

Plusub-Android 框架

使用说明

文件状态： [] 草稿 [√] 正式发布 [] 正在修改	文件标识：	Plusub-REGULER-RD-UR
	当前版本：	1.0
	作 者：	www.blakequ.com
	完成日期：	2015/5/8

版本/状态	作者	参与者	起止日期	备注
正式版	blakequ		2015/5/8	

目录

Plusub-Android 框架.....	1
使用说明.....	1
一 框架概述.....	3
1.1 概述	3
1.2 框架优劣.....	3
二 BasePlusubLib 框架介绍	4
2.1 框架结构.....	4
2.2 框架模块介绍.....	5
2.2.1 Com.plusub.lib	5
2.2.2 Com.plusub.lib.activity.....	5
2.2.3 Com.plusub.lib.adapter.....	5
2.2.4 Com.plusub.lib.service	6
2.2.5 Com.plusub.lib.task	6
2.2.6 Com.plusub.lib.util	6
2.2.7 Com.plusub.lib.bitmap.....	7
2.3 使用方法.....	7
2.3.1 AndroidManifest.xml 基本配置.....	7
2.3.2 继承 BaseApplication.....	8
2.3.3 实现 BaseRequestService	8
2.3.4 基本组件 Activity , Fragment , FragmentActivity 的替换.....	9
2.3.5 ListView 的适配器 BaseArrayListAdapter.....	9
2.3.6 JSON 解析注解	10
2.3.7 视图实例化注解.....	16
三 BasePlusubViewLib	17
3.1 框架结构.....	17
3.2 框架结构介绍.....	17
四 其他.....	22
4.1 关于命名.....	22
4.1.1 包命名.....	22
4.1.2 类命名.....	22
4.2 如何引用系统框架作为依赖库.....	23
4.3 Android Studio 引用依赖库.....	23
4.3.1 引用 BasePlusubLib 库	23
4.3.2 引用 BasePlusubViewLib 库 (包含 BasePlusubLib 库)	23

一 框架概述

1.1 概述

系统框架分为两个基础包：BasePlusLib 和 BasePlusViewLib。

- BasePlusLib：基本库内集成了 [ImageLoader](#)，[EventBus](#)，自定义的 activity 管理，域注解方式的视图初始化和域注解方式的 JSON 数据解析，还有网络数据访问，异步任务管理，及基本的工具类。
- BasePlusViewLib：视图库内集成了基本的视图类型，包括非常常用的下拉刷新组件，浮动视图组件，IDrawer 侧滑组件，App 升级服务等。

注意：其中 BasePlusViewLib 必须依赖库 BasePlusLib。

1.2 框架优劣

每个框架都有一定的优缺点，这需从具体项目进行分辨，下面就本框架的优缺点进行说明。

- 优点：实现简单基础快速上手开发，保证系统的基础稳定性，系统升级维护的灵活性以及代码整体的统一，保证团队协作。最大限度的减少开发的周期，保证项目顺利完成。
- 缺点：系统开发缺少一定的灵活性，性能有所降低。如网络请求使用系统框架是异步请求，JSON 解析采用注解方式等。

总的来说，只要掌握了框架的正确用法，开发必然事半功倍，故而每个学习的都需要掌握框架的正确用法以及编程规范。

二 BasePlusubLib 框架介绍

2.1 框架结构

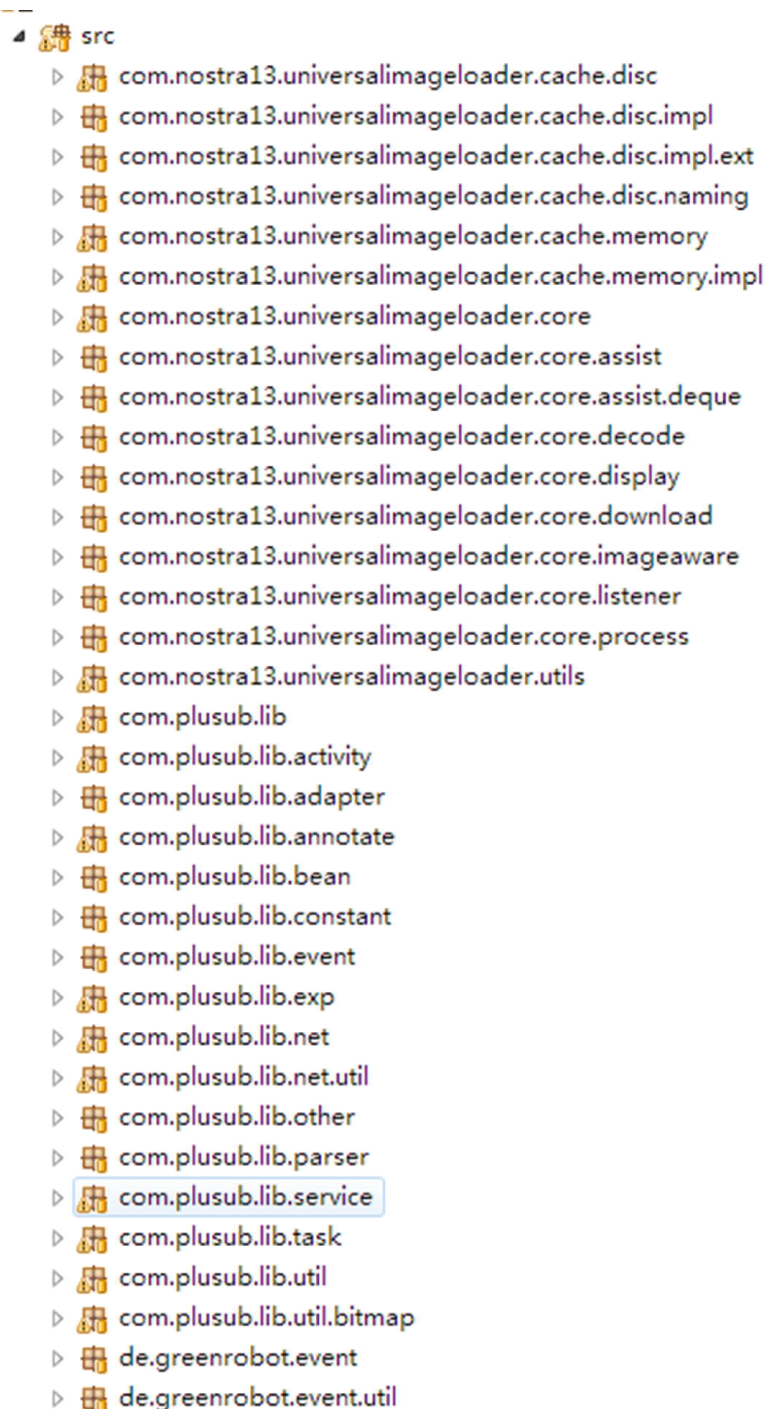


图 2-1 框架结构

如图所示，框架结构中功能模块划分通过包名就可以明确区分下来。

- `com.nostra13.universalimageloader`：这个包下是直接引用 `ImageLoader`，实现图片异步

加载，本地缓存，图片处理为一体的非常好的框架，这里没有进行修改，直接使用。

- de.greenrobot.event :是 EventBus ,是设计模式中的观察者模式(生产/消费者编程模型) 的优雅实现。对于事件监听和发布订阅模式 ,EventBus 是一个非常优雅和简单解决方案，我们不用创建复杂的类和接口层次结构。最大的好处是解耦，保证组件之间的耦合性，实现面向对象编程的一个原则：内紧外送。使用方式可以参看百度搜索。
- com.plusub.lib :是系统框架的核心。里面包含了对 Activity , Fragment , Service , 网络请求，数据缓存，日志系统，图片处理，数据解析，异常处理等的实现。

2.2 框架模块介绍

2.2.1 Com.plusub.lib

- BaseApplication : 为抽象类，继承了 Application。里面主要实现了基本系统信息获取，如版本号等，Activity 管理，退出 App 处理，屏幕锁定，设置 session，未捕获异常处理等。重要方法：
exitApp(Context) : 退出 App 时必须调用，会清除缓存，结束所有堆栈内 activity，完整退出应用，在退出 app 时调用即可。
doUncaughtException : 这个方法是可实现方法，当应用由于未捕获异常异常退出时，应该做的操作，如重启应用或者上传异常信息到服务器等。
getImageCache() : 获取 ImageLoader 的图片载入入口。
setSessionId(String) : 设置 sessionId
setLockScreen(boolean) : 设置是否锁定屏幕为竖屏（默认锁定）
- BaseHelper : 基本帮助类，主要实现了日志打印保存管理，系统日志显示控制。

2.2.2 Com.plusub.lib.activity

- BaseActivity : 继承自 Activity。实现了 Activity 的进一步封装，需要实现的方法是：initData() , initEvent() , setRootView();系统自动调用的顺序是：setRootView, initData, initEvent ; 功能是设置视图，初始化数据，初始化事件。
- BaseFragment : 继承自 Fragment。实现 Fragment 进一步封装。
- BaseFragmentActivity : 继承自 FragmentActivity。实现 FragmentActivity 进一步封装。
- SimpleBrowserActivity : 简单浏览器实现抽象类，需要继承实现，开发者必须首先实现 initWebView(WebView webView)方法。将 webView 返回。若要显示网页，可手动调用 webView.loadUrl(url);
- SimpleWebActivity : 可直接使用的浏览器实现类，使用方法非常简单

```
Intent intent = new Intent(this, SimpleWebActivity.class)
intent.putExtra(SimpleWebActivity.URL, "www.baidu.com");
startActivity(intent);
```

2.2.3 Com.plusub.lib.adapter

- BaseArrayListAdapter : 是继承实现了 BaseAdapter，实现了基本的方法，继承实现时，

只需要实现 getView 方法，填充数据的时候调用 add,refresh,delete 等方法，这些方法会自动调用刷新，不需要自己去刷新界面。

- FragmentPagerAdapter：是 ViewPager+Fragment 实现 Tab 切换的适配器。
- FragmentTabAdapter：是 Fragment 实现 Tab 切换的适配器，可以不使用 ViewPager。

2.2.4 Com.plusub.lib.service

- BaseRequestService：是网络请求抽象类，实现了网络请求从发出请求到将返回结果解析为实体的整个过程。使用时必须实现 addTaskToMap 方法，将请求 url，解析实体都配置到 map 中即可。checkJsonError 方法是判断返回的 json 数据是否正确的函数，如果返回 false 则解析结果为 null，在 TaskMessage 的字段 message 会携带具体错误信息。doErrorOperator 是实现错误处理的函数，如果需要对某个错误进行处理，就需要实现该方法。

2.2.5 Com.plusub.lib.task

- DataRefreshTask：是实现了数据回调接口，如果某个类需要接受返回的 URL 请求数据，必须实现该接口。系统会自动回调 onRefresh 方法。

2.2.6 Com.plusub.lib.util

内部是基本的工具类：

- ArrayUtils：简单数组 V[]类型的工具类，可以快速获取数据的最后和第一个值。
- CacheManager：实现 Http 请求，和 ImageLoader 的缓存管理
- Cn2Spell：实现将汉字转换为字母，转换第一个字母。
- CountTimer：倒计时，只需要实现 OnTimerListener 即可，设定好总的倒计时时间和倒计时的间隔时间，在监听器就可以在每个间隔和结束时回调 onTick 和 onFinish 方法。
- DensityUtils：获取系统屏幕大小密度的基本工具类。
- DownloadManagerPro：对系统下载器 DownloadManager 的进一步封装，和 DownloadManager 结合使用，可以获取下载的具体信息。
- DownloadUtils：带有进度和完成通知回调的独立线程下载工具类。
- FileIntentUtils：文件打开选择器，会自动判定文件类型，并给予合适的选择器并返回。
- FileUtils：文件操作工具类，如 Sdcard 是否可以，文件的读写增删操作。
- HttpRequest：独立的 http 请求工具类，可实现 http 请求。
- ImageSpan：图片资源映射，实现图片资源解析
- ImageUtils：图片工具类，实现图片放大缩小，图片转换，读取图片，图片处理等。
- ListUtils：List 工具类，实现获取 List 数据的位置，大小，空判断，转换等。
- LogUtils：日志工具类，实现日志保存，分级显示等。
- MapUtils：Map 工具类，可以将字符串转换为 map，通过 value 获取 key，map 转换为 json 等。
- MD5Encryptor：MD5 加密
- MediaSpan：实现可点击播放的 ClickSpan

- MediaUtils：获取音频的时间长度。
- NetStateUtils：获取网络状态，网络类型等。
- ObjectUtils：可以时间基本简单类型 int，long 数组与对象 Integer,Long 数组的转换。
- PreferencesUtils：SharedPreferences 的工具类。可以实现各种类型数据的快速保存。
- RandomUtils：获取各种随机数。
- RegexUtils：基本正则表达式判断，如手机号码等。
- ResourceUtils：资源工具类，可以获取 assets,raw 的文件。
- ScreenObserver：屏幕状态监听器，监听屏幕关闭与打开。
- SizeUtils：字节，MB 等转换。
- StringUtils：将 String 转换为各种类型。
- SystemTool：获取系统基本信息工具类。
- TimeUtils：时间转换工具类，格式化时间。
- ToastUtils：取代系统 toast 显示工具类。
- ValidateUtils：合法性验证，包括手机号，身份证等检测。
- ViewUtils：视图工具类，可以获取屏幕截图，创建快捷键。

2.2.7 Com.pluslib.bitmap

Bitmap 的创建、释放、管理的工具类。

2.3 使用方法

在使用框架的时候，必须阅读此处，严格按照使用方法使用。

2.3.1 AndroidManifest.xml 基本配置

要使用框架，必须在配置文件中配置基本的权限等。

```
<!--屏幕适配配置-->
<supports-screens
    android:anyDensity="true"
    android:largeScreens="true"
    android:normalScreens="true"
    android:smallScreens="true"
    android:xlargeScreens="true" />

<!--基本权限配置-->
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RESTART_PACKAGES" />
<uses-permission android:name="android.permission.GET_TASKS" />
```



```

<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.ACCESS_DOWNLOAD_MANAGER" />
<uses-permission android:name="android.permission.DOWNLOAD_WITHOUT_NOTIFICATION" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

```

2.3.2 继承 BaseApplication

必须自己实现 MainApplication 并继承自 BaseApplication ,因为里面涉及框架需要自动调用的方法。

1.需要实现的方法：

- clearApp()：当系统退出时会自动调用该方法，在内部可以实现缓存释放，状态清除等操作。
- doUncaughtException：选择性实现，会捕获未拦截异常，当应用异常退出时是上传异常信息还是重启应用需要在内部实现。

2.需要常使用的方法：

- getImageCache()：是获取 ImageLoader，使用方法非常简单，可以是网络图片
MainApplication.getInstance().getImageCache().displayImage("图片url", ImageView组件, AppConfig.getCommonImageViewOptionsNotScale());
AppConfig 的配置参考 BasePlusLibExample。
- setSessionId(String)：设置 sessionId
- setLockScreen(boolean)：设置是否锁定屏幕为竖屏（默认锁定）

3.注意：写完了，需要主动在 AndroidManifest.xml 中注册

```

<application
    android:name=".MainApplication"
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">

```

2.3.3 实现 BaseRequestService

这是是为了实现网络请求，在内部需要配置请求的 URL，返回 JSON 数据对应的实体信息，请求方式（get 或 post），请求缓存。

1.需要实现的方法：

- addTaskToMap：增加 task 到 map 中，使用方法如下
RequestMap.put(RequestTaskConstant.TASK_DOUBAN,
new RequestEntity(PATH+"/v2/book/search?", BookEntity.class, RequestType.GET));
或者


```
RequestMap.put(RequestTaskConstant.TASK_DOUBAN,  
new RequestEntity(PATH+"/v2/book/search?", BookEntity.class, RequestType.GET), false);
```

第一个参数是请求 ID，这个**必须唯一**

第二个参数是请求实体，里面封装了请求 url，json 数据返回的实体对象

第三个参数是请求方式 GET 或 POST

第四个参数是可选，是否缓存 url

➤ checkJsonError：必须实现

检查 JSON 错误，对于返回的 JSON 字符串的检查，使用方法见 demo

2.其他方法

➤ doErrorOperator：错误处理，非必须实现

如果在请求的过程中返回了异常，会调用这个方法，然后在实现的时候，可以针对具体异常作出相应处理，如 session 超时，则需要重新登陆获取 session，使用方法见 demo。

➤ addNewTask：添加新的请求任务到任务列表

➤ removeTask：从任务列表移除任务。

2.3.4 基本组件 Activity，Fragment，FragmentActivity 的替换

在使用基本组件 Activity，Fragment，FragmentActivity 的时候，不要使用这些组件，都需要替换为 BaseActivity，BaseFragment，BaseFragmentActivity，并实现其中的基本方法，initView()，initData()，initEvent()，setRootView()，inflaterView 这些方法。

调用顺序是 setRootView()，inflaterView，initView()，initData()，initEvent()

➤ initView 需要在里面实现基本的视图初始化

➤ initData 需要在里面实现基本的数据初始化工作

➤ initEvent 需要实现基本的事件，如点击，下拉刷新等

2.3.5 ListView 的适配器 BaseArrayListAdapter

对于简单列表的适配器，不需要实现系统的 BaseAdapter，可以使用 BaseArrayListAdapter 进行替换使用，只需要实现 getView 方法即可，而且使用方式是固定的。里面提供了 add,delete,clear 等数据操作的方法，每个方法被调用时会自动刷新界面，不用调用者去手动刷新界面。BaseAdapter 中的其他方法如 getItem 都已经被实现，可以直接调用，基本使用方法示例：

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    // TODO Auto-generated method stub
    Holder holder = null;
    if (convertView != null) {
        holder = (Holder) convertView.getTag();
    } else {
        convertView = inflater.inflate(R.layout.listitem_book_item, null);
        holder = new Holder(convertView);
        convertView.setTag(holder);
    }
    setData(holder, position);
    return convertView;
}

private void setData(Holder holder, int position) {
    // TODO Auto-generated method stub
    BookEntity book = (BookEntity) getItem(position);
    if (book != null) {
        MainApplication.getInstance().getImageCache().displayImage(book.getImage(), holder.mIvHeader,
            AppConfig.getCommonImageViewOptionsNotScale());
        holder.mTvContent.setText(book.getAuthorInfo());
        holder.mTvDate.setText(book.getPubData());
        holder.mTvTitle.setText(book.getTitle());
    }
}

private class Holder {
    @BindView(id = R.id.list_iv_head)
    ImageView mIvHeader;
    @BindView(id = R.id.list_tv_title)
    TextView mTvTitle;
    @BindView(id = R.id.list_tv_date)
    TextView mTvDate;
    @BindView(id = R.id.list_tv_content)
    TextView mTvContent;

    public Holder(View view) {
        AnnotateUtil.initBindView(this, view);
    }
}
```

如图所示，实现 getView 方法即可，在实现的时候使用了视图缓存，可以百度理解其原理。使用 Holder 去管理视图实例，在 Holder 里面使用了注解的方式去实例化组件，后面会讲解具体用法。

需要替换的地方：列的布局文件 layout 和 holder 中列的具体组件以及 setData 填充数据。

2.3.6 JSON 解析注解

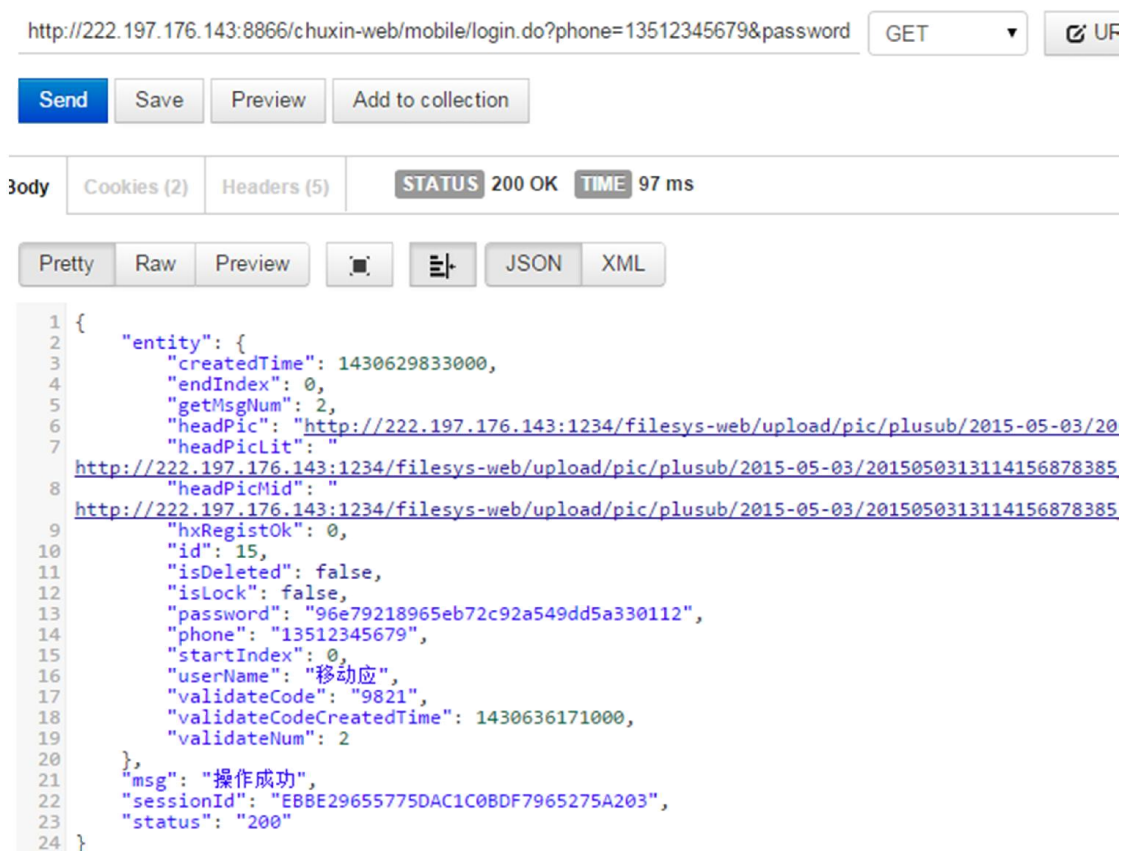
JSON 解析注解实现了 JSON 字符串到实体对象的自动解析，将 JSON 字符串解析为自定义实体的过程，实现了编程方式的简化和快速开发，其关键是如何去注解并建立对应于 JSON 数据的实体类。

2.3.6.1 对应 JSON 数据实体的建立方法

下面按照步骤进行讲解：

➤ 获取 JSON 字符串并进行格式化

这里需要使用的到的工具可以网上自己下载，本地格式化工具 JSONView，或[网站](#)。最好的方式是使用 Chrome 浏览器的插件 [PostMan](#)，或者直接的插件 [Post-handle](#)。获取到数据之后会直接格式化。



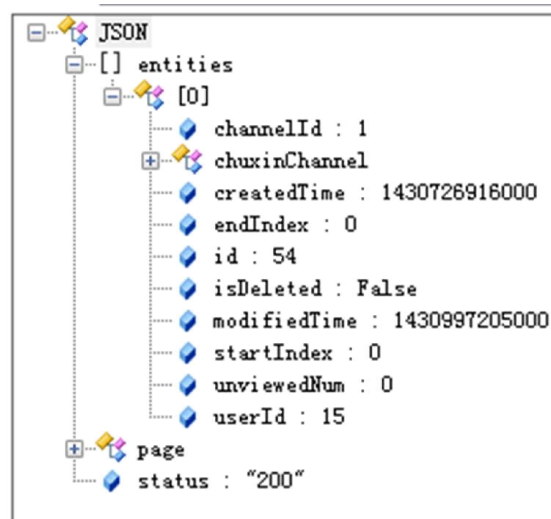
这是请求的数据并自动进行了格式化。

➤ 观察 JSON 数据的结构

JSON 的数据结构都是以键值对的方式进行展示，其中键都是 string 类型的，指明其含义，而值有几种类型：对象（花括号中{}）、数组（方括号中[]）、数字、字符串（引号中）、逻辑值（true or false）以及 null。如下图所示：

```
1 {
2   "entities": [
3     {
4       "channelId": 1,
5       "chuxinChannel": {
6         "content": "eerr ",
7         "descript": "美丽的成都，来了就不想走的城市",
8         "endIndex": 0,
9         "iconUrl": "http://222.197.176.143:1234/file",
10        "iconUrlLit": "http://222.197.176.143:1234/f",
11        "iconUrlMid": "http://222.197.176.143:1234/f",
12        "id": 1,
13        "isDeleted": false,
14        "loveNum": 1,
15        "modifiedTime": 1428722725000,
16        "name": "你好成都",
17        "picture": "http://222.197.176.143:1234/file",
18        "pictureLit": "http://222.197.176.143:1234/f",
19        "pictureMid": "http://222.197.176.143:1234/f",
20        "recordNum": 1,
21        "recordUserHeadPic": "http://222.197.176.143",
22        "recordUserId": 2,
23        "recordUserName": "小初各个",
24        "startIndex": 0,
25        "userId": 1,
26        "userName": "小初妹妹",
27        "viewNum": 1
28      },
29      "createdTime": 1430726916000,
30      "endIndex": 0,
31      "id": 54,
32      "isDeleted": false,
33      "modifiedTime": 1430997205000,
34      "startIndex": 0,
35      "unviewedNum": 0,
36      "userId": 15
37    },
38  ],
39  "page": {
40    "currentPage": 1,
41    "nextPage": 1,
42    "pageSize": 20,
43    "previousPage": 1,
44    "totalPages": 1,
45    "totalRows": 1
46  },
47  "status": "200"
48 }
```

该图中包含了上述的所有类型，其中键为 `entitys` 的值为数组类型（大小为 1）；键为 `chuxinChannel` 和 `page` 的值都是一个对象；键为 `content` 的值为一个字符串；键为 `endIndex` 的值为数字；键为 `isDeleted` 的值为逻辑值。从而我们能清晰的看清楚其 json 数据的整个结构，以直观的方式查看如下：



JSON 数据中分为了 3 层，每一个列的虚线就代表了一层

第一层：键为 entities 的数组；page 为键的对象；键为 status 的字符串

第二层：entities 第 0 个数组，里面有很多数据

第三层：键为 chuxinChannel 的对象

➤ 建立对应的实体对象

实体的结构搞清楚了，那么就要按照上图的结构进行建立对应的实体对象了。

第一层：

```
public class TestEntity {  
  
    @JsonParserField(praserKey="entities", isList=true, classType=ChannelEntity.class)  
    private List<ChannelEntity> channelList;  
    @JsonParserField(praserKey="page")  
    private PageEntity page;  
  
    private String status;  
  
    public List<ChannelEntity> getChannelList() {  
        return channelList;  
    }  
    public void setChannelList(List<ChannelEntity> channelList) {  
        this.channelList = channelList;  
    }  
}
```

对于第一层，三个参数分别对应了第一层的三个 JSON 键（entities,page,status）。如果变量的名字和 JSON 的键是一致的（都是 status），可以不用注解，系统会默认 status 为键。

对于 entities 是列表形式，注解需要明确的指明是 list 并且指明 list 内的实体类型 isList 和 classType。

第二层：

```
public class ChannelEntity {  
  
    @JsonParserField(defaultValue="-1")  
    private int channelId;  
    @JsonParserField(praserKey="chuxinChannel")  
    private ChannelDetailEntity channel; //对应于键chuxinChannel的对象  
    @JsonParserField(defaultValue="0")  
    private long createTime;  
    private int id;  
    @JsonParserField(defaultValue="0")  
    private boolean isDeleted;  
    @JsonParserField(defaultValue="0", praserKey="unviewedNum")  
    private int num; //对应于键unviewedNum  
    private int userId;  
}
```

第二层既有实体也有简单的字符串等基本类型。

第三层：

```
@JsonParserClass(parserRoot="chuxinChannel")  
public class ChannelDetailEntity {  
  
    private String content;  
    private String descript;  
    @JsonParserField(praserKey="iconUrl")  
    private int picUrl;  
}
```

第三层是最内层的实体对象，实体的键为 chuxinChannel,这个需要在类名上注解，使用 JsonParserClass。

➤ 对实体对象进行注解

类注解：只是注解在类上的，如上的 ChannelDetailEntiy 上注解，其中 parserRoot 是必选项，其他都是可选项。


```

public @interface JsonParserClass {

    /**
     * 是否解析对象为数组，默认false
     */
    boolean isList() default false;

    /**
     * 解析的对象所在JSON字符串的根键值，如 "entities": [{...}{...}]，根键值为entities
     * <br>如果没有则不设置
     */
    String parserRoot();

    /**
     * 是否有page信息，如果有则需要设置pageKeyStr，表示其根键值
     */
    boolean isHasPage() default false;

    /**
     * 如果有page信息，则page实体变量的名字
     */
    String pageFieldStr() default "";
}

```

其中 isHasPage 和 pageFieldStr 很少使用可不关注。

参数注解：参数注解是实体对象中具体参数的注解，所有都是可选

```

public @interface JsonParserField {

    /**
     * 对应待解析JSON中的字段的名，默认为 ""
     */
    String praserKey() default "";

    /**
     * 如果没有找到对于的解析key，则设置为该默认值，默认为 ""
     */
    String defaultValue() default "";

    /**
     * 是否解析为数组，如果为数组，<b>则必须设置属性classType</b>，设置数组的实体对象
     * <br><b>注意：如果实体为数组对象，必须是List类型
     */
    boolean isList() default false;

    /**
     * 如果有实体对象，对象的类名
     */
    Class classType() default Object.class;
}

```

参数注解在第一层 TestEntity 中已经使用，按照注释理解使用即可。

➤ 自动生成 Get 和 Set 方法（必须实现）

右键点击—source—Generate Getters and Setters—全选点击完成即可。

2.3.7 视图实例化注解

- 在 Activity 中使用

```
@BindView(id = R.id.main_tab_frame)
private FrameLayout mFrameLayout;
@BindView(id = R.id.main_tab_layout_1, click = true)
private RelativeLayout mRITab1;
@BindView(id = R.id.main_tab_item_label1)
private TextView mTab1Label;
@BindView(id = R.id.main_tab_item_badge1)
private ImageView mTab1Badge;
```

上面就是基本的使用方法，不需要其他的任何配置，其中 id 就是 View 的 id；click 是 View 是否可点击，如果设置为 true 则会主动注册监听器。然后重写（框架已经自动实现了 OnClickListener）Activity 或 Fragment 的 onClick(View v)方法即可：

```
@Override
public void onClick(View v) {
    // TODO Auto-generated method stub
    switch(v.getId()){
        case R.id.main_tab_layout_1:
            setTabSelectedStates(0);
            break;
    }
```

- 在 Adapter 及其他地方使用

```
private class Holder{
    @BindView(id = R.id.list_iv_head)
    ImageView mIvHeader;
    @BindView(id = R.id.list_tv_title)
    TextView mTvTitle;
    @BindView(id = R.id.list_tv_date)
    TextView mTvDate;
    @BindView(id = R.id.list_tv_content)
    TextView mTvContent;

    public Holder(View view){
        AnnotateUtil.initBindView(this, view);
    }
}
```

唯一区别是在构造函数中主动调用 AnnotateUtil.initBindView 方法，其中的 view 是布局文件。

BasePlusubViewLib

3.1 框架结构

```
src
├── com.ikimuhendis.drawer
├── com.plusub.lib.service
├── com.plusub.lib.view
├── com.plusub.lib.view.floatingview
├── com.plusub.lib.view.refresh
└── de.keyboardsurfer.android.widget.crouton
```

里面定义了许多经常使用的基本视图组件。

3.2 框架结构介绍

- Com.plusub.lib.service
里面只有 AppUpgradeService,是用来 App 升级并安装的服务,使用方式可看注释,只需要传入 app 的 url 即可实现 app 的升级与安装。
注意：使用时必须在 manifest 进行注册。下载完成后需要手动停止服务。
- Com.ikimuhendis.drawer
是侧边栏组件实现,使用方法见 Demo



- Com.plusub.lib.view.floatingview
浮动按钮组件,使用见 Demo



- Com.plusub.lib.view.refresh
下拉刷新，上拉加载组件，使用方法见 Demo



可以实现的功能有：下拉刷新，到底部自动加载或手动加载，加载结束后显示结束，超出屏幕显示更多按钮。

- Com.plusub.lib.view
里面有许多常用组件，设计 ListView，Dialog，PopupWindow，TextView，ScrollView，ImageView，ViewPager 等。

```
com.pluslib.view
├── BadgeView.java 12230 15-5-7 下午3:52 yy
├── BaseDialog.java 12230 15-5-7 下午3:52 yy
├── BasePopupWindow.java 12230 15-5-7 下午3:52 yy
├── CircleImageView.java 12230 15-5-7 下午3:52 yy
├── CollapsibleTextView.java 12230 15-5-7 下午3:52 yy
├── ExpandableTextView.java 12230 15-5-7 下午3:52 yy
├── HorizontalListView.java 12230 15-5-7 下午3:52 yy
├── ListViewForScrollView.java 12230 15-5-7 下午3:52 yy
├── PaddingCheckBox.java 12230 15-5-7 下午3:52 yy
├── PPRefreshListener.java 12230 15-5-7 下午3:52 yy
├── PPScrollView.java 12230 15-5-7 下午3:52 yy
├── PPViewPager.java 12230 15-5-7 下午3:52 yy
├── RotateImageView.java 12230 15-5-7 下午3:52 yy
├── RotateLoadingDialog.java 12230 15-5-7 下午3:52 yy
├── RoundImageView.java 12230 15-5-7 下午3:52 yy
├── RoundProgressBar.java 12230 15-5-7 下午3:52 yy
├── ScaleImageView.java 12230 15-5-7 下午3:52 yy
├── ScrollingTextView.java 12230 15-5-7 下午3:52 yy
├── ScrollViewPager.java 12230 15-5-7 下午3:52 yy
├── ViewInjectUtils.java 12232 15-5-7 下午4:18 yy
├── WrapContentHeightViewPager.java 12230 15-5-7 下午3:52 yy
└── ZoomImageView.java 12230 15-5-7 下午3:52 yy
```

其中最常用的：

1.ViewInjectUtils 中的显示 Toast 用于替换系统的 Toast

ViewInjectUtils.showCustomToast 显示 Toast

ViewInjectUtils.showLoadingDialog 显示加载进度 toast

ViewInjectUtils.showLoadingDialogNotCancel 显示无法点击消失的 Toast

ViewInjectUtils.dismissLoadingDialog 进度 Toast 消失

2.BaseDialog 和 BasePopupWindow



实现效果如上图所示,使用方法如下：

显示单个提示：

```
private void showSimpleDialog(){
    AlertDialog mDialogAttention = AlertDialog.Builder(getActivity(),
        R.string.notice).setTitle(
        getString(R.string.msg_delete_confirm), //content
        getString(R.string.cancel), //left button
        new OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                // TODO Auto-generated method stub
                dialog.dismiss();
            }

        },
        getString(R.string.confirm), //right button
        new OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                // TODO Auto-generated method stub
                //do something
                dialog.dismiss();
            }

        }
    );
    // mDialogAttention.setTitleLineVisibility(View.GONE);
    // mDialogAttention.setTitleTextColor(R.color.text_content);
    mDialogAttention.show();
}
```

自定义 List 视图 dialog

```
private void showViewDialog() {
    final SimpleStringAdapter adapter = new SimpleStringAdapter(getActivity());
    String[] array = getResources().getStringArray(R.array.msg_long_click);
    for (int i = 0; i < array.length; i++) {
        StringEntity se = new StringEntity(array[i]);
        adapter.add(se);
    }

    SimpleListDialog mSimpleListDialog = new SimpleListDialog(getActivity());
    mSimpleListDialog.setTitle(getString(R.string.notice));
    mSimpleListDialog.setTitleLineVisibility(View.GONE);
    mSimpleListDialog.setAdapter(adapter);
    mSimpleListDialog.setOnSimpleListItemClickListener(new
    onSimpleListItemClickListener() {
        @Override
        public void onItemClicked(int position) {
            // TODO Auto-generated method stub
            switch (position) {
                case 0:
                    //do something
                    break;
            }
        }
    });
}
```

```
                default:
                    break;
            }
        }
    });
    mSimpleListDialog.show();
}
```

3.其他

BadgeView：组件提示图标

BaseDialog：基本对话框

BasePopupWindow：基本的弹出框

CircleImageView：圆形 ImageView

HorizontalListView：水平 ListView

PaddingCheckBox：修复 checkbox 的 paddingLeft 属性会随着分辨率的不同，空间大小不同的 CheckBox

PPViewPager：不需要每次滑动都加载视图的 ViewPager

ScrollViewPager：可屏蔽滑动事件的 ViewPager

ViewInjectUtils：注入式组件，弹出框

WrapContentHeightViewPager：可自适应高度的 ViewPager

四 其他

4.1 关于命名

4.1.1 包命名

必须以项目或公司的网址作为基础包名+模块+功能

如 com.plusub 是公司网址作为基础包名（注意是反的）











Com.plusub.activity 中 activity 是模块—界面逻辑模块

Com.plusub.db 中 db 是数据库模块

Com.plusub.db.user 中的 user 是 db 下的用户数据库模块

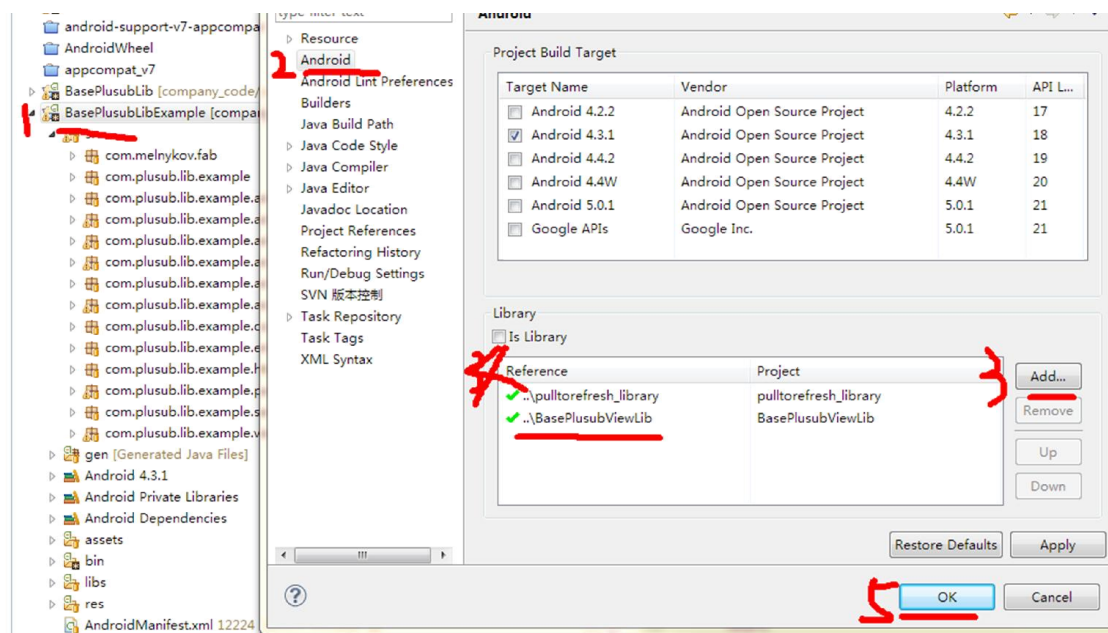
4.1.2 类命名

- 如果是 Activity 或 Fragment 或 Service 则必须以 Activity 或 Fragment 或 Service 开头，后面跟功能名如

- ▷  com.plusub.rentlandapp
- ▲  com.plusub.rentlandapp.activity
 - ▷  MainActivity.java 5540 15-5-7 下午8:25 qh
- ▲  com.plusub.rentlandapp.activity.center
 - ▷  FragmentPersonCenter.java 5540 15-5-7 下午8:25 qh
- ▲  com.plusub.rentlandapp.activity.exchange
 - ▷  FragmentExchange.java 5540 15-5-7 下午8:25 qh
- ▲  com.plusub.rentlandapp.activity.home
 - ▷  FragmentHome.java 5540 15-5-7 下午8:25 qh
- ▷  com.plusub.rentlandapp.activity.landmanager

- 实体类型必须以 Entity 结束
- 其他按照编码规范进行命名

4.2 如何引用系统框架作为依赖库



1. 右键单击工程
2. 选择 android
3. 点击 add , 选择 BasePlusViewLib
4. 添加进入了 Library
5. 点击 ok 即完成依赖库的添加。

4.3 Android Studio 引用依赖库

4.3.1 引用 BasePlusLib 库

1. Maven

```
<dependency>
  <groupId>com.plusub.lib</groupId>
  <artifactId>PlusubBaseLib</artifactId>
  <version>1.0.2</version>
  <type>pom</type>
</dependency>
```
2. Gradle

```
compile 'com.plusub.lib:PlusubBaseLib:1.0.2'
```

4.3.2 引用 BasePlusViewLib 库 (包含 BasePlusLib 库)

1. Maven

```
<dependency>
  <groupId>com.plusub.lib</groupId>
  <artifactId>PlusubBaseViewLib</artifactId>
  <version>1.0.2</version>
  <type>pom</type>
</dependency>
```
2. Gradle

compile 'com.plusub.lib:PlusubBaseViewLib:1.0.2'