台的优先级，因为默认情况下，绑定一个service会提高它的优先级

BIND\_ABOVE\_CLIENT：确保客户端处于前台时，绑定的service也变为前台进程

BIND\_ALLOW\_OOM\_MANAGEMENT：允许系统在低内存等状态下删除该service（这是自己对源码中注释的理解）

BIND\_WAIVE\_PRIORITY：绑定service时不改变其优先级

BIND\_ADJUST\_WITH\_ACTIVITY：系统根据service所绑定的activity的重要程度来调整这个service的优先级。

11、service生命周期？

1.    被启动的服务(startService())的生命周期。

             如果一个Service被某个Activity 调用Context.startService() 方法启动，那么不管是否有Activity使用bindService()绑定或unbindService()解除绑定到该Service，该Service都在后台运行。如果一个Service被多次执行startService()，它的onCreate()方法只会调用一次，也就是说该Service只会创建一个实例，而它的onStartCommand()将会被调用多次(对应调用startService()的次数)。该Service将会一直在后台运行，直到被调用stopService()，或自身的stopSelf方法。当然如果系统资源不足，系统也可能结束服务。

2.    被绑定的服务(bindService())的生命周期。

             如果一个Service被调用 Context.bindService ()方法绑定启动，不管调用bindService()调用几次，onCreate()方法都只会调用一次，而onStartCommand()方法始终不会被调用，这时会调用onBind()方法。当连接建立之后，Service将会一直运行，除非调用Context.unbindService() 断开连接或者之前调用bindService() 的 Context 不存在了(如该Activity被finish)，系统将会自动停止Service，对应onDestroy()将被调用。

3.    被启动又被绑定的服务的生命周期。

             如果一个Service又被启动又被绑定，则该Service将会一直在后台运行。调用unbindService()将不会停止Service，而必须调用stopService()或Service的stopSelf()方法来停止服务。

4.   当服务被停止时清除服务。

            当一个Service被终止时，Service的onDestroy()方法将会被调用，在这里应当做一些清除工作，如停止在Service中创建并运行的线程等。

12、 activity和service的绑定方式以及怎么在Activity 中启动自己对应的Service？

1、activity能进行绑定得益于Serviece的接口。为了支持Service的绑定，实现onBind方法。

2、Service和Activity的连接可以用ServiceConnection来实现。需要实现一个新的ServiceConnection，重现onServiceConnected和OnServiceDisconnected方法，一旦连接建立，就能得到Service实例的引用。

3、执行绑定，调用bindService方法，传入一个选择了要绑定的Service的Intent(显示或隐式)和一个你实现了的ServiceConnection的实例

13、Service的启动方式？

采用Context.startService()方法启动服务，在服务未被创建时，系统会先调用服务的onCreate()方法，接着调用onStart()方法。如果调用startService()方法前服务已经被创建，多次调用startService()方法并不会导致多次创建服务，但会导致多次调用onStart()方法。采用startService()方法启动的服务，只能调用Context.stopService()方法结束服务，服务结束时会调用onDestroy()方法。

采用Context.bindService()方法启动服务，在服务未被创建时，系统会先调用服务的 onCreate()方法，接着调用onBind()方法。这个时候调用者和服务绑定在一起，调用者退出了，系统就会先调用服务的onUnbind()方 法，接着调用onDestroy()方法。如果调用bindService()方法前服务已经被绑定，多次调用bindService()方法并不会导致 多次创建服务及绑定(也就是说onCreate()和onBind()方法并不会被多次调用)。如果调用者希望与正在绑定的服务解除绑定，可以调用 unbindService()方法，调用该方法也会导致系统调用服务的onUnbind()-->onDestroy()方法。

14、谈谈ContentProvider、ContentResolver、ContentObserver之间的关系？

ContentProvider：

\* 四大组件的内容提供者，主要用于对外提供数据

\* 实现各个应用程序之间的（跨应用）数据共享，比如联系人应用中就使用了ContentProvider，你在自己的应用中可以读取和修改联系人的数据，不过需要获得相应的权限。其实它也只是一个中间人，真正的数据源是文件或者SQLite等

\* 一个应用实现ContentProvider来提供内容给别的应用来操作，通过ContentResolver来操作别的应用数据，当然在自己的应用中也可以

ContentResolver：

\* 内容解析者，用于获取内容提供者提供的数据

\* ContentResolver.notifyChange（uri）发出消息

ContentObserver：

\* 内容监听器，可以监听数据的改变状态

\* 目的是观察（捕捉）特定Uri引起的数据库的变化，继而做一些相应的处理，它类似于数据库技术中的触发器（Trigger），当ContentObserver所观察的Uri发生变化时，便会触发它。触发器分为表触发器、行触发器，相应地ContentObsever也分为表ContentObserver、行ContentObserver，当然这是与它所监听的Uri MIME Type有关的

\* ContentResolver.registerContentObserver()监听消息

15、广播的分类？

 分为有序广播和无序广播两类。

无序广播发送代码：

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

public void click(View v){

//启动界面 startActivity

//发送广播 sendBroadcast

Intent intent = new Intent();

intent.setAction("com.itheima.cctv.action.NEWS");

intent.putExtra("data", "我是一个无须的广播");

sendBroadcast(intent);

}

}

无序广播的监听代码：

public class CctvReceiver extends BroadcastReceiver {

private static final String TAG = "CctvReceiver";

@Override

public void onReceive(Context context, Intent intent) {

String data = intent.getStringExtra("data");

Log.d(TAG, "data==="+data);

}

}

<receiver android:name="com.itheima.cctv.CctvReceiver">

<intent-filter >

<!--这里监听的广播就是上面发送广播设置的intent.setAction("com.itheima.cctv.action.NEWS");-->

<action android:name="com.itheima.cctv.action.NEWS"/>

</intent-filter>

</receiver>

有序广播发送：

public class ShengReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

Log.d("vivi", "我是省级部门，我收到的指令是："+getResultData());

//getResultData()是用来获取有序广播里面的数值.这里的信息是:

//主席讲话：每人奖励10斤土豆

setResultData("主席讲话：每人奖励7斤土豆");//有序广播的数值,可以被修改,后面的程序在接受到这个广播,就会变成,现在我们改变的值了

//有序广播传输是可以终止的.但是最终的接受者就算在终止之后,也是可以接受到数据的

//abortBroadcast();

}

}

public class ShiReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

Log.d("vivi", "我是市级部门，我收到的指令是："+getResultData());

//因为上面修改了数据,所以这里获取到的数据是:主席讲话：每人奖励7斤土豆

}

}

<!--有序广播的优先级别使用 android:priority=""来指定,最高是1000,最低是-1000-->

<receiver android:name="com.itheima.region.ShengReceiver">

<intent-filter android:priority="1000">

<action android:name="com.itheima.gov.action.POTATO"/>

</intent-filter>

</receiver>

<receiver android:name="com.itheima.region.ShiReceiver">

<intent-filter android:priority="500">

<action android:name="com.itheima.gov.action.POTATO"/>

</intent-filter>

</receiver>

 有序接收：

public class MyReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

Log.d("vivi", "我是恩恩主席的内线，我收到的指令是："+getResultData());

}

}

启动模式：

1、standard：标准化启动模式

        每启动一个Activity，都会重新创建Activity的新的实例，将其放在栈的顶部。不需要考虑这个实例是否已经存在。

        每一次启动，它的onCreate()、onStart()、onResume()方法都会被依次调用。

2、singleTop：栈顶复用模式

        当前栈中已经有该Activity实例，并且该实例位于栈顶时，会去调用onNewIntent()方法。

        当前栈中已有该Activity的实例但是该实例不在栈顶时，依然会去创建Activity。

        当前栈中不存在该Activity实例时，会去新创建一个该Activity。

应用场景：IM对话框、新闻客户端推送。

3、singleTask：栈内复用模式

         它主要检测【寻找，通过taskAffinity】整个栈中是否已经存在当前想要启动的Activity，存在的话直接将该Activity置于栈顶，之前位于该Activity上面的Activity将被销毁，同时调用onNewIntent()方法，而不存在的话进行创建。

应用场景：应用主界面。

4、singleInstance：

        一个人独享一个任务栈。当该Activity启动时，系统会创建一个新的任务栈，同时将Activity放到这个新的任务栈当中，有别的应用来启动该Activity时，由于栈内复用的特性，不会再去创建相应Activity任务栈，而是这两个应用独享一个Activity实例。

        例如：应用A中现有两个Activity E、Activity F，为standard启动模式，应用B中有一个Activity G，但其启动模式是singleInstance。应用A想用应用B任务栈当中的Activity G，尽管在不同的应用下，但是应用A仍然会直接复用Activity G。

        特性：

                1、以SingleInstance模式启动的Activity具有全局唯一性【全局唯一性即指在整个系统当中只会存在一个这样的实例】；

                2、如果在启动这样一个Activity时，【整个系统都是单例的】，已经存在了一个实例；

                3、以SingleInstance模式启动的Activity具有独占性。

应用场景：呼叫来电。

网络协议：

        协议：【协议指计算机通信网络中两台计算机之间进行通信所必须共同遵守的规定或规则】

        HTTP协议

        基本概念：【超文本传输协议】允许将HTML（超文本标记语言）文档从Web服务器传送到客户端的浏览器。HTTP协议是                                 基于TCP/IP通信协议来传输数据的，可以从服务器端获取图片等数据资源。

        URI：【uniform resource identifier】统一的资源标识符，用来唯一的标识一个资源。强调资源！！！

        组成部分：

                        1）访问资源的命名机制；file

                        2）存放资源的主机名；

                        3）资源自身的名称，由路径表示，着重于强调资源。

         例：file://a:1234/b/c/d.txt   表示资源目标在a主机下的1234端口的b目录下的c目录下的d.txt文件。

        URL：【uniform resource Location】统一资源定位器，是一种具体的URI。即URL可以用来标识一个资源，而且还指明了如                       何定位这个资源。强调路径！！！

        组成部分：

                        1）协议；

                        2）存有该资源的主机IP地址；

                        3）主机资源的具体地址。

        HTTP协议特点：

                                   1）简单快速；

                                   2）无连接；【限制每次链接只处理一个请求，服务器处理完客户的请求之后会收到客户端的应答，再断开链                                          接，节省了重复的时间】；

                                   3）无状态：【没有记忆能力，】

       HTTP协议的request/response请求头原理剖析：

       request有可能经过代理服务器到达web服务器

       代理服务器最主要的作用：提高访问速度【大部分代理服务器都具有缓存功能，当你再次访问前一个网络请求时，就可以直          接从代理服务器中获取，而不需要请求我们的web服务器】。

       HTTP协议容易混淆知识点：

       （1）http1.0与http1.1的具体区别：

                  http处于计算机网络的应用层。

                   1）缓存处理

                   2）带宽优化及网络连接的使用

                   3）Host头使用：1.1上请求操作和响应操作都支持Host头，而且在请求消息中如果没有Host头的话会报告一个400错                      误。

                   4）长连接：在一个TCP连接上，可以传送多个HTTP请求和响应，而不是发送一个HTTP请求就断开一个连接，再发                      送一个HTTP请求再建立一个连接。

                存在的问题：

                                    1）传输数据时，每次都需要重新创建连接，增加了大量的延迟时间；

                                    2）传输数据时，所有传输的内容都是明文，客户端和服务器端都无法验证对方的身份；

                                    3）使用时，header里携带的内容过大，增加了传输成本。

       （2）get / post方法的区别：

                  1）提交的数据：get提交的数据一般会放在我们的URL之后，用“  ？”来分割；而post提交数据都会放在我们entity-                           body【消息主体】当中。

                  2）提交的数据大小是否有限制：get提交的数据是有限制的，因为url是有限制的，不能无限制的输入一个url地址；而                     post方法提交的是body，因此没有限制。

                  3）取得变量的值：get方法通过Request.QueryString()来取得变量的值，而post方法通过Request.Form来取得变量的                     值。

                  4）安全问题：get提交数据一定会带来安全问题

       （3）Cookie和Session的区别：

                  1）cookie【用户身份的标识】：客户端的解决方案。是由服务器发给客户端的特殊信息，而这些信息以文本文件的方                   式存放在客户端，然后客户端每次向服务器发送请求的时候都会带上这些特殊的信息。存放在响应头里面。

                       客户端 向服务端发送一个HTTP Request请求；

                       服务端给客户端一个HTTP Response ，并且把它的cookies设置给我们的客户端；

                       客户端将HTTP Request和cookies打包给我们的服务端；

                       服务端会根据客户端给我们的cookies来进行指定的判断，返回HTTP Response给我们的客户端。

                       此方法弥补了我们HTTP协议无状态的不足。之前当上述请求响应操作完成后，服务端和客户端就断开连接，服务端                        就无法从连接上跟踪我们所想要的客户端。

                  2）session：另一种记录客户状态的限制，cookie保存在客户端浏览器中，而session保存在服务器上。客户端浏览器                     访问服务器时，服务器把客户端信息以某种形式记录在服务器上。

                        创建session；

                        在创建session同时，服务器会为该session生成唯一的session id；

                        在session被创建之后，就可以调用session相关的方法往session中增加内容；

                        当客户端再次发送请求的时候，会将这个session id带上，服务器接收到请求之后就会依据session id找到相                           应的session。

                  3）区别：

                                 存放的位置不同；

                                 存取的方式不同【cookie保存的是Ascll码字符串，而session中能够保存任何类型的数据】；

                                 安全性上的不同【cookie存放在客户端浏览器中，对我们客户端可见，客户端的一些程序就有可能去修改我们                                  cookie的内容，而session则不然，它存储在服务端上，对客户端是透明的，不存在敏感信息泄露的风险】；

                                 有效期上的不同【一般我们会设置cookie过期时间， session依赖 id，若id设置为-1，当关掉浏览器session就                                  会失效】；

                                 对服务器的造成的压力不同【cookie保存在客户端不占用客户资源，session保存在服务端，每一个用户都会                                    产生一个session。在并发很多用户时cookie是一个很好的选择】。

        HTTPS协议：

        基本概念：对工作在以加密连接（SSL / TLS）上的常规HTTP协议。通过在TCP和HTTP之间加入TLS【Transport Layer               Security】来加密。

        SSL / TLS协议：安全传输协议，TLS是SSL的升级版，也是现阶段所使用的协议；

        HTTPS传输速度：

                                      1）通信慢；

                                      2）SSL必须进行加密处理。

        TLS / SSL握手：

                                      1）密码学原理

                                            对称加密：加密数据所用的密钥和解密数据所用的密钥相同。

                                            非对称加密：分私有密钥和公有密钥。

                                      2）数字证书：互联网通讯中标志通讯各方身份信息的一串数字。

                                      3）握手过程

Handler【Android SDK提供给开发者方便进行异步消息处理的类】：

          AsyncTask、retrofit都对Handler进行了封装。

       （1）Handler四大组件

                 1）Message

                       Message是在线程之间传递的消息，它可以在内部携带少量的信息，用于在不同线程之间交换数据。

                       例：Message的what字段、arg1字段、arg2字段来携带整型数据，obj字段携带一个Object对象。

                 2）Handler

                       处理者，它主要用来发送和处理消息。发送消息一般是使用Handler的sendMessage()方法，消息经过处理后，最终                        传递到Handler的handlerMessage()方法中。

                 3）MessageQueue

                      消息队列，它主要用来存放所有通过Handler发送的消息，这部分消息会一直存在于消息队列中，等待被处理。

                      注意：每个线程中只会有一个MessageQueue对象。

                 4）Looper

                     是每个线程中MessageQueue的管家，调用Looper的loop()方法后，就会进入到一个无限循环当中，每当发现                                  MessageQueue中存在一条消息，就会将其取出传递到Handler的handleMessage()方法当中。

                     注意：每个线程中只会有一个Looper对象。

                     异步消息处理流程：

                                                       1）在主线程当中创建一个Handler对象；

                                                       2）重写handleMessage()方法；

                                                       3）当子线程需要进行UI操作时，创建一个Message对象，并通过Handler将消息发送出去；

                                                       4）消息添加到MessageQueue的队列中等待被处理；

                                                       5）Looper在MessageQueue中取出待处理消息，发回Handler的handleMessage()方法中。

                    【由于Handler是在主线程中创建的，因此我们的handleMessage()方法中的代码也会在主线程中执行，避免了异常的                       产生】

   Handler消息机制：

作用：跨线程通信。当子线程中进行耗时操作后需要更新UI时，通过Handler将有关UI的操作切换到主线程中执行。

四要素：

Message（消息）：需要被传递的消息，其中包含了消息ID，消息处理对象以及处理的数据等，由MessageQueue统一列队，最终由Handler处理。

MessageQueue（消息队列）：用来存放Handler发送过来的消息，内部通过单链表的数据结构来维护消息列表，等待Looper的抽取。

Handler（处理者）：负责Message的发送及处理。通过 Handler.sendMessage() 向消息池发送各种消息事件；通过 Handler.handleMessage() 处理相应的消息事件。

Looper（消息泵）：通过Looper.loop()不断地从MessageQueue中抽取Message，按分发机制将消息分发给目标处理者。

具体流程如图

Handler.sendMessage()发送消息时，会通过MessageQueue.enqueueMessage()向MessageQueue中添加一条消息；

通过Looper.loop()开启循环后，不断轮询调用MessageQueue.next()；

调用目标Handler.dispatchMessage()去传递消息，目标Handler收到消息后调用Handler.handlerMessage()处理消息。

---------------------

作者：dsd2333

来源：CSDN

原文：https://blog.csdn.net/m0\_37868230/article/details/81457720

版权声明：本文为博主原创文章，转载请附上博文链接！