**【Android】BaseActivity基类设计**

[](https://www.jianshu.com/u/583b326f3ef8)

[吾非言](https://www.jianshu.com/u/583b326f3ef8) 关注

2018.03.06 09:11\* 字数 1486 阅读 3520评论 17喜欢 226

作者：邹峰立，微博：zrunker，邮箱：[zrunker@yahoo.com](https://link.jianshu.com/?t=mailto%3Azrunker%40yahoo.com)，微信公众号：书客创作，个人平台：[www.ibooker.cc](https://link.jianshu.com/?t=http%3A%2F%2Fwww.ibooker.cc)。

本文选自[书客创作](https://link.jianshu.com/?t=http%3A%2F%2Fwww.ibooker.cc)平台第141篇文章。[阅读原文](https://link.jianshu.com/?t=http%3A%2F%2Fwww.ibooker.cc%2Farticle%2F141%2Fdetail) 。

书客创作

通常在APP开发当中都会自定义一个抽象的BaseActivity，用作Acitvity的基类，主要是用来实现Activity的一些公共属性以及公共方法。一个优秀的BaseActivity，在一定程度上能够优化代码结构，降低耦合度，提高代码可读性，方便修改。

一般情况下BaseActivity，只会将Activity的一些公共部分进行集成，这也是设计BaseActivity基类其中一个准则。当然BaseActivity的设计还取决于当前应用要求。那么如何去设计BaseActivity基类呢？首先要明白BaseActivity基类能够做些什么。

**一、隐藏标题栏**

在Android开发中，隐藏标题栏的方式有很多，例如可以在style.xml中，设置当前应用的主题，如下：

<resources>

<!-- Base application theme. -->

<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

......

<!--无主题-->

<item name="windowNoTitle">true</item>

</style>

</resources>

通过BaseActivity同样可以进行设置，通常设置无标题主题是在Activity的onCreate方法中进行设置。不过这里设置是要分情况的。

1、如果定义的BaseActivity是继承android.support.v7.app.AppCompatActivity，需要通过以下方法进行设置：

// 隐藏标题栏

if (getSupportActionBar() != null)

getSupportActionBar().hide();

2、如果定义的BaseActivity是继承android.app.Activity或者android.support.v4.app.FragmentActivity，需要通过以下方法进行设置：

// 隐藏标题栏

requestWindowFeature(Window.FEATURE\_NO\_TITLE);

**二、状态栏沉浸效果**

状态栏沉浸效果设置方式也有很多种，例如可以在也是在style.xml中，设置当前应用的主题，如下：

<resources>

<!-- Base application theme. -->

<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

......

<!--沉浸效果-->

<item name="android:fitsSystemWindows">true</item>

</style>

</resources>

通过BaseActivity同样可以进行设置，通常设置无标题主题是在Activity的onCreate方法中进行设置。如下：

// 沉浸效果

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

// 透明状态栏

getWindow().addFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_STATUS);

// 透明导航栏

getWindow().addFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_NAVIGATION);

}

**三、定义Activity管理类**

为什么要设置Activity管理类呢？打个简单的例子，当程序退出的时候，是不是要把所有打开的Activity进行关闭，也就是清空堆栈。如果不设置Activity管理类根本就知不知道到底打开了多少个Activity。例如定义一个名为ActivityUtil的管理类，代码如下：

/\*\*

\* Activity工具类

\*

\* @author 邹峰立

\*/

public class ActivityUtil {

private List<Activity> activityList = new ArrayList<>();

private static ActivityUtil instance;

// 单例模式中获取唯一的ExitApplication实例

public static synchronized ActivityUtil getInstance() {

if (null == instance) {

instance = new ActivityUtil();

}

return instance;

}

// 添加Activity到容器中

public void addActivity(Activity activity) {

if (activityList == null)

activityList = new ArrayList<>();

activityList.add(activity);

}

// 移除Activity

public void removeActivity(Activity activity) {

if (activityList != null)

activityList.remove(activity);

}

// 遍历所有Activity并finish

public void exitSystem() {

for (Activity activity : activityList) {

if (activity != null)

activity.finish();

}

// 退出进程

android.os.Process.killProcess(Process.myPid());

System.exit(0);

}

}

这里只是简单的定义一个Activity管理类，在实际开发当中，Activity管理类不会只有这么简单。定义好Activity管理类之后，只需要在BaseActivity每次执行onCreate方法的时候进行添加addActivity，在BaseActivity每次执行onDestroy方法的时候进行移除removeActivity。最后在程序退出的时候执行ActivityUtil类中的exitSystem方法，例如，双击返回键退出程序功能，就可以在应用程序主页使用下面代码实现：

// 设置返回按钮的监听事件

private long exitTime = 0;

public boolean onKeyDown(int keyCode, KeyEvent event) {

// 监听返回键，点击两次退出程序

if (keyCode == KeyEvent.KEYCODE\_BACK && event.getAction() == KeyEvent.ACTION\_DOWN) {

if ((System.currentTimeMillis() - exitTime) > 5000) {

Toast.makeText(getApplicationContext(), "再按一次退出程序", Toast.LENGTH\_LONG).show();

exitTime = System.currentTimeMillis();

} else {

ActivityUtil.getInstance().exitSystem();

}

return true;

}

return super.onKeyDown(keyCode, event);

}

**四、对BACK、HOME等键统一处理**

在BaseActivity中可以对BACK、HOME等键统一处理，例如点击返回键的实现，关闭当前页面：

@Override

public boolean onKeyDown(int keyCode, KeyEvent event) {

// 点击手机上的返回键，返回上一层

if (keyCode == KeyEvent.KEYCODE\_BACK) {

this.finish();

ActivityUtil.getInstance().removeActivity(this);

}

return super.onKeyDown(keyCode, event);

}

**五、对网络状态变化进行实时监听**

在Android系统中，当网络状态改变的时候，系统会发送一个名为CONNECTIVITY\_ACTION的广播，如果要监听网络状态变化，就需要对该广播进行监听。如何监听？可以通过注册广播的方式。首先定义一个名为NetBroadcastReceiver的广播接收器，用来接收系统广播。代码如下：

/\*\*

\* 检查网络状态切换 - 广播接受器

\*

\* @author 邹峰立

\*/

public class NetBroadcastReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

// 如果相等的话就说明网络状态发生了变化

if (ConnectivityManager.CONNECTIVITY\_ACTION.equals(intent.getAction())) {

boolean netWorkState = NetworkUtil.isNetworkConnected(context);

// 接口回调传过去状态的类型

if (BaseActivity.netEvent != null)

BaseActivity.netEvent.onNetChange(netWorkState);

}

}

// 网络状态变化接口

public interface NetChangeListener {

void onNetChange(boolean netWorkState);

}

}

在该类中定义一个对外接口NetChangeListener，该接口用来告诉外界当前网络状态，外界只需要实现该接口，就能够知道当前手机的网络状态。而BaseActivity.netEvent就是基类中实例化网络监听接口NetChangeListener的静态对象，通过该对象就可以将当前的网络状态传递给外界。

当然定义好广播接收器之后，还要在添加到清单文件中。

<!--注册广播-->

<receiver android:name=".broadcastreceiver.NetBroadcastReceiver">

<intent-filter>

<action

android:name="android.net.conn.CONNECTIVITY\_CHANGE"

tools:ignore="BatteryLife" />

</intent-filter>

</receiver>

同时网络状态监听，还需要权限，不要忘记加权限：

<!--网络状态权限-->

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

另外，在该类中NetworkUtil是自定义的网络管理类，主要用来检测当前网络状态，代码如下：

/\*\*

\* 网络工具类

\*

\* create by 邹峰立 on 2016/9/18

\*/

public class NetworkUtil {

public static final int NETTYPE\_WIFI = 0x01;

public static final int NETTYPE\_CMWAP = 0x02;

public static final int NETTYPE\_CMNET = 0x03;

/\*\*

\* 检测网络是否可用

\*/

public static boolean isNetworkConnected(Context context) {

ConnectivityManager cm = (ConnectivityManager) context.getApplicationContext().getSystemService(Context.CONNECTIVITY\_SERVICE);

NetworkInfo ni = null;

if (cm != null) {

ni = cm.getActiveNetworkInfo();

}

return ni != null && ni.isConnectedOrConnecting();

}

/\*\*

\* 获取当前网络类型

\*

\* @return 0：没有网络 1：WIFI网络 2：WAP网络 3：NET网络

\*/

public static int getNetworkType(Context context) {

int netType = 0;

ConnectivityManager connectivityManager = (ConnectivityManager) context.getApplicationContext().getSystemService(Context.CONNECTIVITY\_SERVICE);

NetworkInfo networkInfo = null;

if (connectivityManager != null) {

networkInfo = connectivityManager.getActiveNetworkInfo();

}

if (networkInfo == null) {

return netType;

}

int nType = networkInfo.getType();

if (nType == ConnectivityManager.TYPE\_MOBILE) {

String extraInfo = networkInfo.getExtraInfo();

if (!TextUtils.isEmpty(extraInfo)) {

if (extraInfo.toLowerCase().equals("cmnet")) {

netType = NETTYPE\_CMNET;

} else {

netType = NETTYPE\_CMWAP;

}

}

} else if (nType == ConnectivityManager.TYPE\_WIFI) {

netType = NETTYPE\_WIFI;

}

return netType;

}

}

**六、权限申请封装**

在Android6.0之后的版本，对一些特殊权限的时候就需要动态申请，可以利用BaseActivity基类实现权限申请封装。如何进行封装，首先要明白权限申请的几个步骤：

1. 判断是否有该权限。
2. 申请权限。
3. 对权限申请结果进行处理。

所以对权限申请封装，也就是对这三个步骤进行封装，如下：

/\*\*

\* 权限检查方法，false代表没有该权限，ture代表有该权限

\*/

public boolean hasPermission(String... permissions) {

for (String permission : permissions) {

if (ContextCompat.checkSelfPermission(this, permission) != PackageManager.PERMISSION\_GRANTED) {

return false;

}

}

return true;

}

/\*\*

\* 权限请求方法

\*/

public void requestPermission(int code, String... permissions) {

ActivityCompat.requestPermissions(this, permissions, code);

}

/\*\*

\* 处理请求权限结果事件

\*

\* @param requestCode 请求码

\* @param permissions 权限组

\* @param grantResults 结果集

\*/

@Override

public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {

super.onRequestPermissionsResult(requestCode, permissions, grantResults);

doRequestPermissionsResult(requestCode, grantResults);

}

/\*\*

\* 处理请求权限结果事件

\*

\* @param requestCode 请求码

\* @param grantResults 结果集

\*/

public void doRequestPermissionsResult(int requestCode, @NonNull int[] grantResults) {

}

**七、更改应用程序字体大小**

很多时候，应用程序需要实现修改字体大小功能，或者防止系统字体大小影响应用字体大小。这个实现可以通过在基类的onResume方法中进行操作。

@Override

protected void onResume() {

super.onResume();

Resources resources = this.getResources();

Configuration configuration = resources.getConfiguration();

configuration.fontScale = ConstantUtil.TEXTVIEWSIZE;

resources.updateConfiguration(configuration, resources.getDisplayMetrics());

}

ConstantUtil.TEXTVIEWSIZE是设值的一个静态常量，当TEXTVIEWSIZE=1的时候，会显示系统标准字体大小，这个时候即使系统修改了字体大小，也不会影响到应用程序的字体大小。如果想要字体放大，设值其值>1即可。如果想要字体缩小，设值其值<1即可。

**八、公共方法集成**

几乎每一个Activity都要执行初始化方法，所以可以在BaseActivity基类定义一个私有抽象方法init，然后在onCreate进行调用，这样当继承该基类的Activity，就必须实现init，并在当前Activity的onCreate方法中自动执行。

**最后，给出完整BaseActivity基类代码**

/\*\*

\* BaseActivity是所有Activity的基类，把一些公共的方法放到里面，如基础样式设置，权限封装，网络状态监听等

\* <p>

\* Created by 邹峰立 on 2018/3/5.

\*/

public abstract class BaseActivity extends AppCompatActivity implements NetBroadcastReceiver.NetChangeListener {

public static NetBroadcastReceiver.NetChangeListener netEvent;// 网络状态改变监听事件

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// 隐藏标题栏

if (getSupportActionBar() != null)

getSupportActionBar().hide();

// 沉浸效果

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

// 透明状态栏

getWindow().addFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_STATUS);

// 透明导航栏

getWindow().addFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_NAVIGATION);

}

// 添加到Activity工具类

ActivityUtil.getInstance().addActivity(this);

// 初始化netEvent

netEvent = this;

// 执行初始化方法

init();

}

// 抽象 - 初始化方法，可以对数据进行初始化

protected abstract void init();

@Override

protected void onResume() {

super.onResume();

Resources resources = this.getResources();

Configuration configuration = resources.getConfiguration();

configuration.fontScale = 1;

resources.updateConfiguration(configuration, resources.getDisplayMetrics());

}

@Override

protected void onDestroy() {

// Activity销毁时，提示系统回收

// System.gc();

netEvent = null;

// 移除Activity

ActivityUtil.getInstance().removeActivity(this);

super.onDestroy();

}

@Override

public boolean onKeyDown(int keyCode, KeyEvent event) {

// 点击手机上的返回键，返回上一层

if (keyCode == KeyEvent.KEYCODE\_BACK) {

// 移除Activity

ActivityUtil.getInstance().removeActivity(this);

this.finish();

}

return super.onKeyDown(keyCode, event);

}

/\*\*

\* 权限检查方法，false代表没有该权限，ture代表有该权限

\*/

public boolean hasPermission(String... permissions) {

for (String permission : permissions) {

if (ContextCompat.checkSelfPermission(this, permission) != PackageManager.PERMISSION\_GRANTED) {

return false;

}

}

return true;

}

/\*\*

\* 权限请求方法

\*/

public void requestPermission(int code, String... permissions) {

ActivityCompat.requestPermissions(this, permissions, code);

}

/\*\*

\* 处理请求权限结果事件

\*

\* @param requestCode 请求码

\* @param permissions 权限组

\* @param grantResults 结果集

\*/

@Override

public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {

super.onRequestPermissionsResult(requestCode, permissions, grantResults);

doRequestPermissionsResult(requestCode, grantResults);

}

/\*\*

\* 处理请求权限结果事件

\*

\* @param requestCode 请求码

\* @param grantResults 结果集

\*/

public void doRequestPermissionsResult(int requestCode, @NonNull int[] grantResults) {

}

/\*\*

\* 网络状态改变时间监听

\*

\* @param netWorkState true有网络，false无网络

\*/

@Override

public void onNetChange(boolean netWorkState) {

}

}

**Activity继承实现**

public class MainActivity extends BaseActivity {

private TextView textView;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

initView();

}

private void initView() {

textView = findViewById(R.id.text);

}

@Override

protected void init() {

// 判断权限

if (!hasPermission(Manifest.permission.READ\_PHONE\_STATE)) {

requestPermission(ConstantUtil.PERMISSIONS\_REQUEST\_READ\_PHONE\_STATE, Manifest.permission.READ\_PHONE\_STATE);

}

}

// 处理请求权限结果

@Override

public void doRequestPermissionsResult(int requestCode, @NonNull int[] grantResults) {

switch (requestCode) {

case ConstantUtil.PERMISSIONS\_REQUEST\_READ\_PHONE\_STATE:// 读取手机信息权限

if (grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

// 权限请求成功

Toast.makeText(this, "权限请求成功", Toast.LENGTH\_SHORT).show();

} else {

// 权限请求失败

Toast.makeText(this, "权限请求失败", Toast.LENGTH\_SHORT).show();

}

break;

}

}

// 处理网络状态结果

@Override

public void onNetChange(boolean netWorkState) {

super.onNetChange(netWorkState);

textView.setText(netWorkState ? "有网络" : "无网络");

}

// 设置返回按钮的监听事件

private long exitTime = 0;

public boolean onKeyDown(int keyCode, KeyEvent event) {

// 监听返回键，点击两次退出程序

if (keyCode == KeyEvent.KEYCODE\_BACK && event.getAction() == KeyEvent.ACTION\_DOWN) {

if ((System.currentTimeMillis() - exitTime) > 5000) {

Toast.makeText(getApplicationContext(), "再按一次退出程序", Toast.LENGTH\_LONG).show();

exitTime = System.currentTimeMillis();

} else {

ActivityUtil.getInstance().exitSystem();

}

return true;

}

return super.onKeyDown(keyCode, event);

}

}

[Github地址](https://link.jianshu.com/?t=https%3A%2F%2Fgithub.com%2Fzrunker%2FZBaseActivity)  
[阅读原文](https://link.jianshu.com/?t=http%3A%2F%2Fwww.ibooker.cc%2Farticle%2F141%2Fdetail)