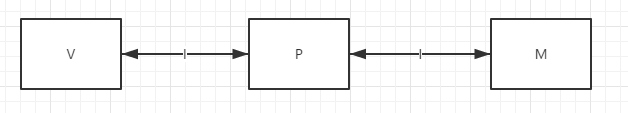
# 架构规范

## 生成MVP模板代码

在Terminal面板上  
执行generate\_MVP 模块名 包名 场景名 -A/F [-M]  
模块名 包名 场景名 -A/-F/-V必填  
-A生成Activity,-F生成Fragment,-V生成View  
-M选填，当有-M时，MVP生成代码会在 包/ 下生成，没有-M时，MVP生成代码会在 包名/场景名 下生成  
  
如generate\_MVP module\_name com.fxchat.geniusrank AccountManager -A  
如generate\_MVP module\_name com.fxchat.geniusrank.userpage Discuss -F -M



V层控件初始化，页面事件监听，显示UI。所有点击事件都需要调到P层，进行业务分发操作。M层处理所有的网络请求，本地存储，数据库相关操作

## 方法命名

1. V层中可以由P层调用的公共方法，加上p2v前缀；同理，还有v2p、p2m、m2p相关方法名前缀。

以登录成功打开首页的流程示例：

v.onCLick -> p.v2pLogin -> p.checkLogin -> m.p2mLoginNet -> p.m2pLogin -> p.loginSuccess -> v.p2vOpenMain

1. M层中方法请根据数据来源，加上相关后缀，网络Net、数据库Db、SharedPreferences的是Sp。例如loginByNet、findCacheByDb、saveBySp。

## 注意事项

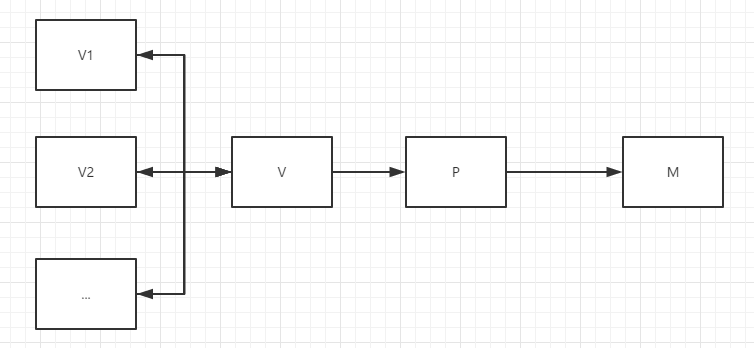
1. 所有调用打开activity，对话框，吐司的操作都在P层，然后再由P层回调到V层编写无业务逻辑的实现代码，P层与M层不允许直接或间接引入Activity或者控件等。注意，A页面一个简单的点击打开B页面的操作，没有什么业务处理，也必须按照这个操作来。

v.onClick -> p.v2pOpenB -> v.p2vOpenB -> B.openSelf

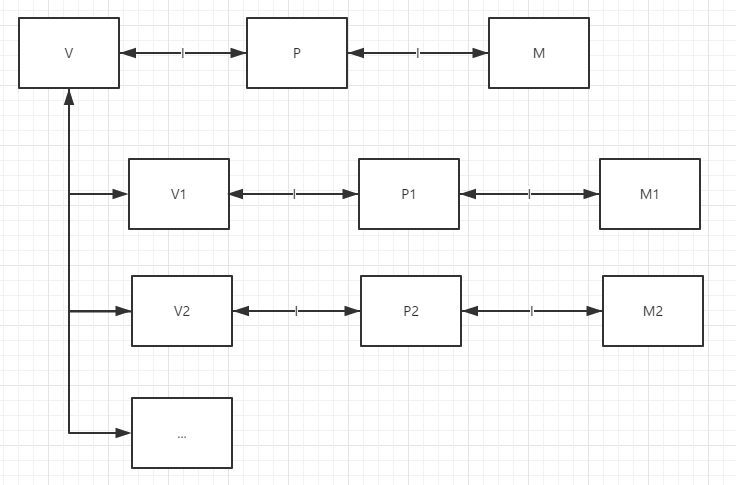
v.onClick -> p.v2pOpenB -> p.pCheckOpenB -> v.p2vOpenB -> B.openSelf

1. V层页面比较复杂时，布局里面控件较多，请在布局中用include引入新的布局，在Activity中可以引入面板Panel的概念，可以按照功能或者布局层级分成不同的面板。

如果整个页面仅仅只是页面复杂，逻辑不复杂，可以划分成1个V，多个Panel，1个P，1个M。



如果整个页面可以划分成多个功能比较独立的面板，请先考虑能不能用fragment对整个页面进行分离，如果不好分离的情况下，再考虑引出不同功能独立的Panel出来。并且针对每个不同的panel定义独立的P层M层。即1个V，1个P，1个M，以及多个Panel，它们各自的P与各自的M。



1. 注意V/P/M层相关之间的调用只能通过接口，只有覆盖接口的相关方法才能用public修饰，其他的请定义成私有方法。

# 命名规范

* 开发前，先确定对应的业务模块，统一模块名module；公共模块 common
* 拜伦基础库类名必须以BL开始，方法名、属性无需遵守

## 控件

1. 有通用简称用通用简称，没有的话用完整英文拼写，禁止使用中文拼音。
2. View 组件的资源 id 需要以 View 的缩写作为前缀

例如tv\_account

1. 变量名以缩写为前缀，采用驼峰法命名

例如tvAccount，mTvAccount

|  |  |
| --- | --- |
| 分类 | 命名 |
| Button | btn\_xxx |
| ImageView | Iv\_xxx |
| CheckBox | cb\_xxx |
| RadioButton | rb\_xxx |
| EditText | et\_xxx |
| TextView | tv\_xxx |
| ScollView | sv/hsv\_xxx |
| ProgressBar | pb\_xxx |
| ListView | lv\_xxx |
| ConstraintLayout | cl\_xxx |
| RelativeLayout | rl\_xxx |
| LinearLayout | ll\_xxx |
| FramLayout | fl\_xxx |
| View | v\_xxx |

## Layout布局

布局文件以下划线分割，小写字母命名；对应类名采用驼峰命名法

模块+布局类型+业务名称

|  |  |
| --- | --- |
| Activity | module\_activity\_xxx |
| Fragment | module\_fragment\_xxx |
| Dialog | module\_dialog\_xxx |
| include | module\_panel\_xxx |
| ListView /GridView/... | module\_item\_xxx |

## Drawable图片资源

模块+业务 + 控件（btn/ll/...）

+ 用途（bg/icon/...）

+ 状态（normal/pressed/checked/...）

+ 颜色（red/white/...）

## Anim 动画

模块名\_逻辑名称\_[方向|序号]

tween 动画资源 ： 尽可能以通用的动画名称命名，如 module\_fade\_in ,

module\_fade\_out , module\_push\_down\_in (动画+方向)；

frame 动画资源：尽可能以模 块+功能命名+序号。如：module\_loading\_grey\_001

## Color颜色

模块名\_逻辑名称\_颜色

<color name="module\_btn\_bg\_color">#33b5e5e5</color>

## dimen

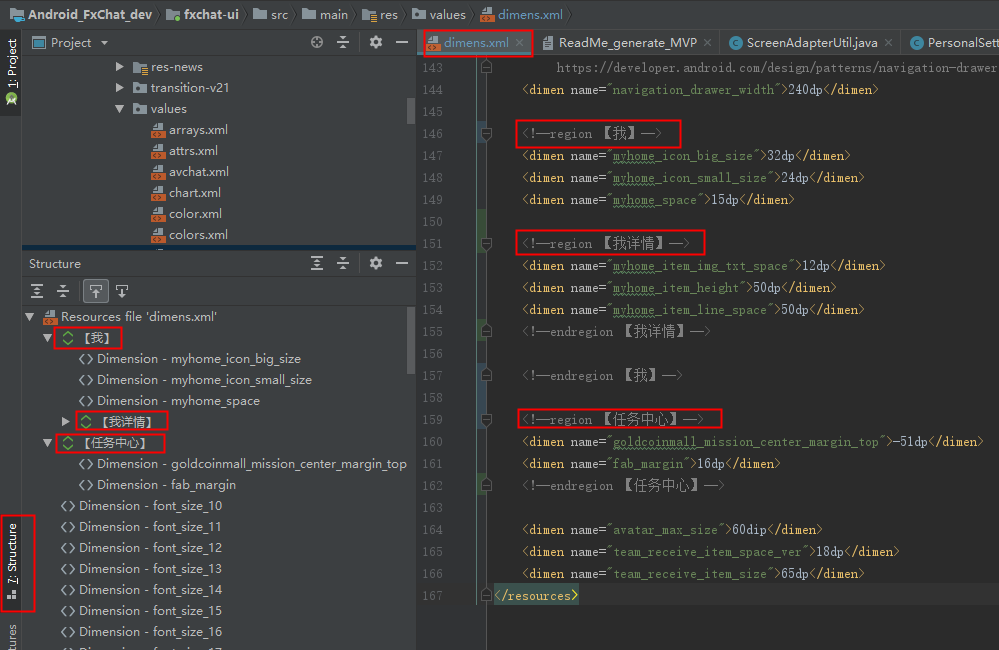
模块名\_描述信息

<dimen name="module\_horizontal\_line\_height">1dp</dimen>

# 代码格式

* 推荐安装阿里巴巴插件Alibaba Java Coding Guidelines 插件进行代码规范检查
* 建议，仔细看一遍《阿里巴巴 Android开发手册》
* 代码提交之前，请先格式化一次再提交

1. If else语句必须加上大括号
2. Switch语句必须加上default判断
3. 对于函数多参数传值，遵守如下约定
4. 参数个数小于等于4个时，可参数依次传值method(args1, args2, args3);
5. 参数个数大于4个时，以需要参数构建的模型传值
6. 关于color，string，style，dimen等，需要按功能模块在文件中用region区分，注释注意成对出现，这样查看起来比较方便。如下图所示，双击左侧某个条目，可以直接定位到对应的模块。

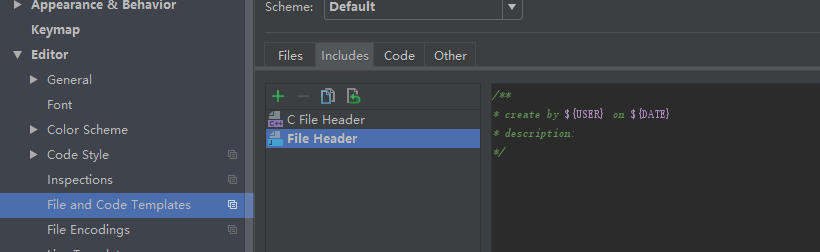


# 注释

尽量去写足够的注释，去描述一下思路，达到看了可以明白某一块代码的效果。

1. 新建类自动生成注释模板

File–>Settings–>Editor–>File and code Template



/\*\*

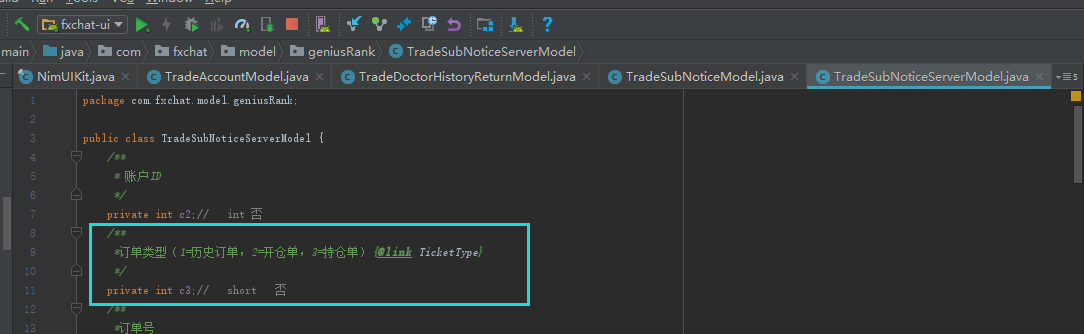
\* create by ${USER} on ${DATE}

\* description:

\*/

1. 注释链接

对象中变量如果是int，并关联一个枚举，需要在注释上加上枚举链接{@link 类名}



注释中也可以直接链接方法名{@link 类名#方法名(参数类型1, 参数类型2)}

1. 注意事项
2. 对于属性注释，位置位于属性顶部，使用，不能在行尾直接注释
3. 接口里面的方法，抽象类的方法必须加上注释
4. 工具类中每个方法必须加上注释。

# 代码管理规范

## 代码库分支管理

我们定义某项目的分支由3部分构成：主线分支（master）、 开发分支（dev）、标签分支（tag)。开发中的任务只能在dev分支上进行，master分支是所有已上线功能代码的集合，tag分支是每一个线上版本对应的节点代码。

开发新项目时，gitlab上会新建一个拜伦名下的项目工程，这个时候需要将工程fork到自己的远程仓库去。

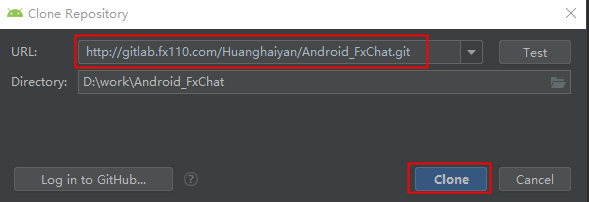


## 代码拉取流程

1. 第一次代码拉取

vcs -> checkout from version control ->git

url为个人远程服务器代码的仓库地址，而不是拜伦仓库下的代码地址，不允许直接提交代码到拜伦仓库。

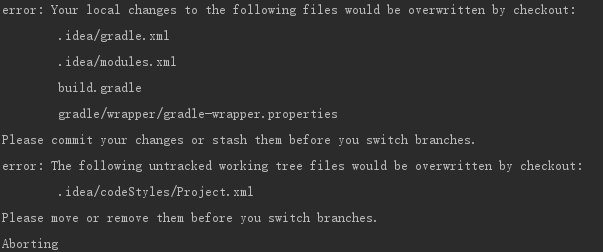


克隆之后，打开工程，在终端命令行输入git branch，发现指向的事master分支，这个时候需要切换到dev分支，使用命令git checkout dev

如果代码拉取失败，这个时候先需要revert一下，将改动的配置文件还原成服务器代码。

vcs -> git ->revert

并且删除工程下对应目录的文件。



之后再重新切换分支就可以了。

1. 关联拜伦仓库，以便随时同步最新代码。

git remote add upstream repo

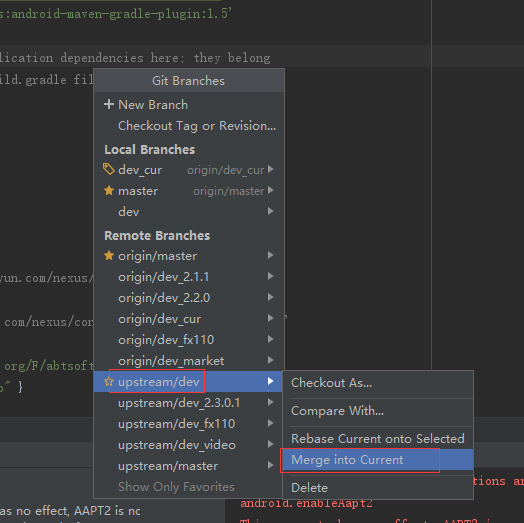
repo为代码的仓库地址

git remote -v

查看当前关联的远程库

1. 更新服务器最新代码

Vcs -> git -> branches ,选择拜伦库下的dev分支，合并到当前分支。



## 代码提交流程

1. 代码提交前，请先同步最新代码，解决冲突

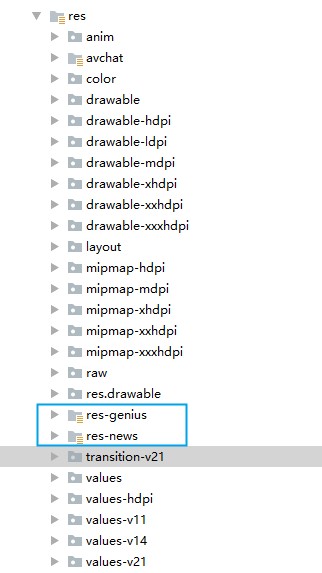
Vcs -> git -> commit File

1. 代码提交时，请预览一下所要提交的内容，与当前描述无关的代码请别提交，注意没有漏提交，注意别提交相关配置文件
2. 如果改了两个需求的内容，请分成两次提交，别一次提交所有代码，确保每次代码提交只涉及1个功能点。
3. 推荐每天下班前将自己远程服务器的代码申请合并到拜伦仓库的代码中，以便别人拉取最新的代码

# 资源分包

组件化开发就不用考虑了。

非组件化开发，不同功能模块建议进行资源分包，这样可以该模块的布局，一些图片资源、shape等等就可以在对应文件夹找到。



# 其他

1. 打开activity请在对应的activity中新建静态方法openselef，

传参方式：

2个参数可以直接在方法里传参。

public static void openself(Context context, int brokerId) {

...

}

超过2个参数，必须新建的一个实体类，专门用来传参

public static void openself(Context context, XXXActivityEntity entity) {

...

}

Activity 间的数据通信，对于数据量比较大的，避免使用 Intent + Parcelable

的方式，可以考虑 EventBus 等替代方案，以免造成 TransactionTooLargeException。

1. 注册/解注册观察者，请注意在Activity或Fragment的生命周期中成对出现
2. 持久化存储应该在 Activity#onPause()/onStop()中实行。
3. 当前Activity的onPause方法执行结束后才会执行下一个Activity的onCreate方法，所以在 onPause 方法中不适合做耗时较长的工作，*这会影响到页面之间的跳转效率*
4. 不要在 Activity#onDestroy()内执行释放资源的工作，例如一些工作线程的销毁和停止，因为 onDestroy()执行的时机可能较晚。可根据实际需要，在Activity#onPause()/onStop()中结合 isFinishing()的判断来执行。
5. Activity的继承方法写在一起，例如onCreate(),onDestory(),onActivityResult()等，方便与查找，以及成对的在Activity生命周期中做订阅取消等操作
6. 不要在 Android 的 Application 对象中缓存数据。
7. 布局中不得不使用 ViewGroup 多重嵌套时，不要使用 LinearLayout 嵌套，改用 RelativeLayout 或者 ConstraintLayout，可以有效降低嵌套数；禁止在设计布局时多次设置子 view 和父 view 中为同样的背景造成页面过度绘制。
8. 不能使用 ScrollView 包裹 ListView/GridView/ExpandableListVIew;因为这样会把 ListView 的所有 Item 都加载到内存中，要消耗巨大的内存和 cpu 去绘制图面。

<?xml version="1.0" encoding="utf-8"?>

<android.support.v4.widget.NestedScrollView>

<LinearLayout>

<ImageView/>

...

<android.support.v7.widget.RecyclerView/>

</LinearLayout>

</android.support.v4.widget.NestedScrollView>

1. 线程池不允许使用 Executors 去创建，而是通过 ThreadPoolExecutor 的方式，这样的处理方式让写的同学更加明确线程池的运行规则，规避资源耗尽的风险。

ThreadPoolExecutor 设置线程存活时间(setKeepAliveTime)，确保空闲时线程能被释放。

Executors 返回的线程池对象的弊端如下：

1) FixedThreadPool 和 SingleThreadPool ： 允许的请求队列长度为Integer.MAX\_VALUE，可能会堆积大量的请求，从而导致 OOM；

2) CachedThreadPool 和 ScheduledThreadPool ： 允许的创建线程数量为Integer.MAX\_VALUE，可能会创建大量的线程，从而导致 OOM。

1. 新建线程时，定义能识别自己业务的线程名称，便于性能优化和问题排查。

public class MyThread extends Thread {

public MyThread(){

super.setName("ThreadName"); …

}

}

1. SharedPreference 中只能存储简单数据类型（int、boolean、String 等），复杂数据类型建议使用文件、数据库等其他方式存储。

相关修改使用 apply 方法进行提交会先写入内存，然后异步写入磁盘，commit 方法是直接写入磁盘。如果频繁操作的话 apply 的性能会优于 commit。仅当需要确定提交结果，并据此有后续操作时，才使用 Editor#commit()。

1. png 图片使用 tinypng 或者类似工具压缩处理，减少包体积。
2. 在 Activity.onPause()或 Activity.onStop()回调中，关闭当前 activity 正在执行的的动画。
3. 在动画或者其他异步任务结束时，应该考虑回调时刻的环境是否还支持业务处理。例如 Activity 的 onStop()函数已经执行，且在该函数中主动释放了资源，此时回调中如果不做判断就会空指针崩溃。
4. 在有强依赖 onAnimationEnd 回调的交互时，如动画播放完毕才能操作页面 ， onAnimationEnd 可能会 因各种异常没被回调 （参考：https://stackoverflow.com/questions/5474923/onanimationend-is-not-getting-called-onanimationstart-works-fine ），建议加上超时保护 或通过 postDelay 替 代onAnimationEnd。

当 View Animation 执行结束时，调用 View.clearAnimation()释放相关资源。

1. 在 AndroidManifest.xml 文件中将 android:allowbackup 属性设置为 false，防止 adb backup 导出数据。