作者：张明云  
链接：https://www.zhihu.com/question/27227425/answer/35881143  
来源：知乎  
著作权归作者所有，转载请联系作者获得授权。

1. 写一个抽象的BaseActivity.java，将初始化抽象为setContentView()、findViews()、getData()、showContent()这四个方法，所有的Activity都继承它：

public abstract class BaseActivity extends Activity{

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

init();

}

public void init(){

setContentView();

findViews();

getData();

showContent();

}

public abstract void setContentView();

public abstract void findViews();

public abstract void getData();

public abstract void showContent();

}

2. 方法返回值不返回null，返回空值，比如：

public static List<String> getFolderFiles( String folderPath ){

List<String> fileList = new ArrayList<String>( );

if( TextUtils.isEmpty( folderPath ) ){

return fileList;

}

File file = new File( folderPath );

if( file.isDirectory( ) ){

File[] files = file.listFiles( );

if( null != files ){

fileList = new ArrayList<String>( );

for( File subFile : files ){

fileList.add( subFile.getPath( ) );

}

}

}

return fileList;

}

3. 在方法顶部做入参判断，入参异常直接返回，比如：

public static List<String> getAssertsFiles( Context context ){

List<String> assertsFileList = new ArrayList<String>( );

if( null == context ){

return assertsFileList;

}

AssetManager assetManager = context.getAssets();

String[] files = null;

try {

files = assetManager.list("");

assertsFileList = Arrays.asList( files );

} catch (IOException e) {

e.printStackTrace( );

}

return assertsFileList;

}

4. 使用List、Map而不是ArrayList、HashMap声明成员变量、作为返回值、作为入参等等，尽量用抽象而不是具体实现。  
  
5. 在任何时候不使用类的finalize方法释放资源；  
  
6. 在finally中关闭文件流、cursor等等；  
  
7. 封装一个DebugUtils类来管理程序中所有的打印：

public class DebugUtils{

public static final String TAG = "Debug";

private DebugUtils( ){

}

public static void println( String printInfo ){

if( Debug.DEBUG\_MODE && null != printInfo ){

System.out.println( printInfo );

}

}

public static void print( String printInfo ){

if( Debug.DEBUG\_MODE && null != printInfo ){

System.out.print( printInfo );

}

}

public static void printLogI( String logInfo ){

printLogI( TAG, logInfo );

}

public static void printLogI( String tag, String logInfo ){

if( Debug.DEBUG\_MODE && null != tag && null != logInfo ){

Log.i( tag, logInfo );

}

}

public static void printLogE( String logInfo ){

printLogE( TAG, logInfo );

}

public static void printLogE( String tag, String logInfo ){

if( Debug.DEBUG\_MODE && null != tag && null != logInfo ){

Log.e( tag, logInfo );

}

}

public static void printLogW( String logInfo ){

printLogW( TAG, logInfo );

}

public static void printLogW( String tag, String logInfo ){

if( Debug.DEBUG\_MODE && null != tag && null != logInfo ){

Log.w( tag, logInfo );

}

}

public static void printLogD( String logInfo ){

printLogD( TAG, logInfo );

}

public static void printLogD( String tag, String logInfo ){

if( Debug.DEBUG\_MODE && null != tag && null != logInfo ){

Log.d( tag, logInfo );

}

}

public static void printLogV( String logInfo ){

printLogV( TAG, logInfo );

}

public static void printLogV( String tag, String logInfo ){

if( Debug.DEBUG\_MODE && null != tag || null != logInfo ){

Log.v( tag, logInfo );

}

}

public static void printLogWtf( String logInfo ){

printLogWtf( TAG, logInfo );

}

public static void printLogWtf( String tag, String logInfo ){

if( Debug.DEBUG\_MODE && null != tag && null != logInfo ){

Log.wtf( tag, logInfo );

}

}

public static void showToast( Context context, String toastInfo ){

if( null != context && null != toastInfo ){

Toast.makeText( context, toastInfo, Toast.LENGTH\_LONG ).show( );

}

}

public static void showToast( Context context, String toastInfo, int timeLen ){

if( null != context && null != toastInfo && ( timeLen > 0 ) ){

Toast.makeText( context, toastInfo, timeLen ).show( );

}

}

public static void printBaseInfo( ){

if( Debug.DEBUG\_MODE ){

StringBuffer strBuffer = new StringBuffer( );

StackTraceElement[ ] stackTrace = new Throwable( ).getStackTrace( );

strBuffer.append( "; class:" ).append( stackTrace[ 1 ].getClassName( ) )

.append( "; method:" ).append( stackTrace[ 1 ].getMethodName( ) )

.append( "; number:" ).append( stackTrace[ 1 ].getLineNumber( ) )

.append( "; fileName:" ).append( stackTrace[ 1 ].getFileName( ) );

println( strBuffer.toString( ) );

}

}

public static void printFileNameAndLinerNumber( ){

if( Debug.DEBUG\_MODE ){

StringBuffer strBuffer = new StringBuffer( );

StackTraceElement[ ] stackTrace = new Throwable( ).getStackTrace( );

strBuffer.append( "; fileName:" ).append( stackTrace[ 1 ].getFileName( ) )

.append( "; number:" ).append( stackTrace[ 1 ].getLineNumber( ) );

println( strBuffer.toString( ) );

}

}

public static int printLineNumber( ){

if( Debug.DEBUG\_MODE ){

StringBuffer strBuffer = new StringBuffer( );

StackTraceElement[ ] stackTrace = new Throwable( ).getStackTrace( );

strBuffer.append( "; number:" ).append( stackTrace[ 1 ].getLineNumber( ) );

println( strBuffer.toString( ) );

return stackTrace[ 1 ].getLineNumber( );

}else{

return 0;

}

}

public static void printMethod( ){

if( Debug.DEBUG\_MODE ){

StringBuffer strBuffer = new StringBuffer( );

StackTraceElement[ ] stackTrace = new Throwable( ).getStackTrace( );

strBuffer.append( "; number:" ).append( stackTrace[ 1 ].getMethodName( ) );

println( strBuffer.toString( ) );

}

}

public static void printFileNameAndLinerNumber( String printInfo ){

if( null == printInfo || !Debug.DEBUG\_MODE ){

return;

}

StringBuffer strBuffer = new StringBuffer( );

StackTraceElement[ ] stackTrace = new Throwable( ).getStackTrace( );

strBuffer.append( "; fileName:" ).append( stackTrace[ 1 ].getFileName( ) )

.append( "; number:" ).append( stackTrace[ 1 ].getLineNumber( ) ).append( "\n" )

.append( ( null != printInfo ) ? printInfo : "" );

println( strBuffer.toString( ) );

}

public static void showStrictMode( ) {

if (DebugUtils.Debug.DEBUG\_MODE) {

StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()

.detectDiskReads().detectDiskWrites().detectNetwork().penaltyLog().build());

StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()

.detectLeakedSqlLiteObjects().detectLeakedClosableObjects().penaltyLog().penaltyDeath().build());

}

}

public static void d(String tag, String msg){

if(DebugUtils.Debug.DEBUG\_MODE){

Log.d(tag, msg);

}

}

public class Debug{

public static final boolean DEBUG\_MODE = true;

}

}

8. 静态类的构造方法私有化，具体实例参见7；  
  
9. 没有必要用硬件加速的页面在AndroidMainfest.xml中将其关掉

android:hardwareAccelerated="false"

10. 在没有使用到多点触控的页面，通过在主题中设置下列两个属性将多点触控关掉：

<item name="android:windowEnableSplitTouch">false</item>

<item name="android:splitMotionEvents">false</item>

11. 定义一个防止视图重复点击的静态类，在需要做防止重复点击的地方加上该判断：

public class BtnClickUtils {

private static long mLastClickTime = 0;

private BtnClickUtils( ){

}

public static boolean isFastDoubleClick() {

long time = System.currentTimeMillis();

long timeD = time - mLastClickTime;

if ( 0 < timeD && timeD < 500) {

return true;

}

mLastClickTime = time;

return false;

}

}

12. 在布局中使用LinearLayout的android:weight属性时，将view的android:layout\_width属性设置为0dp；  
  
13. 涉及到APP的核心数据全部加密处理，在使用的时候解密；  
  
14. 如果你的IDE是Eclipse，用好这几个快捷键，爽歪歪滴：

* CTRL + SHIFT + O
* ALT + SHIFT + M
* ALT + SHIFT + S
* CTRL + O
* CTRL + /
* ALT + /
* CTRL + D
* CTRL + K
* ALT + → 和 ALT + 左箭头
* CTRL + SHIFT + X 和 CTRL + SHIFT + Y
* F5、F6、F7、F8
* F11
* CTRL + M

还有很多很多，先就写在这里吧，希望对题主有帮助。  
**再补一刀：**如果题主有代码洁癖，很注重代码质量，可以详尽阅读SonarQube关于JAVA编码的规则：[Sonar@OSC - 代码质量分析平台](//link.zhihu.com/?target=http%3A//sonar.oschina.net/coding_rules%23s%3DcreatedAt%257Casc%3Dfalse)，如果条件允许，自己配置一下，每周通过Sonar运行一下自己的代码，以提高代码质量。