**Android 第一行代码**

第一章

1. 2003年10月，Andy Rubin等人一起创办了Android公司。2005年8月被谷歌收购。2008年推出了Android系统的第一个版本。
2. Android大致可以分为四层架构，五块区域：
3. Linux内核层
4. 系统运行库层 /android运行时
5. 应用框架层
6. 应用层
7. Android四大组件：活动、服务、广播接收器、内容提供器。
8. Android系统提供主要的东西：四大组件、SQLite数据库、地理位置定位、强大的多媒体、传感器。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 文件名 | src | gen | assets | bin | libs | res |
| 内容 | 存放java代码 | 系统自动生成的，R.java文件 | 随程序打包的文件 | 编译产生的文件，包括安装包 | 第三方jar包 | 资源文件，所有的图片、布局、字符串等 |

1. AndroidManifest.xml文件：
2. 在程序中定义的所有四大组件都需要在这个文件里注册；
3. 添加权限声明；
4. 指定程序最低兼容版本和目标版本。
5. project.properties文件：指定了编译程序时所使用的SDK版本。
6. “@+”用于定义资源，“@”用于引用资源。
7. 使用log的优点：级别区分、过滤器、时间、可控制。

第二章

1. Activity需要在AndroidManifest.xml文件中注册。
2. 加载布局：setContentView()，一般传入布局文件的ID；
3. 隐藏标题栏：requestWindowFeature(Window.FEATURE\_NO\_TITLE)；
4. 获取布局文件中定义的元素：findViewById()；
5. Menu的使用：

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.main, menu);

return true;

}

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.add\_item:

Toast.makeText(this, "You clicked Add", Toast.LENGTH\_SHORT).show();

break;

case R.id.remove\_item:

Toast.makeText(this, "You clicked Remove", Toast.LENGTH\_SHORT).show();

break;

default:

}

return true;

}

1. Intent用于Android程序中各组件之间的交互，用于启动活动、启动服务、以及发送广播等场景。
2. Intent分为显式Intent和隐式Intent。显式直接调用Intent的构造函数；隐式指定AndroidManifest.xml中action和category，然后由系统分析判断要启动哪个activity。
3. Intent传值：intent.putExtra("extra\_data", data);

Intent 取值：Intent intent = getIntent();

String data = intent.getStringExtra("extra\_data");

1. 启动Intent：、
2. startActivity(intent); //不返回结果给上一个活动
3. startActivityForResult(intent, 1); //返回结果给上一个活动

onActivityResult(int requestCode, int resultCode, Intent data);

setResult(RESULT\_OK, intent);

1. Android 活动存放在返回栈中，是一种后进先出的数据结构。
2. 活动的四种状态：
   1. 运行状态：活动位于返回栈的栈顶；
   2. 暂停状态：活动不再处于栈顶位置，但仍然可见。如为占满屏幕的对话框；
   3. 停止状态：不再处于栈顶位置，并且完全不可见；
   4. 销毁状态：活动从返回栈中移除。
3. 活动的生存期：

onCreate()：活动第一次被创建的时候调用。完成初始化操作，如加载布局、事件绑定。

onStart()：活动由不可见变为可见的时候调用。

onResume()：和用户进行交互的时候调用，活动一定位于返回栈的栈顶。

onPause()：去启动或者恢复另一个活动的时候调用。

onStop()：活动完全不可见的时候调用。

onDestroy()：在活动被销毁时调用。

onRestart()：由停止状态变为运行状态之前调用（onRestart—>onStart—>onResume）。

1. onSaveInstanceState(Bundle outState)用于保存活动被回收时的临时数据；并且在onCreate(Bundle savedInstanceState)恢复保持的临时数据。  
   当App设置的targetVersion小于android3.0时在onPause()中会调用onSaveInstanceState(Bundle outState)；无论targetVersion是否小于android3.0，在onStop()中都会调用onSaveInstanceState(Bundle outState)
2. 活动的启动模式：

standand: 无论返回栈中是否存在，每次启动都会创建该活动的一个新的实例。

singleTop：如果返回栈的栈顶是该活动，则可直接使用它，不会再创建新的活动实例。

singleTask：如果返回栈中存在该活动，则将其之上的所有活动出栈后使用该实例。否则创建一个新实例。

singleInstance(开启一个新的返回栈)：多个应用程序共享一个活动实例。

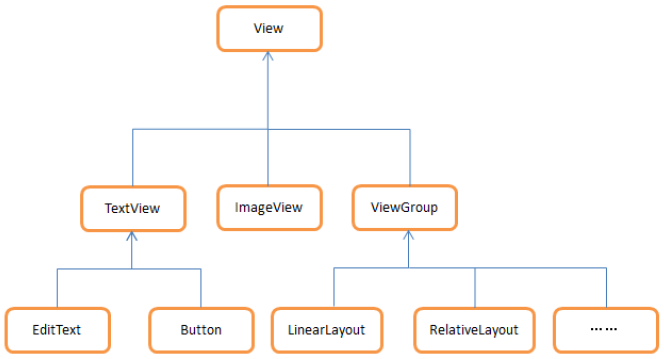
1. 活动管理器：第82页。

第三章

1. android:gravity（是本元素所有子元素的重力方向）

android:layout\_gravity( 是本元素相对于父元素的重力方向 )

1. android:hint属性来指定了一段提示性的文本
2. android:maxLines可以指定EditText的最大行数。
3. 四种常用布局：LinearLayout、RelativeLayout、FrameLayout、TableLayout



1. ListView是典型MVC架构。ListView通过getView()中的view类型参数和ViewHolder类来实现优化。

**ArrayAdapter<String> adapter = new ArrayAdapter<String>( MainActivity.this, android.R.layout.simple\_list\_item\_1, data);**

**ListView listView = (ListView) findViewById(R.id.list\_view);**

**listView.setAdapter(adapter);**

1. px：像素，显示屏的最小元素单元；

pt：磅数，常作为字体的单位；

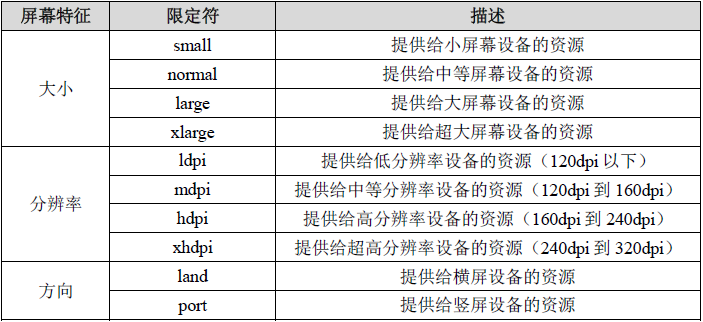
dp：密度无关的像素，可以自适应不同屏幕分辨率；

sp：可伸缩像素，解决文字大小适配问题。

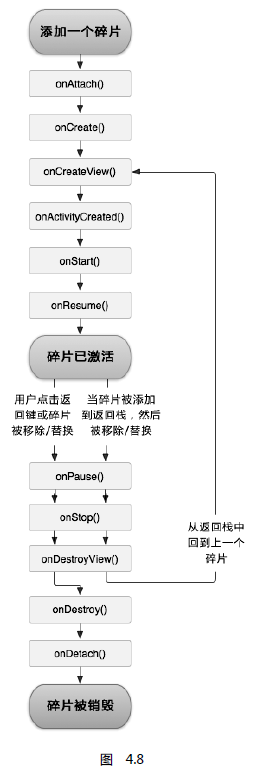
1. 制作Nine-Patch图片：上边框和左边框绘制的部分表示图片拉伸的区域；下边框和右边框绘制的部分则表示内容会被放置的区域。

第四章

1. 碎片（Fragment）是一种可以嵌入在活动当中的UI片段，它能让程序更加合理和充分地利用大屏幕的空间。
2. 动态添加碎片的5个步骤：
3. 创建待添加的碎片实例。
4. 获取到FragmentManager，在活动中可以直接调用getFragmentManager()方法得到。
5. 开启一个事务，通过调用beginTransaction()方法开启。
6. 向容器内加入碎片，一般使用replace()方法实现，需要传入容器的id和待添加的碎片实例。
7. 提交事务，调用commit()方法来完成。
8. Fragment的使用：ViewPage + Fragments(Fragment列表) + FragmentPagerAdapter。
9. 碎片的四个状态：运行状态、暂停状态、停止状态和销毁状态。
10. 碎片的回调：
    1. onAttach(): 当碎片和活动建立关联的时候调用；
    2. onCreateView(): 为碎片创建视图（加载布局）时调用；
    3. onActivityCreated(): 确保与碎片相关联的活动一定已经创建完毕的时候调用；
    4. onDestroyView() ：当与碎片关联的视图被移除的时候调用；
    5. onDetach(): 当碎片和活动解除关联的时候调用。
11. 限定符：



1. 碎片完整生命周期：



第五章

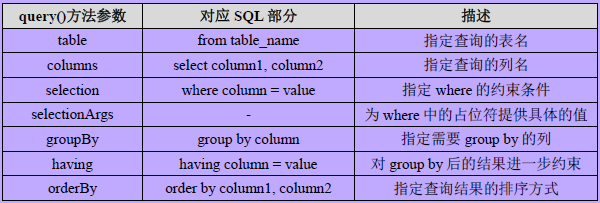
1. Android中的广播主要可以分为两种类型，标准广播（完全异步执行的广播）和有序广播（同步执行的广播）。
2. 注册广播的两种方式：

代码中注册（动态注册）：可以自由地控制注册与注销。

AndroidManifest.xml中注册（静态注册）：实现开机启动。

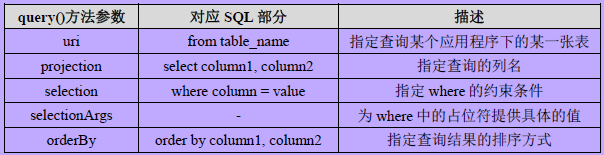
第六章

1. Android实现数据存储的5种方式：
2. 使用SharedPreferences存储数据；
3. 文件存储数据；
4. SQLite数据库存储数据；
5. 使用ContentProvider存储数据；
6. 网络存储数据；
7. SharedPreferences存储是采用键值对的方式，并使用XML格式对数据进行管理。
8. 获取SharedPreferences对象的三种方法：
9. Context类中的getSharedPreferences()方法；
10. Activity类中的getPreferences()方法；
11. PreferenceManager类中的getDefaultSharedPreferences()方法。
12. SharedPreferences实现文件存储的三步骤：
13. 调用SharedPreferences对象的edit()方法来获取一个SharedPreferences.Editor对象。
14. 向SharedPreferences.Editor对象中添加数据，比如添加一个布尔型数据就使用putBoolean方法，添加一个字符串则使用putString()方法，以此类推。
15. 调用commit()方法将添加的数据提交，从而完成数据存储操作。
16. SQLiteOpenHelper类用于数据库的创建和升级，有两个抽象方法onCreate()和onUpgrade()，两个实例方法getReadableDatabase()和getWritableDatabase()。



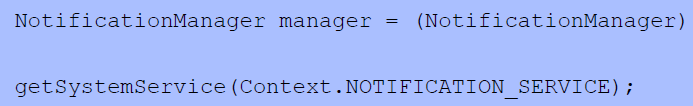
第七章

1. 内容提供器（Content Provider）主要用于在不同的应用程序之间实现数据共享的功能。
2. 内容提供器可以选择只对哪一部分数据进行共享。
3. Context中的getContentResolver()方法获取到ContentResolve类的实例。
4. 内容URI分为权限（authority）和路径（path）两部分。



第八章

1. 通知（Notification）：当应用程序不在前台运行时，可借助通知来发出提示信息。
2. 获取NotificationManager类实例：



1. PendingIntent可以用于启动活动、启动服务以及发送广播等。

第九章

* 1. 服务（Service）是Android实现程序后台运行的解决方案，适合用于去执行那些不需要和用户交互而且还要求长期运行的任务。
  2. 服务的运行不依赖于任何用户界面，而是依赖于创建服务时所在的应用程序进程。
  3. 服务运行在主线程中，一般在服务内部创建子线程来执行具体任务，防止主线程被阻塞。
  4. Android中的异步消息处理主要由四个部分组成：

Message：用于在线程之间传递的消息。

Handler：用于发送和处理消息。

MessageQueue：消息队列，用于存放消息。每个线程中只会有一个MessageQueue对象。

Looper：每个线程中的MessageQueue的管家，每个线程中也只会有一个Looper对象。

子线程中使用handler需要手动绑定Looper对象，详见如下：

<http://www.cnblogs.com/xgjblog/p/5258947.html>

<http://blog.csdn.net/shaoenxiao/article/details/54561753?utm_source=itdadao&utm_medium=referral>

1. 服务需要在AndroidManifest.xml文件中进行注册。
2. 服务中的四个方法：
   1. onCreate()方法会在服务创建的时候调用；
   2. onStartCommand()方法会在每次服务启动的时候调用；
   3. onBind()用bindService()来获取一个服务的持久连接
   4. onDestroy()方法会在服务销毁的时候调用。
   5. 服务启动的两种方法：
   6. startService()-> onCreate()-> onStartCommand()

stopService()或stopSelf()-> onDestroy()

* 1. bindService()->onCreate()->onBind()

unbindService()-->onUnbind()-->onDestroy()

第十章

1. 在Android上发送HTTP请求的方式一般有两种，HttpURLConnection和HttpClient。一般都要开启子线程。
2. 通过HttpURLConnection实现访问网络
3. URL url = new URL("http://www.baidu.com");
4. HttpURLConnection connection = (HttpURLConnection) url.openConnection();
5. 设置HttpURLConnection属性
6. 调用getInputStream()方法就可以获取到服务器返回的输入流。
7. 通过HttpClient实现访问网络
8. HttpClient httpClient = new DefaultHttpClient();
9. 发起GET请求

HttpGet httpGet = new HttpGet("http://www.baidu.com");

httpClient.execute(httpGet);

或者发起POST请求

HttpPost httpPost = new HttpPost("http://www.baidu.com");

List<NameValuePair> params = new ArrayList<NameValuePair>();

params.add(new BasicNameValuePair("username", "admin"));

params.add(new BasicNameValuePair("password", "123456"));

UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params, "utf-8");

httpPost.setEntity(entity);

1. httpClient.execute(httpPost);
2. 获取返回结果HttpEntity entity = httpResponse.getEntity();

String response = EntityUtils.toString(entity, "utf-8");

1. 在网络上传输数据时最常用的格式有两种，XML和JSON。
2. 解析XML格式数据的方法：Pull解析、SAX解析和DOM解析。
3. JSON比XML体积小，但语义性不如XML。
4. 解析JSON使用JSONObject。

JSONArray jsonArray = new JSONArray(jsonData);

JSONObject jsonObject = jsonArray.getJSONObject(i);

String id = jsonObject.getString("id");

第十一章

1. LocationManager locationManager = (LocationManager)getSystemService(Context.LOCATION\_SERVICE);

Location location = locationManager.getLastKnownLocation(LocationManager.NETWORK\_PROVIDER);

List<String> providerList = locationManager.getProviders(true);

locationManager.requestLocationUpdates(provider, 5000, 1, locationListener);

第十二章

1. Android支持的传感器，如光照传感右器、加速度传感器、地磁传感器、压力传感器、温度传感器等。
2. SensorManager senserManager = (SensorManager) getSystemService(Context.SENSOR\_SERVICE);

SensorEventListener listener = new SensorEventListener() {

@Override

public void onAccuracyChanged(Sensor sensor, int accuracy) { }

@Override

public void onSensorChanged(SensorEvent event) { }

};

sensorManager.unregisterListener(listener);

第十二章

1. Application类用于管理程序内的全局状态信息，比如全局Context。
2. 使用Intent来传递对象通常有两种实现方式：Serializable和Parcelable。
3. 实现序列化只需让一个类去实现Serializable接口。