1. Android xml布局文件中尽量给每个控件指定layout\_weight和layout\_height,不然很可能会出现莫名其妙的错误。
2. CharSequence使用时注意，它可以用int赋值给它，但是还是会运行时失败。
3. SimpleAdapter是在具体的外部控件中调用setViewBinder方法将

**private ViewBinder mViewBinder**成员变量赋值，这时在内部就可以调用了，但是外部的具体控件必须实现ViewBinder这个接口定义的方法，这样就可以使得SimpleAdapter自己按照自己的意愿行事，而具体的方法留给外面想使用自己的类去实现。

**这里有种很好的设计思路，就是自己定义内部接口，如果外部想使用自己，那么就请实现自己的借口，而自己可以按照自己的意愿假装一切都是已经实现的进行编写自己的更复杂的逻辑功能。切记，这里说的自己定义接口给别人去实现，并不只是说自己声明接口方法，让外部具体类去实现，还可能是自己定义一个内部接口类，而自己内部定义一个该接口类型的成员，提供一个setter方法，使得外部具体类不必显示实现该接口，只要提供一个该接口的具体实现的实例调用setter方法，就可以使得二者默契工作。这样做的好处一点是，外部实现类即使是已经定义好了，但是任然可以使用，这里的**

**public static interface ViewBinder {**

**boolean setViewValue(View view, Object data, String textRepresentation);**

**}**

**就是一个很好的例子。**

1. Android的布局管理器也是ViewGroup的子类。
2. Android 中android:gravity是设置子元素的对其方式，而androd:layout\_gravity是设置自己在父容器中的对齐方式。
3. 在Activity的onCreate方法中，如果想对读取控件上的用户输入，记得要写在对应 的事件回调函数中获取，这是显然的事情，但是有时大脑短路时也是“郁闷的”。
4. BroadCastReceiver类的onReceive方法不应该定义新线程，而应该是开启新的Service去执行特定的操作，以为BroadCastReceiver的生命周期很短，有可能定义的新线程任务还没有执行完，BroadCastReceiver的生命周期就结束了，导致新的线程也被迫提前终止。
5. 但是BroadCastReceiver类中的OnReceive方法怎么可以操作UI 线程中的控件的呢？
6. 多线程中，并不是同步每个方法就可以保证对变量的互斥访问了，因为可能有些业务逻辑，是有多个同步方法串联调用完成的，这时如果在同步方法调用期间暂停，则就会发生意想不到的错误。
7. 在问到线程同步时，就不单要回到锁相关的知识，还要回答ThreadLocal相关的东西。

但是要注意，多线程中两个问题，互斥访问和线程同步，互斥有加锁和线程本地化存储两种解决方式，但是线程同步，只能通过信号量同步。

总之，ThreadLocal不是用来解决对象共享访问问题的，而主要是提供了保持对象的方法和避免参数传递的方便的对象访问方式。归纳了两点：   
1。每个线程中都有一个自己的ThreadLocalMap类对象，可以将线程自己的对象保持到其中，各管各的，线程可以正确的访问到自己的对象。   
2。将一个共用的ThreadLocal静态实例作为key，将不同对象的引用保存到不同线程的ThreadLocalMap中，然后在线程执行的各处通过这个静态ThreadLocal实例的get()方法取得自己线程保存的那个对象，避免了将这个对象作为参数传递的麻烦。

**public** T get() {

Thread t = Thread.*currentThread*();

ThreadLocalMap map = getMap(t);

**if** (map != **null**) {

ThreadLocalMap.Entry e = map.getEntry(**this**);

**if** (e != **null**)

**return** (T)e.value;

}

**return** setInitialValue();

}

**public** **void** set(T value) {

Thread t = Thread.*currentThread*();

ThreadLocalMap map = getMap(t);

**if** (map != **null**)

map.set(**this**, value);

**else**

createMap(t, value);

}

11. MainActivity中的getIntent有什么用途？

12. sdcard的读取怎样，怎样验证sdcard是否存在？

13. 今天犯了一个很二的问题：两个工程，其中一个工程使用第二个工程编译后打jar包，而在调试第一个工程时候，尽然在修改第二个工程打源码来看结果！！

14 . android 的meta-data

/×**读取Application中的meta-data ×/**

try {

ApplicationInfo appinfo = this.getPackageManager()

.getApplicationInfo(getPackageName(),PackageManager.GET\_META\_DATA);

String metaDataValue = appinfo.metaData.getString("ApplicationMetaData");

Toast.makeText(this,this.getClass().getName() + "TestApplication 的 metaData 值为 "+ metaDataValue, Toast.LENGTH\_LONG).show();

} catch (NameNotFoundException e) {

e.printStackTrace();

}

**/×读取Activity中的meta-data×/**

try {

ActivityInfo infor = this.getPackageManager()

.getActivityInfo(getComponentName(), PackageManager.GET\_META\_DATA);

String metaDataValue = infor.metaData.getString("activityMetaData");

Toast.makeText(this,getComponentName().getClassName()+ "Activity 的 metaData 值为 " + metaDataValue,Toast.LENGTH\_LONG).show();

} catch (Exception e) {

e.printStackTrace();

}

**/× 读取Receiver 中的meta-data ×/**

ComponentName cn = new ComponentName(context, TestReceiver.class);

try {

ActivityInfo info = context.getPackageManager().

getReceiverInfo(cn,PackageManager.GET\_META\_DATA);

Toast.makeText(context,this.getClass().getName()+" Receriver的metaData为 ："+ info.metaData.getString("receiverMetaData"),

Toast.LENGTH\_LONG).show();

} catch (NameNotFoundException e) {

e.printStackTrace();

}

**/× 读取service中的meta-data ×/**

ComponentName cn = new ComponentName(getApplicationContext(),TestService.class);

try {

ServiceInfo info = getApplicationContext().getPackageManager()

.getServiceInfo(cn, PackageManager.GET\_META\_DATA);

Toast.makeText(getApplicationContext(),getClass().getName() + "的 metaData的值为： "+ info.metaData.getString("serviceMetaData"),

Toast.LENGTH\_LONG).show();

} catch (NameNotFoundException e) {

e.printStackTrace();

}

15. java classLoader的东西

16.创建 Gerrit public key

cd ~

ssh-keygen

cat ~/.ssh/id\_rsa.pub

然后将得到打公钥串贴到github的自己账户的公钥列表中。

17. android 中 Application 和Activity的关系

每个android应用都有一个Application实例，在Mainmanifest.xml中可以查阅到，但是每个应用可以有多个Activity。一般Application会被用来定义一些Acitivity中使用的全局变量。还可以在Applicaiton中设置，捕获运行时异常的处理机制。

如：

Thread.*setDefaultUncaughtExceptionHandler*(**new** UncaughtExceptionHandler() {

@Override

**public** **void** uncaughtException(Thread thread, Throwable ex) {

// 获取线程id

Log.*e*("Crash", "crashed 的线程id为："+ Thread.*currentThread*().getId() + "");

}

});s

这样，当线程出现crash时就可以处理异常。

还有一点值得注意的是：Application和Activity都是有MainUI线程启动运行的。 所以他们共用同一个线程。

18. 中兴手机在adb中识别不出来。

执行

lsusb

命令，可以得到下面的信息：

Bus 001 Device 010: ID 19d2:0363 ZTE WCDMA Technologies MSM

执行下面打命令，创建一个udev规则

sudo /etc/udev/rules.d/70-android-usb.rules

编辑其内容如下：

SUBSYSTEM=="usb",ATTRS{idVendor}=="19d2",ATTRS{idProduct}=="0363",MODE="0666"

执行下面的命令，重新启动udev

sudo chmod a+rx /etc/udev/rules.d/70-android-usb.rules

sudo /etc/init.d/udev restart

执行下面的命令，再次尝试查看usb设备列表

adb kill-server

adb devices  
 19 . UncaughtExceptionHandler的使用

参见第17条，注意在Application中中如果想显示toast则应该传入Activity的 Context，Application的Context是不行的,应该是Activity中的Context

20. 重新执行刚刚输入的命令而不必重启或者注销重新登录。

如修改 .bashrc 或者 .netrc时，

执行 source .netrc 或者 . .netrc

即可使得修改的文件生效

21. AsyncTask 的相关知识点

class PullPageAsync extends AsyncTask<URL, Integer, String> {

private Context context;

private ProgressDialog progressDialog;

private int currentPro = 0;

public PullPageAsync(Context context) {

this.context = context;

}

@Override

protected void onPreExecute() {

progressDialog = new ProgressDialog(context);

progressDialog.setTitle("进度更新");

progressDialog.setMax(100);

progressDialog.setProgress(currentPro);

progressDialog.setProgressStyle(ProgressDialog.STYLE\_HORIZONTAL);

progressDialog.show();

}

@Override

protected void onProgressUpdate(Integer... values) {

progressDialog.setProgress(values[0]);

}

@Override

protected String doInBackground(URL... params) {

StringBuilder result = new StringBuilder();

// 每次读取一行，并实时更新读取进度

BufferedInputStream in = null;

try {

URLConnection connection = params[0].openConnection();

in = new BufferedInputStream(connection.getInputStream());

byte[] buffer = new byte[1024];

while (in.read(buffer) != -1) {

result.append(new String(buffer, "utf-8"));

currentPro++;

publishProgress(currentPro \* 10);

}

} catch (Exception e) {

Toast.makeText(context, "尝试下载网络资源失败", Toast.LENGTH\_LONG).show();

Log.e("PullPageAsync", "尝试下载网络资源失败");

} finally {

try {

in.close();

} catch (IOException e) {

e.printStackTrace();

}

}

return result.toString();

}

@Override

protected void onPostExecute(String result) {

super.onPostExecute(result);

progressDialog.setProgress(100);

progressDialog.dismiss();

pageContent.setText(result);

pullContent.setText("总共有 " + result.length() + " 个字节。");

}

}

22.[This Handler class should be static or leaks might occur 解决办法](http://blog.csdn.net/wuleihenbang/article/details/17126371)？为什么？

Handler 类应该为static类型，否则有可能造成泄露。在程序消息队列中排队的消息保持了对目标Handler类的引用。如果Handler是个内部类，那 么它也会保持它所在的外部类的引用。为了避免泄露这个外部类，应该将Handler声明为static嵌套类，并且使用对外部类的弱应用。

说的很明白，因为内部类对外部类有个引用，所以当我们在内部类中保存对外部类的弱引用时，这里的内部类对外引用还是存在的，这是jdk实现的，而只有我们声明为静态内部类，此时是对外部类没有强引用的（这时它和普通的外部类没有什么区别，只是在访问控制上有很大的方便性，它可以直接访问外部类的私有成员），这样当线程中还有Looper 和Looper中的MessageQueue还有Message时（message中保持了对Handler的引用），而此时Activity已经死亡时，就可以对Activity进行回收了。

23. [深度理解Handler消息处理机制](http://www.cnblogs.com/freeliver54/archive/2012/06/19/2554515.html)

疑惑一：在MainActivity中为什么只是类似的写一行如下代码就可以使用handler了呢？

Handler handler = new Handler() {

@Override

public void handleMessage(Message msg) {

// handle the nsg message...

}

};

在Handler类的构造过程中，你可以看到下面的几行代码：

mLooper = Looper.myLooper();

if (mLooper == null) {

throw new RuntimeException("Can't create handler inside thread that has not called Looper.prepare()");

}

**mQueue = mLooper.mQueue; //1**

mCallback = callback;

mAsynchronous = async;

可见这里隐藏着对handler中的MessageQueue和相关的数据结构的初始化赋值，而 **//1**处的代码如果你继续跟进去回发现有如下代码：

**public** **static** Looper myLooper() {

**return** *sThreadLocal*.get();

}

其中 ***sThreadLocal*** *在****Looper*** *类中被声明为 ：***static** **final** ThreadLocal<Looper> *sThreadLocal* = **new** ThreadLocal<Looper>();的形式，可得它是一个静态不可变的**键值，**

这里如果继续跟进去，请**参考第10条**补充理解。之所以没有抛出异常提醒未调用**prepare**，是因为在**MainActivity**启动时它已经调用一次**prepare**，如果你再调用，那么将会抛出运行时异常。

疑惑二：建立过程可以简而言之吗？

线程先调用Looper.prepare() (**注意这行代码一定要在要建立handler的线程运行时调用，一般在run方法中调用，不能在它的构造方法中调用，因为构造方法调用时该线程还没有运行，正在执行的线程还是创建该子线程的父线程的**)，当调用该语句时，那么将会以Looper的静态成员*sThreadLocal****为主键****，在当前正在执行的线程的内部存入一个值为一个****新的****Looper对象****（这也是线程本地存储的本质，其实多个线程并非使用的是同一个对象而只是使用同一个ThreadLocal向外的展示形式罢了），因为Looper中有MessageQueue，handler***中***Looper***引用正是通过***Looper.myLooper(),***而得到当前线程中以***sThreadLocal***为主键存储的***Looper***对象的引用，这也是为什么要求***Looper.prepare***方法和新建***Handler***时（**此时传入*Looper***）一定要在同一个线程中执行。当调用***Looper.loop()***方法时（**也同时在当前线程的中调用**）那么***loop***方法内部将首先从当前的线程中查处该线程中保存的***Looper***对象，然后取出里面的***MessageQueue***对象，使用阻塞的方式进行对队列中的消息进行逐个读取和处理。（关于处理时，为什么就可以回调到***handler***中定义的***handleMessge***方法了呢？请参考下面的）

疑惑三：***Handler***中是什么原理，使得发送的消息就可以直接调用到***handler***中覆写的***handleMessge***方法？

24. 静态内部类的实现机制？

25. 消息读取时，如果有就继续，如果没有就阻塞，时怎么实现的，而有写的消息到来时还可以继续读取消息？

26.Activity/Service中注册的BroadcastReceiver当相应组件销毁时，一定不要忘了unregister掉！！！

27. Service相关学习。

其一：

简单的Service：（本应用使用的）

默认是运行在主线程中的，所以当我们准备在***service***中做些耗时的事情时，还请我们自己创建一个子线程用以去处理任务。如果用***startService***方法启动***service***则***oncreate***方法只会被调用一次（在***service***的生命周期里），而这是我们可以在***onStartCommand***方法的第一参数中获得启动该***Service***时的那个***Intent***。停止***Service***有其它任意组件调用***stopSercvice***或者该***Service***主动调用***stopSelp***方法。

默认有两种方式实现自己的***Service***：

其一：继承 ***Service***类。就如我们正常情况下使用的那样。

其二：继承***IntentService***类。为我们创建***worker***线程。

我们只要实现***IntentService***的***onHandleIntent***方法即可。

值得注意的是，如果还有其它的方法被覆写了，在写完自己的业务逻辑后注意调用父类的***onHandleIntent***回调方法。

总结：不要以为继承***IntentService***是最好的选择，其实它有一个很严重的弊端：当我们想同时执行多个任务时，用***IntentService***的***Worker***线程执行时默认时顺序执行的（串行），没有并发性，而这是恰恰使用***Service***时只要我们使用多线程即可使得多个任务同时进行了。

其二：

绑定类型的Service（bound Service）

1.实现

28. android 中apk包和jar包的区别是什么，怎么达成apk的包?

29.android 中的BaseAdapter的类它的过滤器实现思想是什么？

30.怎么引用apk包中的资源？

31. EditText 中的光标不显示并不是设置失去焦点，只是设置光标显不显示的问题，如下可以达到隐藏输入框的光标。

android:cursorVisible=*"false"*

32. 获取对象的私有成员，并调用它的公有方法。