设计文档及规范

# MVP模式

### 为什么使用MVP模式？

传统的Android MVC开发模式中，Activity、Fragment等View层通常还承担着Controller的职责，需要同时负责逻辑处理和显示，随着界面及逻辑的复杂度不断提升，代码很容易变得复杂和臃肿，可读性及可维护性变差。MVP作为MVC的一种演化模式，可以很好的解决这些问题，它将视图和逻辑层分离开来，减少了它们之间的耦合度，同时使代码结构更加清晰，增加代码的可读性和可维护性。

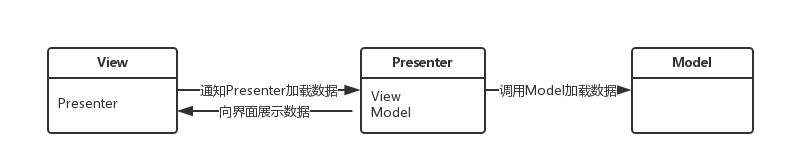
### MVP原理

Model：数据、逻辑操作层，负责网络操作，数据库操作，文件操作等

View：界面展示层，负责界面展示，具体表现为Activity、Fragment、自定义View等

Presenter：中间层，连接Model和View层，同时持有Model和View接口的引用。

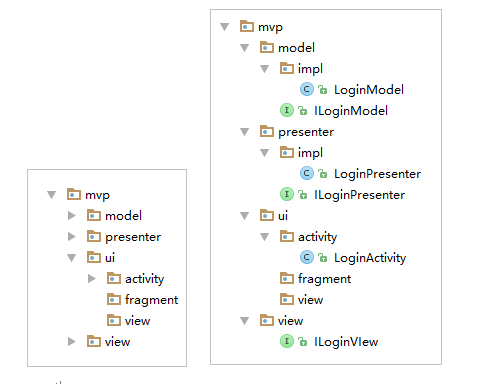
如下图所示，View与Model并不直接交互，而是使用Presenter作为View与Model之间的桥梁。其中Presenter中同时持有View层以及Model层的Interface的引用，而View层持有Presenter层Interface的引用。当View层某个界面需要展示某些数据的时候，首先会调用Presenter层的某个接口，然后Presenter层会调用Model层请求数据，当Model层数据加载成功之后会调用Presenter层的回调方法通知Presenter层数据加载完毕，最后Presenter层再调用View层的接口将加载后的数据展示给用户。这就是MVP模式的整个核心过程。



### 本项目MVP使用规范

一般情况下，将Model层的接口和实现类放在model和model.impl包下，将Presenter层的接口和实现类放在presenter和presenter.impl包下，将View层接口放在view包下，将显示层放在ui下的ui.activity/ui.fragment/ui.view等对应的包下。特殊情况，比如一个相对较大的模块，可以单独创建一个包，然后在该模块包下再分别创建对应的MVP包，将对应的接口和类放在对应的包内。

参考：



如果对MVP模式思想还不太理解的话，建议自行百度学习，附上github几个MVP应用的小例子：

<https://github.com/chentao0707/SimplifyReader>

<https://github.com/antoniolg/androidmvp>

# 相关功能模块

## 多语言支持

### 功能描述

支持根据系统语言切换多界面语言，并且支持通过增加语言文件的方式支持新的语言。

### 使用说明

项目默认语言文件均在res/values/string.xml文件中。需要添加其它语言只需要做如下操作：

1. 将默认的string.xml文件拷贝出来，交给负责翻译的人员进行翻译。
2. 翻译完成后，将文件传回给开发人员
3. 开发人员根据不同语言，将对应的文件打包到app中，部分国家语言对应的存放目录如下：

中文（中国）： res/values-zh-rCN

中文（台湾）： res/values-zh-rTW

中文（香港）： res/values-zh-rHK

英语（美国）： res/values-en-rUS

英语（英国）： res/values-en-rGB

日文： res/values-ja-rJP

法文（法国）： res/values-fr-rFR

德文（德国）： res/values-de-rDE

## 自升级模块

### 功能描述

实现App的自更新，该模块需要网络支持，将新版本或补丁上传到后台服务器，服务器自动推送给客户端，APP运行后发现新版本提示用户升级。

支持热更新，支持推荐升级和强制升级两种方式，支持根据不同渠道不同网络环境进行定制，支持升级提示框界面定制。参考图：



### 使用说明

1. 普通升级：新版本发布后，登陆后台管理，选择应用升级-全量更新，将新版本的apk上传到服务器，配置好升级策略后即可以发布新版本。
2. 热修复升级：开发人员根据两个不同版本制作好差分包，登陆后台管理，选择应用升级-热修复，将制作好的补丁包上传到服务器，同样配置好升级策略后即可发布新的升级包。
3. 主要API说明：

base/AppConfig.java

AUTO\_CHECK\_UPGRADE = true;//是否开启自动检测升级

AUTO\_DOWNLOAD\_ON\_WIFI = false;//是否在WIFI下自动下载升级包

ENABLE\_HOT\_FIX = true;//是否开启热升级

Beta.checkUpgrade();//手动检测更新

UpgradeInfo upgradeInfo = Beta.getUpgradeInfo();//获取升级信息

Beta.upgradeDialogLayoutId = R.layout.upgrade\_dialog; //自定义升级对话框布局

1. 注意事项：
2. 新版本的版本号（versionCode值，build.gradle文件中配置）必须比旧版本大。
3. 需要保存每一次发布的新版本apk以及对应的R.txt和mapping.txt文件，以便后期针对该版本制作补丁包。
4. 每一次制作补丁包的tinkerId值（tinker-support.gradle文件中配置）都必须不同且唯一

## 异常处理模块

### 功能描述

应用发生异常后，能够捕获到异常信息，将信息存在本地或者发送到后台服务器，以便开发人员查看和解决问题。



### 使用说明

默认情况下app会将所有异常Log信息保存到本地文件中，如果有网络的情况下，会将每次异常的Log信息传回给服务器，开发人员登录后台管理，选择异常上报功能，即可查看已经上报的异常。

相关配置base/BaseApplication.java：

@ReportsCrashes(  
 formUri = "http://192.168.0.23:8080/AppTestService/LogServlet",  
 mode = ReportingInteractionMode.*TOAST*,  
 resToastText = R.string.*crash\_toast\_text*,  
 reportSenderFactoryClasses = {CustomeLogSenderFactory.class} ,  
 customReportContent = {*APP\_VERSION\_NAME*, *ANDROID\_VERSION*, *PHONE\_MODEL*, *CUSTOM\_DATA*, *STACK\_TRACE*, *LOGCAT* }  
)

formUri：Log提交的目标服务器地址

mode：异常发生后提示方式，包括Toast、弹窗、通知等方式

resToastText：选用Toast通知对应的字符串

reportSenderFactoryClasses：自定义发送接口类，对应的CustomeLogSenderFactory文件中可以修改Log的处理逻辑，保存本地或者发送给服务器或者其他方式。

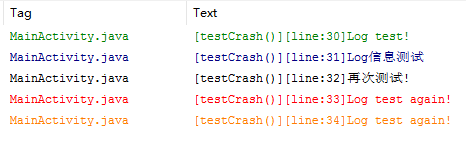
customReportContent：自定义需要添加的一些字段。

## 日记模块

### 功能描述

应用内的所有模块内可以方便的调用日志模块，进行日志的记录。日志记录分为三级：info（友好信息、进度等）、warn（警告）、error（错误）、fatal（崩溃）。

日记将以如下格式输出 ：[方法名称()][line:行号]log信息，Tag为类文件名



### 使用说明

Log开关：base/AppConfig.java

*/\*\* 是否打印Log信息\*/*public static final boolean *SHOW\_LOG* = true;

默认情况下直接调用：

LogUtil.*i*("Log test!");  
LogUtil.*d*("Log信息测试");  
LogUtil.*v*("再次测试!");  
LogUtil.*w*("Log test again!");  
LogUtil.*e*("Log test again!");

如果需要将log信息上报给服务器，使用带report的方法，

LogUtil.*reportI*("Log test!");  
LogUtil.*reportD*("Log信息测试");  
LogUtil.*reportV*("再次测试!");  
LogUtil.*reportW*("Log test again!");  
LogUtil.*reportE*("Log test again!");

如果需要自定义tag标签，使用带tag参数的方法：

LogUtil.*i*("tag", "Log test!");  
LogUtil.*d*("tag", "Log信息测试");  
LogUtil.*v*("tag", "再次测试!");  
LogUtil.*w*("tag", "Log test again!");  
LogUtil.*e*("tag", "Log test again!");

## 数据库操作模块

### 功能描述

数据库操作模块，该模块主要功能由GreenDao框架来支持，GreenDao是一个轻量、快速的Android ORM解决方案，它的特点包括性能最大化（对于Android来说，可能是最快的ORM）、很容易使用APIs、对Android高度优化、最小的内存开销、较小的文件体积等。

### 使用说明

参考：https://github.com/greenrobot/greenDAO

# 相关开源框架

## Butter Knife依赖注入框架

### 功能描述

依赖注入框架，能够方便的绑定各种组件、资源和事件，节约开发人员的时间和精力，同时该框架采用的是编译时注入的方式，不会在运行时对App的性能造成影响。

### 使用说明

//绑定View  
@BindView(R.id.*text*) TextView mTextView;  
//绑定颜色  
@BindColor(R.color.*colorPrimary*) int mColorPrimary;  
//绑定图片资源  
@BindDrawable(R.mipmap.*ic\_launcher*) Drawable mIcLauncher;

//绑定按钮点击事件  
@OnClick(R.id.*btn\_crash*)void testCrash(View view){  
 LogUtil.*i*("tag", "Log test!");  
}

具体请参考：<https://github.com/JakeWharton/butterknife>

## RxJava/RxAndroid异步框架

### 功能描述

异步框架，简洁(逻辑性)，随着代码逻辑越来越复杂，仍可以保持简洁，提高代码可读性和可维护性。

### 使用说明

具体用法请自行百度或参考：

<https://github.com/ReactiveX/RxAndroid>，<https://github.com/ReactiveX/RxJava>

推荐文章：http://gank.io/post/560e15be2dca930e00da1083

## Okgo 网络操作框架

### 功能描述

封装了okhttp的标准RESTful风格的网络框架，支持大文件上传下载，上传进度回调，下载进度回调，表单上传（多文件和多参数一起上传），链式调用，可以自定义返回对象，支持Https和自签名证书，支持超时自动重连，支持cookie的持久化和自动管理，支持五种缓存模式缓存网络数据，支持301和302重定向，扩展了统一的上传管理和下载管理功能。

### 使用说明

参考：https://github.com/jeasonlzy/okhttp-OkGo

## Glide图片加载和缓存框架

### 功能描述

一个开源的图片加载和缓存处理的第三方框架，该框架被广泛的运用于Google官方应用中，是Google推荐的一个图片加载框架。

### 使用说明

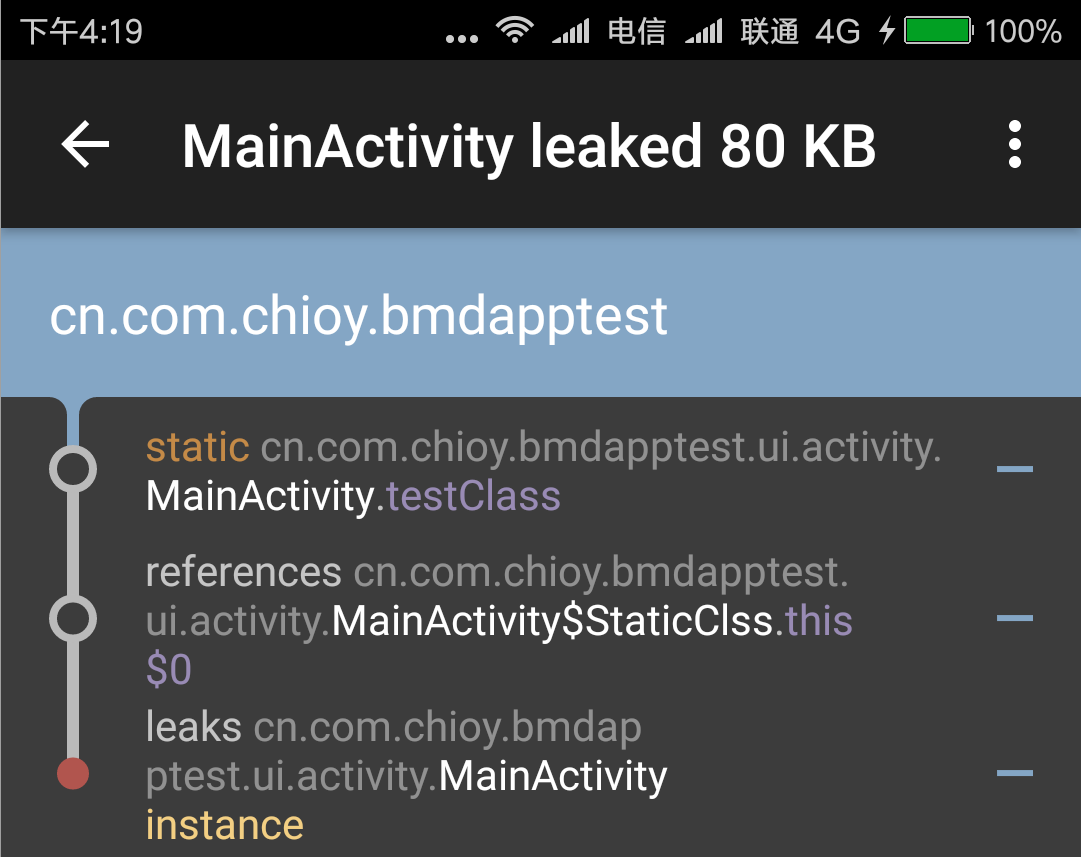
Glide.*with*(MainActivity.this)  
 .load("http://www.test.com/img.jpg") //从URL加载  
 .placeholder(R.mipmap.*ic\_launcher*) //默认图片  
 .error(R.mipmap.*ic\_launcher*) //加载图片失败后显示的图片  
 .into(mImageView); //目标ImageView

参考：https://github.com/bumptech/glide

## Leakcanary内存泄漏监测框架

### 功能描述

开发调试辅助框架，能够检测出代码中出现的内存泄漏，并展现出来。

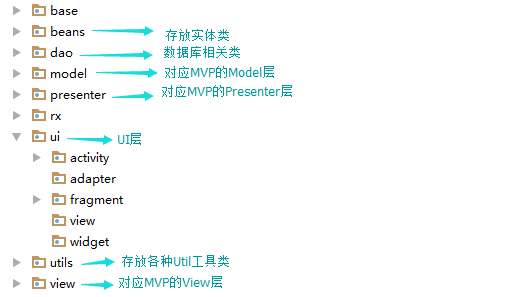


### 使用说明

只需要在应用启动时进行初始化。

//初始化LeakCanary  
if (LeakCanary.*isInAnalyzerProcess*(this)) {  
 return;  
}  
LeakCanary.*install*(this);

# 项目分包说明



# 相关规范

## 命名规范

|  |  |  |
| --- | --- | --- |
| 类型 | 规则 | 备注 |
| 常量 | 1. 全部大写； 2. 中间“\_”连接； | 如：  TAG\_NAME=”text”; |
| 变量 | 1. 首字母小写； 2. Java驼峰命名； 3. 全局变量以字母m作为前缀 | 如：  Context mContext;  TextView textView; |
| 方法 | 1. 首字母小写； 2. Java驼峰命名； | 如：  void testMethod(); |
| 类名 | 1. 首字母大写 2. Java驼峰命名； 3. Activity类以Activity结尾 4. Fragment类以Fragment结尾 5. Service类以Service结尾 6. 广播类以Receiver结尾 7. ContentProvide类以Provider结尾 8. Application类以Application结尾 9. 自定义View以Custom\*\*\*View命名 10. 自定义Adapter以Adapter结尾 11. ViewHolder以Holder结尾 | 如：  Class MyActivity  Class FirstFragment  Class BootReceiver  Class CustomButtonView |
| 接口 | 1. 首字母大写I开头 2. Java驼峰命名 3. 接口实现类以 impl结尾 | 如：  Interface ILoginModel  Class LoginModelImpl |
| 包名 | 1. 所有字母小写 2. 按功能模块进行划分 | 参考项目包名 |
| 布局文件 | 1. 全部小写 2. 中间以”\_”连接 3. Activity对应的布局以\_activity结尾 4. Fragment对应的布局以\_fragment结尾 5. 列表项布局以\_item结尾 | 如：  main\_activity.xml  left\_fragment.xml  news\_item.xml |
| Drawable资源 | 1. 全部小写字母 2. 中间以”\_”连接 3. 同一类型的图片以统一前缀划分 4. 图标以ic\_开头 5. 背景图以bg\_开头 | 如：  ic\_head.png  bg\_main.png |
| 资源文件 | 1. 全部小写 2. 中间以”\_”连接 3. 同一类型资源以统一前缀划分 4. Selector资源已\_selector结尾 5. Menu资源以\_menu结尾 6. 动画资源以\_animator结尾 7. 其它类似 | 如：  btn\_login\_selector.xml  login\_menu.xml  btn\_hide\_animator.xml |
| 后续添加 |  |  |

## 注释规范

|  |  |  |
| --- | --- | --- |
| 类型 | 规则 | 备注 |
| 类注释 | 1. IDE按如下规则添加注释模板   /\*\*  \* 类描述  \*  \* @author 作者  \* @Date 2017/2/22  \*/ |  |
| 方法注释 | /\*\*  \* desc:描述  \* @param 参数名 参数描述  \* @param 参数名2 参数描述  \* @return 返回值类型说明  \* @throws Exception 异常说明  \*/ |  |
| 常量和成员变量 | /\*\*注释信息\*/ |  |
| 内部代码注释 | //这里是注释信息 |  |
| TODO | 代码不完善的地方使用TODO注释，方便后期快速定位 |  |

## 代码风格

|  |  |  |
| --- | --- | --- |
| 类型 | 规则 | 备注 |
| 代码顺序 | 1、Android四大组件各回调方法以生命周期前后调用作为排序规则，如Activity中将onCreate放在开头，将onDestory放在最后 |  |
| 大括号 | 左大括号不换号  右大括号换行 | void testMethod(){  }  If(true){  } |
| 后续添加 |  |  |
|  |  |  |