BaseModel技术分析文档

Base公用库概述：

> 项目中公用的基类，管理类，工具类等等的集合依赖库，与业务逻辑无关，可以随时提供给其余项目使用。

Base库基本说明(按照功能分包)：

- activity: BaseActivity和BaseNetActivity的封装
- binding: 使用databinding的自定义属性绑定的基类封装
- json: 用于解析json数据
- model: 用于存放BaseViewModel中包含的model
- mvvm: 对Mvvm模式的Activity和Fragment的封装
- net: 对网络模块的封装，本项目采用的网络框架为:retrofit + 协程
- util: 通用工具类的统一目录
- view: 通用自定义View的目录
- viewmodel: 基类的viewmodel封装

Base模块中部分功能使用注意事项和示例：

- 如果某个module需要使用databinding，需要在该module的build.gradle的文件中加入

```
android {
      kotlinOptions {
        jvmTarget = JavaVersion.VERSION_1_8
    }

    buildFeatures {
        dataBinding = true
    }

    kapt {
        generateStubs = true
    }
}
```

- BaseNetActivity: 包含了网络请求的基类Activity，封装成了Mvvm+databind+liveData的模式,包含childView,empty,error,load等状态View，并且可以支持自定义异常状态布局。实现了View层与Model层双向绑定

  类定义：

```
/**
 * desc   :包含网络状态，数据状态的基类Activity T:对应界面布局的dataBind对象 VM:对应界面的ViewModel对象
 */
abstract class BaseNetActivity<V : ViewDataBinding, VM : BaseNetViewModel> : BaseVMActivity() {
}
```

关键变量，方法

```
/**
 * 初始化base的dataBind对象，并且注册lifecycle
 */
private val baseBinding: ActivityBaseNetBinding by lazy {
    DataBindingUtil.setContentView<ActivityBaseNetBinding>(this, R.layout.activity_base_net)
        .apply {
            lifecycleOwner = this@BaseNetActivity
        }
}

/**
 * 获取子类布局的ID
 * @return Int 子类布局的ID
 */
protected abstract fun getLayoutId(): Int

/**
 * 初始化子类的ViewModel
 * @return VM 子类的ViewModel
 */
protected abstract fun initViewModel(): VM

/**
 * 定义BaseViewModel
 */
private lateinit var baseViewModel: VM

/**
 * 定义子类的View，用于跟子类的dataBind进行绑定
 */
private var childView: View? = null

/**
```

```
 * 定义子类的dataBing对象
 */
protected lateinit var dataBind: V


override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    //获取子类初始化的ViewModel
    baseViewModel = initViewModel()
    baseBinding.viewModel = baseViewModel
    //开启base的liveData的数据变化监听
    startBaseObserve()
    //初始化子类的布局
    initChild()
    //开启子类的LiveData监听
    startObserve()
    //初始化子类的View
    initView()
    //初始化子类的数据
    initData()
}

/**
 * 初始化子类布局
 */
private fun initChild() {
    if (!baseBinding.baseChildView.isInflated) {
        baseBinding.baseChildView.viewStub?.layoutResource = getLayoutId()
        childView = baseBinding.baseChildView.viewStub?.inflate()
        if (childView != null) {
            dataBind = DataBindingUtil.bind(childView!!)!!
        }
    }
}

/**
 * 开始监控baseViewModel的数据变化，包含网络状态，标题栏，以及错误类的布局加载
 */
private fun startBaseObserve() {

    baseViewModel.baseStatusModel.observe(this, Observer {
        setStatus(it)
    })

    baseViewModel.baseTitleModel.observe(this, Observer {
        setTitle()
```

```
    })
}
```

布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:bind="http://schemas.android.com/apk/res-auto">

    <data>
        <!-- 声明BaseNetActivity所需要的ViewModel -->
        <variable
            name="viewModel"
            type="com.rm.baselisten.viewmodel.BaseNetViewModel" />
    </data>


    <androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/clBaseContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        >

        <!-- BaseNetActivity的TitleBar -->
        <ViewStub
            android:id="@+id/baseTitleLayout"
            android:layout_width="match_parent"
            android:layout_height="48dp"
            android:layout="@layout/base_layout_title"
            bind:title="@{viewModel.baseTitleModel}"
            tools:ignore="MissingConstraints" />

        <!-- BaseNetActivity的子类布局容器 -->
        <ViewStub
            android:id="@+id/baseChildView"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_constraintTop_toBottomOf="@+id/baseTitleLayout"
            />

        <!-- BaseNetActivity的正在加载中布局容器 -->
```

```
        <ViewStub
            android:id="@+id/baseLoad"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            bind:status="@{viewModel.baseStatusModel}"
            />

        <!-- BaseNetActivity的网络错误布局容器 -->
        <ViewStub
            android:id="@+id/baseError"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            bind:status="@{viewModel.baseStatusModel}"
            />

        <!-- BaseNetActivity的空数据布局容器 -->
        <ViewStub
            android:id="@+id/baseEmpty"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            bind:status="@{viewModel.baseStatusModel}"

            />

    </androidx.constraintlayout.widget.ConstraintLayout>


</layout>
```

dataBind使用示例

1:在databinding的布局文件中引入对应的Model

2:在View中使用@{model.xxx} 或者@={model.xxx} 进行数据绑定

3:在对应的加载datadinding布局的地方使用 DataBindingUtil.setContentView()或者DataBindingUtil.bind()将数据与

databind关联起来，再将获取到的ViewModel实体数据赋值给databinding中的viewModel引用，并开启数据监听，即可实现

数据的双向绑定。

小提示：databinding可以使用@BindingAdapter注解自定义属性解析或者@BindingConversion来实现统一的某些View效果

例如，在xml中使用isVisible属性去控制View的显示与隐藏

1:先定义一个BindingViewVisible(名字可以自己定义)的类

```
@BindingAdapter("isVisible")
fun View.isVisible(isVisible: Boolean) {
    visibility = if (isVisible) View.VISIBLE else View.GONE
}
```

2:在xml文件中引入对应的model，并使用isVisible的属性，就相当于这个布局的visible属性与status这个model的属性值相关联了

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>
        <import type="android.view.View"/>
        <import type="com.rm.baselisten.model.BaseNetStatus"/>

        <variable
            name="status"
            type="com.rm.baselisten.model.BaseStatusModel" />
    </data>

    <LinearLayout
        android:id="@+id/ll_loading"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        isVisible="@{status.netStatus == BaseNetStatus.BASE_SHOW_LOADING}"
        android:gravity="center">

        <ProgressBar
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

    </LinearLayout>
```

```
</layout>
```

使用Mvvm+LiveData创建具体的业务：

- 因为项目中使用koin(kotlin官方)依赖注入框架，所以在新建ViewModel时，需要在模块初始化时，加入ViewModel的注入过程(该过程具体流程请参考component-comm的README文件)
- 定义继承BaseNetActivity的具体业务类(示例使用LoginActivity)

```
class LoginActivity : BaseNetActivity<ActivityLoginBinding, LoginViewModel>()
{
}
```

- 初始化loginViewModel(这里可以直接使用koin依赖注入框架的viewModel方法获取对应已经注册过的viewmodel)

```
private val loginViewModel by viewModel<LoginViewModel>()
```

- 开启liveData数据的监听

```
    override fun startObserve() {
        loginViewModel.apply {
            uiState.observe(this@LoginActivity, Observer {

 loginViewModel.baseStatusModel.postValue(BaseStatusModel(BaseNetStatus.BASE_
SHOW_LOADING))

                it.isSuccess?.let {

 loginViewModel.baseStatusModel.postValue(BaseStatusModel(BaseNetStatus.BASE_
SHOW_CONTENT))
                    Toast.makeText(this@LoginActivity, "登陆成功",
Toast.LENGTH_LONG).show()
                }

                it.isError?.let { err ->

 loginViewModel.baseStatusModel.postValue(BaseStatusModel(BaseNetStatus.BASE_
SHOW_CONTENT))
                }
            })
        }
    }
```

布局文件如下

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

<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:binding="http://schemas.android.com/tools">

    <data>

        <variable
            name="viewModel"
            type="com.rm.module_mine.login.LoginViewModel" />

        <variable
            name="title"
            type="com.rm.module_mine.bean.Title" />
    </data>


    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">


        <com.google.android.material.textfield.TextInputLayout
            android:id="@+id/userNameLayout"
            style="@style/MineTextInputLayout"
            android:layout_marginTop="80dp"
            android:hint="@string/mine_username">

            <androidx.appcompat.widget.AppCompatEditText
                android:id="@+id/userNameEt"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:singleLine="true"
                android:text="@={viewModel.userName}"
                binding:afterTextChanged="@{viewModel.verifyInput}" />

        </com.google.android.material.textfield.TextInputLayout>


        <com.google.android.material.textfield.TextInputLayout
            android:id="@+id/pswLayout"
            style="@style/MineTextInputLayout"
```

```
            android:hint="@string/mine_password"
            app:passwordToggleEnabled="true">

            <androidx.appcompat.widget.AppCompatEditText
                android:id="@+id/passwordEt"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="textPassword"
                android:singleLine="true"
                android:text="@={viewModel.passWord}"
                binding:afterTextChanged="@{viewModel.verifyInput}" />

        </com.google.android.material.textfield.TextInputLayout>


        <com.google.android.material.button.MaterialButton
            android:id="@+id/login"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="20dp"
            android:layout_marginTop="20dp"
            android:layout_marginRight="20dp"
            android:background="@color/businessColorPrimary"
            android:enabled="@{viewModel.uiState.enableLoginButton}"
            android:onClick="@{() -> viewModel.login()}"
            android:text="@string/mine_login"
            android:textColor="@color/businessWhite"
            android:textSize="16sp" />

        <com.google.android.material.button.MaterialButton
            android:id="@+id/register"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="20dp"
            android:layout_marginTop="20dp"
            android:layout_marginRight="20dp"
            android:background="@color/businessColorPrimary"
            android:enabled="@{viewModel.uiState.enableLoginButton}"
            android:onClick="@{() -> viewModel.register()}"
            android:text="@string/mine_register"
            android:textColor="@color/businessWhite"
            android:textSize="16sp" />

    </LinearLayout>

</layout>
```