# 1、关于框架

所有项目采用一套通用基础框架

1. 人员切换项目只需要熟悉业务。

2. 任意员工代码结构一致，降低程序员之间沟通以及日后维护的成本

3. 重用大量代码，多人完成的工具类可以放置在框架中

项目框架：<http://code.qdedu.cn/zhl-client/androidBaseLib.git>

框架demo：<http://code.qdedu.cn/zhl-client/KangraooApp.git>

4.框架中涵盖了安全开发，自动化开发等实用功能

5.框架包含多模块

## **1.1架构与基类**

框架设计架构采用mvc，mvp，mvvm多种混合架构来提供给项目，根据不同的业务可选择不同的架构，来增强项目的可适应性，降低代码复杂程度

### **1.1.1mvc基础类**

|  |  |  |
| --- | --- | --- |
| 类名 | 父类 | 说明 |
| BFragment | Fragment |  |
| BLazyFragment | BFragment | 废弃 |
| BActivity | Activity |  |
| IBaseView | 无 | 包含基础View抽象方法 |
| IBackFragment |  | 重写返回键的fragment |

### **1.1.2MVP基础类**

|  |  |  |
| --- | --- | --- |
| 类名 | 父类 | 说明 |
| BMvpActivity | AppCompatActivity | MVP基础activiyt |
| BMvpFragment | Fragment | MVP基础fragment |
| BaseRxPresenter | BasePresenter | 不再使用rxjava废弃 |
| BasePresenter | 无 | 包含基本Presenter方法 |
| IRxPresenter | 无 | 包含基础rx抽象方法 |

### **1.1.3MVVM基础类**

|  |  |  |
| --- | --- | --- |
| 类名 | 父类 | 说明 |
| BVMActivity | BActivity |  |
| BVMFragment | BFragment |  |
| BViewModel | ViewModel |  |
| BStateActionEvent | 无 |  |
| BDVMFragment | BVMFragment | 代有databing的fragment |
| BDVMActivity | BVMActivity | 代有databing的activity |

### 1.1.4架构扩展类

|  |  |  |
| --- | --- | --- |
| 类名 | 父类 | 说明 |
| IAction |  | 业务层扩展类 |
| BasePageAction | IAction | 多页请求业务扩展实现类 |
| IBasePageView |  | 多页请求业务扩展接口 |

## **1.2开源库使用**

### **1.2.1retrofit2**

架构中构建了两种实现方式，以满足不同需求

表单请求与json请求

fun getDefaultFromRetrofit(baseUrl: String): HRetrofit {  
 val factory: MutableList<Converter.Factory> = ArrayList()  
 factory.add(ScalarsConverterFactory.create())  
 factory.add(LibConverterFactory.create())  
 return instance(baseUrl, factory)  
 }  
  
fun getDefaultJsonRetrofit(baseUrl: String): HRetrofit {  
 val factory: MutableList<Converter.Factory> = ArrayList()  
 factory.add(ScalarsConverterFactory.create())  
 factory.add(JsonConverterFactory.create())  
 return HRetrofit.instance(baseUrl, factory)  
}

### **1.2.2logger**

ULog封装了logger

主要方法：

|  |  |
| --- | --- |
| 方法 | 参数 |
| d,e,w,i,v,wtf | 多参数 |
| dm,em,wm,im,vm,wtfm | 具有格式字符，多参数 |
| dt,et,wt,it,vt,wtft | 自定义tag，多参数 |
| dmt,emt,wmt,imt,vmt,wtfmt | 自定义tag，格式字符，多参数 |
| json | 打印json |
| xml | 打印xml |
| o | 打印对象 |
| ds,es,ws,is,vs,wtfs | 简单打印，tag，单参数 |

### **1.2**.3glide

图像工具，框架中将缓存默认存于框架的构建目录中

### **1.2.4okhttp3**

架构中争对请求做了如下处理

1、可自建DNS

2、支持多url请求多header与urlparams添加

UOkHttp是对该第三方的构建，项目中理论建立一个Okhttp对象

默认发送几个header

1. 请求平台platform android

2. 版本说明version 当前app版本

默认请求超时

1. 连接超时10秒

2. 写入超时30秒

3. 读取超时30妙

### 1.2.5Arouter

多模块

### 1.2.6moshi

kotlin 最好用的json解析

### 1.2.7mmkv

替换系统sharepreferces存储以便支持安全与多进程处理

### 1.2.8room

android数据库

### 1.2.9okdownload

断点下载工具

# **二、架构设计与基本结构**

## 2.1架构设计基本思路：

1、模块化

2、资源可复用性

基本库 include ':basektlib'

## 2.2可复用模块

include ':base-module-update' 项目更新

include ':base-module-video' 项目视频库

include ':base-module-share' 项目分享库

include ':base-module-reading' 项目阅读功能

include ':base-module-kefu' 项目客服集成

include ':base-module-reader' 阅读器使用

include ':base-module-web' 网页展示

## 2.3可复用lib

include ':base-lib-picselect' 图片选择库

include ':baselib-common' 公用资源库

## 2.4基本框架结构

app 项目管理包

data 数据层管理包

exception 异常类管理包

service 服务类管理包

tools 工具包

ui 页面管理包

widget 项目view组件管理包

### 2.4.1项目类部分设计

ActivityLifeManager 管理所有activity的生命周期，支持多进程多任务栈管理

AppManager 管理所有app操作

TaskExecutor 项目线程池，基于cpu密集型的单例线程池，支持队列FIFO或LIFO

### 2.4.2数据类部分设计

采用外观设计模式设计

数据&网络层

请求实体类必须继承BParam

BaseLocalDataSource 本地数据源基类（可选择继承）

用于存储到mmvk，数据库，缓存数据，等本地数据

BaseRemoteDataSource网络数据源基类（可选择继承）

用于请求网络数据

BaseRepository 数据仓库基类（可选择继承）

用于切换本地与网络之间的数据，通过数据仓库来缓存网络数据，并切换本地与网络源

### 2.4.3tools工具类设计

常用工具类

数据检查

奔溃工具

文件系统

日志工具

权限工具

安全工具

json解析

字体图标

键盘工具

系统工具

### 2.4.4组件类设计

常用弹框

toolbar组件

空页面组件

# 三、框架其他说明

## **3.1框架gradle工具**

### **3.1.1加固工具**

applicationVariants.all { variant ->  
 variant.outputs.all { output ->// each 改为 all  
 //apk命名格式 如 渠道名\_versionName\_打包时间 QD-READING\_5.0.0.18071802\_2018071910.apk  
 def fileName = "${variant.productFlavors[0].name}\_${buildType.name}\_${variant.versionName}.apk"  
 def outFile = output.outputFile  
 if (outFile != null && outFile.name.endsWith('.apk')) {  
 outputFileName = fileName // output.outputFile 改为 outputFileName  
 }  
 }  
 }

def types = ["assembleAcademy\_releaseRelease" : "academy\_release"]  
 this.gradle.taskGraph.afterTask { task ->//每个task执行结束时会调用这个方法  
 def match = false  
 types.each { k, v ->  
 if (task.name == k) {  
 match = true  
 }  
 }  
 if (match) {  
 def outPath = getBuildDir().getAbsolutePath() + File.separator +  
 "outputs" + File.separator +  
 "apk" + File.separator +  
 types[task.name] + File.separator +  
 "release" + File.separator  
 def appFilePath = outPath + types[task.name] + "\_release\_" + productFlavors.academy\_release.versionName + ".apk"//这里的versionName无法动态获取  
  
 println "appFilePath=" + appFilePath  
  
 if (!new File(appFilePath).exists()) {  
 println "apk not exist"  
 return  
 }  
 executeCmd('java.exe -jar ' + rootDir + File.separator + "jiagu" + File.separator + "autojiagu.jar " + rootDir + " " + appFilePath + " " + file(rootProject.ext.signingConfigs['storeFile']).getCanonicalPath()+ ' ' + rootProject.ext.signingConfigs['storePassword'] + ' ' + rootProject.ext.signingConfigs['keyAlias'] + ' ' + rootProject.ext.signingConfigs['keyPassword'], rootDir)  
 }  
 }def executeCmd(cmd, m) {  
 println "开始加固"  
  
 def p = cmd.execute()  
 println p.text  
 p.waitFor()  
 println p.exitValue()  
  
 println "加固完成"  
}

## **3.2框架自动化工具**

### 3.2.1API自动生成工具

配置文件 api.properties

Update=[\  
 {"title":"versionsByCode","url":"update/v1.0/versionsByCode","type":"GET","remarks":"升级"}\  
 ]

recipe.xml.ftl

<?xml version="1.0"?>  
<recipe>  
 <merge from="ApiMethods.java.sdw" to="data/service/ApiMethods.kt" />  
 <merge from="ApiService.java.sdw" to="data/service/ApiService.kt" />  
 <merge from="DataSource.java.sdw" to="data/source/{{dataSource}}.kt" />  
 <merge from="LocalDataSource.java.sdw" to="data/source/local/{{localDataSource}}.kt" />  
 <merge from="RemoteDataSource.java.sdw" to="data/source/remote/{{remoteDataSource}}.kt" />  
 <merge from="Repository.java.sdw" to="data/source/{{repository}}.kt" />  
 <instantiate from="Params.java.page.sdw"  
 to="data/model/params/{{params}}.kt" />  
 <instantiate from="Response.java.page.sdw"  
 to="data/model/responses/{{response}}.kt" />  
 <instantiate from="DataSource.java.page.sdw"  
 to="data/source/{{dataSource}}.kt" />  
 <instantiate from="LocalDataSource.java.page.sdw"  
 to="data/source/local/{{localDataSource}}.kt" />  
 <instantiate from="RemoteDataSource.java.page.sdw"  
 to="data/source/remote/{{remoteDataSource}}.kt" />  
 <instantiate from="Repository.java.page.sdw"  
 to="data/source/{{repository}}.kt" />  
</recipe>

### 3.2.2UI自动生成工具

配置文件 autotemplate.properties

TestAuto1={"ui":"Activity","type":"mvvm","remarks":"TestAuto1"}  
TestAuto2={"ui":"Activity","type":"mvc","remarks":"TestAuto2"}  
TestAuto3={"ui":"Activity","type":"mvp","remarks":"TestAuto3"}  
  
TestAutoF1={"ui":"Fragment","type":"mvvm","remarks":"TestAutoF1"}  
TestAutoF2={"ui":"Fragment","type":"mvc","remarks":"TestAutoF2"}  
TestAutoF3={"ui":"Fragment","type":"mvp","remarks":"TestAutoF3"}  
  
TestAutoL1={"ui":"Activity","type":"mvvm","biz":"list","remarks":"TestAutoL1"}  
TestAutoL3={"ui":"Activity","type":"mvp","biz":"list","remarks":"TestAutoL3"}  
  
TestAutoFL1={"ui":"Fragment","type":"mvvm","biz":"list","remarks":"TestAutoFL1"}  
TestAutoFL3={"ui":"Fragment","type":"mvp","biz":"list","remarks":"TestAutoFL3"}

类型说明：type：mvvm，mvp，mvc 生成三种架构

mvc架构由于业务简单不涉及list生成

mvvm与mvp涉及 biz 业务生成

remarks：生成注释

ui：生成类型，只生成Fragment和Activity

autorecipe.xml.ftl

<?xml version="1.0"?>  
<recipe>  
  
 <mergexml from="AndroidManifest.xml.sdw" to="AndroidManifest.xml" />  
 <instantiate from="Activity.kt.page.sdw"  
 to="activity/{{activityClass}}.kt" />  
 <instantiate from="Fragment.kt.page.sdw"  
 to="fragment/{{activityClass}}.kt" />  
 <instantiate from="View.kt.page.sdw"  
 to="view/{{viewClass}}.kt" />  
 <instantiate from="Presenter.kt.page.sdw"  
 to="presenter/{{presenterClass}}.kt" />  
 <instantiate from="Adapter.kt.page.sdw"  
 to="adapter/{{adapterClass}}.kt" />  
 <instantiate from="ViewModel.kt.page.sdw"  
 to="viewmodel/{{viewModelClass}}.kt" />  
  
 <instantiateres from="activity\_layout.xml.page.sdw"  
 to="layout/{{layoutName}}.xml" />  
 <instantiateres from="fragment\_layout.xml.page.sdw"  
 to="layout/{{layoutName}}.xml" />  
 <instantiateres from="adapter\_item\_layout.xml.page.sdw"  
 to="layout/{{itemlayoutName}}.xml" />  
  
</recipe>

page.xml,生成ui列表记录

<?xml version="1.0" encoding="UTF-8"?>  
  
<record>   
 <insert ui="TestAutoFL3"/>   
 <insert ui="TestAutoL1"/>   
 <insert ui="TestAuto1"/>   
 <insert ui="TestAuto2"/>   
 <insert ui="TestAutoFL1"/>   
 <insert ui="TestAuto3"/>   
 <insert ui="TestAutoF2"/>   
 <insert ui="TestAutoF1"/>   
 <insert ui="TestAutoF3"/>   
 <insert ui="TestAutoL3"/>   
 <insert ui="TestAutoFToA1"/>   
 <insert ui="TestAutoFToA2"/>   
 <insert ui="TestAutoFToA3"/>   
 <insert ui="TestAutoFToAL1"/>  
 <insert ui="TestAutoFToAL3"/>  
</record>

### 3.2.3快速跳转ui库

project : build.gradle 项目的gradle配置

buildscript { repositories { maven { url 'https://www.jitpack.io' } }

在你的每个需要做容易跳转的模块添加如下配置

1. 你的项目必须要支持 kapt
2. kotlin kapt
3. 你的项目必须支持 @Parcelize 注解 也就是必须添加 apply plugin: 'kotlin-android-extensions'

apply plugin: 'kotlin-android-extensions'apply plugin: 'kotlin-kapt'android {androidExtensions { experimental = true }} kapt com.github.smartbackme.AutoPage:autopage-processor:1.0.8 implementation com.github.smartbackme.AutoPage:autopage:1.0.8

bundle 支持的基本类型都支持（除ShortArray） 以下类型都支持，如果类型不是如下类型，可能会报kapt错误

:Parcelable String Long Int Boolean Char Byte Float Double Short CharSequence CharArray IntArray LongArray BooleanArray DoubleArray FloatArray ByteArray ArrayList<Int> ArrayList<String> ArrayList<CharSequence> ArrayList<:Parcelable> Array<:Parcelable>

## 3.3框架人性化

1、崩溃优化：奔溃后的app不会直接退出，会给当前用户提示信息，之后再退出。

2、开发优化：日志包含每个Activity，Fragment地址，点击日志就可跳转对应页面，减少调试时间。

3、网络优化：网络可以多域名，不同加密算法来请求数据。

4、数据层自动生成：网络层自动通过配置构建，提高网络层开发效率。

5、Debug环境切换：通过注册debug配置环境来切换app环境。

6、Console控制台：悬浮框console日志

7、国际化支持：目前包含中文简体，繁体，日语，英语

8、文件系统：自动构建本地文件系统

## 3.4框架安全性

框架采用Aes对称加密算法

1、mmkv：可选择存储安全的数据，数据一旦存储，无法在其他终端进行解读，只能在当前设备的app中读取。其他设备无法进行备份数据操作。

2、加固配置：集成的app打包自动加固（加固后的项目，无法进行反编译，无法动态调试）

3、异常栈日志：框架用LOG把异常栈输出，通过debug状态来输出，防止通过异常栈来泄漏框架关键信息。

4、网络安全：可有效防止dns劫持

注：项目使用框架的加固方案和aes加密的情况下可保证项目极高的安全性

## 3.5框架性能

1、从启动到展示页面，采用了懒加载以及线程池等策略提升架构性能

2、从运行角度：采用大量单例模式来减少系统创建对象开销

# 四、框架使用说明

参见demo

# 五、基于本框架的代码质量自动化分析

为了方便自动管控部分代码质量，框架提供质量管理分析：

集成分析代码：

1：build.gradle 基本配置

plugins {  
 id("io.gitlab.arturbosch.detekt").version("1.5.0")  
}  
  
allprojects {  
 repositories {  
 google()  
 jcenter()  
 }  
 apply plugin: 'io.gitlab.arturbosch.detekt'  
}  
  
detekt {  
 failFast = true // fail build on any finding  
 buildUponDefaultConfig = true // preconfigure defaults  
 config = files("$projectDir/config/detekt.yml")  
 // point to your custom config defining rules to run, overwriting default behavior  
 baseline = file("$projectDir/config/baseline.xml")  
 // a way of suppressing issues before introducing detekt  
  
 reports {  
 html.enabled = true // observe findings in your browser with structure and code snippets  
 xml.enabled = true // checkstyle like format mainly for integrations like Jenkins  
 txt.enabled = true  
 // similar to the console output, contains issue signature to manually edit baseline files  
 }  
}

运行./gradlew tasks可以看到相关任务被加进去了:

detektdetektBaseline - Creates a detekt baseline on the given --baseline path.detektGenerateConfig - Generate a detekt configuration file inside your project.detektIdeaFormat - Uses an external idea installation to format your code.detektIdeaInspect - Uses an external idea installation to inspect your code.

2 app/build.gradle配置

apply plugin: 'com.android.application'  
apply plugin: 'kotlin-android'  
apply plugin: 'kotlin-android-extensions'  
apply plugin: 'kotlin-kapt'  
  
configurations {  
 ktlint  
}  
  
dependencies {  
 ktlint rootProject.ext.dependencies.tools.ktlint  
 detektPlugins ("io.gitlab.arturbosch.detekt:detekt-formatting:$detekt\_version")}  
  
task ktlint(type: JavaExec, group: "verification") {  
 description = "检查kotlin 风格"  
 classpath = configurations.ktlint  
 main = "com.pinterest.ktlint.Main"  
 args "src/\*\*/\*.kt"  
 // to generate report in checkstyle format prepend following args:  
 // "--reporter=plain", "--reporter=checkstyle,output=${buildDir}/ktlint.xml"  
 // see https://github.com/pinterest/ktlint#usage for more  
}  
check.dependsOn ktlint  
  
task ktlintFormat(type: JavaExec, group: "formatting") {  
 description = "修复 kotlin 风格"  
 classpath = configurations.ktlint  
 main = "com.pinterest.ktlint.Main"  
 args "-F", "src/\*\*/\*.kt"  
}

### klint使用

检查:

./gradlew ktlint

自动修改:

./gradlew ktlintFormat

并不是所有问题都可以自动修改, 有的问题需要手动改.

配置中的这句:

check.dependsOn ktlint

把运行检查加到了check task中. 通过:

./gradlew check --dry-run

可以查看运行check task都会做哪些事情(dry run表示不用实际执行这些任务).

### detekt使用

运行:

./gradlew detekt

进行检测.

detekt {}

块是用来进行自定义的属性设置的.

如果开启了文件输出, 结果在:

app/build/reports/detekt目录下可以查看.并且运行./gradlew check --dry-run可以发现, 运行check的时候也会执行detekt.

可以利用:

./gradlew detektGenerateConfig

生成配置文件: config/detekt/detekt.yml.

在配置文件中可以看到对各种规则的开关状态, 编辑它可以进行定制.

### 添加 git hook

在 commit 代码时检测规则与质量是最好的时机

具体文件地址： config/detekt/pre.commit

#!/usr/bin/env bash## 方案一： https://arturbosch.github.io/detekt/git-pre-commit-hook.html# 将该文件放到project根目录的.git/hooks/目录下，执行 chmod +x pre-commit 为脚本添加可执行权限即可，# 之后的每次commit都会先执行 checkstyle task 进行代码检查## 方案二： https://stackoverflow.com/questions/427207/can-git-hook-scripts-be-managed-along-with-the-repository# 从git 2.9开始, 可以设置:core.hooksPath了.# 可以在repo里面添加一个目录hooks, 然后把git hooks文件放进去track.# 在命令行跑:# git config core.hooksPath [文件路径]# 把找hook文件的目录设置成指定目录就好了.echo "执行 kotlin 代码规范检查..."./gradlew detektEXIT\_CODE=$?if [ $EXIT\_CODE -ne 0 ]; then echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" echo " Detekt failed " echo " Please fix the above issues before committing " echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" exit $EXIT\_CODEfi#echo "执行 checkstyle 任务 "#./gradlew checkstyle

如果 git 版本 > 2.9，在项目更目录下执行

git config core.hooksPath ./config