LoonAndroid

---- 2015 年 12 月 16 日

作者：潘老师

写在文档之前：此文档根据当前编写日期来编写，后期源程序更改将持续跟进。此文档仅限基友内部交流使用，如有其他人拿做其它商业用途产生其它影响，本人概不负责。程序如有雷同，纯属虚构。

```
/**
* 认准商标 谨防假冒
* ---------------------------------------------- <br>
* ooo0.............. <br>
* (....) ... 0ooo... <br>
* .\..(.....(.....)....... <br>
* ..\_)..... )../........ <br>
* .......... (_/.......... <br>
* 微信 gdpancheng <br>
* ---------------------------------------------- <br>
* @author gdpancheng@gmail.com 2014 年 12 月 10 日 上午 10:37:24
*/
```

一 简介

Loonandroid 是一个注解框架，不涉及任何 UI 效果，目的是一个功能一个方法，以方法为最小颗粒度对功能进行拆解。把功能傻瓜化，简单化，去掉重复性的代码，隐藏复杂的实现。以便团队合作或者后期修改变得简单。说框架是夸大了，主要是因为我比较喜欢偷懒，对于一个码农来说，能够偷懒，并且在不影响项目质量的情况下，是不容易的。很多朋友看到注解就就要吐槽，会影响性能什么的。注解，特别是运行中的注解，无法避免的会影响性能。

通过注解自动注入 因为反射会让程序变慢 50~100 毫秒左右，基本感觉不出来.硬件性能好的手机 基本上没影响

这个是否会影响用户的体验感，经过测试无需太大的担心。我是做外包的，初衷是在不影响项目质量的前提下减少我的工作量，而且 BUG 其他人改起来相对比较容易，本工具专属外包码农，如果你想做精细，很在意性能数据，请看看就好。本人从 08 年开始做开发，到现在什么都会点，java,javaWeb,php,android,notejs，比较懒，用到什么才学什么，杂而不精，觉得写的比较烂，大家路过就好。

特别声明：版本 3x 框架还未在实际生产中验证，目前只拥有 demo 和我改造的一个项目，请小心谨慎，如有使用，请告知我，我会跟踪相应 BUG 以及性能，以做到及时修复。

二 特色及功能

2.1 基本功能

- **InLayer 注解**
- **InPlayer 注解**
- **Activity 生命周期注解**
- **InView 注解**
- **InSource 注解**
- **InAll 注解**
- **后台进程注解**
- **方法点击事件注解**
- **基类注解**
- **自动 Fragment 注解**
- **手动 Fragment 注解**

2.2 适配器功能

- **无适配器**
- **无参 baseAdapter**
- **自定义一 adapter**
- **自定义二 adapter**
- **自动绑定一 adapter**
- **自动绑定二 adapter**
- **通用适配器**

2.3 功能简化

- **网络请求模块**
- **输入验证**
- **跨进程通讯**
- **Json 格式化类**
- **倒计时类**

2.4 下拉刷新

- **Listview**

- Grid
- 横向 Scrollview
- 纵向 Scrollview
- 横向 ViewPage
- 纵向 ViewPage
- WebView

2.5 模块类

- 自定义模块 XML 中使用
- 自定义模块变量使用

2.6 组件类

- 获取图片组件
- 登录组件

四 使用步骤

在项目的 Application 中进行初始化

```
public class App extends Application {
```

```
        @Override
        public void onCreate() {
            app = this;
            Ioc.getIoc().init(this);
            super.onCreate();
            }
    }
```

在 assets 目录下面 mvc.properties 的配置设置如下

```
#-----------------------------------------框架基础配置----------------------------
#配置当前屏幕基于哪个分辨率开发 框架里面所有缩放比例全部来源于此 默认 480 800
standard_w=720
standard_h=1280
#-----------------------------------------设置只允许加载到框架中的包名--------
#如果不设置，那么默认遍历 Manifest 中的 package，多个可以以逗号隔开
permit=com.android.demo,com.loonandroid.pc.plug
#-----------------------------------------设置不允许解析的包名-----------------
#如果不设置，那么默认遍历 Manifest 中的 package，多个可以以逗号隔开
limit=com.example.loonandroid2.R
```

五 功能详解

5.1 经典功能（不断增加中）

5.1.1 从摄像头或者相册获取图片

如下：在 application 先注册，然后只需要 implements PluginPhoto 接口,就可以使用组件了
从相册获取照片 photo();如果有额外参数 传入 PhotoConfig 即可
从相机获取照片 camera();如果有额外参数 传入 PhotoConfig 即可
callBack 中可以获得返回的照片和 sd 卡绝对路径

```
@InLayer(R.layout.activity_getphoto)
public abstract class GetPhotoActivity extends Activity implements PluginPhoto {

    @InAll
    Views test;
```

```
    class Views {
        ImageView iv_photo;
        @InBinder(listener = OnClick.class, method = "click")
        Button bt_photo, bt_camera;
    }

    private void click(View v) {
        switch (v.getId()) {
        case R.id.bt_photo:
            //从相册获取图片
            PhotoConfig config = new PhotoConfig();
            config.aspectX = 1;
            config.aspectY = 2;
            config.outputX = 200;
            config.outputY = 400;
            photo(config);
            break;
        case R.id.bt_camera:
            //从相机获取图片
            camera();
            break;
        }
    }

    @Override
    public void callBack(Object... args) {
        Toast.makeText(this, "图片路径："+args[1], Toast.LENGTH_SHORT).show();
        test.iv_photo.setImageBitmap((Bitmap)args[0]);
    }
}
```

5.1.2 登录模块

如下：在 application 先注册，然后只需要 implements PluginLogin 接口,就可以使用组件了
首先在 i 方法中对 config 进行初始化，设置 ID，无需对输入框进行验证，框架会自动验证然后 onValiResult 会自动告诉你验证结果
你只需要在 onValiResult 判断后网络请求即可 ,网络请求成功以后 只要调用 save()方法则自动保存用户名密码，只要调用 getSave()即可获得之前保存的用户名密码

```
@InLayer(R.layout.activity_login)
```

```
public abstract class LoginActivity extends Activity implements PluginLogin{

    @Override
    public void i(LoginConfig config) {
        config.init(R.id.ed_number, R.id.ed_password, R.id.ed_submit,
R.id.ed_remember);
    }

    /**
     * 当点击登陆按钮，会自动获取输入框内的用户名和密码，对其进行验证
     */
    @Override
    public void onValiResult(View view) {
        if (view == null) {
            //验证通过
            App.app.http.u(this).login("aaa", "bbb");
        }else{
            Toast.makeText(this, "账号密码不能为空", Toast.LENGTH_SHORT).show();
        }
    }

    @InHttp(HttpUrl.LOGIN_KEY)
    public void result(ResponseEntity entity){
        if (entity.getStatus() == FastHttp.result_net_err) {
            Toast.makeText(this, "网络请求失败", Toast.LENGTH_SHORT).show();
            return;
        }
        if (entity.getContentAsString()==null||entity.getContentAsString().length()==0)
        {
            Toast.makeText(this, "网络请求失败", Toast.LENGTH_SHORT).show();
            return;
        }
        //解析返回的数据
        HashMap<String, Object> data =
Handler_Json.JsonToCollection(entity.getContentAsString());
        int status = Integer.valueOf(data.get("status").toString());
        if (status == 0) {
            Toast.makeText(this, data.get("data").toString(),
Toast.LENGTH_SHORT).show();
            return;
        }
```

```
        save();
        //-------------------------------------------------------
        //清除保存的数据
        clear("bbb");清除账号 bbb 的缓存
        clear();清除所有缓存
    }
}
```

5.2 基本功能

5.2.1 InLayer 注解

```
@InLayer(R.layout.activity_inlayer)
public class InLayerActivity extends Activity

替代了之前的 setContentView

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_inlayer);
}
```

5.2.2 InPlayer 注解

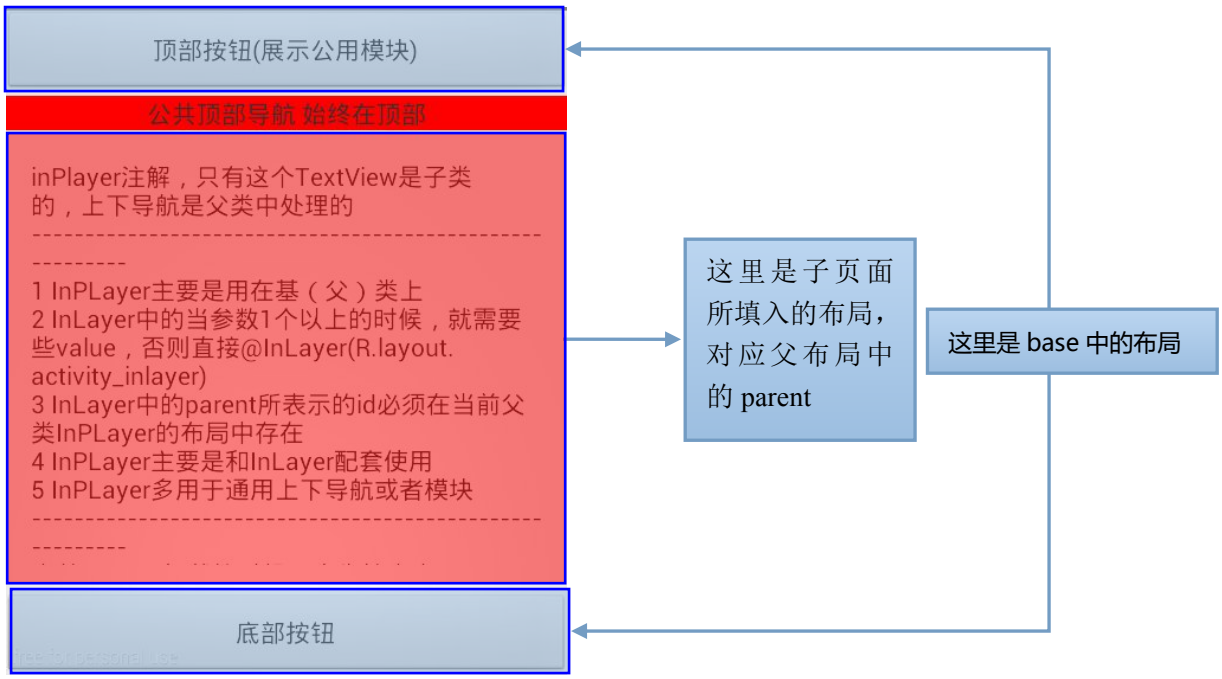
只能用于基类上的注解

```
@InPLayer(R.layout.activity_com)
public class CommonActivity extends Activity
```

在其子类中,需要设置 parent,当然也可以不设置

```
@InLayer(value = R.layout.activity_inplayer, parent = R.id.common)
public class InPLayerActivity extends CommonActivity
```

如果设置了 parent 那么在父类中 设置的 injectPlayer 中必须含有 ID 为 parent 的 View.
如下：



使用场景：可以用来实现上下导航，通用的布局，这样只需要继承 base 即可，base 中的布局以及对应的点击事件可以在 base 中统一初始化或者绑定相应事件，可以结合其他注解一起使用，具体可以参考 demo

5.2.3 Activity 生命周期注解

```
@Init
void creat() {}//init 代替了 OnCreate

@InNewIntent
void newIntent() {}// 2 InNewIntent 代替了 OnNewIntent

@InPause
void pause() {}//InPause 代替了 OnPause

@InRestart
void restart() {}InRestart 代替了 OnRestart

@InStart
void start() {}//InRestart 代替了 OnRestart
```

```
@InStop
void stop() {}//InStop 代替了 OnStop

@InResume
void resume() {}//InResume 代替了 OnResume

@InDestroy
void destroy() {}// InDestroy 代替了 OnDestroy
```

5.2.4 InView 注解

这里就不写代码了

```
@InView
TextView tv_first;// 变量名等于ID

@InView(R.id.tv_first)
TextView first;// 变量名不等于ID

@InView(binder = @InBinder(method = "click", listener = OnClick.class))
Button bt_onclick;// 绑定事件 其中listener必须继承OnListener 类别有单击 长按 触摸 列表点击 选中等

@InView(binder = @InBinder(method = "longClick", listener = OnLongClick.class))
Button bt_long;

@InView(binder = @InBinder(method = "check", listener = OnCompoundChecked.class))
CheckBox cb_select, cb_select1, cb_select2, cb_select3;

@InView(binder = @InBinder(method = "onCheckedChanged", listener = OnRadioChecked.class))
RadioGroup rg_radio;

@InView(binder = @InBinder(method = "itemSelected", listener = OnItemSelected.class))
Spinner sp_spinner;

@InView(binder = @InBinder(method = "clicks", listener = OnItemClick.class))
ListView lv_list;

@InView(binder = @InBinder(method = "onTouch", listener = OnTouch.class))
TextView tv_touch;
```

```
private void click(View v) {
    MakeToast("单击");
}

private void longClick(View v) {
    MakeToast("长按");
}

private void check(CompoundButton buttonView, boolean isChecked) {
    MakeToast("多选");
}

public void clicks(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
    MakeToast("列表单击");
}

private boolean onTouch(View v, MotionEvent event) {
    MakeToast("触摸");
    return true;
}

private void onCheckedChanged(RadioGroup group, int checkedId) {
    MakeToast("单选");
}

public void itemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
    MakeToast("列表选择");
}

public void noneItemSelected(AdapterView<?> arg0) {
    MakeToast("列表么有选择");
}
```

5.2.5 InSource 注解

InSource 可以自动取出 xml 中的资源进行填充

- **资源名称要和变量名一样**
- **支持 Drawable，String，String[]**

```
@InSource
Drawable ic_launcher;
@InSource
String hello_world;
```

5.2.6 InAll 注解

- **InAll 可以绑定 view**
- **InAll 可以绑定 view 的各种事件**
- **InAll 可以绑定 view 的事件有单击 长按 触摸 列表点击 选中等**
- **InAll 事件可以覆盖**
- **InAll 资源可以解析**

1 统一添加点击事件 如果没加@Ignore 标签

```
//除了 AdapterView 和图片字符串以外
// 全部会加上点击事件
@InAll(@InBinder(method = "click", listener = OnClick.class))
Views1 views1;

static class Views1 {
    TextView tv_first;// 变量名等于 ID
    Button bt_long;
    // 加了这个表示忽略 InAll 的点击事件
    @Ignore
    Button bt_onclick;
    CheckBox cb_select, cb_select1, cb_select2, cb_select3;
    RadioGroup rg_radio;
    Spinner sp_spinner;
    ListView lv_list;
    TextView tv_touch;
    Drawable ic_launcher;
    String hello_world;
}
```

2 单独定义即为每个 view 绑定单独的点击事件

```
@InAll
Views2 views2;

static class Views2 {
    TextView tv_first;// 变量名等于 ID
```

```
    @InBinder(method = "longClick", listener = OnLongClick.class)
    Button bt_long;
    @InBinder(method = "click", listener = OnClick.class)
    Button bt_onclick;
    @InBinder(method = "check", listener = OnCompoundChecked.class)
    CheckBox cb_select, cb_select1, cb_select2, cb_select3;
    @InBinder(method = "onCheckedChanged", listener = OnRadioChecked.class)
    RadioGroup rg_radio;
    @InBinder(method = "itemSelected", listener = OnItemSelected.class)
    Spinner sp_spinner;
    @InBinder(method = "clicks", listener = OnItemClick.class)
    ListView lv_list;
    @InBinder(method = "onTouch", listener = OnTouch.class)
    TextView tv_touch;
    Drawable ic_launcher;
    String hello_world;
}
```

3 @InBinder 覆盖@InAll 中的@InBinder

```
@InAll(@InBinder(method = "click", listener = OnClick.class))
Views3 views3;

static class Views3 {
    TextView tv_first;// 变量名等于 ID
    @InBinder(method = "longClick", listener = OnLongClick.class)
    Button bt_long;
    @InBinder(method = "click", listener = OnClick.class)
    Button bt_onclick;
    @InBinder(method = "check", listener = OnCompoundChecked.class)
    CheckBox cb_select, cb_select1, cb_select2, cb_select3;
    @InBinder(method = "onCheckedChanged", listener = OnRadioChecked.class)
    RadioGroup rg_radio;
    @InBinder(method = "itemSelected", listener = OnItemSelected.class)
    Spinner sp_spinner;
    @InBinder(method = "clicks", listener = OnItemClick.class)
    ListView lv_list;
    @InBinder(method = "onTouch", listener = OnTouch.class)
    TextView tv_touch;
    Drawable ic_launcher;
    String hello_world;
```

```
}
```

注意：内部类请用 Static 以降低内部类对当前 Activity 对象的占用导致内存无法释放
好处：1 我们使用 View 的时候，直接可以使用 views. IDE 会自动提示 2 减少 findbyId 代码量

5.2.7 后台进程注解

只要在方法上加了@InBack 其会独立运行在新的线程中

```
@InLayer(R.layout.simple_list_item_1)
public class InBackActivity extends Activity {

    @InView(binder = @InBinder(method = "click", listener = OnClick.class))
    Button bt_test;
    @InView
    TextView tv_test;

    @Init
    public void init(){
        bt_test.setText("点击弹不出 toast，因为是后台进程");
    }

    @InBack
    public void click(View v) {
        bt_test.setText("后台进程");
        //这里要么不会显示要么报错
    }
}
```

使用场景如下：欢迎页面

```
@InLayer(R.layout.activity_first)
public class WelecomeActivity extends Activity {

    /**
     * Init 注解表示 View 初始化完毕后执行的方法<br>
     * InBack 注解该方法运行在后台进程(该方法不能是 private 不能是 final)<br>
     * @author gdpancheng@gmail.com 2015 年 1 月 5 日 下午 5:46:34
```

```
     * @return void
     * @throws InterruptedException
     */
    @Init@InBack
    protected void init() throws InterruptedException {
        Thread.sleep(3000);//正常情况下 当前方法会阻塞屏幕，加上@InBack 就不会
        startActivity(new Intent(WelecomeActivity.this, MenuActivity.class));
        finish();
    }
}
```

5.2.8 方法点击事件注解

Ids 是一个数组，用逗号隔开，表示这些 ids 的 View 绑定事件
listeners 为事件类型
当这些 View 事件触发的时候，会走这个方法
点击事件遵循子类覆盖父类，方法上的注解事件可以覆盖 InView 或者 InAll 上的事件

```
@InListener(ids={R.id.top},listeners={OnClick.class})
private void click(View view){
}
```

5.2.9 基类注解

```
@InLayer(value = R.layout.activity_inplayer, parent = R.id.common)
public class InPLayerActivity extends CommonActivity {
    @InListener(ids={R.id.top},listeners={OnClick.class})
    private void click(View view){
        Toast.makeText(this, "子类中点击了", Toast.LENGTH_SHORT).show();
    }
}
```

说明：对基类中通用模块的 view 进行事件绑定，如果需要在子类中单独处理，则在子类中覆盖即可 注意其必须为 private，其中 ids 是需要绑定事件的 View 的 ID，后面为事件类型。点击事件遵循子类覆盖父类，方法上的注解事件可以覆盖 InView 或者 InAll 上的事件

5.2.10 自动 Fragment 注解

```
@InLayer(value = R.layout.activity_fragment)
public class AutoFragmentActivity extends CommonActivity {

    //创建了一个 Fragment 无需方法 onCreateView
    @InBean
    private AutoFragment fragment;

    /**
     * 当 onCreateView 执行完毕以后 会调用含有{@link Init}注解的方法
     * @author gdpancheng@gmail.com 2015 年 1 月 11 日 下午 4:43:00
     * @return void
     */
    @Init
    void init() {
        startFragmentAdd(fragment);
    }
}
```

下面是父类代码

```
@InPLayer(R.layout.activity_com)
public class CommonActivity extends FragmentActivity {

    public void startFragmentAdd(Fragment fragment) {
        FragmentManager fragmentManager = this.getSupportFragmentManager();
        FragmentTransaction fragment = fragmentManager.beginTransaction();
        fragment .replace(R.id.fl_test, fragment);
        fragment .commit();
    }

    /**
     * 对基类中通用模块的 view 进行事件绑定,如果需要在子类中单独处理 则在子类中覆盖即可 注意其必须为 private 其中 ids 是需要绑定事件的 View 的 ID,后面为事件类型 TODO(这里用一句话描述这个方法的作用)
     *
     * @author gdpancheng@gmail.com 2015 年 11 月 6 日 下午 6:00:52
     * @param view
     * @return void
```

```
     */
    @InListener(ids = { R.id.top, R.id.bottom }, listeners = { OnClick.class })
    private void click(View view) {
        Toast.makeText(this, "父类中点击了", Toast.LENGTH_SHORT).show();
    }
}
```

下面是 fragment 代码

```
/**
 * 只有自动生成或者由{@link BeanFactory}创建的 Fragment 才可以自动绑定布局
 *  <br>----------------------------------------------
 * <br>ooo0.............
 * <br>(....) ... Oooo...
 * <br>.\..(.....(.....).......
 * <br>..\_)..... )../........
 * <br>.......... (_/..........
 * <br>微信 gdpancheng
 * <br>----------------------------------------------
 * @author gdpancheng@gmail.com 2015 年 1 月 11 日 下午 4:32:24
 */
@InLayer(R.layout.activity_com)
public class AutoFragment extends Fragment {

    @InView(binder = @InBinder(listener = OnClick.class, method = "click"))
    Button top;

    @Init
    void init() {
        System.out.println("fragment 初始化完毕");
    }

    @InBack
    private void click(View view) {
        System.out.println("这里是后台进程");
    }

    /**
     * 对 view 的 view 进行事件绑定
     * 其中 ids 是需要绑定事件的 View 的 ID，后面为事件类型
     * TODO(这里用一句话描述这个方法的作用)
```

```
     * @author gdpancheng@gmail.com 2015 年 11 月 6 日 下午 6:00:52
     * @param view
     * @return void
     */
    @InListener(ids={R.id.top,R.id.bottom},listeners={OnClick.class})
    private void l(View view){
        Toast.makeText(view.getContext(), "点击了", Toast.LENGTH_SHORT).show();
    }
}
```

其中除了 activity 生命周期的注解无法在 fragment 中使用以外。其他的包括 Init 注解等都可以在 fagment 中使用，去掉了重复性的代码 oncreatview

手动 Fragment 注解

```
/**
 * 有参的 Fragment <br>
 * 如果无参可以直接使用 BeanFactory.instanceFragment(ParamFragment.class)代替 new ParamFragment()
 * <br>----------------------------------------------
 * <br>ooo0.............
 * <br>(....) ... Oooo...
 * <br>.\..(.....(.....).......
 * <br>..\_)..... )../........
 * <br>.......... (_/..........
 * <br>微信 gdpancheng
 * <br>----------------------------------------------
 * @author gdpancheng@gmail.com 2014 年 12 月 19 日 下午 1:17:32
 */
@InLayer(value = R.layout.activity_fragment)
public class ParamFragmentActivity extends CommonActivity {

    private ParamFragment fragment;

    @Init
    void init() {
        fragment = BeanFactory.instanceFragment(ParamFragment.class, "d");
        startFragmentAdd(fragment);
    }
}
```

带有参构造器

```
@InLayer(R.layout.activity_com)
public class ParamFragment extends Fragment {

    public String aa;

    public ParamFragment(String aa) {
        this.aa = aa;
    }

    @Init
     void init(){
        System.out.println("fragment 初始化完毕"+aa);
    }

    /**
     * 对 view 的 view 进行事件绑定
     * 其中 ids 是需要绑定事件的 View 的 ID，后面为事件类型
     * TODO(这里用一句话描述这个方法的作用)
     * @author gdpancheng@gmail.com 2015 年 11 月 6 日 下午 6:00:52
     * @param view
     * @return void
     */
    @InListener(ids={R.id.top,R.id.bottom},listeners={OnClick.class})
    private void l(View view){
        Toast.makeText(view.getContext(), "点击了", Toast.LENGTH_SHORT).show();
    }
}
```

5.3 适配器功能

5.3.1 无适配器

无需创建 adapter 当 listview 使用了 InView 并且设置了 Item 参数以后

直接从 listview 对象中获取 adapter，然后强转成 LoonAdapter，并从获取数据集合 list，然后设置数据

自动注入的适配器会根据你的数据集合的类型是 hashmap 或者实体类的属性或者 key 来和 view 的 id 进行匹配，一旦匹配会自动绑定。

如果你想在自己绑定则传入 LoonViewDeal 的实现类，然后通过 LoonViewDeal 的回调来实现自己绑定

```
@InLayer(value = R.layout.activity_list)
public class NoAdapterActivity extends CommonActivity implements OnClickListener {

    @InAll(@InBinder(method = "onClick", listener = OnClick.class))
    Views views;

    private BaseAdapter mAdapter;
    private ArrayList<Good> mListItems;

    class Views {
        TextView test;
        Button top;
        @InView(item = R.layout.simple_list_item_1)
        ListView list;
    }

    @Init
    void init() {
        mAdapter = (BaseAdapter) views.list.getAdapter();
        LoonAdapter<Good> loon = (LoonAdapter<Good>) mAdapter;
        mListItems = loon.getData();

        //模拟数据
        for (String title : mStrings) {
            Good good = new Good();
            good.setTv_test(title);
            mListItems.add(good);
        }
```

```
        loon.setDeal(deal);
    }

    LoonViewDeal<Good> deal = new LoonViewDeal<Good>() {
        @Override
        public boolean dealView(Good t, ViewHolder viewHolder) {
            viewHolder.setData(R.id.bt_test, "按钮");
            return false;
        }
    };

    @Override
    public void onClick(View arg0) {
        Toast.makeText(NoAdapterActivity.this, "点击", Toast.LENGTH_SHORT).show();
    }

    private String[] mStrings = { "Abbaye de Belloc", "Abbaye du Mont des Cats", "Abertam",
"Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au Chablis", "Afuega'l Pitu",
"Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler", "Abbaye de Belloc", "Abbaye du
Mont des Cats", "Abertam", "Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au
Chablis", "Afuega'l Pitu", "Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler" };
}
```

5.3.2 无参 baseAdapter

上一个是在 listview 上增加一个 item 的属性，这个例子是在 Baseadapter 字段上增加了 InBean 的注解 InBean 的参数是适配器的 item 布局

```
@InLayer(value = R.layout.activity_list)
public class BaseAdapterActivity extends CommonActivity implements OnClickListener {

    @InAll(@InBinder(method = "onClick", listener = OnClick.class))
    Views views;

    @InBean(R.layout.simple_list_item_1)
    private BaseAdapter mAdapter;

    class Views {
        TextView test;
        Button top;
        ListView list;
```

```
    }

    @Init
    void init() {
        //从 BaseAdapter 中获得回掉自定义处理对象
        LoonAdapter<HashMap<String, String>> loon = (LoonAdapter<HashMap<String,
String>>) mAdapter;
        //然后设置设置 View 的拦截处理方法
        loon.setDeal(deal);
        // 模拟数据
        for (String title : mStrings) {
            HashMap<String, String> data = new HashMap<String, String>();
            data.put("tv_test", title);
            loon.getData().add(data);
        }
        views.list.setAdapter(mAdapter);
    }

    LoonViewDeal<HashMap<String, String>> deal = new LoonViewDeal<HashMap<String,
String>>() {
        @Override
        public boolean dealView(HashMap<String, String> t, ViewHolder viewHolder) {
            viewHolder.setData(R.id.bt_test, "按钮" +
viewHolder.getPosition()).setOnClickListener(BaseAdapterActivity.this);
            return false;
        }
    };

    @Override
    public void onClick(View arg0) {
        Toast.makeText(BaseAdapterActivity.this, ((Button) arg0).getText(),
    Toast.LENGTH_SHORT).show();
    }

    private String[] mStrings = {"Abbaye de Belloc", "Abbaye du Mont des Cats", "Abertam",
"Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au Chablis", "Afuega'l Pitu",
"Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler", "Abbaye de Belloc", "Abbaye du
Mont des Cats", "Abertam", "Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au
Chablis", "Afuega'l Pitu", "Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler" };
}
```

5.3.3 自定义一 adapter

自动注入一个自定义适配器这个适配器的 Item 为 InBean 的参数
activity 如下：

```
@InLayer(value = R.layout.activity_list)
public class CustomAdapterActivity extends CommonActivity implements OnClickListener {

    /**
     * 对 View 布局进行自动注入
     */
    @InAll(@InBinder(method = "onClick", listener = OnClick.class))
    Views views;

    /**
     * 自动创建了一个以@R.layout.simple_list_item_1 布局为 item 的适配器
     */
    @InBean(R.layout.simple_list_item_1)
    private CustomAdapter mAdapter;

    /**
     * 包含了当前类布局中 所需要 find 出来的所有的 View
     *  <br>----------------------------------------------
     * <br>ooo0..............
     * <br>(....) ... Oooo...
     * <br>.\..(.....(.....).......
     * <br>..\_)..... )../........
     * <br>.......... (_/..........
     * <br>微信 gdpancheng
     * <br>----------------------------------------------
     * @author gdpancheng@gmail.com 2015 年 1 月 11 日 下午 4:47:12
     */
    class Views {
        TextView test;
        Button top;
        ListView list;
    }

    @Init
    void init() {
        // 模拟数据 模拟了 N 条数据
```

```
        for (String title : mStrings) {
            Good good = new Good();
            good.setTv_test(title);
            mAdapter.getData().add(good);
        }
        //把当前类的点击事件传递给注入的 adapter
        mAdapter.setL(this);
        views.list.setAdapter(mAdapter);
    }

    /**
     * listview item 的点击事件会走这里
     */
    @Override
    public void onClick(View arg0) {
        Toast.makeText(CustomAdapterActivity.this, ((Button) arg0).getText(),
Toast.LENGTH_SHORT).show();
    }

    /**
     * 数据源
     */
    private String[] mStrings = { "Abbaye de Belloc", "Abbaye du Mont des Cats", "Abertam",
"Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au Chablis", "Afuega'l Pitu",
"Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler", "Abbaye de Belloc", "Abbaye du
Mont des Cats", "Abertam", "Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au
Chablis", "Afuega'l Pitu", "Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler" };
}
```

自定义 adapter 如下：

**写法分为两种
**
1　CustomAdapter 实现 LoonAdapter 接口，并声明 LoonAdapter 接口的泛型
**该泛型为你绑定到该 Item 的来源数据的类型
**
2　如果你不实现 LoonAdapter 接口，当这个适配器注入的时候 也会自动实现 LoonAdapter 接口，
**那么你在使用的时候，就必须强转
**

```
public abstract class CustomAdapter extends BaseAdapter implements LoonAdapter<Good> {
    View.OnClickListener l;
```

```
    @Override
    public boolean dealView(Good good, ViewHolder viewHolder) {
        viewHolder.setData(R.id.bt_test, "按钮");
        return false;
    }

    public void setL(View.OnClickListener l) {
        this.l = l;
    }
}
```

5.3.4 自定义二 adapter

自动注入了一个自定义适配器 该自定义适配器没有实现 LoonAdapter 接口

```
@InLayer(value = R.layout.activity_list)
public class CustomAdapterActivity2 extends CommonActivity implements OnClickListener {

    /**
     * 对 View 布局进行自动注入
     */
    @InAll(@InBinder(method = "onClick", listener = OnClick.class))
    Views views;

    @InBean(R.layout.simple_list_item_1)
    private CustomAdapter2 mAdapter;

    /**
     * 包含了当前类布局中 所需要 find 出来的所有的 View
     *  <br>----------------------------------------------
     * <br>ooo0.............
     * <br>(....) ... Oooo...
     * <br>.\..(.....(.....).......
     * <br>..\_)..... )../........
     * <br>.......... (_/..........
     * <br>微信 gdpancheng
     * <br>----------------------------------------------
     * @author gdpancheng@gmail.com 2015 年 1 月 11 日 下午 4:47:12
     */
    class Views {
        TextView test;
```

```
        Button top;
        ListView list;
    }

    @Init
    void init() {
        // 模拟数据 模拟了 N 条数据
        for (String title : mStrings) {
            Good good = new Good();
            good.setTv_test(title);
            ((LoonAdapter<Good>) mAdapter).getData().add(good);
        }
        //把当前类的点击事件传递给注入的 adapter
        mAdapter.setL(this);
        views.list.setAdapter(mAdapter);
    }

    /**
     * listview item 的点击事件会走这里
     */
    @Override
    public void onClick(View arg0) {
        Toast.makeText(CustomAdapterActivity2.this,"哈哈", Toast.LENGTH_SHORT).show();
    }

    /**
     * 数据源
     */
    private String[] mStrings = { "Abbaye de Belloc", "Abbaye du Mont des Cats", "Abertam",
"Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au Chablis", "Afuega'l Pitu",
"Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler", "Abbaye de Belloc", "Abbaye du
Mont des Cats", "Abertam", "Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au
Chablis", "Afuega'l Pitu", "Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler" };
}
```

自定义 adapter 如下：

**写法分为两种
**
1　CustomAdapter 实现 LoonAdapter 接口，并声明 LoonAdapter 接口的泛型
**该泛型为你绑定到该 Item 的来源数据的类型
**
2　如果你不实现 LoonAdapter 接口，当这个适配器注入的时候 也会自动实现 LoonAdapter 接口，
**那么你在使用的时候，就必须强转
**

```
public abstract class CustomAdapter2 extends BaseAdapter {

    View.OnClickListener l;

    public boolean dealView(Good good, ViewHolder viewHolder) {
        viewHolder.setData(R.id.bt_test, "按钮");
        return false;
    }

    public void setL(View.OnClickListener l) {
        this.l = l;
    }

}
```

5.3.5 自动绑定一 adapter

```
@InLayer(value = R.layout.activity_list)
public class SimpleAdapterActivity extends CommonActivity{
    @InAll
    Views views;

    private BaseAdapter mAdapter;

    class Views {
        TextView test;
        Button top;
        @InView(item = R.layout.simple_list_item_1)
        ListView list;
    }

    @Init
```

```
    void init() {
        mAdapter = (BaseAdapter) views.list.getAdapter();
        LoonAdapter<Good> loon = (LoonAdapter<Good>) mAdapter;
        //
--------------------------------------------------------------------------------
        // 模拟数据
        for (String title : mStrings) {
            Good good = new Good();
            good.setTv_test(title);
            loon.getData().add(good);
        }
        //
--------------------------------------------------------------------------------
    }

    private String[] mStrings = { "Abbaye de Belloc", "Abbaye du Mont des Cats", "Abertam",
"Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au Chablis", "Afuega'l Pitu",
"Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler", "Abbaye de Belloc", "Abbaye du
Mont des Cats", "Abertam", "Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au
Chablis", "Afuega'l Pitu", "Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler" };
}
```

5.3.6 自动绑定二 adapter

```
@InLayer(value = R.layout.activity_list)
public class SimpleAdapterActivity2 extends CommonActivity {

    @InAll
    Views views;

    @InBean(R.layout.simple_list_item_1)
    private BaseAdapter mAdapter;

    class Views {
        TextView test;
        Button top;
        ListView list;
    }

    @Init
```

```
    void init() {
        LoonAdapter<HashMap<String, String>> loon = (LoonAdapter<HashMap<String,
String>>) mAdapter;
        //
--------------------------------------------------------------------------------
        // 模拟数据
        for (String title : mStrings) {
            HashMap<String, String> data = new HashMap<String, String>();
            data.put("tv_test", title);
            loon.getData().add(data);
        }
        //
--------------------------------------------------------------------------------
        views.list.setAdapter(mAdapter);
    }

    private String[] mStrings = { "Abbaye de Belloc", "Abbaye du Mont des Cats", "Abertam",
"Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au Chablis", "Afuega'l Pitu",
"Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler", "Abbaye de Belloc", "Abbaye du
Mont des Cats", "Abertam", "Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au
Chablis", "Afuega'l Pitu", "Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler" };
}
```

5.3.7 通用适配器

```
@InLayer(value = R.layout.activity_list)
public class SimpleAdapterActivity3 extends CommonActivity implements OnClickListener {

    @InAll
    Views views;
    ArrayList<Good> arrayList = new ArrayList<Good>();

    class Views {
        TextView test;
        Button top;
        ListView list;
    }

    @Init
    void init() {
```

```
        // 模拟数据

        for (String title : mStrings) {
            Good good = new Good();
            good.setTv_test(title);
            arrayList.add(good);
        }
        views.list.setAdapter(new CommonAdapter<Good>(this, arrayList,
R.layout.simple_list_item_1) {
            @Override
            public void convert(ViewHolder helper, Good item) {
                helper.setData(R.id.tv_test, item.getTv_test());

    helper.getView(R.id.bt_test).setOnClickListener(SimpleAdapterActivity3.this);
            }
        });
    }

    @Override
    public void onClick(View arg0) {
        Toast.makeText(SimpleAdapterActivity3.this, ((Button) arg0).getText(),
Toast.LENGTH_SHORT).show();
    }

    private String[] mStrings = { "Abbaye de Belloc", "Abbaye du Mont des Cats", "Abertam",
"Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au Chablis", "Afuega'l Pitu",
"Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler", "Abbaye de Belloc", "Abbaye du
Mont des Cats", "Abertam", "Abondance", "Ackawi", "Acorn", "Adelost", "Affidelice au
Chablis", "Afuega'l Pitu", "Airag", "Airedale", "Aisy Cendre", "Allgauer Emmentaler" };
    }
```

5.4 功能简化

5.4.1 网络请求模块

1 首先要在 App 中进行初始化，并设置网络请求核心

```
public class App extends Application {

    public static App app;
    public Http<HttpInterFace> http;


    @Override
    public void onCreate() {
        app = this;
        Ioc.getIoc().init(this);
        http = new Http<HttpInterFace>(HttpInterFace.class);
        IocListener.newInstance().setHttpListener(listener);
        super.onCreate();
    }

    //框架自带请求核心
    public IocHttpListener<ResponseEntity> listener = new
IocHttpListener<ResponseEntity>() {

        @Override
        public ResponseEntity netCore(NetConfig config) {
            System.out.println("拦截请求："+config);
            InternetConfig netConfig = InternetConfig.defaultConfig();
            ResponseEntity reslut = null;
            switch (config.getType()) {
            case GET:
                reslut = FastHttp.get(config.getUrl(), config.getParams(),
netConfig);
                break;
            case POST:
                reslut = FastHttp.post(config.getUrl(), config.getParams(),
netConfig);
                break;
            case FORM:
                reslut=FastHttp.form(config.getUrl(), config.getParams(), new
HashMap<String, File>(), netConfig);
                break;
            case WEB:
                netConfig.setMethod(config.getMethod());
                netConfig.setName_space(config.getName_space());
```

```
                    reslut = FastHttp.webServer(config.getUrl(),
config.getParams(), netConfig, "post");
                    break;
                }
                System.out.println("拦截结果："+reslut);
                return reslut;
            }
        };

        public static class Result {
            public final static int OK = 0;
            public final static int ERROR = 1;
            int status = 0;
            public String object;
        }
        //OKhttp 自带请求核心
        public IocHttpListener<Result> listener2 = new IocHttpListener<Result>() {

            final OkHttpClient client = new OkHttpClient();

            @Override
            public Result netCore(NetConfig config) {
                System.out.println("拦截请求："+config);
                Result result = new Result();
                System.out.println(config);
                switch (config.getType()) {
                case GET: {
                    try {
                        Request request = new
Request.Builder().url(config.getUrl()).build();
                        Response response = client.newCall(request).execute();
                        if (!response.isSuccessful()) {
                            result.status = Result.ERROR;
                            break;
                        }
                        Headers responseHeaders = response.headers();
                        for (int i = 0; i < responseHeaders.size(); i++) {
                            System.out.println(responseHeaders.name(i) + ": " +
responseHeaders.value(i));
                        }
                        result.status = Result.OK;
```

```
                result.object = response.body().string();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
            break;
        case POST: {
            try {
                StringBuffer buffer = new StringBuffer();
                for (String key : config.getParams().keySet()) {
                   if (buffer.length() != 0) {
                    buffer.append("&");
                   }
                   buffer.append(key+"="+config.getParams().get(key));
                }
                MediaType MEDIA_TYPE_MARKDOWN =
MediaType.parse("text/x-markdown; charset=utf-8");
                Builder builder = new
Request.Builder().url(config.getUrl());
                builder.post(RequestBody.create(MEDIA_TYPE_MARKDOWN,
buffer.toString()));
                Request request = builder.build();
                Response response = client.newCall(request).execute();
                if (!response.isSuccessful()) {
                    result.status = Result.ERROR;
                    break;
                }
                Headers responseHeaders = response.headers();
                for (int i = 0; i < responseHeaders.size(); i++) {
                    System.out.println(responseHeaders.name(i) + ": " +
responseHeaders.value(i));
                }
                result.status = Result.OK;
                result.object = response.body().string();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
            break;
        case FORM:{
            try {
```

```
                            StringBuffer buffer = new StringBuffer();
                            for (String key : config.getParams().keySet()) {
                                if (buffer.length() != 0) {
                                    buffer.append("&");
                                }
                                buffer.append(key+"="+config.getParams().get(key));
                            }
                            MediaType MEDIA_TYPE_MARKDOWN =
    MediaType.parse("text/x-markdown; charset=utf-8");
                            Builder builder = new
    Request.Builder().url(config.getUrl());
                            builder.post(RequestBody.create(MEDIA_TYPE_MARKDOWN,
    buffer.toString()));
                            Request request = builder.build();
                            Response response = client.newCall(request).execute();
                            if (!response.isSuccessful()) {
                                result.status = Result.ERROR;
                                break;
                            }
                            Headers responseHeaders = response.headers();
                            for (int i = 0; i < responseHeaders.size(); i++) {
                                System.out.println(responseHeaders.name(i) + ": " +
    responseHeaders.value(i));
                            }
                            result.status = Result.OK;
                            result.object = response.body().string();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                        break;
                    case WEB:
                        //这里是 soap 协议
                        break;
                    }
                    System.out.println("拦截结果："+result);
                    return result;
                }
            };
        }
```

先初始化 `public Http<HttpInterFace> http = new Http<HttpInterFace>(HttpInterFace.class);`
切换请求核心只需要 `IocListener.newInstance().setHttpListener(listener);`

HttpInterFace 如下

```
@InNet(HttpUrl.class)
public interface HttpInterFace {

    //同步
    @InForm(HttpUrl.FILES)
    public <T> T login(@InParam("id") String id,LinkedHashMap<String, File> params);

    @InGet
    public <T> T login(LinkedHashMap<String, Object> params);

    @InPost
    public <T> T asyncLoginPost(String name, String password);

    @InPost(HttpUrl.LOGIN)
    public <T> T postStr(@InParam String json);

    @InWeb(method="getRegionCountry",space="http://WebXml.com.cn/")
    public <T> T asyncWeb();

    @InWeb
    public <T> T asyncLoginWeb(@InParam("theRegionCode") String theRegionCode,NetConfig
config);

    //异步
    @InPost(HttpUrl.LOGIN)
    public void login(@InParam("username") String name, @InParam("password") String
password);

    @InPost(HttpUrl.LOGIN)
    public void loginPost(@InParam LinkedHashMap<String, Object> params);

    @InPost
    public void login(@InParam String json);

    @InWeb
    public void LoginWeb(String json);
```

```
    @InForm
    public void LoginForm(String name, String password);
}
```

方法如果有返回值 则是同步 这里是有返回值则是同步
返回值的类型是根据你在 listener 泛型的类型
否则是异步
方法名必须要和 HttpUrl 中的 url 名称一样，方法名小写，HttpUrl 字段名必须大写
否则则需要在 HttpInterFace 注解里面设置 URL

如果一个页面有多个网络请求 需要回调则需要在 HttpUrl 中设置 KEY，KEY 必须以_KEY 结尾，全部大写

HttpUrl 代码如下：

```
public static class HttpUrl {
        /** 系统 URL **/
        public static final String BASIC = "http://cs2.137home.com/index.php?m=api&a=";
        public static final String LOGIN = BASIC + "login";
        public static final String LONGIN_POST = BASIC + "login";
        public static final String ASYNCLOGINPOST = BASIC + "login";
        public static final String ASYNCWEB =
"http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx";
        public static final String ASYNCLOGINWEB =
"http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx";
        public static final String LOGINFORM = BASIC + "login";
        public static final String ASYNCLOGINFORM = BASIC + "login";
        public static final String ASYNCLOGINDISTRIBUTE = BASIC + "login";
        public static final String FILES =
"http://cs2.137home.com/index.php?m=api&a=edit_avatar";
        public static final int LOGIN_KEY = 0;
    }
```

以上是所有配置。下面将介绍网络请求调用

同步

```
Object object = App.app.http.u(AHttpModule.this).login(params);
```

异步

```
App.app.http.u(AHttpModule.this).login("aaa", "bbbb");
```

回调

根据你在 app 设置的请求核心不同 result 的参数不同，否则无法回调，网络请求结果会回调到此方法

```
@InHttp
public void result(ResponseEntity entity) {
    v.tv_result.setText(entity.getContentAsString());
}

@InHttp
public void result2(Result result) {
    v.tv_result.setText(result.object);
}
```

详细应用请参考 demo

5.4.2 输入验证

可以模仿 VaPassword 等构建自己的验证器

```
@InLayer(R.layout.activity_vali)
public class ValidatorActivity extends Activity {

    @InAll
    Views views;

    static class Views {
        @InVa(value=VaPassword.class,index=1)
        EditText tv_password;
        @InVa(value=VaPasswordConfirm.class,index=2)
        EditText tv_passwordconfirm;
        @InVa(value=VaEmail.class,index=3)
        EditText tv_email;
        @InVa(value=VaMobile.class,index=4)
        EditText tv_mobile;
        @InVa(value=VaDate.class,index=5)
        EditText tv_data;
        @InVa(value=VaWeb.class,index=6)
        EditText tv_web;
        @InVa(value=VaCard.class,index=7)
        EditText tv_card;
        @InVa(msg = "不能为空",empty=false,index=8)
        EditText tv_notnull;
```

```
            @InVa(reg=Regex.LET_NUM_UNLINE_REG,msg="请输入字母数字或下划线
",empty=false,index=9)
            EditText tv_number;
            @InBinder(listener=OnClick.class,method="click")
            Button bt_onclick;
        }

        @Init
        public void init() {
        }

        public void click(View view) {
            Validator.verify(this);
        }

        @InVaOK
        private void onValidationSucceeded() {
            Toast.makeText(this, "验证成功", Toast.LENGTH_SHORT).show();
        }

        @InVaER
        public void onValidationFailed(ValidatorCore core) {
            System.out.println(core.getMsg());
            if
(core.getView()!=null&&EditText.class.isAssignableFrom(core.getView().getClass())) {
                EditText editText = core.getView();
                editText.requestFocus();
                editText.setFocusable(true);
                editText.setError(core.getMsg());
            }
        }
    }
```

5.4.3 跨进程通讯

这里是跨进程通讯实例，框架集成了 tinybus
只有 activity 和 fragment 中可以不用注册 tinybus
其他类中需要自己手动绑定以及解绑

TinyBus.send(new User());//发送消息

接收消息

```
@Subscribe(mode = Mode.Background)
    public void event(User event) {
        System.out.println("这里是后台进程");
    }

    @Subscribe
    public void event2(User event) {
        bt_test.setText("点击之后");
        System.out.println("这里是前台进程");
    }
```

5.4.4 Json 格式化类

Json 转实体类：

```
Parent event = Handler_Json.JsonToBean(Parent.class, parent_json.toString());
```

Json 转集合：

```
HashMap<String, Object> object = Handler_Json.JsonToCollection(parent_json.toString());
```

5.4.5 倒计时类

参考 demo

六 用到的技术

用到的技术有，反射，动态代理，java 接口代理等等

七 友情链接（使用了 loonandroid 1X 版本已上线项目）


面吧：一个朋友开发的软件，转为程序员面试使用的神器，帮忙宣传下，谢谢。

古道网：是一款记录、分享古代道路的手机应用


酷鱼: 新财富集团的 APP 资本人的社交圈


泵阀通：是由明作网络在移动互联网领域针对泵阀产业推出的跨时代新媒体


大歌星：娱乐 K 歌应用，把手机成为录间棚，原版伴奏供你选择。


拍手吧：高品质音乐，独特的游戏，海量电子书，超好玩的应用软件以及社交体验


宝贝计划：“宝贝计划”是大连地区专注于儿童教育领域的 APP。