**Volley（网络请求）**

**implementation'com.android.volley:volley:1.1.1'**

RequestQueue queue = Volley.newRequestQueue(this);  
 String url = "";  
 JsonObjectRequest request = new JsonObjectRequest(GET, url, object, new Response.Listener<JSONObject>(),

new Response.ErrorListener());

queue.add(request);

**停止请求：**

**方法一：**

request.setTag(TAG);

queue.add(request);

//为请求添加一个标签TAG，在activityonStop里停止包含TAG

的请求

@Override

protected void onStop () {

super.onStop();

if (mRequestQueue != null) {

mRequestQueue.cancelAll(TAG);

}

}

**方法二：**

**停止所有请求：**

requestQueue.cancelAll(this)

**透明状态栏**

implementation 'com.jaeger.statusbarutil:library:1.5.1'

StatusBarUtil.setTransparent(this);

//如果是白色背景加上下面这一句将状态栏改成黑色字体

//getWindow().getDecorView().setSystemUiVisibility(SYSTEM\_UI\_FLAG\_LIGHT\_STATUS\_BAR);

**防止手机转屏页面重建数据丢失**

<activity  
android:name=".xxxActivity"

<!---在活动下面加上这句话就可以防止转屏丢失数据--->  
android:configChanges="orientation|keyboard|keyboardHidden|screenSize">

</activity>

使用DataBinding也可以保持数据

**安卓P允许访问http页面**

<application

android:usesCleartextTraffic="true">

</application>

**ViewModel、DataBinding**

dataBinding{  
 enabled true  
}

**Activity使用方法：**

MyViewModel viewModel;

ActivityViewModelBinding binding;

binding = DataBindingUtil.setContentView(this, R.layout.activity\_view\_model);

viewModel = ViewModelProviders.of(this).get(MyViewModel.class);

binding.setData(viewModel);

binding.setLifecycleOwner(this);

**XML绑定语法**

@{data.xxx}

@{()->data.xxx()}

**与视图Text控件绑定时一定注意LiveData类型，不是String要进行转型（这个错误编译不会报错，但是运行就会FC）：**

android:text="@{String.valueOf(data.number)}"

**阻止按钮自动大写**

**style.xml**

<item name="textAllCaps">false</item>

**Room**

依赖：  
      def room\_version = "2.2.2"  
      implementation "androidx.room:room-runtime:$room\_version"  
      annotationProcessor "androidx.room:room-compiler:$room\_version"

      testImplementation "androidx.room:room-testing:$room\_version"

//下面这三个一般用不到

//implementation "androidx.room:room-ktx:$room\_version"  
      //implementation "androidx.room:room-rxjava2:$room\_version"  
      //implementation "androidx.room:room-guava:$room\_version"

1.新建一个@Entity XXX类，定义表格格式

2.定义一个@Dao接口interface xxxDao，编写数据库操作语句

@Insert

@Update

@Delete

@Query(“”)

3.定义一个抽象类xxxDataBase

@DataBase(entities = {xxx.class},

version = x,

exportSchema = boolen)

里面定义一个抽象方法

abstract xxxDao getxxxDao;

调用：

xxxDataBase x =

Room.databaseBuilder(this,xxxDataBase.class,"name")

//允许在主线程运行，一般不允许这样写

//.allowMainThreadQueries()

.build();

xxxDao xx = x.getxxxDao;

**防止键盘遮挡输入框**

**将布局放到ScrollView中，然后在MainFest里Activity里面加入：**

android:windowSoftInputMode="adjustResize"

**RecyclerView**

**Adapter：**

//定义一个内部类ViewHolder

//Adapter继承RecyclerView.Adapter<xxxAdapter.ViewHolder>

public class xxxAdapter extends RecyclerView.Adapter<xxxAdapter.ViewHolder> {  
 public xxxAdapter(List<xxx> lists){  
 }  
 static class ViewHolder extends RecyclerView.ViewHolder {  
 public ViewHolder(@NonNull View itemView) {  
 super(itemView);

//初始化item上的控件  
 }  
 }

// IDE会引导生成三个需要实现的方法

public ViewHolder onCreateViewHolder(){

View view = LayoutInflater.from(parent.getContext())  
 .inflate(R.layout.item\_recycler,parent,false);  
 ViewHolder holder = new ViewHolder(view);  
 return holder;

}

public void onBindViewHolder(){

//绑定视图，为item上的控件赋值

}

public int getItemCount(){

//返回item个数

}  
}

**具体调用：**

List<xxx> lists; //封装数据

RecyclerView recylcler = ..;

recycler.setLayoutManager(new LinearLayoutManager(this));  
// 如果可以确定每个item的高度是固定的，设置下面这个选项可以提高性能  
recycler\_words.setHasFixedSize(true);  
//Adapter  
xxxAdapter adapter = new xxxAdapter(lists);  
recycler.setAdapter(adapter);

**广播接收器**

**发送广播时：**

高版本的android系统，对于广播的接收变得越来越严格，发送广播时要在Intent后面加上.setComponent参数

Intent intent =new Intent( "com.xxx.name" );//参数是广播接收器name

//参数1是广播接收器所在的包名，参数2是广播接收器name

intent.setComponent(new ComponentName("com.xxx","com.xxx.name"));

sendBroadcast(intent );

**Retrofit（网络请求）**

//创建一个接口，通过注解定义请求方式格式等

public interface RetrofitUtil {

//第二段请求地址

//加上第一段请求地址意思就是请求 https://a.b.b/de  
 @GET("de")  
 Call<UpdateInfo> getUpdateInfo();  
}

//使用

//构建一个Retrofit

Retrofit retrofit = new Retrofit  
 .Builder()

//这里是使用Gson将请求结果封装成对象  
 .addConverterFactory(GsonConverterFactory.create())

//第一段请求地址  
 .baseUrl("https://a.b.b/")  
 .build();  
RetrofitUtil retrofitUtil = retrofit.create(RetrofitUtil.class);  
Call<UpdateInfo> call = retrofitUtil.getUpdateInfo();

//异步请求  
call.enqueue(new Callback<UpdateInfo>() {});

**EventBus（活动间发送接收信息）**

**发送：**

EventBus.getDefault().postSticky(city\_id);  
startActivity(new Intent(this.getActivity(), B.class));

**接收：**

//接收前先注册

EventBus.getDefault().register(this);

//活动销毁时反注册

protected void onDestroy() {  
 super.onDestroy();  
 EventBus.getDefault().unregister(this);  
}

//通过注解使用

//(线程模式，是否Stiky粘性注册)

@Subscribe(threadMode = ThreadMode.MAIN,sticky = true)  
public void eventGetCityId(String city\_id) {  
 this.city\_id = city\_id;  
}

**改变Menu三个小点颜色**

**item下面可以继续包含Menu**

<item  
 android:icon="@drawable/ic\_more\_white\_24"  
 app:showAsAction="always"  
 android:title="About">  
 <menu>  
 <item  
 android:id="@+id/m\_item\_about"  
 android:title="关于"/>  
 </menu>  
</item>

**锁屏显示页面**

//onCreat()

getWindow().addFlags(WindowManager.LayoutParams.FLAG\_SHOW\_WHEN\_LOCKED //锁屏显示

| WindowManager.LayoutParams.FLAG\_DISMISS\_KEYGUARD //解锁

| WindowManager.LayoutParams.FLAG\_KEEP\_SCREEN\_ON //保持屏幕不息屏

| WindowManager.LayoutParams.FLAG\_TURN\_SCREEN\_ON);//点亮屏幕

//startActivity

val pm = getSystemService(Context.POWER\_SERVICE) as PowerManager

val mKeyguardManager = getSystemService(Context.KEYGUARD\_SERVICE) as KeyguardManager

val wakeLock = pm.newWakeLock(PowerManager.ACQUIRE\_CAUSES\_WAKEUP or PowerManager.SCREEN\_DIM\_WAKE\_LOCK, "wakeLock")

wakeLock.acquire()

val mKeyguardLock = mKeyguardManager.newKeyguardLock("")

mKeyguardLock.disableKeyguard()

val i = Intent(this,CameraActivity::class.java)

i.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK or Intent.FLAG\_ACTIVITY\_EXCLUDE\_FROM\_RECENTS)

startActivity(i)

**查看内存**

/\*\*

\* 打印当前手机内存信息应用的内存信息

\*/

@TargetApi(Build.VERSION\_CODES.JELLY\_BEAN)

public static void printMemoryInfo()

{

final String TAG="MemoryUtils.printMemoryInfo()";

//打印当前APP内存信息

//开启了 android:largeHeap="true" 后,就是启用了流氓应用的内存限制

//打印当前应用内存信息

Runtime rt = Runtime.getRuntime();

Log.d(TAG, "APP当前内存状态: 最大可申请内存:" + rt.maxMemory() / 1024/1024 +"MB 已申请内存:" + rt.totalMemory() / 1024/1024 + "MB 空闲内存:"+rt.freeMemory()/1024/1024+"MB");

}

/\*\*

\* 获得app可用内存的字节数 这个类不需要try,catch理论上不会报错

\*/

@TargetApi(Build.VERSION\_CODES.HONEYCOMB)

public static long getAppSurplusMe()

{

final String TAG="MemoryUtils.getAppSurplusMe()";

Runtime rt = Runtime.getRuntime();

//一下参数单位为字节数

long totalMemory=rt.totalMemory();//这个是已经申请的内存,等于已经使用的内存加上空闲内存

long maxMemory=rt.maxMemory();//最大内存限制

long freeMemory=rt.freeMemory();

//假如最大内存限制是64M,已经申请了34M,空闲4M,那么其实当前使用的是:(34-4)M,而实际当前有效可使用的内存是:64-(34-4)=34;

//64-(34-4)=34 请允许我引用高数老师的那句话:"同理可得" 64-34+4

//so

long surplusMemory=maxMemory-totalMemory+freeMemory;

Log.d(TAG, "系统当前内存状态: 最大可申请内存:" + rt.maxMemory() / 1024/1024 +"MB 已申请内存:" + rt.totalMemory() / 1024/1024 + "MB 空闲内存:"+rt.freeMemory()/1024/1024+"MB");

return surplusMemory;

}

/\*\*

\* 获得手机可用内存的字节数 这个类不需要try,catch,理论上不会报错

\*

\* 这个方法要慎用 容易导致崩溃 特别在引导页的时候 低内存手机容易发生崩溃

\*

\* @param context

\* @return

\*/

public static long getPhoneSurplusMe(Context context)

{

ActivityManager am = (ActivityManager)context.getSystemService(context.ACTIVITY\_SERVICE);

ActivityManager.MemoryInfo mi = new ActivityManager.MemoryInfo();

am.getMemoryInfo(mi);

return mi.availMem;

}

**传感器**

**#define SENSOR\_TYPE\_ACCELEROMETER 1 //加速度**

**#define SENSOR\_TYPE\_MAGNETIC\_FIELD 2 //磁力**

**#define SENSOR\_TYPE\_ORIENTATION 3 //方向**

**#define SENSOR\_TYPE\_GYROSCOPE 4 //陀螺仪**

**#define SENSOR\_TYPE\_LIGHT 5 //光线感应**

**#define SENSOR\_TYPE\_PRESSURE 6 //压力**

**#define SENSOR\_TYPE\_TEMPERATURE 7 //温度**

**#define SENSOR\_TYPE\_PROXIMITY 8 //接近**

**#define SENSOR\_TYPE\_GRAVITY 9 //重力**

**#define SENSOR\_TYPE\_LINEAR\_ACCELERATION 10//线性加速度**

**#define SENSOR\_TYPE\_ROTATION\_VECTOR 11//旋转矢量**

**1 加速度传感器**

加速度传感器又叫G-sensor，返回x、y、z三轴的加速度数值。

该数值包含地心引力的影响，单位是m/s^2。

将手机平放在桌面上，x轴默认为0，y轴默认0，z轴默认9.81。

将手机朝下放在桌面上，z轴为-9.81。

将手机向左倾斜，x轴为正值。

将手机向右倾斜，x轴为负值。

将手机向上倾斜，y轴为负值。

将手机向下倾斜，y轴为正值。

加速度传感器可能是最为成熟的一种mems产品，市场上的加速度传感器种类很多。

手机中常用的加速度传感器有BOSCH（博世）的BMA系列，AMK的897X系列，ST的LIS3X系列等。

这些传感器一般提供±2G至±16G的加速度测量范围，采用I2C或SPI接口和MCU相连，数据精度小于16bit。

**2 磁力传感器**

磁力传感器简称为M-sensor，返回x、y、z三轴的环境磁场数据。

该数值的单位是微特斯拉（micro-Tesla），用uT表示。

单位也可以是高斯（Gauss），1Tesla=10000Gauss。

硬件上一般没有独立的磁力传感器，磁力数据由电子罗盘传感器提供（E-compass）。

电子罗盘传感器同时提供下文的方向传感器数据。

**3 方向传感器**

方向传感器简称为O-sensor，返回三轴的角度数据，方向数据的单位是角度。

为了得到精确的角度数据，E-compass需要获取G-sensor的数据，

经过计算生产O-sensor数据，否则只能获取水平方向的角度。

方向传感器提供三个数据，分别为azimuth、pitch和roll。

azimuth：方位，返回水平时磁北极和Y轴的夹角，范围为0°至360°。

0°=北，90°=东，180°=南，270°=西。

pitch：x轴和水平面的夹角，范围为-180°至180°。

当z轴向y轴转动时，角度为正值。

roll：y轴和水平面的夹角，由于历史原因，范围为-90°至90°。

当x轴向z轴移动时，角度为正值。

电子罗盘在获取正确的数据前需要进行校准，通常可用8字校准法。

8字校准法要求用户使用需要校准的设备在空中做8字晃动，

原则上尽量多的让设备法线方向指向空间的所有8个象限。

手机中使用的电子罗盘芯片有AKM公司的897X系列，ST公司的LSM系列以及雅马哈公司等等。

由于需要读取G-sensor数据并计算出M-sensor和O-sensor数据，

因此厂商一般会提供一个后台daemon来完成工作，电子罗盘算法一般是公司私有产权。

**4 陀螺仪传感器**

陀螺仪传感器叫做Gyro-sensor，返回x、y、z三轴的角加速度数据。

角加速度的单位是radians/second。

根据Nexus S手机实测：

水平逆时针旋转，Z轴为正。

水平逆时针旋转，z轴为负。

向左旋转，y轴为负。

向右旋转，y轴为正。

向上旋转，x轴为负。

向下旋转，x轴为正。

ST的L3G系列的陀螺仪传感器比较流行，iphone4和google的nexus s中使用该种传感器。

**5 光线感应传感器**

光线感应传感器检测实时的光线强度，光强单位是lux，其物理意义是照射到单位面积上的光通量。

光线感应传感器主要用于Android系统的LCD自动亮度功能。

可以根据采样到的光强数值实时调整LCD的亮度。

**6 压力传感器**

压力传感器返回当前的压强，单位是百帕斯卡hectopascal（hPa）。

**7 温度传感器**

温度传感器返回当前的温度。

**8 接近传感器**

接近传感器检测物体与手机的距离，单位是厘米。

一些接近传感器只能返回远和近两个状态，

因此，接近传感器将最大距离返回远状态，小于最大距离返回近状态。

接近传感器可用于接听电话时自动关闭LCD屏幕以节省电量。

一些芯片集成了接近传感器和光线传感器两者功能。

下面三个传感器是Android2新提出的传感器类型，目前还不太清楚有哪些应用程序使用。

**9 重力传感器**

重力传感器简称GV-sensor，输出重力数据。

在地球上，重力数值为9.8，单位是m/s^2。

坐标系统与加速度传感器相同。

当设备复位时，重力传感器的输出与加速度传感器相同。

**10 线性加速度传感器**

线性加速度传感器简称LA-sensor。

线性加速度传感器是加速度传感器减去重力影响获取的数据。

单位是m/s^2，坐标系统与加速度传感器相同。

加速度传感器、重力传感器和线性加速度传感器的计算公式如下：

加速度 = 重力 + 线性加速度

**11 旋转矢量传感器**

旋转矢量传感器简称RV-sensor。

旋转矢量代表设备的方向，是一个将坐标轴和角度混合计算得到的数据。

RV-sensor输出三个数据：

x\*sin(theta/2)

y\*sin(theta/2)

z\*sin(theta/2)

sin(theta/2)是RV的数量级。

RV的方向与轴旋转的方向相同。

RV的三个数值，与cos(theta/2)组成一个四元组。

RV的数据没有单位，使用的坐标系与加速度相同。

举例：

sensors\_event\_t.data[0] = x\*sin(theta/2)

sensors\_event\_t.data[1] = y\*sin(theta/2)

sensors\_event\_t.data[2] = z\*sin(theta/2)

sensors\_event\_t.data[3] = cos(theta/2)

GV、LA和RV的数值没有物理传感器可以直接给出，

需要G-sensor、O-sensor和Gyro-sensor经过算法计算后得出。

算法一般是传感器公司的私有产权。

参考文献：

android source code hardware\libhardware\include\hardwaresensor.h

<http://www.dzsc.com/data/html/2010-11-29/87454.html>

**状态栏**

// 隐藏状态栏

getWindow().addFlags(WindowManager.LayoutParams.FLAG\_FULLSCREEN);

// 显示状态栏

getWindow().clearFlags(WindowManager.LayoutParams.FLAG\_FULLSCREEN);

**控制状态栏和导航栏显示，setSystemUiVisibility**

View.SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE：全屏显示时保证尺寸不变。

View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN：Activity全屏显示，状态栏显示在Activity页面上面。

View.SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION：效果同View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN

View.SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION：隐藏导航栏

View.SYSTEM\_UI\_FLAG\_FULLSCREEN：Activity全屏显示，且状态栏被隐藏覆盖掉。

View.SYSTEM\_UI\_FLAG\_VISIBLE：Activity非全屏显示，显示状态栏和导航栏。

View.INVISIBLE：Activity伸展全屏显示，隐藏状态栏。

View.SYSTEM\_UI\_LAYOUT\_FLAGS：效果同View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN

View.SYSTEM\_UI\_FLAG\_IMMERSIVE\_STICKY：必须配合View.SYSTEM\_UI\_FLAG\_FULLSCREEN和View.SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION组合使用，达到的效果是拉出状态栏和导航栏后显示一会儿消失。

// 全屏展示

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.JELLY\_BEAN) {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.LOLLIPOP) {

// 全屏显示，隐藏状态栏和导航栏，拉出状态栏和导航栏显示一会儿后消失。

activity.getWindow().getDecorView().setSystemUiVisibility(

View.SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE

| View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN

| View.SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION

| View.SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION

| View.SYSTEM\_UI\_FLAG\_FULLSCREEN

| View.SYSTEM\_UI\_FLAG\_IMMERSIVE\_STICKY);

} else {

// 全屏显示，隐藏状态栏

activity.getWindow().getDecorView().setSystemUiVisibility(View.SYSTEM\_UI\_FLAG\_FULLSCREEN);

}

}

// 非全屏显示，显示状态栏和导航栏

activity.getWindow().getDecorView().setSystemUiVisibility(View.SYSTEM\_UI\_FLAG\_VISIBLE);