代码.note

```
getObjectClass获取类，在从class当中根据方法名称和参数类型取得方法id，最后通过cal
   l【开头】回调Java。
```

1. 日志

```
#include <iostream>
#include <android/log.h>
#define TAG "JNISTUDY"
// __VA_ARGS__ 代表 ...的可变参数
#define LOGD(...) __android_log_print(ANDROID_LOG_DEBUG, TAG, __VA_ARGS__);
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR, TAG, __VA_ARGS__);
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO, TAG, __VA_ARGS__);
int age = 99; // 实现
void show() {
LOGI("show run age:%d\n", age);
}
```

2.native调用java层

2.1 数组

```
testArrayAction(JNIEnv *env, jobject thiz,jint coujstring text_info,
jintArray ints,jobjectArray strs) {
int countInt = count; // jint本质是int，所以可以用int接收
LOGI("参数一 countInt:%d\n", countInt);
const char * textInfo = env->GetStringUTFChars(text_info, NULL);
// 把int[] 转成 int*
int* jintArray = env->GetIntArrayElements(ints, NULL);
jsize size = env->GetArrayLength(ints);

for (int i = 0; i < size; ++i) {
```

```
*(jintArray+i) += 100; // C++的修改，影响不了Java层
}
/**
* 0: 刷新Java数组，并 释放C++层数组
* JNI_COMMIT: 只提交 只刷新Java数组，不释放C++层数组
* JNI_ABORT: 只释放C++层数组
*/
env->ReleaseIntArrayElements(ints, jintArray, 0);

// ③: jobjectArray 代表是Java的引用类型数组，不一样
jsize strssize = env->GetArrayLength(strs);
for (int i = 0; i < strssize; ++i) {
jstring jobj = static_cast<jstring>(env->GetObjectArrayElement(strs
, i));

// 模糊：isCopy内部启动的机制
// const char* GetStringUTFChars(jstring string, jboolean* isCopy)
const char * jobjCharp = env->GetStringUTFChars(jobj, NULL);
LOGI("参数四 引用类型String 具体的：%s\n", jobjCharp);
// 释放jstring
env->ReleaseStringUTFChars(jobj, jobjCharp);
}
}
```

2.2 对象

```
_putObject(JNIEnv *env, jobject thiz, jobject student, jstring str) {
const char * strChar = env->GetStringUTFChars(str, NULL);
env->ReleaseStringUTFChars(str, strChar);
// 1.寻找类 Student 两种方式
// jclass studentClass = env->FindClass("com/test/java/Student");
jclass studentClass = env->GetObjectClass(student); // 第二种

// 2.找方法ID，Student类里面的函数规则 签名
jmethodID setName = env->GetMethodID(studentClass, "setName", "(Ljava/lang/
  String;)V");
jmethodID getName = env->GetMethodID(studentClass, "getName", "()Ljava/lan
  g/String;");
jmethodID showInfo = env->GetStaticMethodID(studentClass, "showInfo", "(Lja
  va/lang/String;)V");

// 3.根据方法ID和类对象 调用方法 setName
jstring value = env->NewStringUTF("AAAA");
env->CallVoidMethod(student, setName, value);
// 4.调用返回值 getName
jstring getNameResult = static_cast<jstring>(env->CallObjectMethod(student,
  getName));
const char * getNameValue = env->GetStringUTFChars(getNameResult, NULL);
LOGE("调用到getName方法，值是:%s\n", getNameValue);

// 5.调用静态showInfo
jstring jstringValue = env->NewStringUTF("静态方法你好，我是C++");
env->CallStaticVoidMethod(studentClass, showInfo, jstringValue);
}
```

2.3 对象进阶

```
//Native层创建java对象
MainActivity_insertObject(JNIEnv *env, jobject thiz) {
// 1.通过包名+类名的方式 拿到 Student class 凭空拿class
const char *studentstr = "com/test/java/Student";
jclass studentClass = env->FindClass(studentstr);

// 2.通过student的class 实例化此Student对象 C++ new Student
jobject studentObj = env->AllocObject(studentClass);
// AllocObject 只实例化对象，不会调用对象的构造函数

// 方法签名的规则
jmethodID setName = env->GetMethodID(studentClass, "setName", "(Ljava/lang/
  String;)V");
jmethodID setAge = env->GetMethodID(studentClass, "setAge", "(I)V");

// 调用方法
jstring strValue = env->NewStringUTF("Derry");
env->CallVoidMethod(studentObj, setName, strValue);
env->CallVoidMethod(studentObj, setAge, 99);

// env->NewObject() // NewObject 实例化对象，会调用对象的构造函数
// ===== 下面是 Person对象 调用person对象的 setStudent 函数等

// 4.通过包名+类名的方式 拿到 Student class 凭空拿class
const char *personstr = "com/test/java/Person";
jclass personClass = env->FindClass(personstr);
jobject personObj = env->AllocObject(personClass);
// setStudent 此函数的 签名 规则
jmethodID setStudent = env->GetMethodID(personClass, "setStudent",
"(Lcom/derry/as_jni_project/Student;)V");
// 向Person类里面注入Student对象
env->CallVoidMethod(personObj, setStudent, studentObj);
}
```

3. 全局引用

```
jclass dogClass; // 你以为这个是全局引用，实际上他还是局部引用

extern "C"
JNIEXPORT void JNICALL testQuote(JNIEnv *env, jobject thiz) {
if (NULL == dogClass) {
/*const char * dogStr = "com/test/java/Dog";
dogClass = env->FindClass(dogStr);*/

// 升级全局引用： JNI函数结束也不释放，反正就是不释放，必须手动释放
// ----- 相当于： C++ 对象 new、手动delete
const char * dogStr = "com/test/java/Dog";
jclass temp = env->FindClass(dogStr);
dogClass = static_cast<jclass>(env->NewGlobalRef(temp));
// 提升全局引用
// 记住：用完了，如果不用了，马上释放，C C++ 工程师的赞美
env->DeleteLocalRef(temp);
}

// <init> V 是不会变的
// 获取Dog类的构造函数一
jmethodID init = env->GetMethodID(dogClass, "<init>", "()V");
jobject dog = env->NewObject(dogClass, init);
// 构造函数2
init = env->GetMethodID(dogClass, "<init>", "(I)V");
dog = env->NewObject(dogClass, init, 100);
// 构造函数3
init = env->GetMethodID(dogClass, "<init>", "(II)V");
dog = env->NewObject(dogClass, init, 200, 300);

env->DeleteLocalRef(dog); // 释放
}

// JNI函数结束，会释放局部引用 dogClass虽然被释放，但是还不等于NULL，只是一个
//悬空指针而已，所以第二次进不来IF，会奔溃

extern int age; // 声明age
extern void show(); // 声明show函数
```

```
// 手动释放全局引用
extern "C"
JNIEXPORT void JNICALL delQuote(JNIEnv *env, jobject thiz) {
if (dogClass != NULL) {
LOGE("全局引用释放完毕，上面的按钮已经失去全局引用，再次点击会报错");
env->DeleteGlobalRef(dogClass);
dogClass = NULL; // 最好指向NULL的地址，不要去成为悬空指针
}
show();
}
```

4. 动态注册与静态注册

> 1. 默认情况下，就是静态注册，静态注册是最简单的方式，NDK开发过程中，基本上使用静态注册
> 2. Android 系统的C++源码：基本上都是动态注册（麻烦）
> 3. 静态注册：
>   优点：开发简单
>   缺点
>   a.JNI函数名非常长
>   b.捆绑 上层 包名 + 类名
>   c.运行期 才会去 匹配JNI函数，性能上 低于 动态注册

```
const char *mainActivityClassName = "com/test/java/MainActivity";

void dynamicMethod01() {
LOGD("动态注册的函数 dynamicMethod01...");
}

int dynamicMethod02(JNIEnv *env, jobject thiz, jstring valueStr) {
const char *text = env->GetStringUTFChars(valueStr, nullptr);
env->ReleaseStringUTFChars(valueStr, text);
return 200;
}

/* 结构体
typedef struct {
const char* name; // 函数名
const char* signature; // 函数的签名
void* fnPtr; // 函数指针
} JNINativeMethod;
*/
static const JNINativeMethod jniNativeMethod[] = {
{"dynamicJavaMethod01", "()V", (void *) (dynamicMethod01)},
{"dynamicJavaMethod02", "(Ljava/lang/String;)I", (int *)
  (dynamicMethod02)},
};
```

```


// 调用JNI_OnLoad函数
extern "C" JNIEXPORT jint JNI_OnLoad(JavaVM *javaVm, void *) {
::jVm = javaVm;
JNIEnv *jniEnv = nullptr;
int result = javaVm->GetEnv(reinterpret_cast<void **>(&jniEnv), JNI_VERSION
  _1_6);
jclass mainActivityClass = jniEnv->FindClass(activityClassName);
// RegisterNatives(Class, 我们的数组==jniNativeMethod, 注册的数量 = 2)
jniEnv->RegisterNatives(mainActivityClass,jniNativeMethod,
sizeof(jniNativeMethod) / sizeof(JNINativeMethod));
return JNI_VERSION_1_6;
}
```

5.线程

> 1. JavaVM全局，绑定当前进程， 只有一个地址
> 2. JNIEnv线程绑定，不能夸线程会崩溃， 绑定主线程，绑定子线程
> 3. jobject 不能夸线程会崩溃，谁调用JNI函数，谁的实例会给jobject

```
package com.sankuai.moviepro.common.utils;

import android.content.Context;
import android.graphics.Color;
import android.graphics.LinearGradient;
import android.graphics.Paint;
import android.graphics.Rect;
import android.graphics.RectF;
import android.graphics.Shader;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.GradientDrawable;
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.StateListDrawable;
import android.graphics.drawable.shapes.RoundRectShape;

import androidx.annotation.ColorRes;

import org.jetbrains.annotations.NotNull;
```

```
public class DrawableUtils {

    /**
     * 圆角矩形Drawable
     *
     * @param color
     * @param padding
     * @param inset      内部矩形与外部矩形的距离
     * @param innerRadii 内部矩形弧度
     * @return drawable
     */
    public static Drawable getRoundShapeDrawable(int color, Rect padding, RectF inset,
float[] innerRadii) {
        ShapeDrawable drawable = new ShapeDrawable();
        // 外部矩形弧度
        float radius = DimenUtils.dp2px(2);
        float[] outerR = new float[]{radius, radius, radius, radius, radius, radius, radius,
radius};
        RoundRectShape shape = new RoundRectShape(outerR, inset, innerRadii);
        drawable.setShape(shape);
        drawable.getPaint().setColor(color);
        drawable.setPadding(padding.left, padding.top, padding.right, padding.bottom);
        return drawable;
    }

    /**
     * 默认的圆角矩形Drawable
     *
     * @param color
     */
    public static Drawable getRoundShapeDrawable(int color) {
        int xPadding = DimenUtils.dp2px(4);
        int yPadding = DimenUtils.dp2px(1);
        Rect padding = new Rect(xPadding, yPadding, xPadding, yPadding);
        return getRoundShapeDrawable(color, padding, null, null);
    }
    /**
     * 默认的Stroke圆角矩形Drawable
     *
     * @param color
     */
    public static Drawable getRoundStrokeShapeDrawable(int color) {
        float radius = DimenUtils.dp2px(2);
        return getRoundStrokeShapeDrawable(color, radius);
    }

    /**
     * 默认的Stroke圆角矩形Drawable
     *
     * @param color
     */
    public static Drawable getRoundStrokeShapeDrawable(int color, float radius) {
        ShapeDrawable drawable = new ShapeDrawable();
        float[] outerR = new float[]{radius, radius, radius, radius, radius, radius, radius,
radius};
        RoundRectShape shape = new RoundRectShape(outerR, null, null);
        drawable.setShape(shape);
        drawable.getPaint().setColor(color);
        drawable.getPaint().setStyle(Paint.Style.STROKE);
        return drawable;
    }

    /**
     * left\top\right\bottom padding相同的情况
     * @return 内描边Drawable
     */
    public static Drawable getRoundStrokeShapeDrawable(int color, float radius, float
width, int padding){
        Rect rectPadding = new Rect(padding, padding, padding, padding);
        return getRoundStrokeShapeDrawable(color, radius, width, rectPadding);
    }

    /**
```

```
     * @return 内描边Drawable
     */
    public static Drawable getRoundStrokeShapeDrawable(int color, float radius, float
width, int left, int top, int right, int bottom){
        Rect rectPadding = new Rect(left, top, right, bottom);
        return getRoundStrokeShapeDrawable(color, radius, width, rectPadding);
    }

    public static Drawable getRoundStrokeShapeDrawable(int color, float radius, float
width, Rect rectPadding){
        ShapeDrawable drawable = new ShapeDrawable();
        float[] outerR = new float[]{radius, radius, radius, radius, radius, radius, radius,
radius};
        RoundRectShape shape = new RoundRectShape(outerR, null, null);
        drawable.setShape(shape);
        Paint paint = drawable.getPaint();
        paint.setAntiAlias(true);
        paint.setColor(color);
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(width);
        drawable.setPadding(rectPadding);
        return drawable;
    }

    /**
     * 圆角矩形Drawable
     *
     * @param color
     */
    public static Drawable getRoundShapeDrawable(int color, float radius) {
        ShapeDrawable drawable = new ShapeDrawable();
        float[] outerR = new float[]{radius, radius, radius, radius, radius, radius, radius,
radius};
        RoundRectShape shape = new RoundRectShape(outerR, null, null);
        drawable.setShape(shape);
        drawable.getPaint().setColor(color);
        drawable.getPaint().setStyle(Paint.Style.FILL);
        return drawable;
    }


    /**
     * 圆角矩形Drawable
     * 前2个 左上角， 3 4 ， 右上角， 56， 左下， 78 ，右下
     *
     * @param color
     */
    public static Drawable getRoundShapeDrawable(int color, float leftTop, float rightTop,
float leftBottom, float rightBottom) {
        ShapeDrawable drawable = new ShapeDrawable();
        float[] outerR = new float[]{leftTop, leftTop, rightTop, rightTop, leftBottom,
leftBottom, rightBottom, rightBottom};
        RoundRectShape shape = new RoundRectShape(outerR, null, null);
        drawable.setShape(shape);
        drawable.getPaint().setColor(color);
        drawable.getPaint().setStyle(Paint.Style.FILL);
        return drawable;
    }

    public static Drawable getGradientDrawable(Context context,
GradientDrawable.Orientation orientation, @ColorRes int startColor, @ColorRes int
endColor) {
        return getGradientDrawable(context, orientation, 0,  startColor, endColor);
    }

    /**
     * 现仅有开始和结束两种颜色的线性渐变，若为多颜色时，可使用可变参数去扩展该方法
     * @param context
     * @param orientation 渐变方向
     * @param radius 圆角
     * @param startColor
     * @param endColor
     * @return 渐变drawable
     */
    public static Drawable getGradientDrawable(Context context,
GradientDrawable.Orientation orientation, float radius, @ColorRes int startColor,
@ColorRes int endColor) {
        int color = context.getResources().getColor(startColor);
        int color2 = context.getResources().getColor(endColor);
        int[] colors = {color, color2};
        GradientDrawable drawable = new GradientDrawable(orientation, colors);
        drawable.setShape(GradientDrawable.RECTANGLE);
        drawable.setCornerRadius(radius);
        drawable.setGradientType(GradientDrawable.LINEAR_GRADIENT);
        return drawable;
    }

    /**
     * java代码：状态选择器
     */
    public static Drawable createSelectedSelector(Drawable pressed, Drawable normal) {
        StateListDrawable drawable = new StateListDrawable();
        drawable.addState(new int[]{android.R.attr.state_selected}, pressed);
        drawable.addState(new int[]{}, normal);
        return drawable;
    }

    public static Drawable getRoundShapeLinearGradientDrawable(@NotNull String
startColor, @NotNull String endColor, float startX, float endX, float startY, float endY,
float radius) {
        LinearGradient linearGradient = new LinearGradient(startX, startY, endX, endY,
Color.parseColor(startColor), Color.parseColor(endColor), Shader.TileMode.CLAMP);
        ShapeDrawable drawable = new ShapeDrawable();
        float[] outerR = new float[]{radius, radius, radius, radius, radius, radius, radius,
radius};
        RoundRectShape shape = new RoundRectShape(outerR, null, null);
        drawable.setShape(shape);
        drawable.getPaint().setShader(linearGradient);
        drawable.getPaint().setStyle(Paint.Style.FILL);
        return drawable;
    }
```

```
}

public abstract class MvpPresenter<V extends ILoadView> extends BasePresenter {

    private WeakReference<V> viewRef;
    public CompositeSubscription compositeSubscription;

    public MvpPresenter() {
        super();
        compositeSubscription = new CompositeSubscription();
    }

    public void attachView(V view) {
        viewRef = new WeakReference<>(view);
    }

    @Nullable
    public V getView() {
        return viewRef == null ? null : viewRef.get();
    }

    public boolean isViewAttached() {
        return viewRef != null && viewRef.get() != null;
    }

    public void detachView() {
        if (viewRef != null) {
            viewRef.clear();
            viewRef = null;
        }
    }

    public abstract void loadData(boolean refresh);



    public void onDestroy() {
```

```
    detachView();
}

public void onDestroyView() {
    clearSubscriptions();
}



protected final <T> void addSubscription(Observable<T> observable) {
    addSubscription(doRequest(observable));
}


protected void addSubscription(Subscription s) {
    if (s != null && compositeSubscription != null) {
        compositeSubscription.add(s);
    }
}

protected final void removeSubscription(Subscription s) {
    if (s != null && compositeSubscription != null) {
        s.unsubscribe();
        compositeSubscription.remove(s);
    }
}

public final void clearSubscriptions() {
    if (compositeSubscription != null) {
        compositeSubscription.clear();
    }
}

private <T> Subscription doRequest(Observable<T> observable) {
    Action1 onNext = getOnNextAction();
    Action1 onError = getOnErrorAction();
    return doRequest(observable, onNext, onError);
    }


    protected Subscription doRequest(final Observable observable, final Action1 onNext,
final Action1 onError) {

        final Observable obWrap = observable.subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread());
        final Action1<Throwable> mtsiErr = new Action1<Throwable>() {
            @Override
            public void call(Throwable throwable) {

            }
        };
        if (onError != null) {
            return obWrap.subscribe(onNext, mtsiErr);
        } else {
            throw new IllegalArgumentException("onError can not be null");
        }
    }


    protected Subscription doRequest(Observable observable, Action1 onNext,
Action1<Throwable> onError, Action0 onFinal) {
        Observable ob = observable.subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread());
        if (onFinal == null) {
            onFinal = Actions.empty();
        }
        if (onError != null) {
            return ob.doAfterTerminate(onFinal).subscribe(onNext, onError);
        } else {
            throw new IllegalArgumentException("onError can not be null");
        }
    }
    protected final void addSubscription(Observable observable, Action1 onNext, Action1
onError) {
        addSubscription(doRequest(observable, onNext, onError));
    }

    // 在处理完成后执行 onFinal
    protected final void addSubscription(Observable observable, SubscribeAction action) {
        if (action.onPre() != null) {
            action.onPre().call();
        }
        addSubscription(doRequest(observable, action.onNext(), action.onError(),
action.onFinal()));
    }

    protected Action1 getOnNextAction() {
        Action1 onNext = new Action1() {
            @Override
            public void call(Object data) {
                if (isViewAttached()) {
                    getView().setData(formatData(data));
                }
            }
        };
        return onNext;
    }

    //复写了formatData 如果手动调用setData 需要手动调用formatData
    protected Object formatData(Object data) {
        return data;
    }

    protected Action1 getOnErrorAction() {
        Action1<Throwable> onError = new Action1<Throwable>() {
            @Override
            public void call(Throwable throwable) {
                if (isViewAttached()) {
                    getView().handleThrowable(throwable);
                }
            }
        };
        return onError;
    }
public class StatusProvider implements View.OnClickListener {
    /**
     * 空状态提示
     */
    public int emptyImageId = R.drawable.component_new_empty_statue;
    public String emptyString =
MovieProApplication.getContext().getString(R.string.noinfo_default);
    public View customEmptyView;   //自定义的空状态View

    /**
     * 网络异常状态提示
     */
    public int errorImageId = R.drawable.component_network_error_new;
    public String errorString =
MovieProApplication.getContext().getString(R.string.error_net);
    public View customErrorView;   //自定义的网络异常View

    /**
     * 服务器异常状态提示
     */
    public int serverErrorImageId = R.drawable.component_get_lost_new;
    public String serverErrorString =
MovieProApplication.getContext().getString(R.string.error_server);
    public View customServerErrorView;     //自定义的服务器异常View

    private View statusView;
    private ProgressDialog mProgress;
    private OverViewDialog mOverView;
    private LayoutInflater layoutInflater;
    private RefreshListener refreshListener;
    private View mCircleProgressContainer;
```

```
@Inject
public StatusProvider(LayoutInflater layoutInflater) {
    this.layoutInflater = layoutInflater;
}

/**
 * 点击刷新的监听
 *
 * @param refreshListener
 */
public void setRefreshListener(RefreshListener refreshListener) {
    this.refreshListener = refreshListener;
}

/**
 * 空状态
 *
 * @param parent
 * @return
 */
public View getEmptyView(ViewGroup parent) {
    if (customEmptyView == null) {
        return getStatusResult(emptyImageId, emptyString, parent);
    } else {
        return customEmptyView;
    }
}

/**
 * 网络异常状态
 *
 * @param parent
 * @return
 */
public View getErrorView(ViewGroup parent) {
    if (customErrorView == null) {
        return getStatusResult(errorImageId, errorString, parent);
        } else {
            return customErrorView;
        }
    }

    /**
     * 服务器异常状态
     *
     * @param parent
     * @return
     */
    public View getServerErrorView(ViewGroup parent) {
        if (customServerErrorView == null) {
            return getStatusResult(serverErrorImageId, serverErrorString, parent);
        } else {
            return customServerErrorView;
        }
    }

    /**
     * 转猫头
     *
     * @param context
     * @param parent
     * @return
     */
    public FrameLayout getProgressContainer(Context context, ViewGroup parent) {
        FrameLayout pframe = new FrameLayout(context);
        pframe.addView(layoutInflater.inflate(R.layout.movie_progress, parent, false), new
FrameLayout.LayoutParams(FrameLayout.LayoutParams.WRAP_CONTENT,
FrameLayout.LayoutParams.WRAP_CONTENT, Gravity.CENTER));
        return pframe;
    }

    /**
     * 转菊花 啊。。Dialog 方式
     *
     * @param fragmentManager
     */
    public void showCircleProgress(FragmentManager fragmentManager) {
        if (mProgress == null) {
            mProgress = new ProgressDialog();
        }
        if (!mProgress.isAdded()) {
            FragmentTransaction transaction = fragmentManager.beginTransaction();
            transaction.add(mProgress, "progress");
            transaction.commitAllowingStateLoss();
            fragmentManager.executePendingTransactions();
        }
    }

    /**
     * 隐藏转菊花  Dialog 方式
     *
     * @param fragmentManager
     */
    public void hideCircleProgress(FragmentManager fragmentManager) {
        if (mProgress != null && mProgress.getDialog() != null && mProgress.isAdded()) {
            mProgress.dismissAllowingStateLoss();
            fragmentManager.executePendingTransactions();
        }
    }

    /**
     * 转菊花 啊。。View 方式
     *
     * @param parent
     */
    public void showCircleProgressView(FrameLayout parent) {
        if (mCircleProgressContainer == null) {
            mCircleProgressContainer = layoutInflater.inflate(R.layout.circle_progressbar,
parent, false);
            mCircleProgressContainer.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {

                }
            });
            parent.addView(mCircleProgressContainer);
        }
        mCircleProgressContainer.setVisibility(View.VISIBLE);
    }

    /**
     * 隐藏转菊花  View 方式
     */
    public void hideCircleProgressView() {
        if (mCircleProgressContainer != null)
            mCircleProgressContainer.setVisibility(View.GONE);
    }

    public void showOverView(FragmentManager fragmentManager) {
        if (mOverView == null) {
            mOverView = new OverViewDialog();
        }
        if (!mOverView.isAdded()) {
            FragmentTransaction transaction = fragmentManager.beginTransaction();
            transaction.add(mOverView, "overview");
            transaction.commitAllowingStateLoss();
            fragmentManager.executePendingTransactions();
        }
    }

    public void hideOverView(FragmentManager fragmentManager) {
        if (mOverView != null && mOverView.getDialog() != null && mOverView.isAdded()) {
            mOverView.dismissAllowingStateLoss();
            fragmentManager.executePendingTransactions();
        }
    }

    private void getStatusViewNoSet(ViewGroup parent) {
        if (statusView == null && layoutInflater != null) {
            statusView = layoutInflater.inflate(R.layout.status_layout, parent, false);
            statusView.setOnClickListener(this);
        }
    }

    private View getStatusResult(int imageRes, String text, ViewGroup parent) {
        getStatusViewNoSet(parent);
        if (statusView == null) {
            return new View(parent.getContext());
        }
        ((ImageView) statusView.findViewById(R.id.statusImg)).setImageResource(imageRes);
        ((TextView) statusView.findViewById(R.id.statusTxt)).setText(text);
        return statusView;
    }

    /**
     * 通过Throwable 获取状态View
     *
     * @param t
     * @param parent
     * @return
     */
    public View getStatusView(Throwable t, ViewGroup parent) {
        if (t instanceof RetrofitException
                && ((RetrofitException) t).kind == RetrofitException.SERVER) {
            return getServerErrorView(parent);
        } else if (t instanceof EmptyDataException) {
            return getEmptyView(parent);
        } else {
            return getErrorView(parent);
        }
    }

    public View getClickErrorView(View.OnClickListener clickListener){
        if (layoutInflater != null) {
            View view = layoutInflater.inflate(R.layout.knb_error,null);
```

```
            view.setOnClickListener(clickListener);
            return view;
        }
        return null;

    }

    public View getClickServerView(View.OnClickListener clickListener){
        if (layoutInflater != null) {
            View view = layoutInflater.inflate(R.layout.knb_error,null);

((ImageView)view.findViewById(R.id.no_info_img)).setImageResource(serverErrorImageId);
            view.setOnClickListener(clickListener);
            return view;
        }
        return null;
    }
    @Override
    public void onClick(View v) {
        if (refreshListener != null) {
            refreshListener.refresh();
        }
    }

    public void detch() {
        refreshListener = null;
        statusView = null;
        mProgress = null;
        mOverView = null;
        mCircleProgressContainer = null;
    }

    public interface RefreshListener {
        void refresh();
    }
}
}
```

```
public class BaseFragment extends Fragment {
    protected EventBus eventBus;
    protected Navigator navigator;
    private Unbinder unbinder;
    public StatusProvider statusProvider;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        navigator = getAppComponent().navigator().get();
        eventBus = EventBus.getDefault();
        registerEventBus(configDefaultRigsterFlags());
    }


    @Override
    public void onViewCreated(final View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        unbinder = ButterKnife.bind(this, view);
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        unbinder.unbind();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        statusProvider.detch();
        eventBus.unregister(this);
    }
    protected int configDefaultRigsterFlags() {
        return DefaultRigisterFlags.NOT_NEED_DEFAULT_REGISTER;
    }



    private void registerEventBus(int flag) {
        if (eventBus.isRegistered(this))
            return;
        if (flag == DefaultRigisterFlags.STICKY_SUBSCRIBER) {
            eventBus.registerSticky(this);
        } else if (flag == DefaultRigisterFlags.NEED_DEFAULT_REGISTER) {
            eventBus.register(this);
        }
    }

}

public abstract class MvpFragment<P extends MvpPresenter> extends
BaseFragment
        implements BaseMvpCallback<P>{

    protected  P presenter;
    public abstract P createPresenter();

    @Override
    public P getPresenter() {
        return presenter;
    }

    @Override
    public void setPresenter(@Nullable P presenter) {
        this.presenter = presenter;
    }

    @Override
    public ILoadView getMvpView() {
        return (ILoadView) this;
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        presenter = createPresenter();
        getPresenter().onCreate(savedInstanceState);
    }




    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        getPresenter().onViewCreated(view, savedInstanceState);
        getPresenter().attachView(getMvpView());
    }


    public void onResume(){
        super.onResume();
        if (!isHidden()) {
            getPresenter().onResume();
        }
    }


    public void onDestroyView(){
        super.onDestroyView();
        getPresenter().onDestroyView();
        getPresenter().detachView();
    }
}
```

```
public interface BaseMvpCallback<P extends MvpPresenter> {

    P createPresenter();

    P getPresenter();

    void setPresenter(P presenter);

    ILoadView getMvpView();
```