Android 学习笔记－让我们快速上手吧

Google 的 Android SDK 发布也有一段时间了，一直想研究一下却苦于找不到时间。利用这个周末，开始强迫自己再次进入学习状态，原因很简单：我看好开放的 gPhone。

SDK 的下载与安装并不复杂，网上也有不少同学已经进入状态了，我就不再重复了吧。

今天主要讨论的，还是永远不变的话题：Hello World.

1.最简单的 HelloWorld

安装了 SDK 后，直接生成一个 Android Project，一句代码不用写，就能跑出一个最简单的 HelloWorld 例程。我们看一下它的代码：

```
public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setTheme(android.R.style.Theme_Dark);
        setContentView(R.layout.main);
}
```

看上去实在很简单，只有两句话而已。关键在这个 R.layout.main 上，凭直觉，这应该是定义的资源。的确，在 R.java 中只是定义了一个 static int 而已，真正的资源描述在 res/layout/main.xml文件里（注意：这里的 R.java 不要手工编辑，每次 build project 时它都会根据 res 下的资源描述被自动修改）。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView id="@+id/txt"
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World"
    />
</LinearLayout>
```

这个文件很好读，一个描述了这是一个线性排列的布局，android:orientation=vertical 表示所有组件将纵向排布。而经典的 Hello World 是用一个 TextView 来展示的。

由此，我们知道，Android 的程序从一个 Activity 派生出来，并且从它的 onCreate 开始启动；Android 里要显示的组件用 XML 文件描述而不用在代码中硬编码（这是一个好的习惯，我们应该从一开始就坚持下去）；

2.让 Button 来说 Hello World

上面的例子是 ADT 自动生成的代码，似乎与我们一点关系也没有。那我们来改一下代码，因为在 windows 平台上的 Helloworld 经常是由一个按钮触发的，所以，我们想第二个 Helloworld 应该是这样的：加一个按钮和文本输入框，单击按钮后在原来的 TextView 后面加上输入框中输入的文字。

第一步是，增加一个 Button 和一个 EditText，与 TextView 一样，它们也在 main.xml 里描述一下：

```
 <EditText id="@+id/edt"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text=""
    />
<Button id="@+id/go"
```

```
        android:layout_width="wrap_content" android:layout_height="wrap_con
tent"
        android:text="@string/go">
        <requestFocus />
    </Button>
```

这里有两个地方要注意：id=@+id/go，这表示需要一个唯一的 UID 来作为 Button 的 ID，它的引用名是 go。还有一个是 android:text=@string/go 表示这个按钮的文本不是直接写有 main.xml 里了，而是来源于另一个资源描述文件 strings.xml 里，本例中的 strings.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">helloTwo</string>
    <string name="tit_dialog">提示</string>
    <string name="msg_dialog">你好，中国</string>
    <string name="ok_dialog">确定</string>
    <string name="go">浏览</string>
</resources>
```

然后，在代码里（onCreate 函数中）我们加上以下代码（简单起见，用了嵌套类）：

```
Button btn = (Button)findViewById(R.id.go);
        btn.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                EditText edt=(EditText)helloTwo.this.findViewById(R.id.edt);
```

```
                TextView txt= (TextView)helloTwo.this.findViewById(R.id.txt
);
                txt.setText(getString(R.string.msg_dialog)+edt.getText());

            }
        });
```

为铵钮增加一个 onClick 事件处理器，在点击事件中，设置 txt 的文本为 R.string.msg_dialgo+edt.getText()。

这里的关键是两个函数的使用： findViewById(R.id.go)可以根据资源的名称加载 View 类型的资源，同样用函数 getString(R.string.msg_dialog)可以加载字符串资源。

编译，run 一下看看效果。

3. 再让菜单 Say Hello

从 API 文档中我们看到 Activity 中有两个函数：onCreateOptionsMenu 和 onOptionsItemSelected，显示，这个 OptionsMenu 就是所谓的上下文菜单（在 GPhone 的模拟器上，有个键专用于弹出这个菜单）。下面我们就为这个 HelloWorld 例子加上一个菜单，并且让它可以 Say hello。

这次，我们不涉及到资源的描述文件了，而是直接使用这两个函数来实现，其实代码也很简单，所以，我们再增加一个退出应用的功能（否则每次都是按取消键退出应用显示太不专业了）。

代码如下：

```
   public boolean onCreateOptionsMenu(Menu menu)
    {
        super.onCreateOptionsMenu(menu);
        menu.add(0,1,"say hello");
```

```
        menu.add(0,2,"exit");
        return true;
    }
    public boolean onOptionsItemSelected(Item item)
    {
        super.onOptionsItemSelected(item);

        int id = item.getId();
        switch(id){
        case 1:
            AlertDialog.show(this,getString(R.string.app_name),
                             getString(R.string.msg_dialog), getString(R.stri
ng.ok_dialog), true);
            break;
        case 2:
            finish();
            break;
        }
```

在 CreateOptionsMenu 时，我们简单地增加两个菜单项，menu.add(组 ID,项 ID,显示文本)，（注意：这里我直接将文字写在代码里，这并不提倡）。然后，在 OptionsItemSelected 事件中，我们根据选中的菜单项做相应处理，如果选中 1，则弹出一个对话框显示资源文件中的“你好，中国”，如果选中 2 则退出应用。

AlertDialog.show 是一个静态方法，类似于我们在 WIN 平台上经常使用的 MessageBox 一样，很方便的。

来源：http://www.sf.org.cn/Android/lumen/20976.html

Android 学习笔记(2)－初识 Activity

根据文档的解释，Activity 是 Android 开发中非常重要的一个基础类。我把它想像

成 J2ME 中的 Display 类，或者是 Win32平台上的 Form 类，也许不准确，但是它的重要性我觉得应该是一样的（当然，如果我们写的是一个没有界面的应用，例如后台运行的服务之类的，可以不用 Display 的）。

1. 在一个 Activity 中使用多个 View

如果把 Activity 看作 MVC 中的 Control？它负责管理 UI 和接受事件（包括用户的输入），虽然说一个 Activity 通常对应一个屏幕，但事实上，我们是可以只用一个 Activity 管理多个不同的 View 来实现简单的逻辑。

首先，我们增加一个新的资源描述 layout/second.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView id="@+id/txt"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello 中国"
    />
 <Button id="@+id/go2"
        android:layout_width="wrap_content" android:layout_height="wrap_con
tent"
        android:text="back">
        <requestFocus />
```

```
    </Button>
</LinearLayout>
```

除了一个“Hello 中国”以外，增加一个按钮可以返回前一个界面。然后，在代码中我们要为 helloTwo 增加两个方法，setViewOneCommand 和 setViewTwoCommand，分别处理一下在不同界面时，从资源里加载组件并为组件绑定一个事件处理器。

```
  public void setViewOneCommand()
    {
        Button btn = (Button)findViewById(R.id.go);
        btn.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                helloTwo.this.setContentView(R.layout.second);
                helloTwo.this.setViewTwoCommand();
            }
        });
        Button btnExit=(Button)findViewById(R.id.exit);
        btnExit.setOnClickListener(new View.OnClickListener(){
            public void onClick(View v){
                helloTwo.this.finish();
            }
        });
    }
    public void setViewTwoCommand()
```

```
    {
        Button btnBack=(Button)findViewById(R.id.go2);
         btnBack.setOnClickListener(new View.OnClickListener(){
             public void onClick(View v){
                 helloTwo.this.setContentView(R.layout.main);
                 helloTwo.this.setViewOneCommand();
             }
        });
    }
```

最后，我们需要在 onCreate 的时候，也就是启动后的 main 界面上设置一下按钮事件处理器。新的 onCreate 方法如下：

```
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setTheme(android.R.style.Theme_Dark);
        setContentView(R.layout.main);
        setViewOneCommand();
    }
```

编译，运行，OK。

2. 还是回到正道上，多个 Activity 之间的跳转

Android 中提供一个叫 Intent 的类来实现屏幕之间的跳转，按文档的说法，似乎他们也建议采用这种方法，Intent 的用法比较复杂，现在我先看看它最简单的用法。

先在应用中增加两个 Activity，这需要修改 AndroidManifest.xml文件了，如下：

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

```
    package="cn.sharetop.android.hello.three">
    <application android:icon="@drawable/icon">
        <activity class=".HelloThree" android:label="@string/app_name">
            <intent-filter>
                <action android:value="android.intent.action.MAIN" />
                <category android:value="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
        <activity class=".HelloThreeB" android:label="@string/app_name">
        </activity>
    </application>
 </manifest>
```

很简单，就是加一个标签而已，新标签的 class 是.HelloThreeB，显示的应用标题与前一个 Activity 一样而已，然后第二步就是修改一个 HelloThree 类的实现，在 onCreate 方法中绑定按钮的事件处理器：

```
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setTheme(android.R.style.Theme_Dark);
        setContentView(R.layout.main);
        setViewOneCommand();
    }
    public void setViewOneCommand()
    {
```

```
        Button btn = (Button)findViewById(R.id.go);
        btn.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                Intent intent = new Intent();
                intent.setClass(HelloThree.this, HelloThreeB.class);
                startActivity(intent);
                finish();
            }
        });
        Button btnExit=(Button)findViewById(R.id.exit);
        btnExit.setOnClickListener(new View.OnClickListener(){
            public void onClick(View v){
                HelloThree.this.finish();
            }
        });
    }
```

这里的跳转功能用 Intent 来操作，它的最简单用法就是用函数 setClass()设置跳转前后两个 Activity 类的实例，然后调用 Activity 自己的 startActivity(intent)即可。最后一句 finish()表示将当前 Activity 关掉（如果不关掉会如何？你可以自己试一下看效果，事实上有时我们是不需要关掉当前 Activity 的）。

然后，我们同样弄一个 Activity 类 HelloThreeB，代码与前面的差不多，只是将 setClass 的两个参数反一下，这样就可以简单地实现在两个 Activity 界面中来回切换的

功能了。

3. 如果我想在两个 Activity 之间进行数据交换，怎么办？

前例中的 startActivity()只有一个参数，如果需要向新打开的 Activity 传递参数，我们得换一个函数了， Android 提供了 startSubActivity(Intent,int)这个函数来实现这个功能。

函数原型为： public void startSubActivity(Intent intent, int requestCode)

这里的 requestCode 用来标识某一个调用，一般由我们定义一个常量。

如何把参数传过去呢？ Intent 类在提供 setClass()函数的同时也提供了一个 setData()函数。

函数原型为：public Intent setData(ContentURI data)

参数类型是 ContentURI，它的详细内容下回再分析，现在就把它当成一个 String 类型来用吧。

参数带到新的 Activity 后，同样用 Activity.getIntent()函数可以得到当前过来的 Intent 对象，然后用 getData()就取到参数了。

把参数带回来的方法是 Activity.setResult()，它有几个形式，现在先看最简单的一个吧。

函数原型是：public final void setResult(int resultCode, String data)

resultCode 是返回代码，同样用来标识一个返回类型，而 data 则是它要返回的参数。

在原来的 Activity 中的事件处理回调函数 onActivityResult，会被系统调用，从它的参数里可以得到返回值。

函数原型为： protected void onActivityResult(int requestCode, int resultCode,String data, Bundle extras)

这里的 requestCode 就是前面启动新 Activity 时的带过去的 requestCode，而 resultCode

则关联上了 setResult 中的 resultCode，data 是参数，extras 也是一个很重要的东西，后面再研究一下它的作用。

下面，我们来看一下代码吧，先看看 HelloThree 中的代码：

```
    public void setViewOneCommand()
    {
        Button btn = (Button)findViewById(R.id.go);
        btn.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                try
                {
                    Intent intent = new Intent();
                    intent.setClass(HelloThree.this, HelloThreeB.class);
                    intent.setData(new ContentURI("One"));

                    startSubActivity(intent,REQUEST_TYPE_A);
                }
                catch(Exception ex){}
            }
        });
        Button btnExit=(Button)findViewById(R.id.exit);
        btnExit.setOnClickListener(new View.OnClickListener(){
            public void onClick(View v){
```

```
                HelloThree.this.finish();
            }
        });
    }
    protected void onActivityResult(int requestCode, int resultCode,
            String data, Bundle extras)
    {
        if (requestCode == REQUEST_TYPE_A) {
            if (resultCode == RESULT_OK) {
                Log.v(TAG,data);
                TextView txt = (TextView)findViewById(R.id.txt);
                txt.setText(data);
            }
        }
    }
```

这里的 REQUEST_TYPE_A 是我们定义的一个常量。在 onActivityResult 中用它与 RESULT_OK 一起作为条件判断如何处理返回值，这里只是简单将 TextView 显示值换成传来的字串。

再来看看另一个 HelloThreeB 类的实现代码：

```
    private Intent i;
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.second);
```

```
        i = getIntent();

        android.util.Log.v(TAG,"onCreate");
        Button btn = (Button)findViewById(R.id.go);
         btn.setOnClickListener(new View.OnClickListener(){
             public void onClick(View v){
                String result=HelloThreeB.this.i.getData().toString()+" And
 Two";
                HelloThreeB.this.setResult(RESULT_OK,result);
                finish();
             }
        });

        TextView v = (TextView)findViewById(R.id.txt);
        v.setText("Param is "+i.getData().toString());

    }
```

在按钮处理事件中，从 Intent 取出参数，处理一下再用 setResult 返回给前一个 Activity 即可。

编译运行即可。

来源：http://www.sf.org.cn/Android/lumen/20977.html

Android 学习笔记(3)－Activity 的生命周期

注意到在 Activity 的 API 中有大量的 onXXXX 形式的函数定义，除了我们前面用到的 onCreate 以外，还有 onStart，onStop 以及 onPause 等等。从字面上看，它们是一些

事件回调，那么次序又是如何的呢？其实这种事情，自己做个实验最明白不过了。在做这个实验之前，我们先得找到在 Android 中的 Log 是如何输出的。

显然，我们要用的是 android.util.log 类，这个类相当的简单易用，因为它提供的全是一些静态方法：

```
Log.v(String tag, String msg);          //VERBOSE
Log.d(String tag, String msg);         //DEBUG
Log.i(String tag, String msg);          //INFO
Log.w(String tag, String msg);      //WARN
Log.e(String tag, String msg);       //ERROR
```

前面的 tag 是由我们定义的一个标识，一般可以用“类名_方法名“来定义。

输出的 LOG信息，如果用 Eclipse+ADT 开发，在 LogCat 中就可以看到，否则用 adb logcat 也行，不过我是从来都依赖于 IDE 环境的。

好了，现在我们修改前面的 HelloThree代码：

```
    public void onStart()
    {
        super.onStart();
        Log.v(TAG,"onStart");
    }
    public void onStop()
    {
        super.onStop();
        Log.v(TAG,"onStop");
    }
    public void onResume()
```

```
    {
        super.onResume();
        Log.v(TAG,"onResume");
    }
    public void onRestart()
    {
        super.onRestart();
        Log.v(TAG,"onReStart");
    }
    public void onPause()
    {
        super.onPause();
        Log.v(TAG,"onPause");
    }
    public void onDestroy()
    {
        super.onDestroy();
        Log.v(TAG,"onDestroy");
    }
    public void onFreeze(Bundle outState)
    {
        super.onFreeze(outState);
        Log.v(TAG,"onFreeze");
    }
```

在 HelloThreeB 中也同样增加这样的代码，编译，运行一下，从 logcat 中分析输出的日志。

在启动第一个界面 Activity One 时，它的次序是：

onCreate (ONE) - onStart (ONE) - onResume(ONE)

虽然是第一次启动，也要走一遍这个 resume 事件。然后，我们点 goto 跳到第二个 Activity Two 中（前一个没有关闭），这时走的次序是：

onFreeze(ONE) - onPause(ONE) - onCreate(TWO) - onStart(TWO) - onResume(TWO) - onStop(ONE)

说明，第二个 Activity Two 在启动前，One 会经历一个：冻结、暂停的过程，在启动 Two 后，One 才会被停止？

然后，我们再点 back 回到第一个界面，这时走的次序是：

onPause(TWO) - onActivityResult(ONE) - onStart(ONE) - onRestart(ONE) - onResume(ONE) - onStop(TWO) - onDestroy(TWO)

说明，返回时，Two 没有经历冻结就直接暂停了，在 One 接收参数，重启后，Two 就停止并被销毁了。

最后，我们点一下 Exit 退出应用，它的次序是：

onPause(ONE) - onStop(ONE) - onDestroy(ONE)

说明如果我们用了 finish 的话，不会有 freeze，但是仍会经历 pause - stop 才被销毁。

这里有点疑问的是：为什么回来时先是 Start 才是 Restart？可是文档中的图上画的却是先 restart 再 start 的啊？不过，后面的表格中的描述好象是正确的，start 后面总是跟着 resume（如果是第一次）或者 restart（如果原来被 stop 掉了，这种情况会在 start 与 resume 中插一个 restart）。

下面不跑例子了，看看文档吧。

1.Android 用 Activity Stack 来管理多个 Activity，所以呢，同一时刻只会有最顶上的那个 Activity 是处于 active 或者 running 状态。其它的 Activity 都被压在下面了。

2.如果非活动的 Activity 仍是可见的（即如果上面压着的是一个非全屏的 Activity 或透明的 Activity），它是处于 paused 状态的。在系统内存不足的情况下，paused 状态的 Activity 是有可被系统杀掉的。只是不明白，如果它被干掉了，界面上的显示又会变成什么模样？看来下回有必要研究一下这种情况了。

3.几个事件的配对可以比较清楚地理解它们的关系。Create 与 Destroy 配成一对，叫 entrie lifetime，在创建时分配资源，则在销毁时释放资源；往上一点还有 Start 与 Stop 一对，叫 visible lifetime，表达的是可见与非可见这么一个过程；最顶上的就是 Resume 和 Pause 这一对了，叫 foreground lifetime，表达的了是否处于激活状态的过程。

4.因此，我们实现的 Activity 派生类，要重载两个重要的方法：onCreate()进行初始化操作，onPause()保存当前操作的结果。

除了 Activity Lifecycle 以外，Android 还有一个 Process Lifecycle 的说明：

在内存不足的时候，Android 是会主动清理门户的，那它又是如何判断哪个 process 是可以清掉的呢？文档中也提到了它的重要性排序：

1.最容易被清掉的是 empty process，空进程是指那些没有 Activity 与之绑定，也没有任何应用程序组件（如 Services 或者 IntentReceiver）与之绑定的进程，也就是说在这个 process 中没有任何 activity 或者 service 之类的东西，它们仅仅是作为一个 cache，在启动新的 Activity 时可以提高速度。它们是会被优先清掉的。因此建议，我们的后台操作，最好是作成 Service 的形式，也就是说应该在 Activity 中启动一个 Service 去执行这些操作。

2.接下来就是 background activity 了，也就是被 stop 掉了那些 activity 所处的 process，那些不可见的 Activity 被清掉的确是安全的，系统维持着一个 LRU 列表，多个处于 background 的 activity 都在这里面，系统可以根据 LRU 列表判断哪些 activity

是可以被清掉的，以及其中哪一个应该是最先被清掉。不过，文档中提到在这个已被清掉的 Activity 又被重新创建的时候，它的 onCreate 会被调用，参数就是 onFreeze 时的那个 Bundle。不过这里有一点不明白的是，难道这个 Activity 被 killed 时，Android 会帮它保留着这个 Bundle 吗？

3.然后就轮到 service process 了，这是一个与 Service 绑定的进程，由 startService 方法启动。虽然它们不为用户所见，但一般是在处理一些长时间的操作（例如 MP3 的播放），系统会保护它，除非真的没有内存可用了。

4.接着又轮到那些 visible activity 了，或者说 visible process。前面也谈到这个情况，被 Paused 的 Activity 也是有可能会被系统清掉，不过相对来说，它已经是处于一个比较安全的位置了。

5.最安全应该就是那个 foreground activity 了，不到迫不得已它是不会被清掉的。这种 process 不仅包括 resume 之后的 activity，也包括那些 onReceiveIntent 之后的 IntentReceiver 实例。

在 Android Application 的生命周期的讨论中，文档也提到了一些需要注意的事项：因为 Android 应用程序的生存期并不是由应用本身直接控制的，而是由 Android 系统平台进行管理的，所以，对于我们开发者而言，需要了解不同的组件 Activity、Service 和 IntentReceiver 的生命，切记的是：如果组件的选择不当，很有可能系统会杀掉一个正在进行重要工作的进程。

来源：http://www.sf.org.cn/Android/lumen/20978.html

Android 学习笔记(4)－学习 Intent 的使用

刚看到 Intent 的时候，我的确有点困惑：从字面上来说，它表示一种意图和目的；从使用上看，它似乎总是用于 Activity 之间的切换；而从它所在包 android.content 来看，它似乎与内容有关。所以，我想或许可以这样理解它： Intent 类绑定一次操作，它负责携带这次操作所需要的数据以及操作的类型等。

如果是这样的话，是否可以将它与事件处理联想起来？即一个 Intent 类似于一个 Event。从 Intent 的两个最重要的成员操作类型（Action）和数据（Data）来看，似乎是有道理的。文档中说，Intent 的 Action 的取值主要是一些定义好了的常量，例如 PICK_ACTION，VIEW_ACTION，EDIT_ACTION 之类的，而 Data 则是一个 ContentURI 类型的变量，这一点，我们前面提到过。

而且文档中说 Intent 分为两大类，**显性的**（Explicit ）和**隐性的**（Implicit）。在前面的例子中，我们在两个 Activity 之间跳转时初步使用了 Intent 类，当时是用 setClass 来设置 Intent 的发起方与接收方，它被称为显性的 Intent，而隐性的 Intent 则不需要用 setClass 或 setComponent 来指定事件处理器，利用AndroidMenifest.xml 中的配置就可以由平台定位事件的消费者。

一般来说，intent 要定位事件的目的地，无外乎需要以下几个信息：

1.种类（category），比如我们常见的 LAUNCHER_CATEGORY 就是表示这是一类应用程序。

2.类型（type），在前面的例子中没用过，表示数据的类型，这是隐性 Intent 定位目标的重要依据。

3.组件（component），前面的例子中用的是 setClass，不过也可以用 setComponent 来设置 intent 跳转的前后两个类实例。

4.附加数据（extras），在 ContentURI 之外还可以附加一些信息，它是 Bundle 类型的对象。

Implicit Intent 的使用相对有点麻烦，我们来做一个例子。首先，我们需要增加一个类：HelloThreeProvider，它必须实现于 ConentProvider 接口，所以代码如下：

```
  public class HelloThreeProvider extends ContentProvider {
|
    public boolean onCreate() {
|       return true;
```

```
    }

    public int delete(ContentURI url, String where, String[] whereArgs) {
        return 0;
    }
    public ContentURI insert(ContentURI url, ContentValues initialValues){
        return url;
    }
    public Cursor query(ContentURI url, String[] projection, String selecti
on,
            String[] selectionArgs, String groupBy, String having, String s
ort) {
        return null;
    }
    public int update(ContentURI url, ContentValues values, String where, S
tring[] whereArgs) {
        return 0;
    }

    public String getType(ContentURI url) {
        return "vnd.sharetop.hello.three/vnd.hello.three";
    }
}
```

这里面有一堆方法要实现，因为它们都是 ContentProvider 中的 abstract 方法，但

是今天的例子中它们多半没有什么用处，只是一个 getType 方法我们让它不管什么 url 都返回一个表示 Intent 所携带的数据类型是我们定义的一个长字串：vnd.sharetop.hello.three/vnd.hello.three。

然后,在 AndroidMenifest.xml 中我们将上面这个 HelloThreeProvider 类加入应用程序:

```
 <application android:icon="@drawable/icon">
         <provider class="HelloThreeProvider" android:authorities="cn.sharetop.android.hello" />
        <activity class="HelloThree" android:label="@string/app_name">
            <intent-filter>
                <action android:value="android.intent.action.MAIN" />
                 <category android:value="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity class="HelloThreeB" android:label="bbb">
        <intent-filter>
             <action android:value="android.intent.action.VIEW" />
             <category android:value="android.intent.category.DEFAULT" />
              <type android:value="vnd.sharetop.hello.three/vnd.hello.three" />
        </intent-filter>
        </activity>
    </application>
```

相对于前面的例子，主要修改了 HelloThreeB 的配置，包括增加了一个<category>

标签表示这是一个一般性的 activity 而已。增加了<action>标签，定义它负责处理 VIEW_ACTION 类型的操作。增加了<type>标签给出一个数据类型的定义串 vnd.sharetop.hello.three/vnd.hello.three。最主要的是在<application>下增加的那个<provider>标签，有个 authorities 属性，我们给的值是 cn.sharetop.android.hello，待一会我们再说它的用处。

最后就是修改以前的跳转代码如下：

```
Intent intent = new Intent();
intent.setData(new ContentURI("content://cn.sharetop.android.hello/one"));
intent.setAction(intent.VIEW_ACTION);
startActivity(intent);
```

现在我们的 setData 里的东西可与以前不一样的，是吧？注意到它的格式了吗？content://是个协议头，固定这样写就行了。然后就是那个 authorities 中定义的串了，再后面就是我们自定义的东西了，我这里很简单的写个 one，其它还可以更长一点，如 one/101 之类的。它负责去关联上那个 provider 类。另外，增加了 setAction 的调用设置操作为 VIEW_ACTION，与 Menifest 中的<action>又挂上了。Android 平台负责根据 Intent 的 Data 信息中的 authorities，找到 ContentProvider，然后 getType，用 type 和 intent 中的 Action 两个信息，再找到可以处理这个 intent 的消费者。

OK，编译运行。

其实，如果是在一个应用内部，这种隐性的 intent 实在有点别扭，个人觉得，这种松藕合的实现方法，只适用于那些较大的系统或者多个不同的应用之间的调用，可手机上又有什么“较大”的系统呢？无非是可以与不同来源的多个应用之间方便地互操作而已，那么会是什么样的场景呢？比如，给 QQ 好友发送 gmail 邮件，用 GoogleMap 查找 QQ 好友所在的位置？看上去挺不错的。

关于这个 ContentProvider，其实还有话说，它主要是的那些看似数据库操作的方法

我们都没真正去实现呢。不过今天就到这里了，等下回再去研究吧。

来源：http://www.sf.org.cn/Android/lumen/20979.html

Android 学习笔记(5)－关于 ListActivity 的简单体验

今天学习点轻松的内容吧，看看 android.app 包里的几个类。首先是这个在平台自的例子中被广泛使用的 ListActivity。这个类其实就是一个含有一个 ListView 组件的 Activity 类。也就是说，如果我们直接在一个普通的 Activity 中自己加一个 ListView 也是完全可以取代这个 ListActivity 的，只是它更方便而已，方便到什么程度呢？来做个例子瞧瞧。

```
  public class HelloTwoB extends ListActivity...{
    public void onCreate(Bundle icicle) ...{
        super.onCreate(icicle);
        setTheme(android.R.style.Theme_Dark);
        setContentView(R.layout.mainb);
        List<String> items = fillArray();
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,R.layo
ut.list_row,items);

        this.setListAdapter(adapter);
    }
    private List<String> fillArray()    ...{
        List<String> items = new ArrayList<String>();
        items.add("日曜日");
        items.add("月曜日");
```

```
        items.add("火曜日");
        items.add("水曜日");
        items.add("木曜日");
        items.add("金曜日");
        items.add("土曜日");
        return items;
    }
    @Override
    protected void onListItemClick(ListView l, View v, int position, long i
d)
    ...{
        TextView txt = (TextView)this.findViewById(R.id.text);
        txt.setText("あすは "+l.getSelectedItem().toString()+"です。");
    }
}
```

的确可以简单到只需准备一个 List 对象并借助 Adapter 就可以构造出一个列表。重载 onListItemClick 方法可以响应选择事件，利用第一个参数可以访问到这个 ListView 实例以得到选中的条目信息。这里有一点要说明的，就是如果更简单的话，其实连那个 setContentView 都可以不要了，Android 也会自动帮我们构造出一个全屏的列表。但是本例中我们需要一个 TextView 来显示选中的条目，所以我们需要一个 layout.mainb 描述一下这个列表窗口。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView id="@+id/text"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text=""
    />
<ListView id="@id/android:list"
    android:layout_width="fill_parent"
    android:layout_height="0dip"
    android:layout_weight="1"
    android:drawSelectorOnTop="false"
    />
</LinearLayout>
```

这里需要注意的是那个 ListView 的 ID，是系统自定义的 android:list，不是我们随便取的，否则系统会说找不到它想要的 listview 了。然后，在这个 listview 之外，我们又增加了一个 TextView，用来显示选中的条目。

再来说说这里用到的 ArrayAdapter，它的构造函数中第二个参数是一个资源 ID，ArrayAdapter 的 API 文档中说是要求用一个包含 TextView 的 layout 文件，平台用它来显示每个选择条目的样式，这里的取值是 R.layout.list_row，所以，我们还有一个 list_row.xml 文件来描述这个布局，相当简单。

```
<?xml version="1.0" encoding="utf-8"?>
  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        >
    <TextView id="@+id/item"
              xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"/>
    <TextView id="@+id/item2"
              xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"/>
  </LinearLayout>
```

从 ArrayAdapter 上溯到 BaseAdapter，发现还有几个同源的 Adapter 也应该可以使用，象 SimpleAdapter 和 CursorAdapter，还是做个例子来实验一下吧。

先看看 SimpleAdapter，说是 simple 却不 simple。

首先看看这个 fillMaps 方法，基本上就明白这个 simpleAdapter 是怎么回事了，在有些场合它还是挺有用的，可以为每个条目绑定一个值：

```
 private List<HashMap<String, String>> fillMaps()
    ...{
        List<HashMap<String, String>> items = new ArrayList<HashMap<String,
String>>();

        HashMap<String,String> i = new HashMap<String,String>();
```

```
        i.put("name","日曜日");
        i.put("key", "SUN");
        items.add(i);
        HashMap<String,String> i1 = new HashMap<String,String>();
        i1.put("name","月曜日");
        i1.put("key", "MON");
        items.add(i1);
        HashMap<String,String> i2 = new HashMap<String,String>();
        i2.put("name","火曜日");
        i2.put("key", "TUE");
        items.add(i2);
        HashMap<String,String> i3 = new HashMap<String,String>();
        i3.put("name","水曜日");
        i3.put("key", "WED");
        items.add(i3);
        HashMap<String,String> i4= new HashMap<String,String>();
        i4.put("name","木曜日");
        i4.put("key", "THU");
        items.add(i4);
        HashMap<String,String> i5 = new HashMap<String,String>();
        i5.put("name","金曜日");
        i5.put("key", "FRI");
        items.add(i5);
        HashMap<String,String> i6 = new HashMap<String,String>();
```

```
        i6.put("name","土曜日");
        i.put("key", "SAT");
        items.add(i6);

        return items;
    }
```

然后，在 HelloTwoB 中的 onCreate 函数中，修改代码，有几个不同：items 的元素是 HashMap 实例，这是一点变化，然后构造函数除了要求 items 以外，还要求提供一个 string[]来说明用 hash 表中的哪个字段显示在列表中，而后是一个资源 ID 的数组。我的代码是这样的：

```
 //SimpleAdapter demo
 List<HashMap<String, String>> items = fillMaps();
SimpleAdapter adapter=new SimpleAdapter(this,items,R.layout.list_row,new String[]{"name"},new int[]{R.id.item});
```

编译跑一下可以看到结果了，是吧？只是显示的文字不太对，再改一下：

```
 protected void onListItemClick(ListView l, View v, int position, long id)
    {
        TextView txt = (TextView)this.findViewById(R.id.text);
        txt.setText("あす
は "+((HashMap)l.obtainItem(position)).get("key").toString()+"です。");
    }
```

这样就好多了，其实一般情况下我们都是用 ListView 中的 obtainItem 取得当前选中的条目，然后转成 List 中的对应类型来使用的。

上面的例子中只显示 name 对应的值，其实你也可以试一下这样：

```
SimpleAdapter adapter=new SimpleAdapter(this,items,R.layout.list_row,new String[]{"name","key"},new int[]{R.id.item,R.id.item2});
```

看看是什么效果。

再看看那个 CursorAdapter 吧，它的列表中元素要求是 Cursor，这东西与 DB 有关，不过最简单的 DB 就是通讯簿。先从 Contacts.People 入手吧，同样修改代码：

```
 //CursorAdapter demo
Cursor mCursor = this.getContentResolver().query(Contacts.People.CONTENT_URI, null, null, null, null);
SimpleCursorAdapter adapter=new SimpleCursorAdapter(this,R.layout.list_row,mCursor,new String[]{Contacts.People.NAME},new int[]{R.id.item});
```

因为单纯的CursorAdapter是抽象类,所以我用的是它的子类SimpleCursorAdapter，很好理解，先用 ContentResolver 查询通讯簿得到一个游标，然后告诉 SimpleCursorAdapter 要用其中的 People.NAME 作为显示项来构造出一个 adapter 即可。

现在的 onListItemClick 也不一样了，如下：

```
    protected void onListItemClick(ListView l, View v, int position, long id)
    {
        TextView txt = (TextView)this.findViewById(R.id.text);
        Cursor c = (Cursor)l.obtainItem(position);
        txt.setText("SEL = "+c.getString(c.getColumnIndex(Contacts.People.NUMBER)));
    }
```

这里同样是先用 obtainItem 取到游标，然后用从记录中取出想要的字段显示即可。

在做这个例子时，因为权限的问题我们还得修改一下 AndroidManifest.xml 文件，让我们的应用可以访问到通讯簿：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cn.sharetop.android.hello.two">
    <uses-permission id="android.permission.READ_CONTACTS" />
    <application android:icon="@drawable/icon">
 ... ...
```

来源：http://www.sf.org.cn/Android/lumen/20980.html

Android 学习笔记(6)—关于 Dialog 的简单体验

继续 android.app 中的几个类的学习，今天的内容是那几个 Dialog 的体验。

注意到 android.app 包下除了 Dialog（可用于制作复杂的对话框）以外，还包括了几个系统定义好的对话框类，如 DatePickerDialog、TimePickerDialog 及 AlertDialog。

其中 AlertDialog 我上回用过一次，基本上就那样子了，今天看看另外两个对话框的使用吧。

首先是 DatePickerDialog 类，修改代码如下：

```
  public class HelloTwoC extends Activity implements OnClickListener, OnDat
eSetListener {
    public HelloTwoC() {
        super();
    }
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setTheme(android.R.style.Theme_Dark);
        setContentView(R.layout.mainc);
```

```
|
|     Button btn = (Button)findViewById(R.id.date);
|     btn.setOnClickListener(this);
├ }
| @Override
⊟⊞ public void onClick(View v) {
|   Calendar d = Calendar.getInstance(Locale.CHINA);
|   d.setTime(new Date());
|   DatePickerDialog dlg=new DatePickerDialog(this,this,d.get(Calendar.YEAR),
d.get(Calendar.MONTH),d.get(Calendar.DAY_OF_MONTH),d.get(Calendar.DAY_OF_WEE
K));
|   dlg.show();
├ }
| @Override
⊟⊞ public void dateSet(DatePicker dp, int y, int m, int d) {
|  TextView txt = (TextView)findViewById(R.id.text);
|  txt.setText(Integer.toString(y)+"-"+Integer.toString(m)+"-"+Integer.toStr
ing(d));
├ }
└}
```

很简单的,无非是需要一个OnDateSetListener接口的实现而已,在它里面的dateSet方法中就可以得到选择的日期了。而 TimePickerDialog 与 DatePickerDialog 使用如出一辙，就不多说了。

看看另一个 ProgressDialog 的用法吧，这个类与 AlertDialog 一样包含了多个

static 的方法，所以使用起来是非常方便的。比如说，如果我们需要用它来表示一个长时间的操作，很简单的用一句话就可以了：

```
ProgressDialog.show(this,null, "operation running...",true,true);
```

URL：http://www.sf.org.cn/Android/lumen/20981.html

Android 学习笔记(7)—关于 Service 和 Notification 的体验

大略地看了一下 android.app 下的 Service 类，觉得它与 Activity 非常相似，只是要注意几个地方：

1.生命周期，Service 的从 onCreate()->onStart(int,Bundle)->onDestroy()显得更为简单。但是它的 onStart 是带参数的，第一个 ID 可用来标识这个 service，第二个参数显示是用来传递数据的了。比较 Activity，传递数据的 Bundle 是在 onCreate 就带进入的。

2.Service 的启动由 Context.startService 开始，其实 Activity 或者 Service 都是 Context 的派生类。结束于 Context.stopService()或者它自己的 stopSelf()。

3.Service 还有一个与 Activity 不一样的是它可以由另一个 Context 去绑定一个已存在的 Service。就是这个方法 Context.bindService()，被绑定的 Service 要求是已经 onCreate 了但可以没有 onStart。在 Service 类中有个抽象方法 getBinder()可以得到这个 IBinder 对象。关于这方面的细节，以后再看，这里只做个记录罢。

4.与 Service 有关的还有一个安全的问题，可以在 AndroidManifest.xml 中用<uses-permission>标签来声明一个 Service 的访问权限，关于Android的安全问题也留待以后再解决吧。

我一直相信一种水到渠成的学习方法，先从最简单的东西入手，就不会觉得学习很枯燥了。

下面来做个例子。

修改AndroidManifest.xml 文件，增加一个 Activity 和一个 Service：

```
        <activity class=".HelloTwoD" android:label="hello_two_d">
        </activity>
        <service class=".HelloTwoDService" />
```

HelloTwoD.java 的代码比较简单，如下：

```
public class HelloTwoD extends Activity implements OnClickListener
{
 public HelloTwoD()
    {
  super();
    }
 public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setTheme(android.R.style.Theme_Dark);
        setContentView(R.layout.maind);

        Button btn = (Button)findViewById(R.id.btnTest);
        btn.setOnClickListener(this);
    }

 @Override
 public void onClick(View arg0) {
  // 用一个显式的 Intent 来启动服务
  Intent i = new Intent();
  i.setClass(this, HelloTwoDService.class);
```

```
    //带上我的名字
    Bundle b= new Bundle();
    b.putString("name", "sharetop");
    this.startService(i,b);
  }

}
```

当然要启动这个 HelloTwoD，也需要在我最初的那个 HelloTwo 中加一点代码（我就不罗嗦了）。再看看那个 HelloTwoDService 是如何实现的：

```
public class HelloTwoDService extends Service {
  public Timer timer;
  public final String TAG="HelloTwoDService_TAG";
  public void onCreate() {
          super.onCreate();

          Log.d(TAG,"onCreate");

          timer = new Timer(true);
      }
  @Override
  public IBinder getBinder() {
    // TODO Auto-generated method stub
    return null;
  }
```

```
| public void onStart(int startId, Bundle arg)
┼┼ {
|  //看看 startId 是什么内容
|  if(arg!=null)
|   Log.d(TAG,"onStart "+Integer.valueOf(startId).toString()+" from "+arg.ge
tString("name"));
|  else
|   Log.d(TAG,"onStart with null Bundle");
|
┼┼         timer.schedule(new TimerTask(){
┼┼          public void run(){
|          //表示一下我的存在
|          Log.d(TAG,"say from a timer.");
|          //停掉自己这个服务
|          HelloTwoDService.this.stopSelf();
├          }
├        },5000);
├ }
| public void onDestroy()
┼┼ {
|  Log.d(TAG,"onDestroy");
├ }
└}
```

这里我用一个定时器 timer 来延时 5 秒钟显示消息，否则立即就显示出来觉得不象

一个后台服务了。用日志输出那个 onStart 中的 startId 看看，原来只是一个标识而已。

下面来个简单的 NotificationManager 吧，看了看 API 文档，觉得最简单地恐怕就是那个 NotificationManager.notifyWithText()了，修改上面的 run 方法如下：

```
        timer.schedule(new TimerTask(){
         public void run(){
         NotificationManager manager=(NotificationManager)getSystemService
(NOTIFICATION_SERVICE);
         manager.notifyWithText(1001, "わたしは SHARETOP です。", Notificati
onManager.LENGTH_LONG, null);
         HelloTwoDService.this.stopSelf();
        }
       },5000);
```

再试试看效果。太简单了，Notification 主要是用于后台服务用来通知前台，所以，Android 提供了三类不同的通知方式，notifyWithText 可以简单地显示一个字串，而 notifyWithView 稍复杂点，可以有一个 view 来构造这个显示信息框，而最灵活的就是那个 notify(int id, Notification notification)了，参数 notification 是类 Notification 的实例。

修改一下刚才的那个 run 方法，如下：

```
        timer.schedule(new TimerTask(){
         public void run(){

         NotificationManager manager=(NotificationManager)getSystemService
(NOTIFICATION_SERVICE);
         Notification nf = new Notification(R.drawable.icon,"这是信息的详
```

```
细描述",null,"信息的标题",null);
|           manager. notify(0,nf);
|
|           HelloTwoDService.this.stopSelf();
|         }
└        },5000);
```

这里创建一个 Notification 的实例 nf，构造函数的第一个参数是那个显示在状态栏（也就是 Android 手机上面的那一条显示信号强度、电池电量等信息的位置）的图标。后面可以有

一个标题和点击以后的详细信息，这是字串形式，还可以有一个 Intent 用来表示点击后可以发生一个跳转行为。

URL：http://www.sf.org.cn/Android/lumen/20982.html

Android 学习笔记(8) — Grid View 与 Image View

简单一点吧，就瞧瞧那个 Grid 的效果，Android提供了一个 Grid View，不过从 Epidemic 中看来，它似乎与 PC 上的 GRID 差别还是挺大的，更像那个 Icon View 的感觉。不知道Android中如何实现表格界面？虽然在移动终端上，表格一般不会有谁使用，大家似乎更倾向于使用 List View，而 Android 对于 List View 则有更简单的实现 List Activity。

废话不说，还是自己写几句代码来实验一下。

```
<Grid View id="@+id/grid"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dip"
    android:verticalSpacing="10"
```

```
    android:horizontalSpacing="10"
    android:numColumns="auto_fit"
    android:columnWidth="60"
    android:stretchMode="columnWidth"
    android:gravity="center"
    />
```

从描述文件中的这些属性来看，与表格非常类似，除了 padding 和 spacing 以外，它还多了那个 gravity，这里是 center 表示单元格中的内容居中放，在类 Grid View 中也提供了方法 setGravity(int)来实现这个效果。

接着，我们沿用以前那个 fillMaps 方法来构造 SimpleAdapter，以前将这个 adapter 赋给 List Activity，现在同样的 Adapter，却是赋给了 Grid View，效果又会是怎样呢？

```
List<HashMap<String, String>> items = fillMaps();
Grid View grd=(Grid View)this.findViewById(R.id.grid);
SimpleAdapter adapter=new SimpleAdapter(this,items,R.layout.list_row,new String[]{"name"},new int[]{R.id.item});
grd.setAdapter(adapter);
```

我觉得 Grid View 并不象表格，倒更象 Icon View，下面试试用图像作为 Grid View 的内容。现在，不能用简单 Adapter 了，得自己弄一个 ImageAdapter，就让它衍生于 BaseAdapter 类吧。

```
public class ImageAdapter extends BaseAdapter {
        //这是资源 ID 的数组
        private Integer[] mThumbIds = {
                R.drawable.a,R.drawable.b,R.drawable.c,
                R.drawable.d,R.drawable.e,R.drawable.f,
```

```
                R.drawable.g,R.drawable.h,R.drawable.i
                };
        public ImageAdapter(Context c) {
            mContext = c;
        }
        public int getCount() {
            return mThumbIds.length;
        }
        public Object getItem(int position) {
            return position;
        }
        public long getItemId(int position) {
            return position;
        }
        public View getView(int position, View convertView, ViewGroup pare
nt) {
            Image View i = new Image View(mContext);
            //设置图像源于资源 ID。
      i.setImageResource(mThumbIds[position]);
            i.setAdjustViewBounds(true);
            i.setBackground(android.R.drawable.picture_frame);
            return i;
        }
        private Context mContext;
```

```
    }
```

很简单，只要重载几个方法就可以了，关键是那个 getView 方法，它负责构建出每个单元格中的对象实例。这里我们构造的是一个 Image View 实例。

然后就是同样的将这个 Adapter 赋给 Grid View 即可，大家可以看看效果，注意在做这个例子前，先放几个小图片到 res/drawable 目录下，buildproject 一下就可以得到那个 R.drawable.a 了（这里的 a 是图像文件名，如 a.png）。

在 getView 方法中我们使用了 Image View 类，这又是一个 widget。除了上面用到的几个方法以外，还有以下几个方法值得注意：

与图像来源有关的方法，我们只用了资源文件的方式。

```
    //不同的图像来源
public void setImageBitmap(Bitmap bm)
public void setImageDrawable(Drawable drawable)
public void setImageResource(int resid)
public void setImageURI(ContentURI uri)
```

图像效果的操作。

```
//颜色过滤
public void setColorFilter(int color, Mode mode)
//矩阵变换
public void setImageMatrix(Matrix matrix)
//透明度
public void setAlpha(int alpha)
```

具体的使用可以参考 API，动手试一下就差不多了。

URL：http://www.sf.org.cn/Android/lumen/20983.html

Android 学习笔记(9)－开始做一个数独游戏[上]

不想再写 Hello123 了，今天开始做一个数独小游戏，因为这个游戏比较简单应该容易上手，就作为我学习 Android 之后的第一个程序比较合适。

初步的设计是只有一个界面（如下图），然后用绿色字体表示题目中有的固定的数字，黄色字体显示玩家输入的数字，而红色字体则是程序判断输入错误后的显示。另外模式分为三种：普通写入、标记（玩家用蓝色小方块标记当前单元格可以输入的数字）、排除模式（玩家指定数字，游戏自动判断一下这个数字肯定不能输入的单元格，将它反相显示出来）。

准备工作就是做一张背景图（棋盘）和三套数字小图片（红、绿、黄）即可。

首先建立工程 sudo，程序主体类 MainActivity 以后，再修改一下那个 main.xml 文件，去掉 TextView 标签即可。因为我们会自己定义一个 View，所以不再需要它了。程序不大，所以不打算做过多的类，下面把几个类的分工描述一下：

1、MainActivity，主体类，负责处理键盘事件和维护一个题库。
2、MainView，显示类，负责维护并显示当前棋局，它的核心在于它的 onDraw 函数。
3、GridCell 和 Question 两个实体类，分别描述了棋盘单元格的信息和题目信息。
4、Helper 类，助手类，封装一些函数，如读写记录、自动解题函数等。

在 MainActivity 中的 onCreate 中，加几句话就可以让游戏全屏显示了。如下：

```
  setTheme(android.R.style.Theme_Dark);
  requestWindowFeature(Window.FEATURE_NO_TITLE);

getWindow().setFlags(WindowManager.LayoutParams.NO_STATUS_BAR_FLAG,WindowManager.LayoutParams.NO_STATUS_BAR_FLAG);
```

主要来看看 MainView 类的代码吧，它的 onDraw 负责显示当前棋局，涉及到的 API 主要是 android.graphics 中的 Canvas 和 Paint。

一是显示图片的方法，因为图片来源于资源，所以显示它的代码如：

```
Bitmap bmp = BitmapFactory.decodeResource(this.getResources(),R.drawable.grid);
canvas.drawBitmap(bmp, 0, 0, null);
```

这是显示背景，如果是数字呢，如何将数字 1 与 R.drawable.a1 资源关联呢？

```
private int[] thumbNormal=new int[]{0,
        R.drawable.a1,R.drawable.a2,R.drawable.a3,R.drawable.a4,R.drawable.a5,
        R.drawable.a6,R.drawable.a7,R.drawable.a8,R.drawable.a9
    };
```

然后就简单地加载即可了。

```
Bitmap b = BitmapFactory.decodeResource(this.getResources(),this.thumbNormal[this.grid[i].value]);
canvas.drawBitmap(b, xx, yy, null);
```

二是显示文本的方法，刚才显示图像的 drawBitmap 中最后一个参数直接给了 null，因为我们实在没有什么效果需要给图像的，但是文本则不同，我们用 Paint 来控制文本的样式。

```
Paint paintText=new Paint();
paintText.setFlags(Paint.ANTI_ALIAS_FLAG);
paintText.setColor(Color.WHITE);
... ...
canvas.drawText(Long.toString(this.ti.code), xx, yy, paintText);
```

三是画一下框的方法，同样是用 Paint 来做的。

```
Paint paintRect = new Paint();
paintRect.setColor(Color.RED);
paintRect.setStrokeWidth(2);
paintRect.setStyle(Style.STROKE);

Rect r=new Rect();
r.left=this.curCol*CELL_WIDTH+GRID_X;
r.top=this.curRow*CELL_WIDTH+GRID_Y;
r.bottom=r.top+CELL_WIDTH;
r.right=r.left+CELL_WIDTH;

canvas.drawRect(r, paintRect);
```

如果不 setStyle 为 Style.STROKE，则缺省为填充模式。

四是反相显示的方法，更简单了，就是一句话了：

```
Paint paintHint=new Paint();
paintHint.setXfermode(new PixelXorXfermode(Color.WHITE));
```

URL：http://www.sf.org.cn/Android/lumen/20984.html

Android 学习笔记(10)－开始做一个数独游戏[中]

继续，今天讨论的是记录文件的读写。因为原来在 Brew 平台上实现的数独将题库是一个二进制文件，所以在Android就直接拿那个文件来用了。

计划实现两个函数，先是 LoadTiList()，加载题库，先装题库文件放在资源里，然后从资源里加载它作为一个 DataInputStream 即可。代码也没几行，如下：

```
   public static boolean LoadTiList(MainActivity me) {
|        DataInputStream in = null;
|        try
        {
|           in = new DataInputStream(me.getResources().openRawResource(R.ra
w.ti));
|
|           byte[] bufC4=new byte[4];
|           byte[] bufC81=new byte[81];
|
|           //总个数
|           in.read(bufC4,0,4);
|           int len=((int)bufC4[3]<<24)+((int)bufC4[2]<<16)+((int)bufC4[1]<
<8)+(int)bufC4[0];
|
|           for(int i=0;i<len;i++)
```

```
            {
                Question ti = new Question();

                //代码
                in.read(bufC4,0,4);
                ti.code=(long) (((long)bufC4[3]<<24)+((long)bufC4[2]<<16)+(
(long)bufC4[1]<<8)+(long)bufC4[0]);
                //时间
                in.read(bufC4,0,4);
                SharedPreferences sp = me.getPreferences(Context.MODE_WORLD
_READABLE);
                ti.time=sp.getLong(Long.toString(ti.code), 0);
                //数据
                in.read(bufC81,0,81);
                for(int j=0;j<81;j++)ti.data[j]=bufC81[j];
                me.tiList.add(ti);
            }
            in.close();
        }
         catch(Exception ex){
            return false;
        }
         finally{
             try{in.close();}catch(Exception e){}
```

```
        }
        return true;
    }
```

这里最麻烦的是因为 java 里没有 unsigned 类型，所以会溢出，比较郁闷，这个没有解决，只能是生成题库文件里注意一下了，不能与brew平台共用那个题库文件了。

二是保存记录，在brew平台我直接用一个文件搞定，读写它，但是 android 不能这样了，因为 ti.dat 是从资源中加载的，所以只能是静态的，不可修改，那记录只能放入 preferences 中了，代码如下：

```
    public static boolean SaveTiList(MainActivity me)
    {
        try
        {
            SharedPreferences sp=me.getPreferences(Context.MODE_WORLD_WRITEABLE);
            Question ti = me.gridView.ti;
            sp.edit().putLong(Long.toString(ti.code),ti.time);
            sp.edit().commit();
        }
        catch(Exception ex){
            return false;
        }
        return true;
    }
```

SharePreferences 可以按 key-value 来保存记录，所以 key 用题目的 code，则 value

就是解它所用的时间了。

Android 不能直接访问 app 目录下的文件，所以不能够象brew那样将数据文件放在程序目录下供它读写，而在 Activity 中提供的两个函数 openFileOutput 和 openFileInput，虽可用来读写文件，但是总是不太方便。

另外，用 SQLite 其实也不方便，因为手机中弄这些东西，事实上用处不大。

URL: http://www.sf.org.cn/Android/lumen/20985.html

Android 学习笔记(11)－开始做一个数独游戏[下]

继续，最后再讨论一下定时器的实现。

本来很简单的一件事，直接用 java.util.timer 应该就够用了，但是发现在它的 task 中无法去 invalidate 我们的 MainView，很郁闷。这一点的处理说明 Android还是相对线程安全的。

折腾良久，明白了非得再做一个 Handler，才能在线程中操作界面元素。所以，代码比brew复杂了一点。

先还是用 Timer 和 TimerTask 来做，如下：

```
 public TimerHandler timerHandler;
 public Timer timer;
public MyTimerTask    task;

... ...

timer=new Timer(true);
task=new MyTimerTask(this);

... ...
```

那个 MyTimerTask 是 MainActivity 的一个内嵌类，实现如下：

```
    private class MyTimerTask extends TimerTask
    {
        private MainActivity me;
        private int a=0;

         public MyTimerTask(MainActivity p){
            me=p;
        }
         public void run(){
            me.gridView.time++;
            Log.d("MyTask",Integer.toString(me.gridView.time));

            timerHandler.sendEmptyMessage(0);
        }
    }
```

这里做两件事，一是将 gridView 中的 time 加一，二是发送一个消息通知 timerHandler。原来我在这里直接让 MainView 去刷新屏幕，发现不行，所以就改成这样处理了。

然后就是如何实现 TimerHandler 类的，也不复杂，就是让它去刷新一下屏幕即可。

```
public class TimerHandler extends Handler {

    private MainView me;
     public TimerHandler(MainView m){
        me=m;
    }
```

```
|
|    @Override
|     public void handleMessage(Message msg) {
|        Log.d("Ti",msg.toString());
|        me.invalidate();
|    }
|
}
```

如此一来，就顺了。

在 MainView 中的 onDraw，根据当前的 time 值显示成 00:00:00 的格式即可。

另外，发现 Android 的模拟器运算速度不如 BREW 的模拟器，相当的慢。

URL: http://www.sf.org.cn/Android/lumen/20986.html

Android 学习笔记(12)－开始做一个数独游戏[补充]

再补充一点吧，如果需要给游戏加上背景音乐，其实也是非常容易的事情。因为 Android提供了一个 MediaPlayer 类可以方便的播放音乐文件。

android.media.MediaPlayer 类没有构造函数，一般是用它的静态方法 create 生成实例，简单地告诉它音乐文件的资源 ID 即可（支持 mp3/wav/midi 等）。

首先，我们得建立一个 Service，就叫 MediaService 吧，它的代码如下：

```
   Android/UploadFiles_8448/200804/20080419152842539.gif" align=top>
Android/UploadFiles_8448/200804/20080419152843671.gif"
align=top>public class MediaService extends Service implements MediaPlayer.O
nErrorListener,MediaPlayer.OnCompletionListener,MediaPlayer.OnPreparedListen
er {
|
|      ... ...
```

```
    private MediaPlayer player;

    @Override
     protected void onDestroy() {
        // TODO Auto-generated method stub
        super.onDestroy();
         if(player!=null){
            player.stop();
            player.release();
        }
    }

    @Override
     protected void onStart(int startId, Bundle arguments) {
        // TODO Auto-generated method stub
        Log.d("Media","onStart");

        player=MediaPlayer.create(this.getApplication(),R.raw.tonhua);
         if(player!=null){
            player.setAudioStreamType(AudioSystem.STREAM_MUSIC);

            player.setOnCompletionListener(this);
            player.setOnPreparedListener(this);
            player.setOnErrorListener(this);
```

```
|
|            player.prepareAsync();
|        }
|    }
|
|    @Override
|     public void onCompletion(MediaPlayer arg0) {
|        // TODO Auto-generated method stub
|        Log.d("Media","finished.");
|    }
|
|    @Override
|     public void onPrepared(MediaPlayer arg0) {
|        // TODO Auto-generated method stub
|        Log.d("Media","prepared.");
|        player.start();
|    }
|
|    @Override
|     public void onError(MediaPlayer arg0,int what, int extra) {
|
|        Log.d("Media","onError");
|        player.stop();
|    }
└}
```

这个服务主要就是用一个 MediaPlayer 去播放资源中的 tonghua（一首 MP3 音乐）。次序一般是先 create 出这个实例，然后 prepare 一下（如果是文件直接 prepare，如果是流则最好异步 parepareAsync），接着就可以 start 了，同步可以直接 start，异步则必须放到 onPrepared 中再 start。

在 MainActivity 中启动这个服务即可。

```
mediaServiceIntent= new Intent();
mediaServiceIntent.setClass(this, MediaService.class);


 this.startService(mediaServiceIntent, new Bundle());
```

当前，在 Activity 停止时也别忘了将这个 Service 停掉，而在 Service 停止时关掉 MediaPlayer。

在模拟器上试了，效果不是太好，声音有点断断续续，不知道是不是我的解码器的问题（Vista 系统）。

URL: http://www.sf.org.cn/Android/lumen/20987.html

消息机制，异步和多线程

有了 framework 后，我们不用面对赤裸裸的 OS API，做一些重复而繁杂的事情。但天下没有免费的午餐，我们还是需要学会高效正确的使用不同的 framework，很多处理某一特定问题的手法在不同的 framework 中，用起来都会有所不同的。

在Android中，下层是 Linux 的核，但上层的 java 做的 framework 把这一切封装的密不透风。以消息处理为例，在 MFC 中，我们可以用 PreTranslateMessage 等东东自由处理消息，在 C#中，Anders Hejlsberg 老大说了，他为我们通向底层开了一扇“救生窗”，但很遗憾，在Android中，这扇窗户也被关闭了（至少我现在没发现...）。

在Android中，你想处理一些消息（比如：Keydown 之类的...），你必须寻找 Activity

为你提供的一些重载函数（比如 onKeyDown 之类的...）或者是各式各样的 listener（比如 OnKeyDownListner 之类的...）。这样做的好处是显而易见的，越多的自由就会有越多的危险和越多的晦涩，条条框框画好了，用起来省心看起来省脑，这是一个设计良好的 framework 应该提供的享受。对于我目前的工程而言，我没有什么 BT 的需求在当前 API 下做不到的，google 的设计 ms 还是很 nice 的。

但世界是残酷的，有的时候我们还是必须有机制提供消息的分发和处理的，因为有的工作是不能通过直接调用来同步处理的，同时也不能通过 Activity 中内嵌的消息分发和接口设定来做到，比如说事件的定时触法，异步的循环事件的处理，高耗时的工作等等。在 Android 中，它提供了一些蛮有意思的方式来做这件事情（不好意思，我见不多识不广，我没见过类似玩法，有见过的提个醒 && 嘴下超生^_^），它有一个 android.os.Handler 的类，这个类接受一个 Looper 参数，顾名思义，这是一个封装过的，表征消息循环的类。默认情况下，Handler 接受的是当前线程下的消息循环实例，也就是说一个消息循环可以被当前线程中的多个对象来分发，来处理（在 UI 线程中，系统已经有一个 Activity 来处理了，你可以再起若干个 Handler 来处理...）。在实例化一个 handlerInstance 之后，你可以通过 sendMessage 等消息发送机制来发送消息，通过重载 handleMessage 等函数来分发消息。但是！该 handlerInstance 能够接受到的消息，只有通过 handlerInstance.obtainMessage 构造出来的消息（这种说法是不确切的，你也可以手动 new 一个 Message，然后配置成该 handlerInstance 可以处理的，我没有跟进去分析其识别机制，有兴趣的自己玩吧^_^）。也就是说 A, B, C, D 都可以来处理同一线程内的消息分发，但各自都只能处理属于自己的那一份消息，这抹杀了 B 想偷偷进入 A 领地，越俎代庖做一些非份之事的可能（从理论上看，B 还是有可能把消息伪装的和 A 他们家的一样，我没有尝试挑战一下 google 的智商，有 BT 需求的自行研究^_^）。这样做，不但兼顾了灵活性，也确保了安全性，用起来也会简单，我的地盘我做主，不用当心伤及无辜，左拥右抱是一件很开心的事情。。。

很显然，消息发送者不局限于自己线程，否者只能做一些定时，延时之类的事情，岂不十分无趣。在实例化 Handler 的时候，Looper 可以是任意线程的，只要有 Handler 的指针，任何线程也都可以 sendMessage（这种构造方式也很有意思，你可以在 A 线程里面传 B 线程的 Looper 来构造 Handler，也可以在 B 线程里构造，这给内存管理的方法带来很大的变数...）。但有条规则肯定是不能破坏的，就是非 UI 线程，是不能触碰 UI 类的。在不同平台上有很多解决方式（如果你有多的不能再多的兴趣，可以看一下很久很久以前我写的一个,不 SB 不要钱)。我特意好好跟了一下 android 中的 AsyncQueryHandler 类，来了解 google 官方的解决方案。

AsyncQueryHandler 是 Handler 的子类，文档上说，如果处理 ContentProvider 相关的内容，不用需要自行定义一套东西，而可以简单的使用 async 方式。我想指代的就应该是 AsyncQueryHandler 类。该类是一个典型的模板类，为 ContentProvider 的增删改查提供了很好的接口，提供了一个解决架构，final 了一些方法，置空了一些方法。通过派生，实例化一些方法（不是每个对 ContentProvider 的处理多需要全部做增删改查，我想这也是该类默认置空一些方法而不是抽象一些方法的原因），来达到这个目的。在内部，该类隐藏了多线程处理的细节，当你使用时，你会感觉异常便利。以 query 为例，你可以这么来用：

// 定义一个 handler，采用的是匿名类的方式，只处理 query，因此只重写了 onQueryComplete 函数：

```
queryHandler = new AsyncQueryHandler(this.getContentResolver()){
// 传入的是一个 ContentResolver 实例，所以必须在 OnCreate 后实例化该 Handler 类
 @Override
    protected void onQueryComplete(int token, Object cookie, Cursor cursor)
{
```

```
        // 在这里你可以获得一个 cursor 和你传入的附加的 token 和 cookie。
        // 该方法在当前线程下（如果传入的是默认的 Looper 话），可以自由设定
    UI 信息
    }
};
```

// 调用时只需要调用 startQuery(int token, Object cookie, ContentURI uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)函数即可：

queryHandler.startQuery(token, cookie, uri, projection, selection, selectionArgs, sortBy);

可见，该类的使用是多么简单（其实现可不会很容易，因为我尝试做了一次造车轮的工作*_*），比直接用 Handler 简单无数倍。但让我倍感孤独的是，不知道是没人做异步的 ContentProvider 访问，还是这个类使用太过于弱智（这个使用方法可是我摸索了半天的啊，难道我真的如此的弱@_@），抑或是大家都各有高招，从 SDK 到网上，没有任何关于该类的有点用的说明。而我又恰巧悲伤的发现，这个类其实有很多的问题，比如他吃掉异常，有错误时只是简单的返回 null 指针（这个其实不能怪他，你可以看看这里...）；当你传一个 null 的 ContentResolver 进去的时候，没有任何异常，只是莫名其妙的丢弃所有消息，使你陷入苦苦的等待而不知其因；更愤慨的是，他的 token 传递竟然有 Bug（难道还是我使用不对&_&），从 startXX 传入的 token，到了 onXXComplete 里面一律变成 1,而文档上明明写着两个是一个东西(我的解决方法是用 cookie 做 token，这个不会丢*_*）。不过我暂时还没有遗弃它的打算，虽然没人理睬，虽然有一堆问题，虽然我按图索骥造了个新轮子，但为了节省剩下的一些无聊的工作，我决定苟且偷生了。。。

还是习惯性跑题了，其实，我是想通过我对这个类的无数次 Debugger 跟进，说说它

的多线程异步处理的解决策略的。他的基本策略如下：

1. 当你实例化一个 AsyncQueryHandler 类时（包括其子类...），它会单件构造一个线程（后面会详述...），这个线程里面会构建一个消息循环。

2. 获得该消息循环的指针，用它做参数实例化另一个 Handler 类，该类为内部类。至此，就有了两个线程，各自有一个 Handler 来处理消息。

3. 当调用 onXXX 的时候，在 XXX 函数内部会将请求封装成一个内部的参数类，将其作为消息的参数，将此消息发送至另一个线程。

4. 在该线程的 Handler 中，接受该消息，并分析传入的参数，用初始化时传入的 ContentResolver 进行 XXX 操作，并返回 Cursor 或其他返回值。

5. 构造一个消息，将上述返回值以及其他相关内容绑定在该消息上，发送回主线程。

6. 主线程默认的 AsyncQueryHandler 类的 handleMessage 方法（可自定义，但由于都是内部类，基本没有意义...）会分析该消息，并转发给对应的 onXXXComplete 方法。

7. 用户重写的 onXXXComplete 方法开始工作。

这就是它偷偷摸摸做过的事情，基本还是很好理解的。我唯一好奇的是它的线程管理方式，我猜测他是用的单件模式。第一个 AsyncQueryHandler 的实例化会导致创建一个线程，从此该线程成为不死老处男，所有的 ContentResolver 相关的工作，都由该线程统一完成。个人觉得这种解决方式很赞。本来这个线程的生命周期就很难估量，并且，当你有一个 ContentProvider 的请求的时候，判断你会做更多的类似操作并不过分。就算错了，花费的也只是一个不死的线程（与进程同生死共存亡...），换来的却是简单的生命周期管理和无数次线程生死开销的节约。同时另外一个很重要的问题，他并会涉及到单件中数据同步的问题，每个类都有各自的 Handler 类，彼此互不干扰，分发可以分别进行。当多个数据请求的时候，在同一个 ContentResolver 上进行的可能微乎其微，这就避免了堵塞。总而言之，这套解决办法和 Android 的整体设计算是天作之合了。

所以建议，如果你有什么非 ContentProvider 操作，却需要异步多线程执行的话，模拟

一套，是个不错的策略，当然，具体情况具体分析，生搬硬套是学不好马列主义的。。。

URL: http://www.sf.org.cn/Android/lumen/21075.html

显示控件使用

Android的界面显示同样也是基于控件的。通常是用View（包括ViewGroup）控件配上 XML 的样式来做的。具体细节不想说了，可以参考 Samples 里的 ApiDemos/View，和 View 的 Doc，以及 Implementing a UI 这篇 Doc。其他还有很多，感觉算是 SDK 讲述的最多的内容。

从控件的使用上,和网页的设计类似,尽量用parent_width之类的抽象长度,用Theme来做风格，抽取所有的字串等信息做本地化设计。相关内容参看 Implementing a UI 就好。

一类比较重要的是数据绑定控件。如果做过 ASP.Net 会从中看到很多类似的地方。一个支持数据绑定的控件，比如 List View。可以通过一个 ListAdapter 绑定到一个数据源上。ListAdapter 是一个抽象类，主要的实现类包括 SimpleAdapter 和 SimpleCursorAdapter。前者是绑定一个静态的 Array，后者是绑定一个动态的 Cursor。Cursor 前面说过，是一个指向数据源的随机迭代器，将 View 绑定到 Cursor 通常要设置这样几个参数。一个是每一行的样式，称作 Row Layout，其实就是一个普通的 Layout 的 XML 文件。还有就是一个列和现实控件的对应关系。那个控件显示哪个列的值，这是需要配置的。为了定制一个良好的数据显示控件，最简单你可以定制很 PP 的 Row Layout，复杂一点就是可以重载绑定控件 View，或者是适配器 ListAdapter。如果是一个数据显示密集的应用，且你对 UI 有些追求，这个工作估计是必不可少的。

一个主要用于显示数据内容的 Activity，可以选择派生自 List Activity。它提供了一个具有 List View 的 Layout，还有 simple_list_item_1, simple_list_item_2, two_line_list_item 等默认的 Row Layout，还有一些比较不错的 API，和可供响应选择 Item 的事件。可以满足你比较基础的需求。如果你觉得只有一个 List View 的界面太突

兀，你可以为这个 List Activity 指定一个 Layout，需要注意的是，你需要提供一个 id 为@android:id/list 的 ListView 控件，避免 Activity 在内部偷偷寻找该控件的时候失败。

除了这些要求，做好 UI 还有注意易用性和效率。快捷键是一个比较不错的选择，在 Activity 中调用 setDefaultkeyMode(SHORTCUT_DEFAULT_KEYS)，可以开启快捷键模式，然后你可以将菜单绑定到指定快捷键上就 OK 了。个人觉得 Tip 也是一个比较重要的东西，但目前观察看来，这个东西只能够自己提供了。界面的动态性有时候是不可避免的，比如说菜单就是一个需要经常根据光标位置提供不同的选项。这个东西Android很人道的考虑到了，你可以参看 NodeList 这个 Sample。它采取的应该是一个静态模拟动态的方式，这样有助于提高速度。你也可以利用 ViewInflate，动态从一个 XML 创建一个控件。成本据 Doc 说很大，不到万不得已不要使用。

URL: http://www.sf.org.cn/Android/lumen/21073.html

Intent 消息传递

在前面写Android的 ContentProvider 时候，可以看到那是基于观察者模式的一个消息传递方法。每一个 Cursor、ContentResolver 做为一个小的注册中心，相关观察者可以在这个中心注册，更新消息由注册中心分发给各个观察者。而在 MFC 或 Winform 中，都会形成一个消息网，让消息在网中流动，被各节点使用、吃掉或者在出口死掉。

相比之下，我个人觉得基于 Intent 的Android核心消息传递机制是有所不同的。它应该会有一个全局性的注册中心，这个注册中心是隐性的，整个Android系统中就那么一个。所有的消息接收者，都被隐形的注册到这个中心。包括 Activity，Service 和 IntentReceiver。其实说隐形注册是不确切的，所有注册都还是我们手动告诉注册中心的，只是与传统的方式不一样，我们通常不是通过代码，而是通过配置文件来做。在应用的 Manifest 中，我们会为一些 Activity 或 Service 添加上 Intent-filter，或在配置文件中添加<receiver></receiver>项。这其实就相当于向系统的注册中心，注册了相关

的 Intent-filter 和 receiver（这个事情完全可以通过代码来做，只是这样就失去了修改的灵活性）。

当程序有一个消息希望发出去的时候，它需要将消息封装成一个 Intent，并发送。这时候，应该是有一个统一的中心（恩，有可能 Android 底层实现的时候不是，但简单这样看是没问题的...）接受到这个消息，并对它进行解析、判定消息类型（这个步骤降低了耦合...），然后检查注册了相匹配的 filter 或 receiver，并创建或唤醒接收者，将消息分发给它。这样做有很多好处。虽然这种传递有的时候不如点对点的传递快（这有些需要速度的地方，我们看到 Android 会通过直接通信来做），但有时候又因为它只经过一跳（姑且这么叫吧...），比复杂的流动又要更快。更重要的是，它耦合性低，在手机平台这种程序组件多变的条件下使用十分适合。并且它可以很容易实现消息的精确或模糊匹配，弹性很大。（我个人曾想在开发一个 C++二次平台的时候引入这样的机制，但在 C++中，建立一套完整的数据 marshal 机制不容易，相比之下，用 java 来做会简单很多...）

恩，废话说了很多，具体讲讲 Android 中 Intent 的使用。当你有一个消息需要传递，如果你明确知道你需要哪个 Activity 或者其他 Class 来响应的话，你可以指定这个类来接受该消息，这被称为显性发送。你需要将 Intent 的 class 属性设置成目标。这种情况很常见，比如 startActivity 的时候，会清楚当前 Activity 完了应该是哪个 Activity，那就明确的发送这个消息。

但是，有的时候你并不确定你的消息是需要具体哪个类来执行，而只是知道接收者该符合哪些条件。比如你只需要有一个接收者能显示用户所选的数据，而不想制定某个具体的方法，这时候你就需要用到隐形发送（传统上，我们可能会考虑用多态，但显然这种方式更为灵活...）。在 Android 中，你可以为 Intent 指定一个 action，表示你这个指令需要处理的事情。系统为我们定义了很多 Action 类型，这些类型使系统与我们通信的语言（比如在 Activity 里面加一个 Main 的 filter，该 activity 就会做成该应用的

入口点），当然你也可以用于你自己的应用之间的通信（同样当然，也可以自定义...）。强烈建议，在自己程序接收或发出一个系统 action 的时候，要名副其实。比如你响应一个 view 动作，做的确实 edit 的勾当，你发送一个 pick 消息，其实你想让别人做 edit 的事，这样都会造成混乱。当然只有 Action 有时候是不够的，在 Android 中我们还可以指定 catalog 信息和 type/data 信息，比如所有的显示数据的 Activity，可能都会响应 View action。但很多与我们需要显示的数据类型不一样，可以加一个 type 信息，明确的指出我们需要显示的数据类型，甚至还可以加上一个 catalog 信息，指明只有你只有按的是“中键”并发出这样的消息才响应。

从上面可以看出,Android 的 Intent 可以添加上 class, action, data/type, catalog 等消息，注册中心会根据这些信息帮你找到符合的接收者。其中 class 是点对点的指示，一旦指明，其他信息都被忽略。Intent 中还可以添加 key/value 的数据，发送方和接收方需要保持统一的 key 信息和 value 类型信息，这种数据的 marshal 在 java 里做，是不费什么力气的。

Android 的 Intent 发送，可以分成单播和广播两种。广播的接收者是所有注册了的符合条件的 IntentReceiver。在单播的情况下，即使有很多符合条件的接收者，也只要有一个出来处理这个消息就好（恩，个人看法，没找到确切条款或抉择的算法，本来想实验一下，没来得及...），这样的情况很容易理解，当你需要修改某个数据的时候，你肯定不会希望有十个编辑器轮流让你来处理。当广播不是这样，一个 receiver 没有办法阻止其他 receiver 进行对广播事件的处理。这种情况也很容易理解，比如时钟改变了，闹钟、备忘录等很多程序都需要分别进行处理。在自己的程序的使用中，应该分清楚区别，合理的使用。

URL: http://www.sf.org.cn/Android/lumen/21072.html

ContentProvider 数据模型概述

Android的数据（包括 files, database 等...）都是属于应用程序自身，其他程序

无法直接进行操作。因此，为了使其他程序能够操作数据，在Android中，可以通过做成ContentProvider 提供数据操作的接口。其实对本应用而言，也可以将底层数据封装成ContentProvider，这样可以有效的屏蔽底层操作的细节，并且是程序保持良好的扩展性和开放性。

ContentProvider，顾名思义，就是数据内容的供应者。在Android中它是一个数据源，屏蔽了具体底层数据源的细节，在 ContentProvider 内部你可以用 Android 支持的任何手段进行数据的存储和操作，可能比较常用的方式是基于 Android 的 SQLite 数据库（恩，文档中和示例代码都是以此为例）。无论如何，ContentProvider 是一个重要的数据源，可以预见无论是使用和定制 ContentProvider 都会很多。于是花了点时间仔细看了看。

数据库操作

从我目前掌握的知识来看，SQLite 比较轻量（没有存储过程之类的繁杂手段），用起来也比较简单。实例化一个 SQLiteDatabase 类对象，通过它的 APIs 可以搞定大部分的操作。从 sample 中看，Android 中对 db 的使用有一种比较简单的模式，即派生一个 ContentProviderDatabaseHelper 类来进行 SQLiteDatabase 对象实例的获取工作。基本上， ContentProviderDatabaseHelper 类扮演了一个 singleton 的角色，提供单一的实例化入口点，并屏蔽了数据库创建、打开升级等细节。在 ContentProvider 中只需要调用 ContentProviderDatabaseHelper 的 openDatabase 方法获取 SQLiteDatabase 的实例就好，而不需要进行数据库状态的判断。

URI

像进行数据库操作需要用 SQL 一样，对 ContentProivder 进行增删改查等操作都是通过一种特定模式的 URI 来进行的（ig：content: //provider/item/id），URI 的能力与 URL 类似，具体细节可以查看 SDK。建立自己的 ContentProvider，只需要派生 ContentProivder 类并实现 insert, delete, update 等抽象函数即可。在这些接口中比

较特殊的是 getType(uri)。根据传入的 uri，该方法按照 MIME 格式返回一个字符串（==!没听过的诡异格式...）唯一标识该 uri 的类型。所谓 uri 的类型，就是描述这个 uri 所进行的操作的种类，比如 content://xx/a 与 content://xx/a/1 不是一个类型（前者是多值操作，后者是单值），但 content://xx/a/1 和 content://xx/a/2 就会是一个类型（只是 id 号不同而已）。

在 ContentProvider 通常都会实例化一个 ContentURIPraser 来辅助解析和操作传入的 URI。你需要事先（在 static 域内）为该 ContentURIPraser 建立一个 uri 的语法树，之后就可以简单调用 ContentURIPraser 类的相关方法进行 uri 类型判断（match 方法），获取加载在 uri 中的参数等操作。但我看来，这只是在使用上简化了相关操作（不然就需要自己做人肉解析了...），但并没有改变类型判定的模式。你依然需要用 switch...case...对 uri 的类型进行判断，并进行相关后续的操作。从模式来看，这样无疑是具有强烈的坏味道，类似的 switch...case...代码要出现 N 此，每次一个 ContentProvider 做 uri 类型的增减都会需要遍历修改每一个 switch...case...，当然，如果你使用模式（策略模式...）进行改造对手机程序来说无疑是崩溃似的（类型膨胀，效率降低...），所以，只能是忍一忍了（恩，还好不会扩散到别的类中，维护性上不会有杀人性的麻烦...）。

增删改查

ContentProvider 和所有数据源一样，向外提供增删改查操作接口，这些都是基于 uri 的指令。进行 insert 操作的时候，你需要传入一个 uri 和 ContentValues。uri 的作用基本就限于指明增减条目的类型（从数据库层面来看就是 table 名），ContentValues 是一个 key/value 表的封装，提供方便的 API 进行插入数据类型和数据值的设置和获取。在数据库层面上来看，这应该是 column name 与 value 的对应。但为了屏蔽 ContentProvider 用户涉及到具体数据库的细节，在 Android 的示例中，用了一个小小的模式。它为每一个表建一个基于 BaseColumn 类的派生类（其实完全可以不派生自

BaseColumn，特别当你的表不基于默认的自动 id 做主键的时候），这个类通常包括一个描述该表的 ContentURI 对象和形如 public static final TITLE = "title"这样的 column 到类数据的对应。从改变上角度来看，你可以修改 column 的名字而不需要更改用户上层代码，增加了灵活性。 insert 方法如果成功会返回一个 uri，该 uri 会在原有的 uri 基础上增加有一个 row id。对于为什么使用 row id 而不是 key id 我想破了脑袋。到最后，我发现我傻了，因为 ContentProvider 不一定需要使用数据库，使用数据库对应的表也可以没有主键，只有 row id，才能在任何底层介质下做索引标识。

但，基于 row id 在删除和修改操作是会造成一定的混乱。删除和修改操作类似。删除操作需要传入一个 uri，一个 where 字串，一组 where 的参数（做条件判定...），而修改操作会多一个 ContentValues 做更新值。着两个操作的 uri 都支持在末尾添加一个 row id。于是混乱就出现了。当在 where 参数中指明了 key id，而在 uri 中提供了 row id，并且 row id 和 key id 所指函数不一致的时候，你听谁的？示例代码中的做法是完全无视 row id（无语...），如此野蛮的方式我估计也只能在示例中出现，在实际中该如何用，恩，我也不知道。幸运的是，我看了下上层对 ContentProvider 的删除操作，其实都不会直接进行，而是通过调用 Cursor 的 delete 方法进行，在这前提下，我想 Cursor 会处理好这些东西吧。

最后一个操作是查询操作，可以想见，查询的参数是最多的，包括 uri 和一组条件参数。条件参数类型和标准的 sql 类似，包括 sort, projection 之类的。从这些参数到 sql 语句的生成，可以寻求 QueryBuilder 类的帮助，它提供了一组操作接口，简化了参数到 sql 的生成工作，哪怕你不懂 sql 都完全没有问题（这话说的我自己都觉得有点悬...）。查询返回一个 Cursor。Cursor 是一个支持随机读写的指针，不仅如此，它还提供了方便的删除和修改的 API，是上层对 ContentProvider 进行操作一个重要对象，需要仔细掌握（Cursor 还可以绑定到 view 上，直接送显，并与用户进行交互，真是程序越往上，封装越好，工作越机械没有复杂性了...）。

数据模型

在与界面打交道的 Cursor、ContentResolver 等数据操作层中，大量采用观察者模式建立数据层与显示层的联系。一个显示层的视图，可以做成某一种观察者注册到 Cursor 或 ContentResolver 等数据中间层中，在实现底层 ContentProvider 中，我们需要特别注意在对数据进行修改操作（包括增删改...）后，调用相应类型的 notify 函数，帮助表层对象进行刷新（还有一种刷新方式是从一个 view 发起的）。可以看到 Android 的整体数据显示框架有点像 MVC 的方式（贫瘠了...叫不出名）。Cursor、ContentResolver 相当于控制层，数据层和显示层的交互通过控制层来掌管，而且控制层很稳定不需要特别定制，通常工作只在定制数据层和显示层空间，还是比较方便和清晰的。

一个设计问题

现在有个设计问题，比如我要扩充一个已有的 ContentProvider（第三方提供），我是建立一个 ContentProvider，只保留第三方 ContentProvider 的 key 信息，并为其添加更多的信息，在表层维护这两个 ContentProvider 的联系好；还是建议一个 ContentProvider，以第三方的 ContentProvider 做一部分底层数据源，像表层提供一个 ContentProvider 好。

前者无疑在实现上简单一些，如果第三方改变，灵活性也更好，只是需要仔细维护表层的相关代码。后者实现上需要付出大量的苦力劳动，当表层使用会简单多了。我举棋不定，期待你的意见。。。

自定义 ContentProvider 的语义

ContentProvider 中，最重要的就是 query 操作。query 根据输入返回一个符合条件的 Cursor。这就可能出现以下几种情况：1. 查询成功，包含几个正确的结果；2. 查询失败，没有符合的结果；3. 输入错误， 触发了某个异常；4. 没能查询到结果，但无法确定是输入错误还是查询失败。第一种情况是我们最需要的，当然是需要正确维系的，而最后一种情况在大部分应用中应该不会出现（但在我的应用中会的*_#），而第二种第

三种是比较常见的。

经过我的测试，系统的 ContentProvider 维持这样的语义：如果是情况 2，返回正常的 Cursor，并且，其 count 为 0，相当于 empty cursor；如果是情况 3，不抛出任何异常，返回 null 的 Cursor。这样的话明明白白写出来是很好理解的，但由于没有官方的文档说明，在自定义的时候经常会误用。比如在某些情况下，用 null 表征查询失败，用抛出异常来描述错误的输入。

返回 empty cursor，如果是通过 databasecursor 自然会有 db 帮你维护，但是如果返回 ArrayListCursor，MergeCursor 或其他自定义的 Cursor，就需要自己维系了。ArrayListCursor 可以通过 new ArrayListCursor(Columns, new ArrayList(){})来提供。其中 Columns 一定不为 null。MergeCursor 不能以 new MergeCursor(new Cursor[]{}) 来创建，而需要通过 new MergeCursor(new Cursor[]{aEmptyCursor, ...}来维系（其实很好理解，我呆了...）。自定义的 Cursor 也一定要提供生成 empty cursor 的方式。

如果将 ContentProvider 作为一个单独的 module 来理解，不通过异常而是通过 null 来返回 MS 是有好处的。在 module 的出口吃掉所有异常，虽然不能提供足够的信息（异常信息全部写入日志），但可能会使上层使用更简单。但在 Android 中，我并没有感觉到这一点。作为 ContentProvider 的上层函数，ListActivity.managedQuery、ListView.setListAdapter 等，根本不能处理一个 null 的 Cursor，在 ListView 中这会触发一个异常。更无语的是，当你把一个 null Cursor 设置为 manage 的后。它不会立即抛异常，而是在 OnFreeze 等生命周期函数的时候，因无法处理 null Cursor 而抛出一个异常。这使得你根本无法在当地 catch 该异常，换句话，ListActivity 的 manageCursor 根本是个无法使用的函数。你必须用 getContext().query()获得 Cursor，然后判定该 Cursor 是否 null，在进行 startManagingCursor 进行绑定。这远不如直接用异常进行错误路径的处理来的统一和方便。

当然,有些东西我们是不能改变的,只能去适应。对于自定义的 cursor, ContentProvider,

最重要的，是在无人造错误输入的情况下返回 empty cursor，而不是 null。至于使用 null 响应还是异常响应上，我个人觉得还是和系统同步为好，虽然别扭，但至少统一不容易有歧义。

此外，ContentProvider 还有很多细致的语义。比如返回的 Cursor 需要绑定一个 URI，以便自动响应更新。自定义的更新需要支持 deleteRow 等操作语义等等。

PS：而上层的 ListView，更是陷阱重重。首先绑定到 ListView 的 Cursor 必须有_id 项，否则会有异常抛出。如果做过.net 的开发，这一点是可以想到的，但是，这种问题应该在文档中写明。另外，在 ListView 中，如果你不绑定一个数据源，你一定不能在 layout 中添加涉及内容的属性。比如 android:height="wrap_content"，这会在 onMeasure 的时候抛出异常。