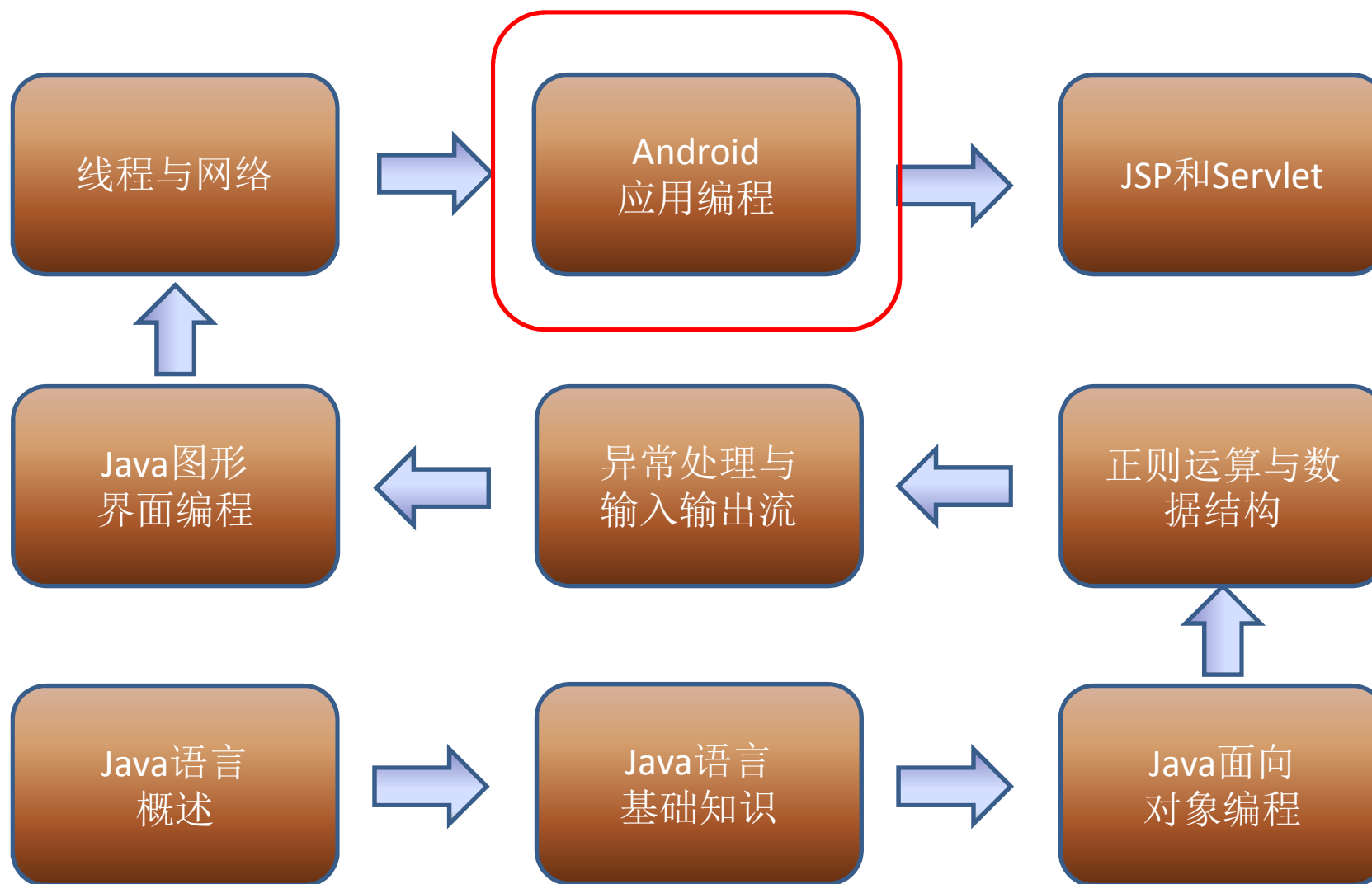




Android 应用编程



课程内容安排



课前思考

- 如何进行手机界面布局？
- 如何不用写Java代码就进行界面布局？
- 如何响应用户操作？
- 如何在不同界面之间进行跳转？
- 如何采集手机上的传感器信息？



Handset Manufacturers



Mobile Operators



open
handset
alliance

Software



Semiconductor



Commercialization





Android的发展

- 2005年7月 Google收购Android公司
- 2007年11月5日 OHA成立
- 2007年11月12日 发布第一版Android SDK
- 2008年8月28日 Android Market 上线
- 2008年9月23日 T-Mobile G1 上市
- 2008年9月23日 Android 1.0 SDK release 1 发布





Android版本进化



Cupcake
Android 1.5



Donut
Android 1.6



Eclair
Android 2.0/2.1



Froyo
Android 2.2



Gingerbread
Android 2.3



Honeycomb
Android 3.0

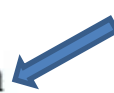


Ice Cream Sandwich
Android 4.0

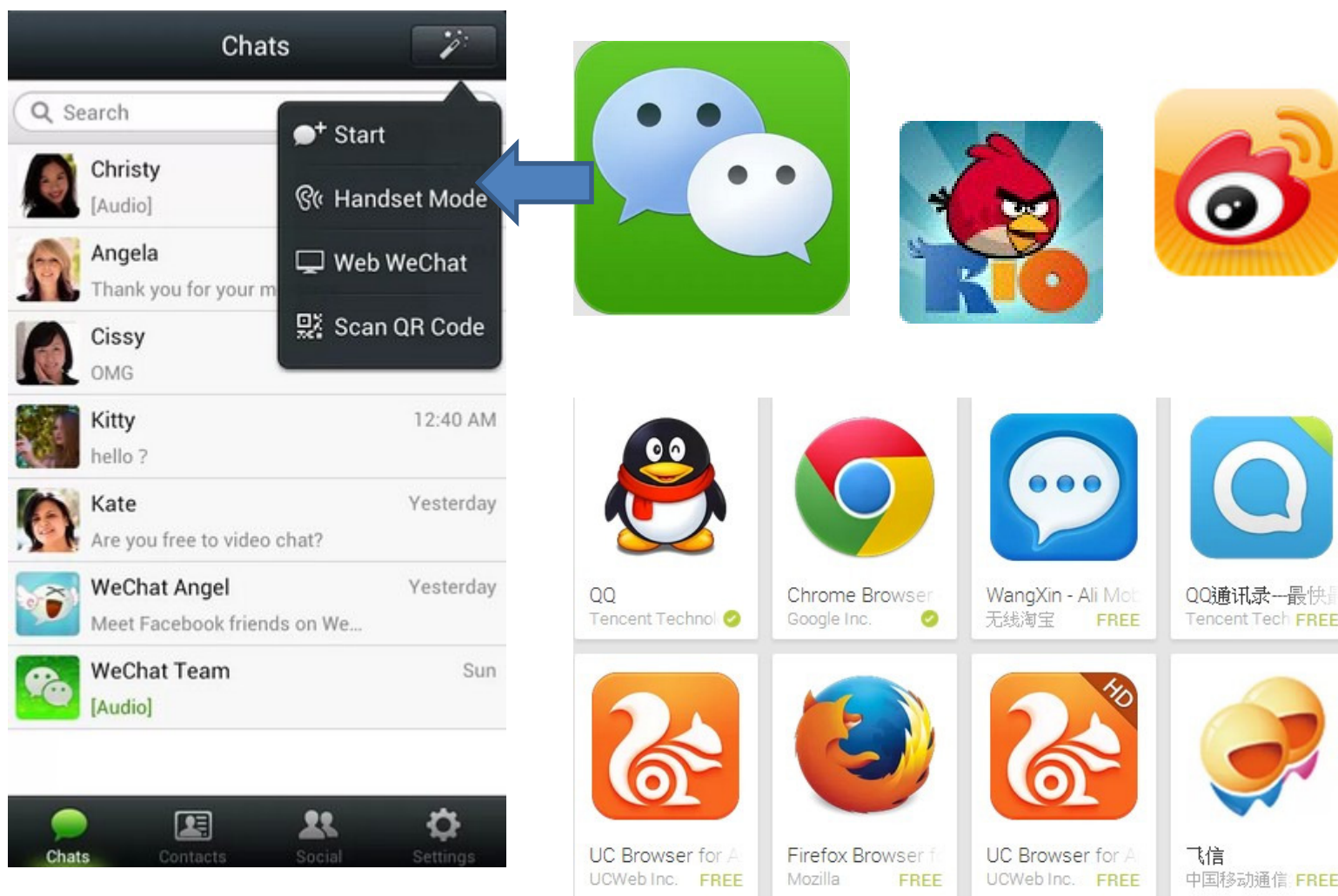


Jelly Bean
Android 4.1
& Android 4.2

最新
Android 4.3



网络上比较流行的应用

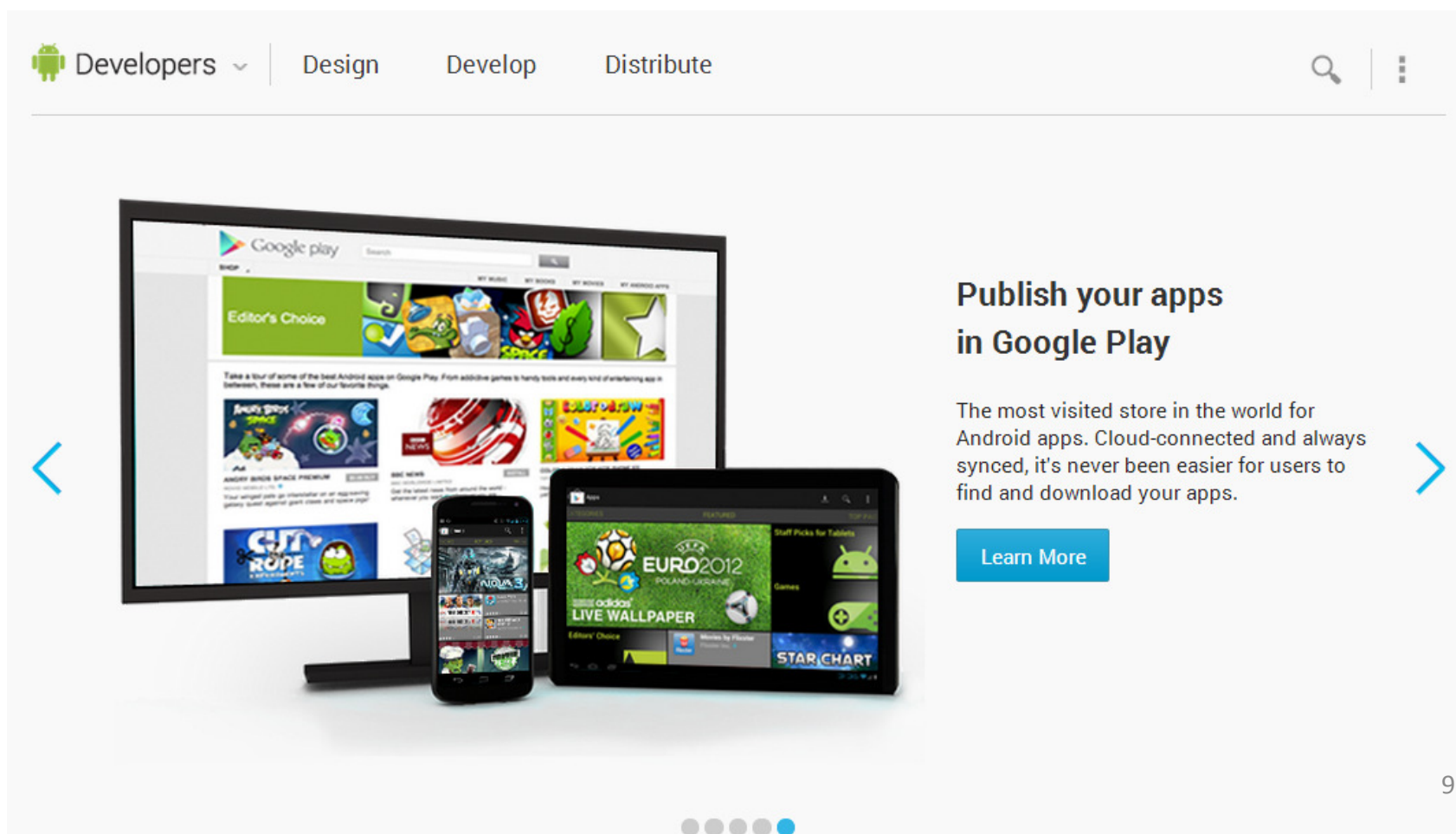


网络上比较流行的游戏



Android 开发者网站

- <http://developer.android.com>



Developers | Design | Develop | Distribute

Publish your apps in Google Play

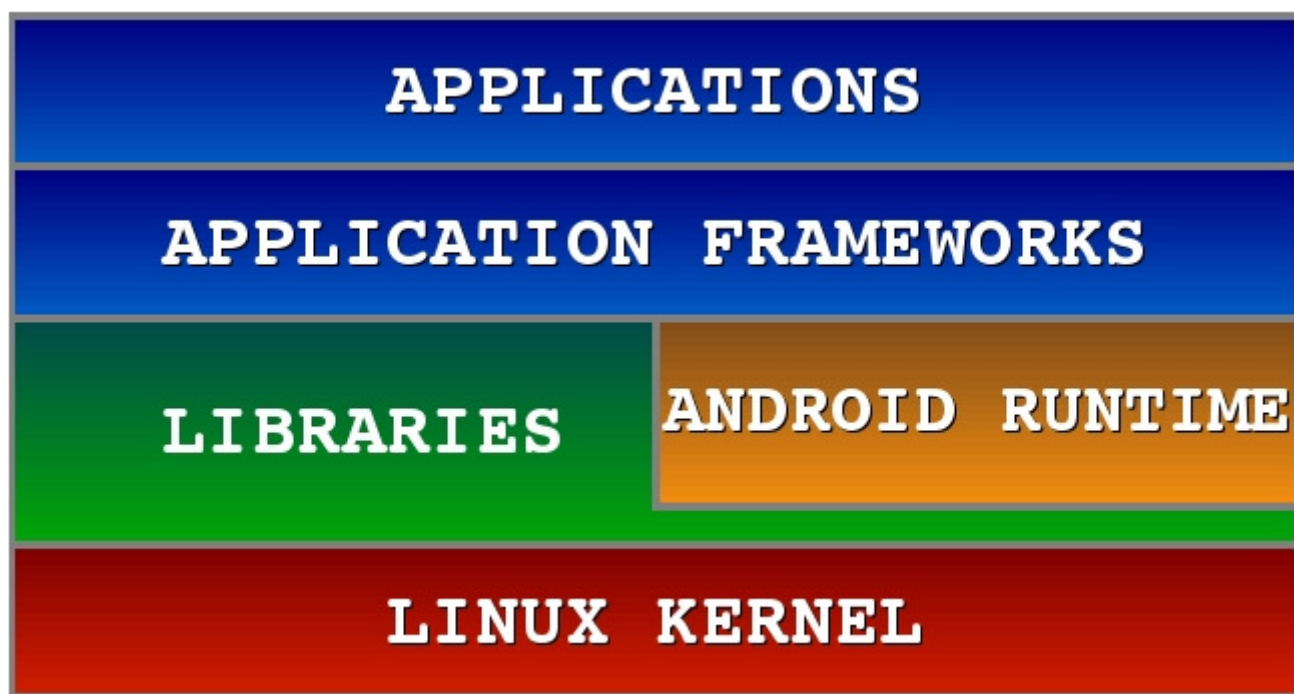
The most visited store in the world for Android apps. Cloud-connected and always synced, it's never been easier for users to find and download your apps.

[Learn More](#)

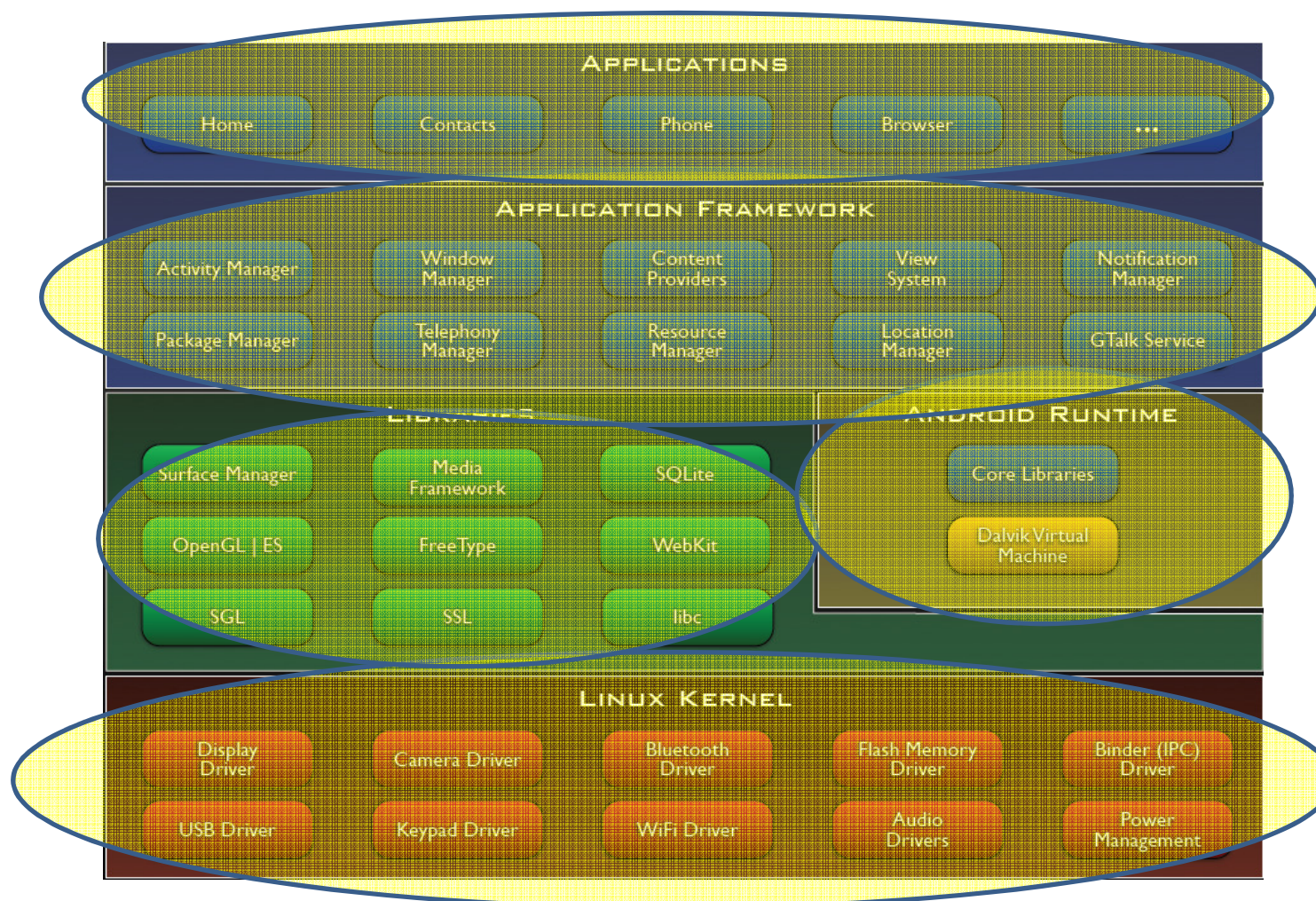
9



Android系统架构



Android系统架构





Android 四大组件

- Activity
- Service
- Content provider
- Broadcast receiver





Android 组件一

- **Activity**

停留在应用表面层。

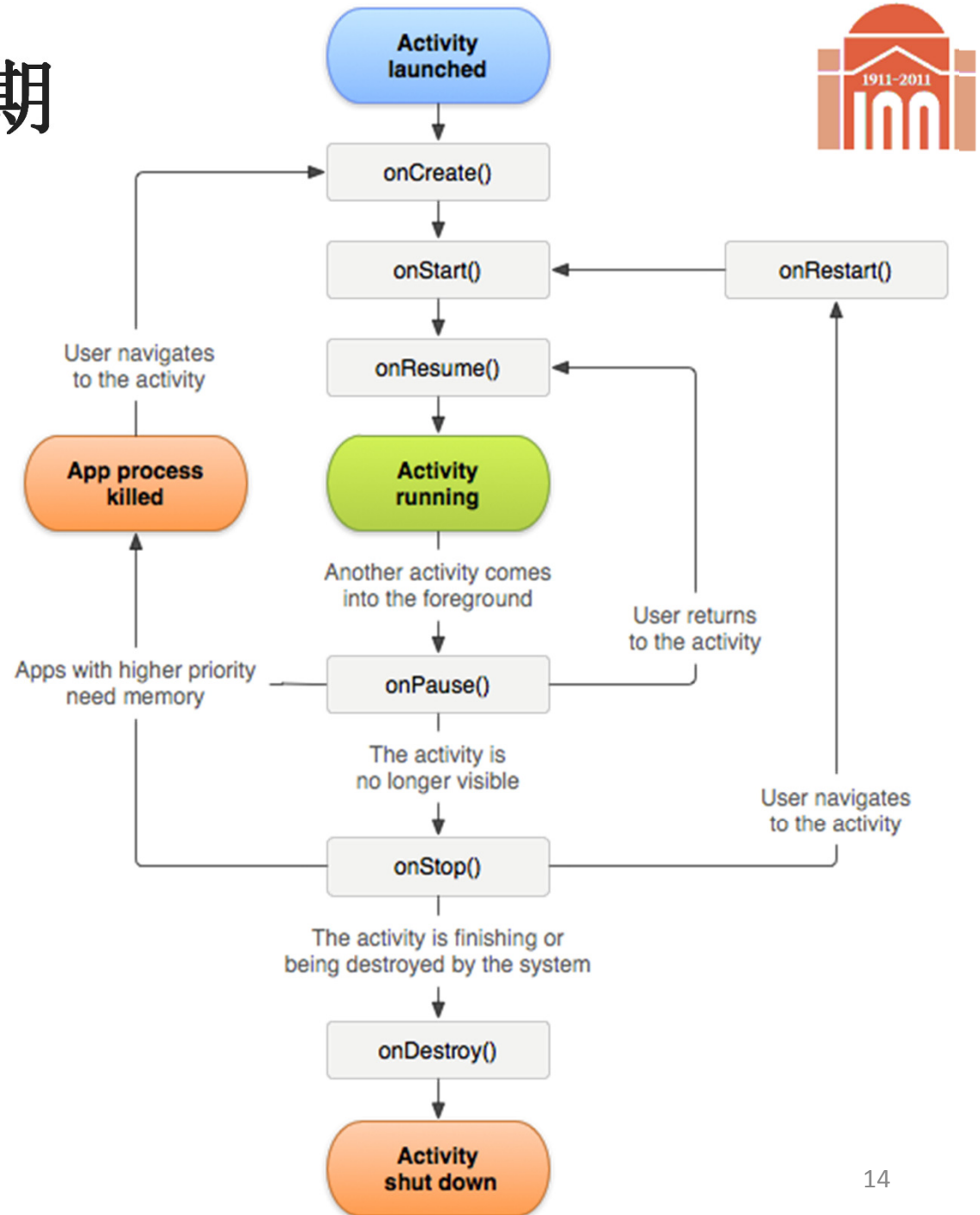
一个Activity通常就是一个单独的屏幕，它上面可以显示一些控件也可以监听并处理用户的事件做出响应。

Activity之间通过Intent进行通信

继承自Activity类

Activity生命周期

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onRestart()
- onDestroy()





Activity生命周期举例

@Override

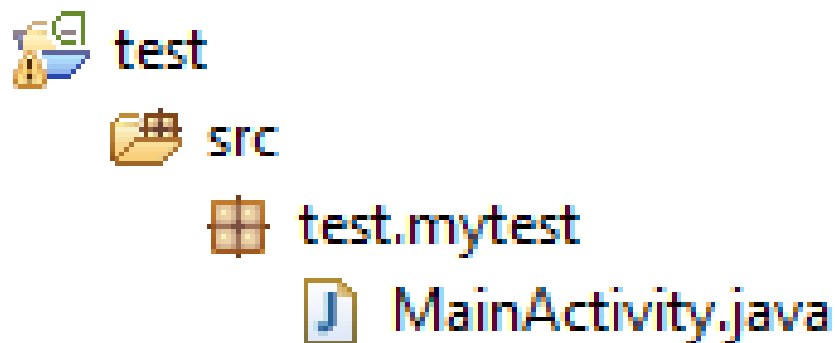
```
public void onPause() {  
    super.onPause(); // Always call the superclass method  
    // Release the Camera , we don't need it when paused  
    // and other activities might need to use it.  
    if (mCamera != null) {  
        mCamera.release()  
        mCamera = null;  
    }  
}
```

@Override

```
public void onResume() {  
    super.onResume(); //Always call the superclass method  
    /* Get the Camera instance as the activity achieves full user focus*/  
    if (mCamera == null) {  
        // Local method to handle camera init  
        initializeCamera();  
    }  
}
```



新建一个Activity



```
public class MainActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        // TODO Auto-generated method stub  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
}
```



用Java代码写事件处理

```
@Override
public void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Button sendbtn = (Button) findViewById(R.id.sendbtn);
    sendbtn.setOnClickListener(new OnClickListener(){
        public void onClick(View arg0) {
            // TODO Auto-generated method stub
            EditText editText = (EditText) findViewById(R.id.editText1);
            String message = editText.getText().toString();
            TextView txtResult = (TextView) findViewById(R.id.txtresult);
            txtResult.setText(message);
        }
    }); }
```



Activity事件处理

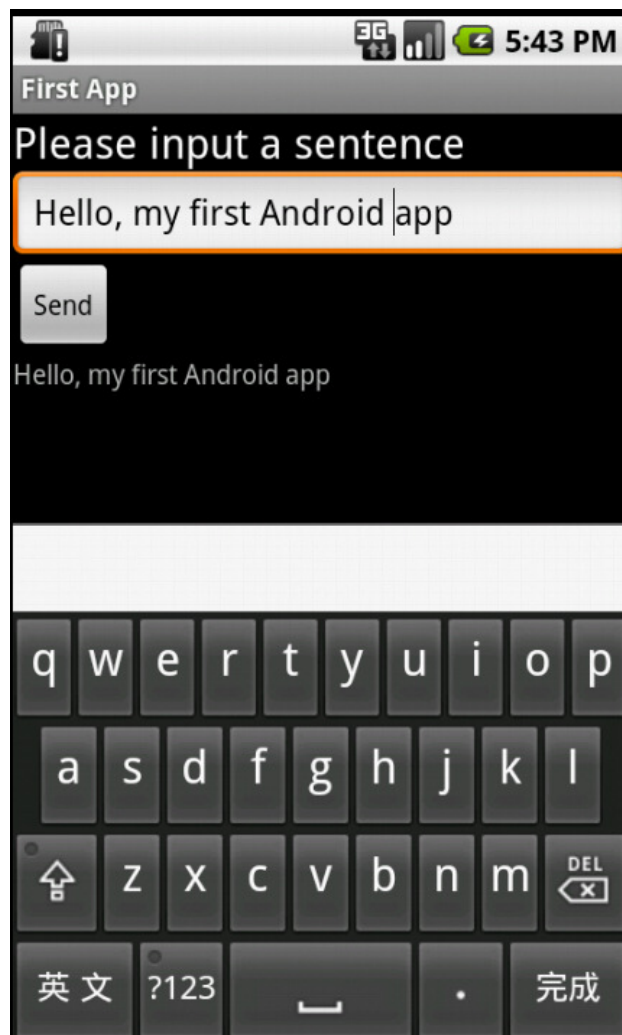
- 把内容显示到另一个TextView中

```
/** Called when the user selects the Send button */  
public void sendMessage(View view){  
    EditText editText =  
(EditText)findViewById(R.id.editText1);  
    String message = editText.getText().toString();  
    TextView txtResult = (TextView)  
findViewById(R.id.txtresult);  
    txtResult.setText(message);  
}
```



在模拟器上运行应用

- 运行程序





Android 组件二

- Service

没有可见的用户界面，但能够长时间运行于后台

服务不能自己运行，需要通过
`startService()` 或 `bindService()` 启动服务

继承自 `Service` 类



Service运行模式

- Service主要以两种模式运行。
 - 启动(Started)

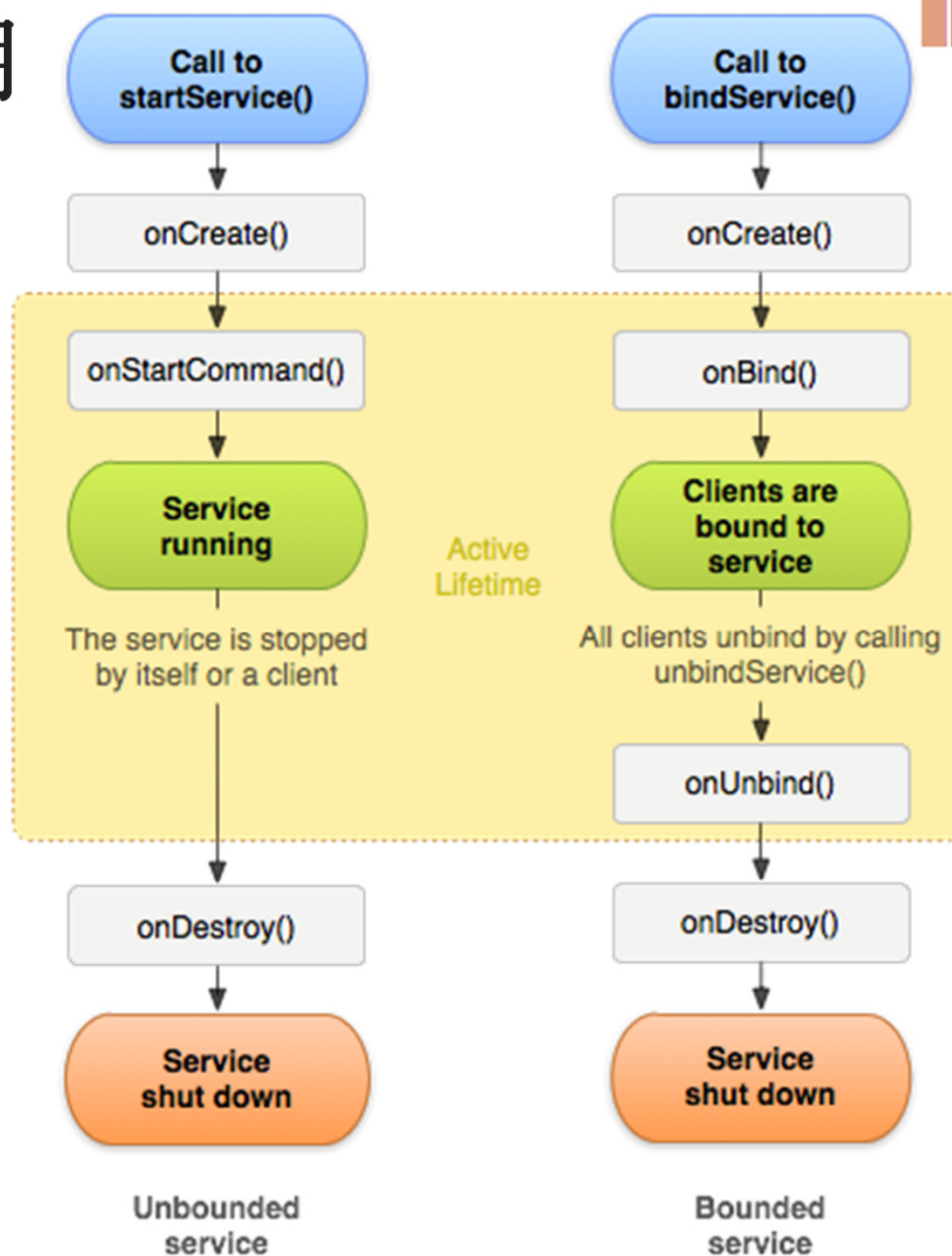
当调用startService()方法运行一个服务的时候，Service在启动模式下运行。
 - 绑定(Bound)

当调用bindService()方法去运行一个服务的时候，Service在绑定模式下运行。

Service生命周期

左: Started
右: Bound

Service是运行在主线程中的，这意味着如果要做大量阻塞任务需要在Service当中开启新的线程。





Service示例

```
public class HelloService extends Service {  
    private Looper mServiceLooper;  
    private ServiceHandler mServiceHandler;  
  
    // Handler that receives messages from the thread  
    private final class ServiceHandler extends Handler {  
        public ServiceHandler(Looper looper) {  
            super(looper);  
        }  
    }  
}
```

@Override



```
public void handleMessage(Message msg) {  
    // Normally we would do some work here, like download a file.  
    // For our sample, we just sleep for 5 seconds.  
    long endTime = System.currentTimeMillis() + 5*1000;  
    while (System.currentTimeMillis() < endTime) {  
        synchronized (this) {  
            try {  
                wait(endTime - System.currentTimeMillis());  
            } catch (Exception e) {  
            }  
        }  
    }  
    // Stop the service using the startId, so that we don't stop  
    // the service in the middle of handling another job  
    stopSelf(msg.arg1);  
} }
```



@Override

```
public void onCreate() {  
    // Start up the thread running the service. Note that we create a  
    // separate thread because the service normally runs in the process's  
    // main thread, which we don't want to block. We also make it  
    // background priority so CPU-intensive work will not disrupt our UI.  
    HandlerThread thread = new HandlerThread("ServiceStartArguments",  
        Process.THREAD_PRIORITY_BACKGROUND);  
    thread.start();  
  
    // Get the HandlerThread's Looper and use it for our Handler  
    mServiceLooper = thread.getLooper();  
    mServiceHandler = new ServiceHandler(mServiceLooper);  
}
```



@Override

```
public int onStartCommand(Intent intent, int flags, int startId) {  
    Toast.makeText(this, "service starting", oast.LENGTH_SHORT).show();  
  
    /*For each start request, send a message to start a job and deliver the  
       start ID so we know which request we're stopping when we finish the  
       job */  
    Message msg = mServiceHandler.obtainMessage();  
    msg.arg1 = startId;  
    mServiceHandler.sendMessage(msg);  
  
    // If we get killed, after returning from here, restart  
    return START_STICKY;  
}
```




@Override

```
public IBinder onBind(Intent intent) {  
    // We don't provide binding, so return null  
    return null;  
}
```

@Override

```
public void onDestroy() {  
    Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();  
}  
}
```



Service示例

- 在AndroidManifest文件中声明你的Service

```
<manifest ... >
```

```
...
```

```
<application ... >
```

```
    <service android:name=".ExampleService" />
```

```
...
```

```
</application>
```

```
</manifest>
```



Service示例

- 在其他某Activity中，调用startService方法运行我们前面写好的service.

```
Intent intent = new Intent(this,  
HelloService.class);  
startService(intent);
```



Android 组件三

- Content provider

ContentProvider提供了一种数据共享访问的方式。当应用继承ContentProvider类，并重写该类用于提供数据和存储数据的方法，就可以向其他应用共享其数据。

- 为存储和读取数据提供了统一的接口
- 应用程序之间可以实现数据共享
- android内置的许多数据都是使用ContentProvider形式，供开发者调用(如视频，音频，图片，通讯录等)



Content Provider 常用的接口

- query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
- Insert(Uri uri, ContentValues values)
- update(Uri uri, ContentValues values, String where, String[] selectionArgs)
- delete(Uri url, ContentValues valeus, String where, String[] selectionArgs)



Content Provider 常用的接口

其中URI 就是数据的访问地址， Uri主要包含了两部分信息：

1. 需要操作的ContentProvider
2. 对ContentProvider中的什么数据进行操。下面是一个URI的例子：

`content://contacts/people/`

该地址指定的是全部的联系人数据

其中“content://”是schema部分，由Android规定。“contacts”是主机名部分，用于唯一表示这个ContentProvider.后面“people”是路径部分，用来表示我们要操作的数据。



Content Provider 示例

- 获取系统数据是Content Provider 一个常用的应用场景。下面以获取联系人数据为例。
 - 创建一个content.xml的layout文件，里面放一个Button和TextView组件，id分别取为btnget和tv;
 - 创建MainContentProvider类，继承自Activity.
 - 在AndroidManifest.xml中添加如下permission:
<uses-permission
android:name="android.permission.READ_CONTACT
S"/>



// MainContentProvider类代码

```
public class MainContentProvider extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.content);  
        Button b1 = (Button) findViewById(R.id.btnget);  
        final TextView tv = (TextView) findViewById(R.id.tv);  
        OnClickListener ocl = new OnClickListener() {  
            public void onClick(View v) {  
                //创建ContentResolver类对象  
                ContentResolver contentResolver = getContentResolver();  
                // 获得所有的联系人
```



```
// 通过contentResolver.query获得所有的联系人
Cursor cursor =
contentResolver.query(ContactsContract.Contacts.CONTENT_URI
, null, null, null, null);
// 循环遍历
String rlt = "";
if (cursor.moveToFirst()) {
    int idColumn =
cursor.getColumnIndex(ContactsContract.Contacts._ID);
    int displayNameColumn =
cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME);
    do {
        // 获得联系人的ID号
        String contactId = cursor.getString(idColumn);
        // 获得联系人姓名
```



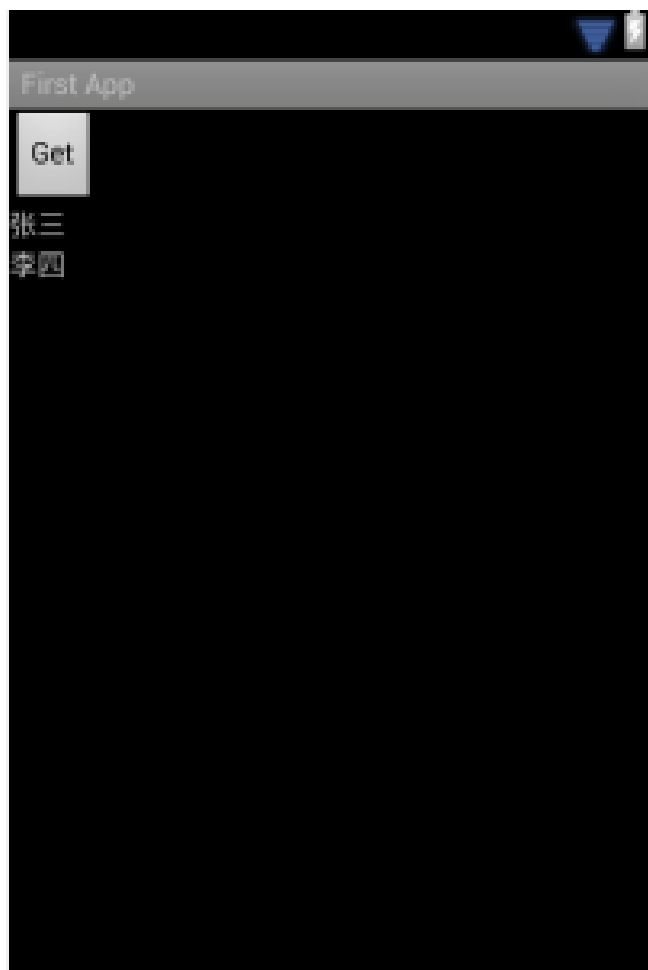
```
// 获得所有的联系人
Cursor cursor =
contentResolver.query(ContactsContract.Contacts.CONTENT_URI
, null, null, null, null);
// 循环遍历
String rlt = "";
if (cursor.moveToFirst()) {
    int idColumn =
cursor.getColumnIndex(ContactsContract.Contacts._ID);
    int displayNameColumn =
cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME);
    do {
        // 获得联系人的ID号
        String contactId = cursor.getString(idColumn);
        // 获得联系人姓名
```



```
// 获得联系人姓名
String displayName = cursor.getString(displayNameColumn);
rIt += displayName + "\r\n ";
} while(cursor.moveToNext());
}
//把获取的全部信息放到TextView中显示
tv.setText(rIt);
}
};
//注册按钮点击事件监听器
b1.setOnClickListener(ocl);
}
```



程序运行结果



- 点击**Get**按钮之后，应用会把全部联系人显示到下面**TextView**中。
- 扩展本应用也可以把用户电话，短信等等信息获取出来。



Android 组件四

- Broadcast Receiver

接收并处理广播通知。多数的广播是系统发起的，如地域变换、电量不足、来电来信等。程序也可以播放一个广播。

利用注册一个Broadcast Receiver来监听到这些Intent并获取Intent中的数据。



实现广播事件

在程序中构造好一个Intent, 然后调用
sendBroadcast进行广播即可, 代码如下:

```
public static final String NEW_BROADCAST =  
"test.mytest.NEW_BROADCAST";  
Intent intent = new Intent(New_BROADCAST);  
Intent.putExtra("data1",someData);  
Intent.putExtra("data2",someData);  
sendBroadcast(intent);
```




实现一个Broadcast Receiver

- 创建一个类继承自BroadcastReceiver类，并重写这个类当中的onReceive方法。
- 注册和注销BroadcastReceiver
 - 两种注册方式都可以
 - AndroidManifest.xml文件中进行注册
 - 代码中直接进行注册
 - 需要的权限一定要写到AndroidManifest.xml中
 - 注销
 - unregisterReceiver();



Broadcast Receiver示例

//实现一个Broadcast Receiver, 当收到短信时
//提示信息。

```
public class MyBroadcastReceiver extends BroadcastReceiver {  
    // action 名称  
    String SMS_RECEIVED = "android.provider.Telephony.SMS_RECEIVED" ;  
    public void onReceive(Context context, Intent intent) {  
        if (intent.getAction().equals( SMS_RECEIVED )) {  
            // 相关处理 : 地域变换、电量不足、来电来信;  
            Toast.makeText(new MainActivity(), "SMS received !",  
Toast.LENGTH_LONG);  
        }  
    }  
}
```



Broadcast Receiver 示例

在AndroidManifest中注册

```
<application>
```

```
<receiver android:name=".MyBroadcastReceiver" >
```

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

```
    <action android:name="android.provider.Telephony.SMS_RECEIVED" />
```

```
  </intent-filter>
```

```
</receiver>
```

```
</application>
```

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```



其他相关组件

- Intent
- SQLite
- 采集感知数据



Intent

- 思考问题
 1. 如何在Activity之间进行切换
 2. 如何在Activity之间传递数据



What is an Intent?

- Android官方API里对Intent的定义： An intent is an abstract description of an operation to be performed.
- 一个Intent就是一次对将要执行的操作的抽象描述。具体有3种形式：
 - 通过startActivity方法来启动一个新的Activity
 - 通过broadcast Intent机制可以将一个Intent发送给其他BroadcastReceiver.
 - 和后台的Service进行交互

Intent对一种操作的抽象描述



- Component name
- Action
- Data
- Category
- Type
- Flags
- Extras



创建Intent

```
Intent intent = new Intent(Context, Class);
```

The constructor used here takes two parameters:

- A Context as its first parameter (this is used because the Activity class is a subclass of Context)
- The Class of the app component to which the system should deliver the Intent (in this case, the activity that should be started)



创建Intent

```
Intent intent = new Intent(this,  
    DisplayMessageActivity.class);  
EditText editText = (EditText)  
    findViewById(R.id.edit_message);  
String message = editText.getText().toString();  
intent.putExtra("mykey", message);  
startActivity(intent);
```



从Intent接收数据

```
Intent intent = getIntent();
```

```
String message = intent.getStringExtra("mykey");
```



Intent 的两种形式

- 直接 Intent

指定了 `component` 属性的 Intent. 通过指定具体的组件类，通知应用启动对应的组件。

- 间接 Intent

没有指定 `component` 属性的 Intent。这些 Intent 需要包括足够的信息，以便系统根据这些信息，在所有的可用组件中，确定满足此 Intent 的组件。



Intent 的两种形式

- 启动一个指定的Activity

```
Intent intent = new  
Intent(CurrentActivity.this, OtherActivity.class);  
startActivity(intent);
```

- 启动一个未指定的Activity

```
Intent intent = new Intent(Intent.ACTION_DIAL,  
Uri.parse("tel:115-1345"));  
startActivity(intent);
```



Intent filter

对于间接Intent, Android通过解析将Intent 映射给可以处理此Intent的Activity, Receiver或服务. Intent 解析机制主要是通过查找已注册在AndroidManifest.xml中的所有IntentFilter 及其中定义的Intent, 最终找到匹配的Intent.

解析过程一般是通过Intent的action, type, category三个属性来进行判断的。



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="test.mytest"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="7"/>
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```



SQLite

- Android 在运行时集成了 SQLite，使得 Android 应用能够方便的利用它来完成数据库操作。
- 建议有一定数据库编程基础的同学学习这一部分。



SQLite

- SQLite 一个开源的嵌入式数据库，它支持 SQL 语言，并且只利用很少的内存就有很好的性能。
- Android 在运行时集成了 SQLite，所以每个 Android 应用程序都可以使用 SQLite 数据库。由于 JDBC 会消耗太多的系统资源，Android 提供了一些新的 API 来使用 SQLite 数据库。
- 数据库存储在 `data/< 项目文件夹 >/databases/` 下。



创建 SQLite 数据库

- Android 不自动提供数据库。在 Android 应用程序中使用 SQLite，必须自己创建数据库，然后创建表、索引，填充数据。Android 提供了 SQLiteOpenHelper 辅助类帮助创建一个数据库，创建数据库需要继承 SQLiteOpenHelper 类。



创建 SQLite 数据库

- SQLiteOpenHelper 类根据开发应用程序的需要，封装了创建和更新数据库使用的逻辑。SQLiteOpenHelper 的子类，至少需要实现三个方法：
- 构造函数，调用父类 SQLiteOpenHelper 的构造函数。这个方法需要四个参数：上下文环境（例如，一个 Activity），数据库名字，一个可选的游标工厂（通常是 Null），一个代表你正在使用的数据库模型版本的整数。
- onCreate () 方法，它需要一个 SQLiteDatabase 对象作为参数，根据需要对这个对象填充表和初始化数据。
- onUpgrade() 方法，它需要三个参数，一个 SQLiteDatabase 对象，一个旧的版本号和一个新的版本号，这样你就可以清楚如何把一个数据库从旧的模型转变到新的模型。



示例代码

```
public class DBHelper extends SQLiteOpenHelper {  
    public DBHelper(Context context, String name, CursorFactory  
        factory, int version) {  
        super(context, name, factory, version);  
    }  
    public DBHelper(Context context){  
        this(context, DBInfo.DB.DB_NAME, null, DBInfo.DB.  
VERSION);  
    }  
    .....  
}
```



示例代码

```
//获得SQLiteDatabase进行数据库操作
dbHelper = new DBHelper(context);
db = dbHelper.getWritableDatabase();
//创建表
db.execSQL("CREATE TABLE mytable (_id INTEGER PRIMARY KEY
AUTOINCREMENT, title TEXT, value REAL);");
//插入语句
db.execSQL("INSERT INTO widgets (name, inventory) VALUES
('Sprocket', 5)");
```



示例代码

//查询语句

```
Cursor result=db.rawQuery("SELECT ID, name, inventory FROM  
mytable");
```

```
result.moveToFirst();
```

```
while (!result.isAfterLast()) {
```

```
int id=result.getInt(0);
```

```
String name=result.getString(1);
```

```
int inventory=result.getInt(2);
```

```
// do something useful with these
```

```
result.moveToNext();
```

```
}
```

```
result.close();
```



采集感知数据

Android 系统对传感器数据采集和管理提供了一组丰富的接口选项，我们可以利用这些传感器监控我们的环境。

Android 通过Sensor和SensorManager类抽象了这些传感器，通过这些类可以使用Android设备的传感器。



Sensor类

- Sensor类表示一个传感器，可以使用SensorManager类的getSensorList方法获取所有可用的传感器。getSensorList方法返回的是一个List<Sensor>
- Sensor所提供的服务有
 - int TYPE_ACCELEROMETER 三轴加速度感应器 返回三个坐标轴的加速度 单位m/s²
 - int TYPE_ALL 用于列出所有感应器



采集感知数据

- int TYPE_GRAVITY 重力感应器
- int TYPE_LIGHT 光线感应器 单位 lux 勒克斯
- int TYPE_LINEAR_ACCELERATION 线性加速度

其他的这里不一一列出。

一般都通过SensorManager类操作Sensor类。



SensorManager类

- SensorManager 允许你访问设备的感应器。通过传入参数SENSOR_SERVICE参数调用Context.getSystemService方法可以获得一个sensor的实例。
- 确保当你不需要的时候要关闭感应器。忽略这一点肯能导致几个小时就耗尽电池，注意当屏幕关闭时，系统不会自动关闭感应器。

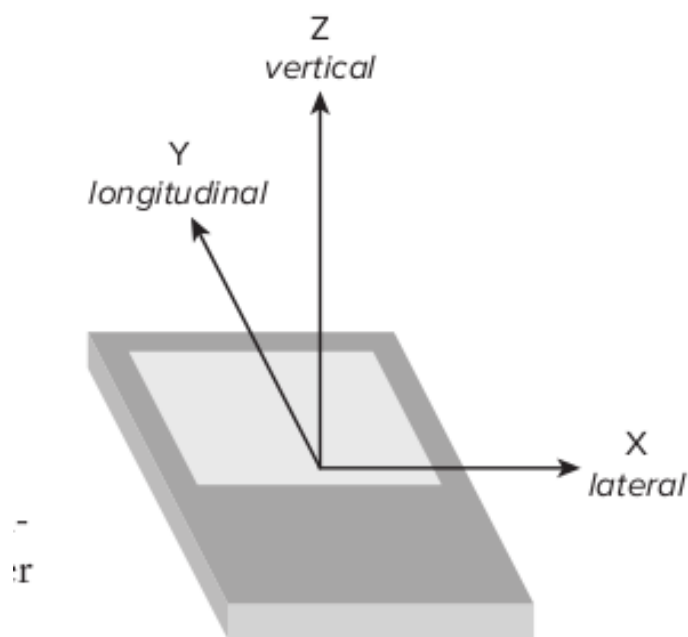
采集感知数据示例

- 采集加速度感应器数据
可以通过加速度感应器获得三个浮点型

x-axis

y-axis

z-axis





采集感知数据示例

- X表示左右移动的加速度
- Y表示前后移动的加速度
- Z表示垂直方向的加速度



采集感知数据

```
public class SensorDemoActivity extends Activity {  
    /** Called when the activity is first created. */  
    //设置LOG标签  
    private static final String TAG = "sensor";  
    private SensorManager sm;  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        //创建一个SensorManager来获取系统的传感器服务  
        sm =  
(SensorManager) getSystemService(Context.SENSOR_SERVICE);  
        //选取加速度感应器
```



//选取加速度感应器

```
int sensorType = Sensor.TYPE_ACCELEROMETER;
```

```
/*
```

```
 * 最常用的一个方法 注册事件
```

```
 * 参数1 : SensorEventListener监听器
```

```
 * 参数2 : Sensor 一个服务可能有多个Sensor实现，此处调用  
getDefaultSensor获取默认的Sensor
```

```
 * 参数3 : 模式 可选数据变化的刷新频率
```

```
 */
```

```
sm.registerListener(myAccelerometerListener,sm.getDefaultSensor(sensorType),SensorManager.SENSOR_DELAY_NORMAL);
```

```
}
```



```
/*
 * SensorEventListener接口的实现， 需要实现两个方法
 * 方法1 onSensorChanged 当数据变化的时候被触发调用
 * 方法2 onAccuracyChanged 当获得数据的精度发生变化的时候
 被调用， 比如突然无法获得数据时
 */
final SensorEventListener myAccelerometerListener = new
SensorEventListener(){

    //复写onSensorChanged方法
    public void onSensorChanged(SensorEvent sensorEvent){
        if(sensorEvent.sensor.getType() ==
Sensor.TYPE_ACCELEROMETER){
            Log.i(TAG,"onSensorChanged");
        }
    }
}
```



```
//图解中已经解释三个值的含义
//X Y Z分别对应values[0]到[2]
    float X_lateral = sensorEvent.values[0];
    float Y_longitudinal = sensorEvent.values[1];
    float Z_vertical = sensorEvent.values[2];
    Log.i(TAG, "\n heading "+X_lateral);
    Log.i(TAG, "\n pitch "+Y_longitudinal);
    Log.i(TAG, "\n roll "+Z_vertical);
}
}
//复写onAccuracyChanged方法
public void onAccuracyChanged(Sensor sensor , int accuracy){
    Log.i(TAG, "onAccuracyChanged");
}
};
}
```



创建HelloWorld



搭建Android开发环境

在开始之前需要搭建好Android开发环境。

1. 安装Android SDK
 2. 安装Eclipse的ADT插件
 3. 安装SDK Tools和platforms
- 具体安装步骤参考说明文档。

或者直接下载官方提供的
ADT bundle

<http://developer.android.com/sdk/index.html>

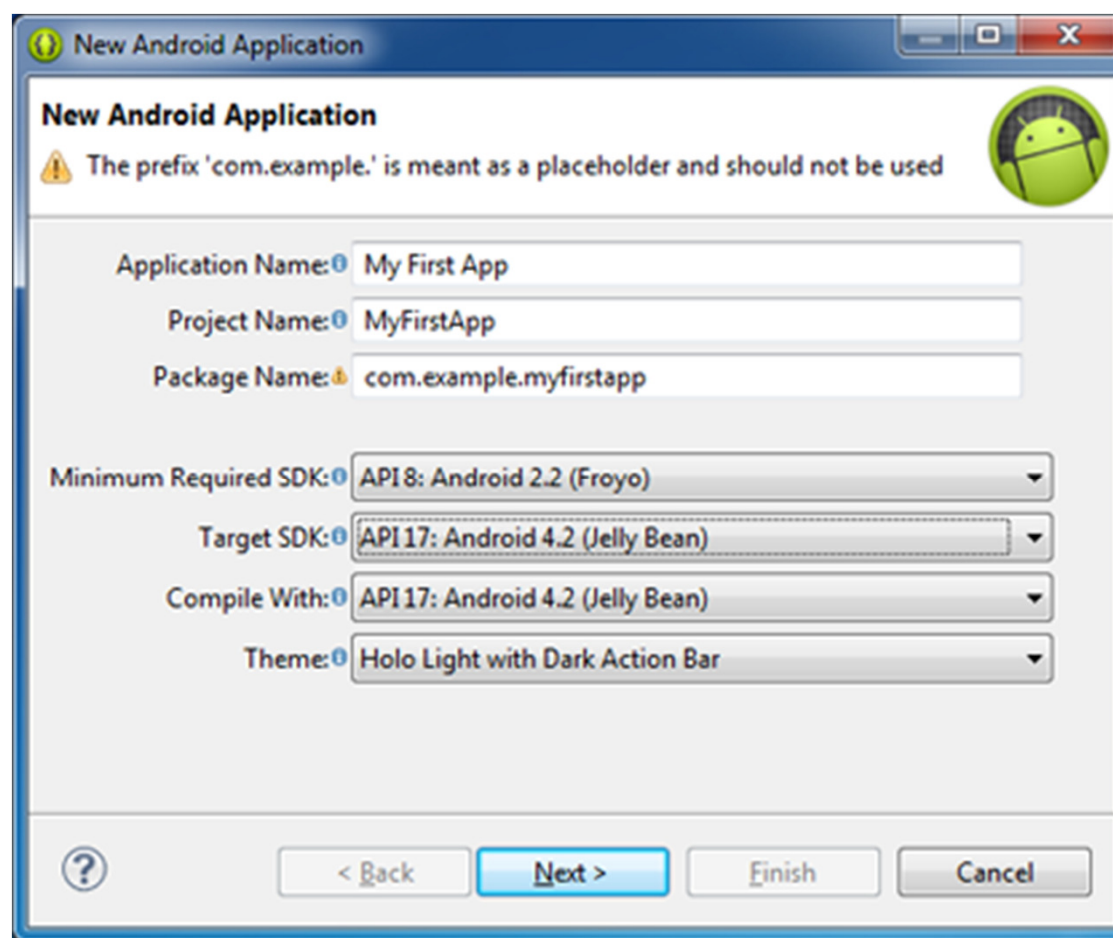




创建第一个Android应用

新建项目

点击 New, 选择Android Application Project

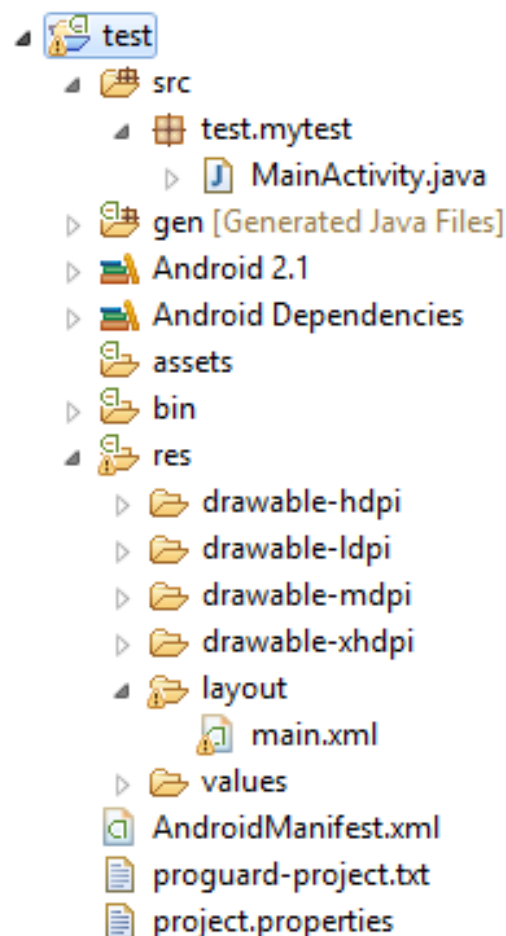




Android项目目录结构

一个Android 项目的目录结构

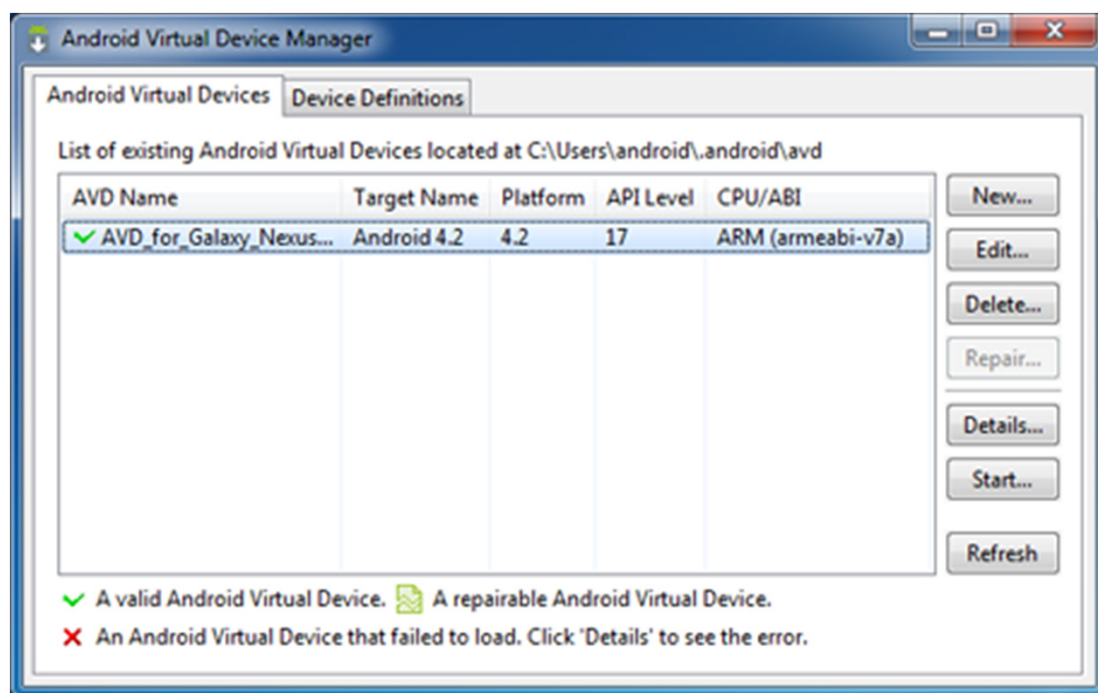
- AndroidManifest.xml
- src/
 - drawable/
 - layout/
 - values/
- res/





打开Android OS模拟器

- 创建Android虚拟设备（AVD），并打开模拟器





构建一个简单的界面

打开res/layout/目录下的activity_main.xml文件

添加一个文本字段

```
<EditText android:id="@+id/edit_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

添加一个按钮

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send" />
```

生成界面





添加按钮响应事件

activity_main.xml 向Button添加[android:onClick](#) 属性:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

在MainActivity.java 类中添加响应方法:

```
public void sendMessage(View view) {
    //add methods here
    ***
}
```



绑定一个Intent

在sendMessage() 方法中添加:

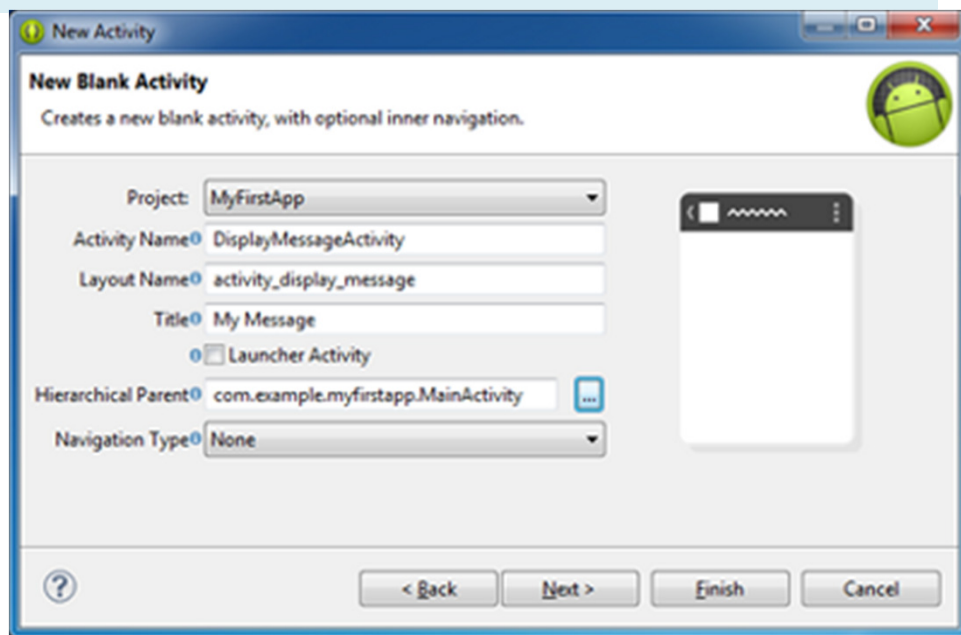
```
Intent intent = new Intent(this,
    DisplayMessageActivity.class);
EditText editText = (EditText)
    findViewById(R.id.edit_message);
String message = editText.getText().toString();
intent.putExtra(EXTRA_MESSAGE, message);
```

在MainActivity.java 类中添加EXTRA_MESSAGE常量:

```
public class MainActivity extends Activity {
    public final static String EXTRA_MESSAGE =
    "com.example.myfirstapp.MESSAGE";
    ...
}
```



新建 DisplayMessageActivity 类:



在AndroidManifest.xml中声明新建的Activity

```
<application ... >
...
<activity
    android:name="com.example.myfirstapp.DisplayMessageActivity"
    android:label="@string/title_activity_display_message"
    android:parentActivityName="com.example.myfirstapp.MainActivity" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```




接收Intent

在DisplayMessageActivity 类中 [onCreate\(\)](#) 方法中添加:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Get the message from the intent
    Intent intent = getIntent();
    String message = intent.getStringExtra(
        MainActivity.EXTRA_MESSAGE);

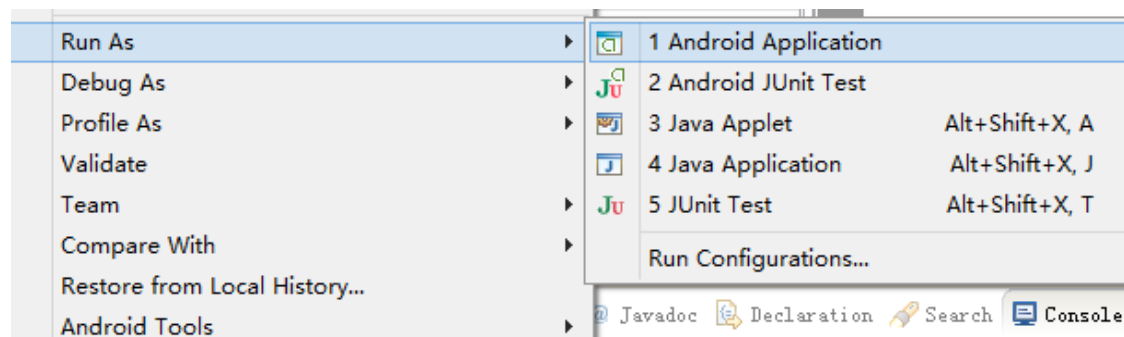
    // Create the text view
    TextView textView = new TextView(this);
    textView.setTextSize(40);
    textView.setText(message);

    // Set the text view as the activity layout
    setContentView(textView);
}
```



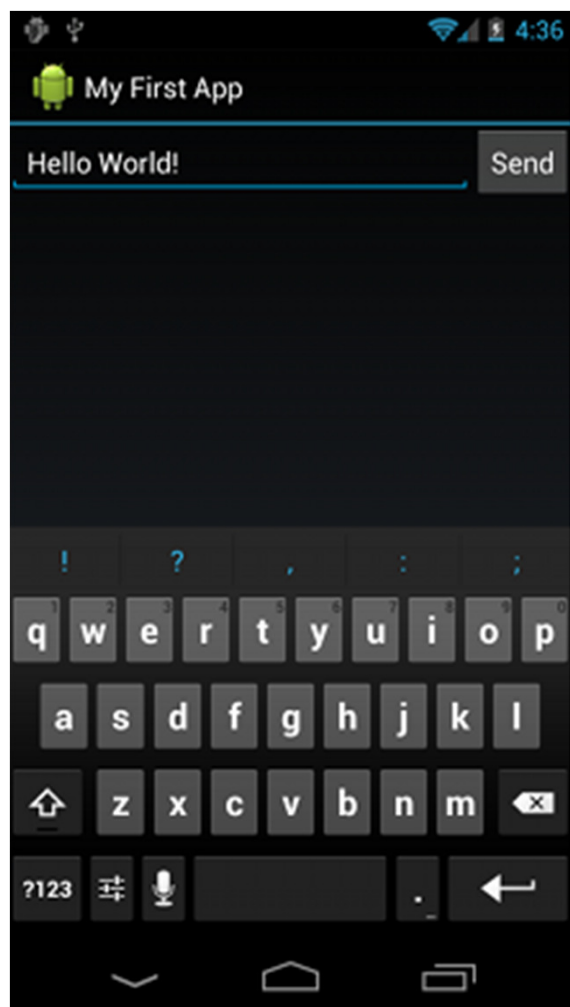
运行APP

Run as Android Application:



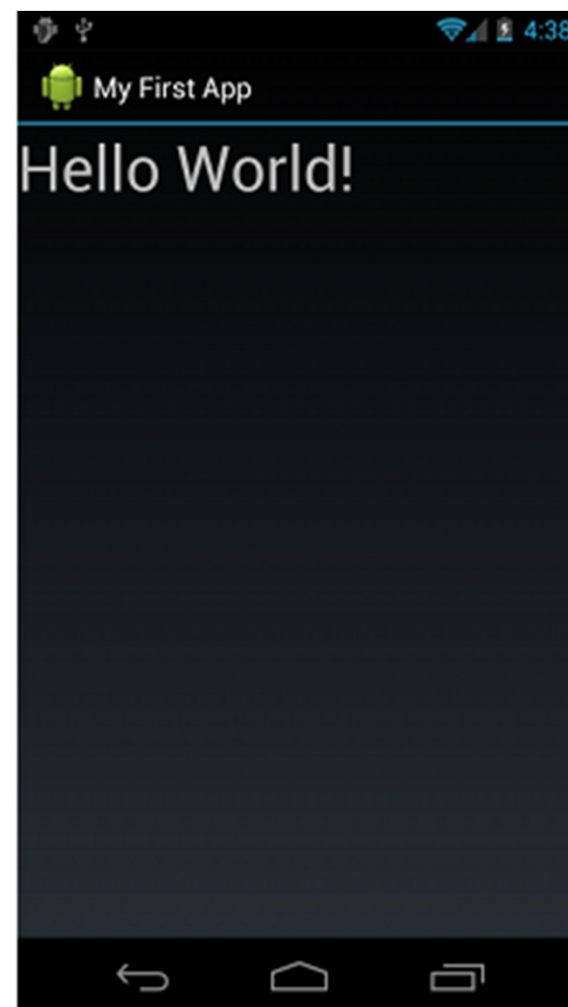


显示



MainActivity

输入文本
点击Send按钮



DisplayMessageActivity



小结

- 四大组件
**Activity, Service,
Content provider,
Broadcast receiver**
- 其他组件
Intent, SQL, Sensor
- 布局控件
- 事件响应





谢谢，再见