目录

早期的Android混沌架构带来了哪些问题?

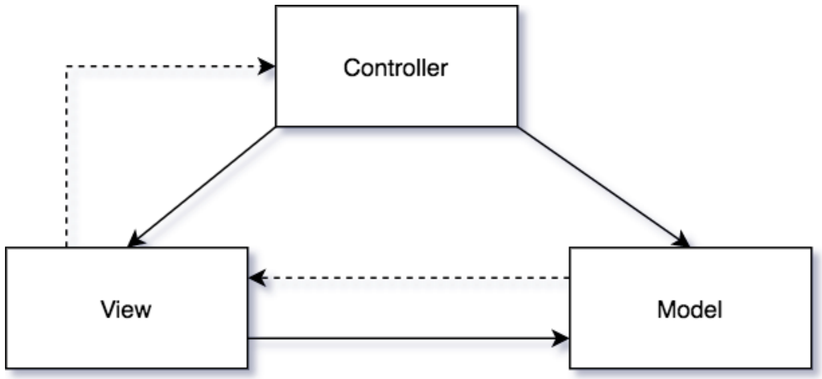
```
// God Activity
public final class UsersActivity extends ListActivity {
	@Override protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		//...
		new ListUsers().execute();
	}

	private final class ListUsers extends AsyncTask<Void, Void, Void> {
		@Override protected Void doInBackground(Void... voids) {
			// final SQLiteOpenHelper sqLiteOpenHelper = ...
			// JsonObjectRequest jsObjRequest = new JsonObjectRequest
			// (Request.Method.GET, url, null, new Response.Listener<JSONObject>() {
			// MySingleton.getInstance(this).addToRequestQueue(jsObjRequest);
			// showData(user);
			return null;
		}
	}
}
```

MVC(Model－view－controller)

MVC模式（Model－view－controller）是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。

MVC模式最早由Trygve Reenskaug在1978年提出[1]，是施乐帕罗奥多研究中心（Xerox PARC）在20世纪80年代为程序语言Smalltalk发明的一种软件架构。**MVC模式**的目的是实现一种动态的程序设计，使后续对程序的修改和扩展简化，并且使程序某一部分的重复利用成为可能。除此之外，此模式透过对复杂度的简化，使程序结构更加直观。软件系统透过对自身基本部分分离的同时也赋予了各个基本部分应有的功能。

-- Wikipedia



通用MVC模式

MVC(Model - view - controller)

问题:

MVC(Model - view - controller)

问题:

没有一个标准架构, 不同的人能设计出不同的MVC架构.

MVC(Model - view - controller)

问题:

没有一个标准架构, 不同的人能设计出不同的MVC架构.

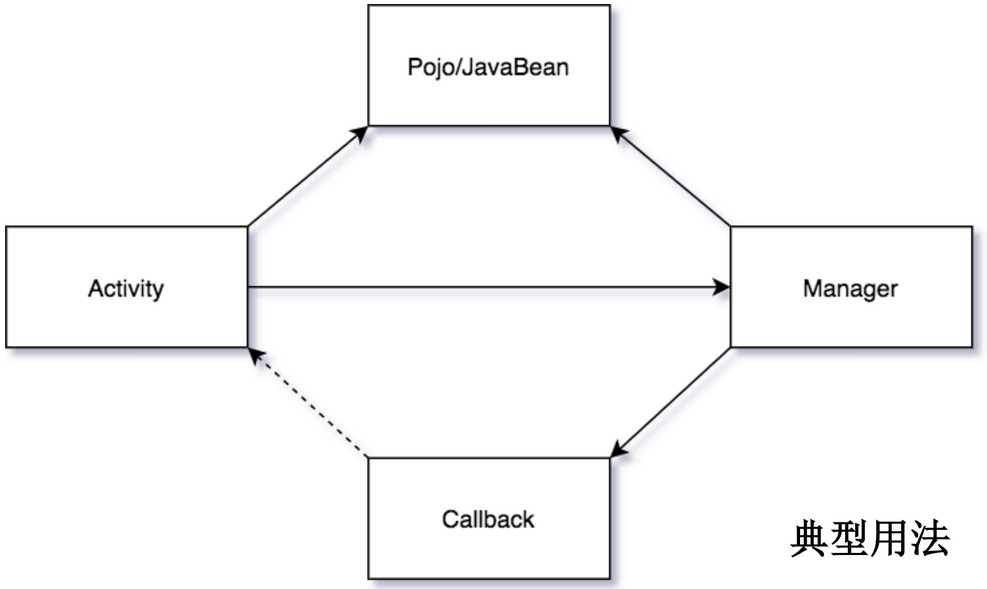
Actvity到底属于View还是Controller?

MVC(Model - view - controller)

问题:

没有一个标准架构, 不同的人能设计出不同的MVC架构.

Actvity到底属于View还是Controller?



典型用法

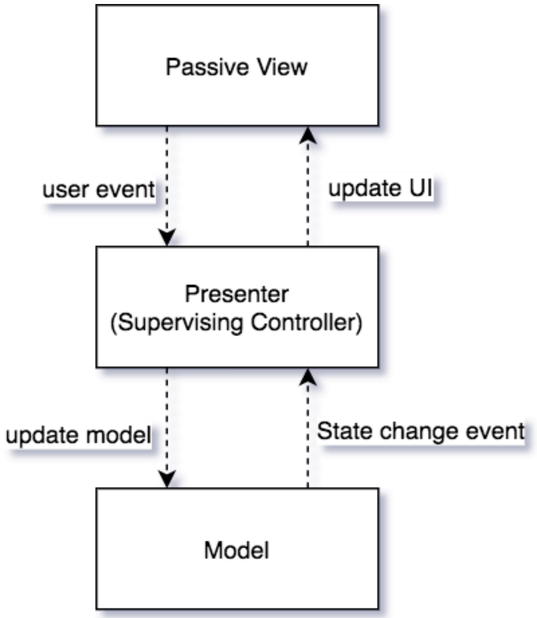
MVP(Model-view-presenter)

Model-view-presenter，简称MVP，是电脑软件设计工程中一种对针对MVC模式，再审议后所延伸提出的一种软件设计模式,被广泛用于便捷**自动化单元测试**和在呈现逻辑中**改良分离关注点**（separation of concerns）。

MVP软件模式起源于1990年代初期，Taligent，Apple，IBM和Hewlett-Packard的合资企业。[2] MVP是Taligent 基于C ++的CommonPoint环境中用于应用程序开发的基础编程模型。该模式后来由Taligent迁移到Java，并在Taligent首席技术官Mike Potel的论文中得到推广。[3]
在1998年Taligent停产之后，Dolphin Smalltalk的 Andy Bower和Blair McGlashan 修改了MVP模式，为其Smalltalk用户界面框架奠定了基础。[4] 2006年，Microsoft开始将MVP纳入其文档和.NET框架中用于用户界面编程的示例。[5] [6]
Martin Fowler [7] 和Derek Greer的另一篇文章详细讨论了MVP模式的演变和多种变体，包括MVP与其他设计模式（例如MVC）的关系。[8]

-- Wikipedia

MVP(Model-view-presenter)



MVP(Model-view-presenter)

2016年，Google发布了Android Architecture Blueprints，用于展示Android应用的架构模式，其中就有todo-mvp，定义了一套标准的架构设计原型(MVP)

后来加入了Android Architecture Component，用于演示各种架构组件的组合. 在2018年Google I/O，将Android Architecture Component作为其中一部分并入Jetpack

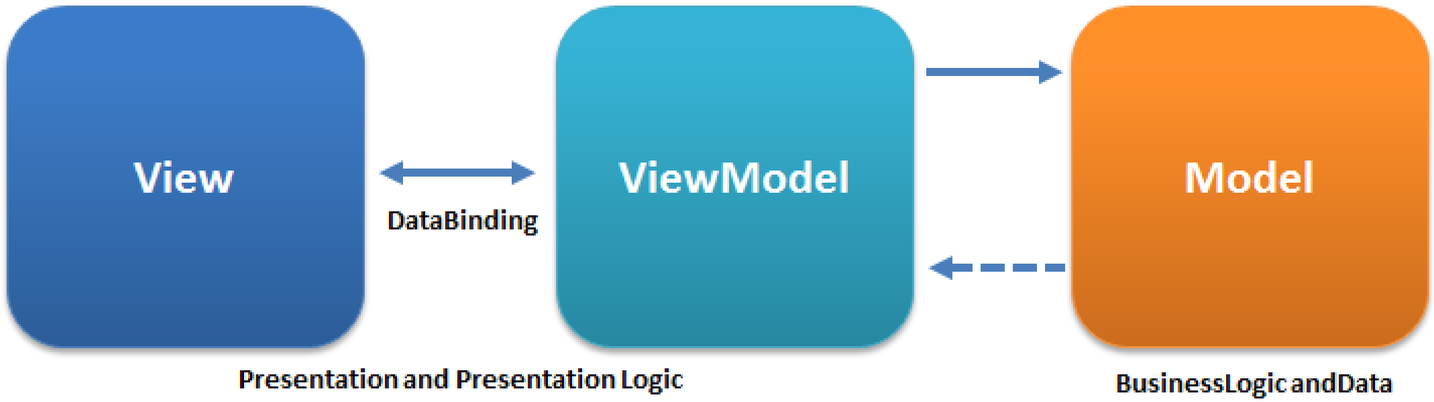
MVVM（Model－view－viewmodel）

MVVM（Model－view－viewmodel）是一种软件架构模式。
MVVM有助于将图形用户界面的开发与业务逻辑或后端逻辑的开发分离开来，这是通过标记语言或GUI代码实现的。
MVVM的视图模型(ViewModel)是一个值转换器，[1] 这意味着视图模型(ViewModel)负责从模型中暴露（转换）数据对象，以便轻松管理和呈现对象。

在这方面，视图模型(ViewModel)比视图做得更多，并且处理大部分视图的显示逻辑。[1] 视图模型可以实现中介者模式，组织对视图所支持的用例集的后端逻辑的访问。

-- Wikipedia

MVVM（Model－view－viewmodel）



MVVM 模式

DataBinding组件 和 lifeCycle组件

DataBinding 和 ViewModel

2015年Google I/O, 发布preview 版的databinding组件

2016年3月, 发布了Android Architecture Blueprints, 包含了DataBinding

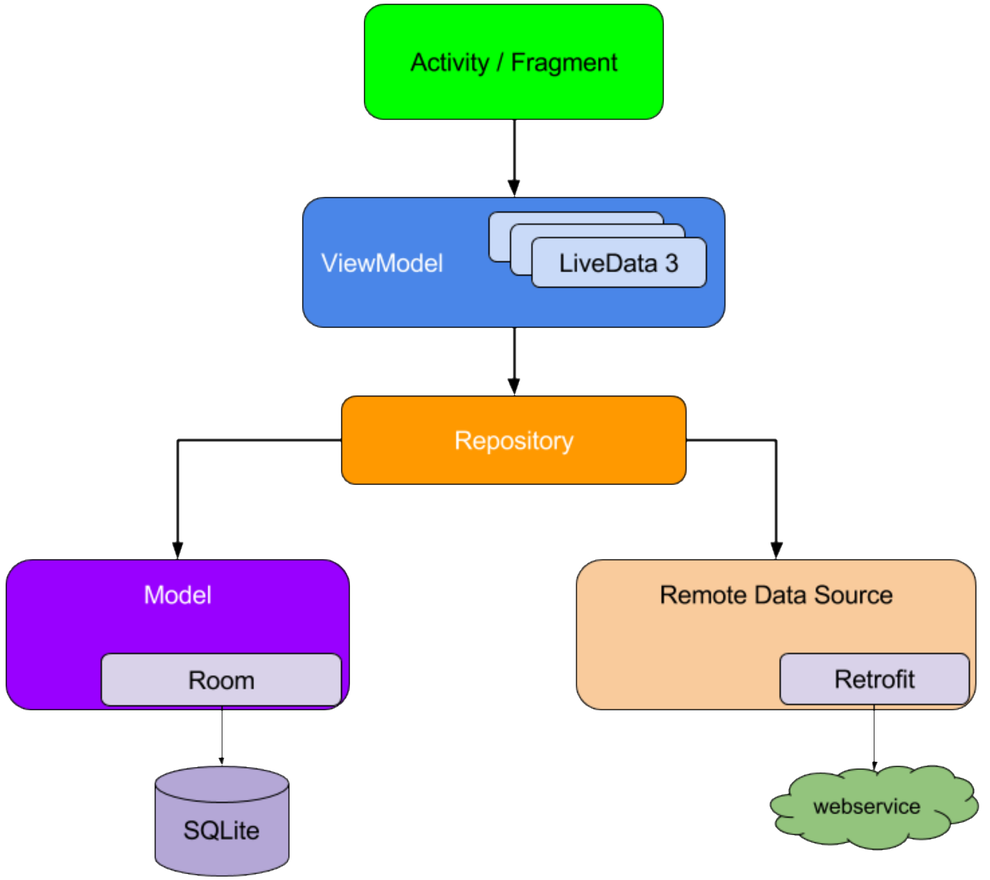
2017年Google I/O, 发布了Architecture Component. 其中就包含 ViewModel/Room/LiveData

同年, 在Android Architecture Blueprints加入了MVVM框架, 但是RxJava, LiveData, DataBinding是分开的, 需要自己设计整合

2018年失踪了一整年几乎没有代码提交(怀疑是去整jetpack去了)

2019年, 发布了V2版本的Blueprint

如何设计

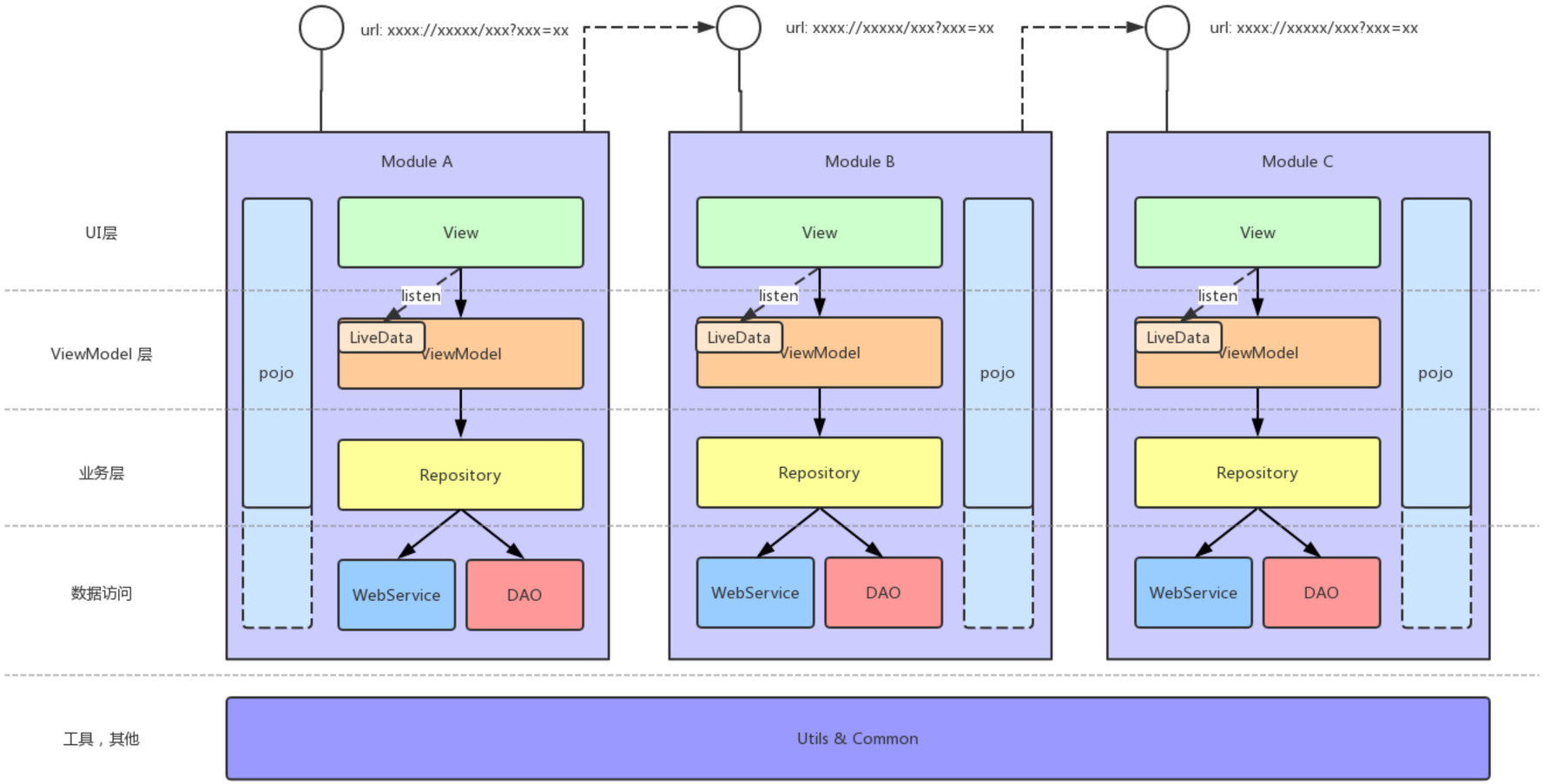


Google 推荐的一种设计

设计原则

- 分离关注点
- 通过ViewModel驱动UI

我们的框架


url: xxxx://xxxxx/xxx?xxx=xx
url: xxxx://xxxxx/xxx?xxx=xx
url: xxxx://xxxxx/xxx?xxx=xx
Module A
Module B
Module C
UI层
ViewModel 层
业务层
数据访问
工具，其他
View
listen
LiveData
ViewModel
pojo
Repository
WebService
DAO
Utils & Common

架构简介

App采用纵向分模块，横向分层的方式设计，将App分为若干模块，每个模块分为4层，模块中各个子模块职责如下：

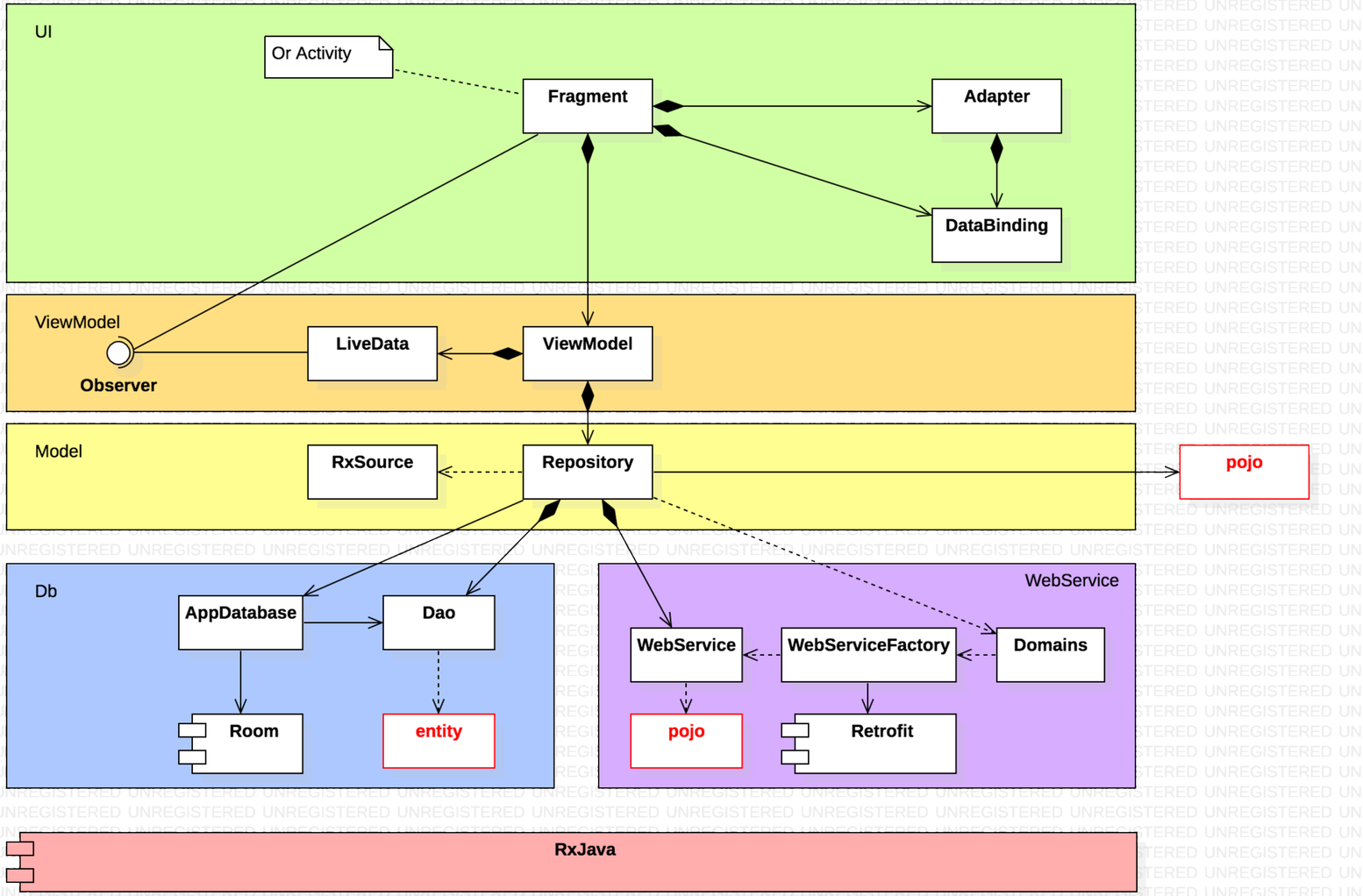
- View： 负责UI的展示，以及界面的跳转，负责监听LiveData的改变，此外DataBinding也是放到View这一层的
- ViewModel：负责存放UI展示用的数据，使用LiveData作为存储，该层不处理任何UI相关逻辑，通过数据驱动UI的改变
- WebService：网络数据访问，并封装成Java对象
- DAO：本地数据库数据访问，并封装成Java对象
- Repository：负责组装WebService和DAO的数据接口，封装业务接口给上层使用
- pojo：作为数据对象在View/ViewModel,Repository中传递使用，如果不需要转换的话，也可以直接作为WebService的pojo和Dao的entity使用

模块之间进行隔离，使用路由框架(ARouter)进行UI跳转

架构设计

- 基于分离关注点的原则
- 定义每一层职责
- 使用响应式编程（RxJava）作为层与层之间数据的传递
- 使用LiveData作为ViewModel通知UI的桥梁

详细设计-引入RxJava



各层的关注点

Dao：

- 负责提供对数据库的访问，返回RxJava的Source对象（~~Observable~~, Single, ~~Maybe~~, Completable...），这里强烈建议使用Single
- 使用Room实现

各层的关注点

WebService：

- 负责对网络进行访问，返回RxJava的Source对象（~~Observable~~, Single(推荐), ~~Maybe~~, Completable...）
- 基于Retrofit实现

各层的关注点

Repository:

- 将WebService和DAO的数据进行组合,串联, 将数据返回给上层
- 通常来说, 返回RxJava的Source对象

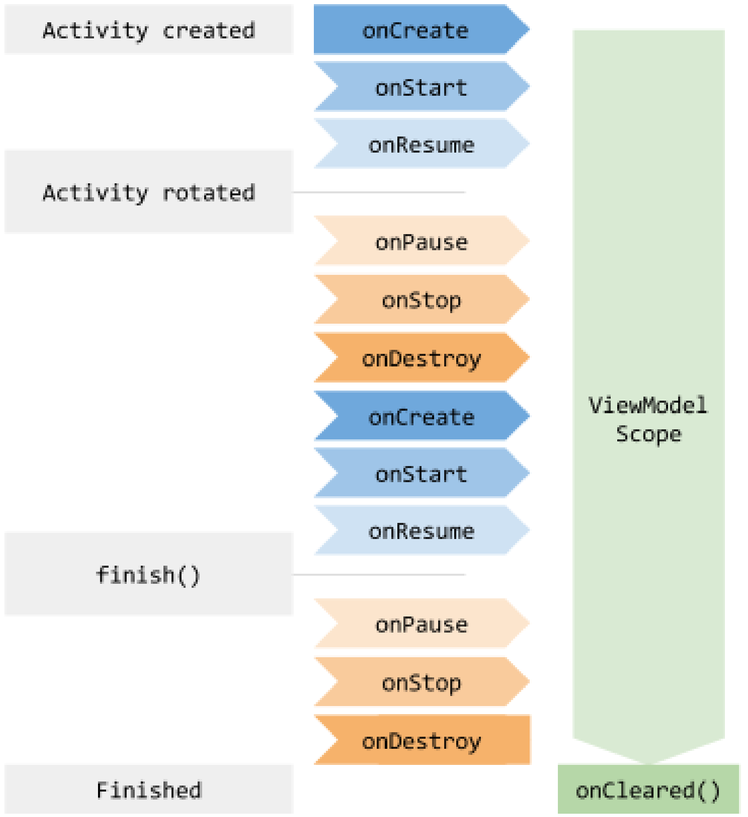
ViewModel

ViewModel：

- 负责管理View所使用的数据，不关注UI，也不持有ui的对象
- 负责向Repository发出调用，得到RxJava的Source对象，此时网络请求尚未开始
- 订阅（Subscribe）并监听Source对象，此时会进行开始网络请求，数据库访问
- Source发射结束后，将结果更新到LiveData中
- 不关注UI的逻辑!!!

Android ViewModel

- 生命周期持续到LifeCycle组件的完成时(Finished，不是销毁时)
- 在configChange事件时，能够保留到下次onCreate
- 在生命周期结束时，会调用onCleard
- 实现原理
  - #onRetainNonConfigurationInstance()
  - #getLastNonConfigurationInstance()



ViewModel周期

问题-共享ViewModel

只能够在同一个Activity中的Fragment中共享

本质上是获取上一层(Activity)的ViewModel

```
public class SharedViewModel extends ViewModel {
    private final MutableLiveData<Item> selected = new MutableLiveData<Item>();

    public void select(Item item) {
        selected.setValue(item);
    }

    public LiveData<Item> getSelected() {
        return selected;
    }
}

public class MasterFragment extends Fragment {
    private SharedViewModel model;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        model = ViewModelProviders.of(getActivity()).get(SharedViewModel.class);
        itemSelector.setOnClickListener(item -> {
            model.select(item);
        });
    }
}

public class DetailFragment extends Fragment {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        SharedViewModel model = ViewModelProviders.of(getActivity()).get(SharedViewModel.class);
        model.getSelected().observe(this, { item ->
           // Update the UI.
        });
    }
}
```

UI

- 负责渲染
- 负责通知ViewModel发起数据变化的请求(请求更新,写入,etc…)
  - 监听LiveData数据并改变对应的ui
  - 使用DataBinding直接绑定LiveData，可以使用双向绑定，但需要注意循环调用的问题
- 最复杂的一层

开发约束

- 禁止反向依赖
- 禁止跨层调用

其他组件

Dagger/butterKnife:

- butterKnife核心能力databinding都能实现
- Dagger上手过于复杂，并且会增加项目复杂度不利于新人理解，如果对这块都比较的团队，可以使用.

Paging:

- 推荐使用，这里不是必须就没放了，一般需要做二次封装

Navigation:

- 新业务可以考虑接入

确定需求

- 使用公开Api：https://gank.io/api
- 拉取四组，其中：
  - 福利 1条
  - Android 20条
  - 前端20条
  - 视频1条
- 分组展示
- 存入db，下次启动时候优先从db获取

「Talk is cheap. Show me the code」

Web Service

```
public interface GankService {

}
```

```
public interface GankService {

    String CATEGORY_FULI = "福利";
    String CATEGORY_ANDROID = "Android";
    String CATEGORY_VIDEO = "休息视频";
    String CATEGORY_FRONT_END = "前端";

    /**
     * 获取某一个分类下的数据, 可以返回Observable, Flowable, Maybe, 建议返回Single
     * @param category 分类名
     * @param count 请求数量
     * @param page 请求的页面序号
     */
    @GET("data/{category}/{count}/{page}")
    Single<ResultJson> getByCategory(
            @Path("category") String category,
            @Path("count") int count,
            @Path("page") int page);
}
```

```
public interface GankService {

    String CATEGORY_FULI = "福利";
    String CATEGORY_ANDROID = "Android";
    String CATEGORY_VIDEO = "休息视频";
    String CATEGORY_FRONT_END = "前端";

    /**
     * 获取某一个分类下的数据, 可以返回Observable, Flowable, Maybe, 建议返回Single
     * @param category 分类名
     * @param count 请求数量
     * @param page 请求的页面序号
     */
    @GET("data/{category}/{count}/{page}")
    Single<ResultJson> getByCategory(
            @Path("category") String category,
            @Path("count") int count,
            @Path("page") int page);
}
```

```java
/**
 * 用于生成WebService对象的辅助类
 */
public class WebServiceFactory {
    /**
     * 创建Web Service Object的入口
     * @param clz WebService的类
     * @return 对应WebService对象
     */
    public static <T> T service(Class<T> clz) {
        Retrofit retrofit = new Retrofit
                .Builder()
                // 域名
                .baseUrl("https://gank.io/api/")
                // 支持RxJava
                .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                // 支持Gson
                .addConverterFactory(GsonConverterFactory.create())
                .build();
        return retrofit.create(clz);
    }
}
```

DAO

```
public abstract /* interface */ class GankDao {

}
```

```
@Dao
public abstract /* interface */ class GankDao {

}
```

```java
@Dao
public abstract class GankDao {

    // 插入数据
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    abstract void insertGank(List<Gank> ganks);

    // 清空整个表
    @Query("DELETE FROM Gank")
    abstract void nukeTable();

    /**
     * 清空整个表, 再插入数据
     */
    @Transaction
    public void nukeThenInsert(List<Gank> ganks) {
        nukeTable();
        insertGank(ganks);
    }

    /**
     * 返回全部数据
     */
    @Query("SELECT * FROM Gank")
    public abstract Single<List<Gank>> getAll();

}
```

```java
@Database(entities = {Gank.class}, version = 1, exportSchema = true)
public abstract class AppDatabase extends RoomDatabase {

    private static AppDatabase instance;

    /**
     * 获取gank表DAO
     */
    public abstract GankDao gankDao();

    public static AppDatabase getInstance() {
        if (instance == null) {
            synchronized (AppDatabase.class) {
                if (instance == null) {
                    instance = Room.databaseBuilder(DemoApplication.getINSTANCE(), AppDatabase.class,
            "demo.db").build();
                    return instance;
                }
            }
        }
        return instance;
    }

}
```

Repository

```java
public class GankRepository {

    private final GankDao dao;
    private final GankService service;

    public GankRepository() {
        service = WebServiceFactory.service(GankService.class);
        dao = AppDatabase.getInstance().gankDao();
    }

    /**
     * 只用于单元测试的构造函数
     * @param service
     * @param dao
     */
    @VisibleForTesting
    GankRepository(GankDao dao, GankService service) {
        this.service = service;
        this.dao = dao;
    }

}
```

```java
/**
 * 获取某一个分组的数据
 * @param category 分组名
 * @param count 需要的数据条数
 * @return Single
 */
Single<List<Gank>> getByCategory(String category, int count) {
    return service.getByCategory(category, count, 1)
            .subscribeOn(Schedulers.io()) // IO(网络线程)线程执行
            .observeOn(Schedulers.single()) // db线程执行下面的
            .map(new Function<ResultJson, List<Gank>>() { // 转成list形式
                @Override
                public List<Gank> apply(ResultJson resultJson) throws Exception {
                    return convertToGank(resultJson.getResults());
                }
            })
            .doOnSuccess(new Consumer<List<Gank>>() { // 成功的话, 需要插入数据库
                @Override
                public void accept(List<Gank> results) throws Exception {
                    dao.nukeThenInsert(results);
                }
            });
}
```

```
/**
 * 获取某一个分组的数据
 * @param category 分组名
 * @param count 需要的数据条数
 * @return Single
 */
Single<List<Gank>> getByCategory(String category, int count) {
    return service.getByCategory(category, count, 1)
            .subscribeOn(Schedulers.io()) // IO(网络线程)线程执行
            .observeOn(Schedulers.single()) // db线程执行下面的
            .map(new Function<ResultJson, List<Gank>>() { // 转成list形式
                @Override
                public List<Gank> apply(ResultJson resultJson) throws Exception {
                    return convertToGank(resultJson.getResults());
                }
            })
            .doOnSuccess(new Consumer<List<Gank>>() { // 成功的话, 需要插入数据库
                @Override
                public void accept(List<Gank> results) throws Exception {
                    dao.nukeThenInsert(results);
                }
            });
}
```

```
/**
 * 获取某一个分组的数据
 * @param category 分组名
 * @param count 需要的数据条数
 * @return Single
 */
Single<List<Gank>> getByCategory(String category, int count) {
    // 转成list形式
    // 成功的话, 需要插入数据库
    return service.getByCategory(category, count, 1)
            .subscribeOn(Schedulers.io()) // IO(网络线程)线程执行
            .observeOn(Schedulers.single()) // db线程执行下面的
            .map(json -> convertToGank(json.getResults()))
            .doOnSuccess(results -> dao.nukeThenInsert(results));
}
```

```
/**
 * 获取某一个分组的数据
 * @param category 分组名
 * @param count 需要的数据条数
 * @return Single
 */
Single<List<Gank>> getByCategory(String category, int count) {
    // 转成list形式
    // 成功的话, 需要插入数据库
    return service.getByCategory(category, count, 1)
            .subscribeOn(Schedulers.io()) // IO(网络线程)线程执行
            .observeOn(Schedulers.single()) // db线程执行下面的
            .map(json -> convertToGank(json.getResults()))
            .doOnSuccess(dao::nukeThenInsert);
}
```

```kotlin
/**
 * 获取某一个分组的数据
 * @param category 分组名
 * @param count 需要的数据条数
 * @return Single
 */
fun getByCategory(
    category: String?,
    count: Int
): Single<List<Gank>> { // 转成list形式
    // 成功的话, 需要插入数据库
    return service.getByCategory(category, count, 1)
        .subscribeOn(Schedulers.io()) // IO(网络线程)线程执行
        .observeOn(Schedulers.single()) // db线程执行下面的
        .map { convertToGank(it.results) }
        .doOnSuccess { dao.nukeThenInsert(it) }
}
```

确定需求

- 使用公开Api：https://gank.io/api
- 拉取四组，其中：
  - 福利 1条
  - Android 20条
  - 前端20条
  - 视频1条
- 分组展示
- 存入db，下次启动时候优先从db获取

```
    /**
     * 从网络获取今日数据
     */
    @SuppressLint("CheckResult")
    @VisibleForTesting(otherwise = VisibleForTesting.PRIVATE)
    Single<TodayGanks> getTodayGanksFromWeb() {
        return Single.zip( // 这里是一次性发所有请求, 也可以一条一条发
                service.getByCategory(GankService.CATEGORY_FULI, 1, 1).subscribeOn(Schedulers.io()), // 福利
                service.getByCategory(GankService.CATEGORY_ANDROID, 20, 1).subscribeOn(Schedulers.io()), // Android
                service.getByCategory(GankService.CATEGORY_FRONT_END, 20, 1).subscribeOn(Schedulers.io()), // 前端
                service.getByCategory(GankService.CATEGORY_VIDEO, 1, 1).subscribeOn(Schedulers.io()), // 视频
                new Function4<ResultJson, ResultJson, ResultJson, ResultJson, TodayGanks>() {
                    @Override
                    public TodayGanks apply(ResultJson t1, ResultJson t2, ResultJson t3, ResultJson t4) throws Exception {
                        List<Gank> fuliList, androidList, frontList, videoList;
                        if (!t1.isError() && !t2.isError() && !t3.isError() && !t4.isError()) {
                            fuliList = convertToGank(t1.getResults());
                            androidList = convertToGank(t2.getResults());
                            frontList = convertToGank(t3.getResults());
                            videoList = convertToGank(t4.getResults());
                            return new TodayGanks(fuliList, androidList, frontList, videoList);
                        } else {
                            throw new IOException("Get Error From Server");
                        }
                    }
                })
//                .subscribeOn(Schedulers.io()) // 网络线程执行上面的(没必要)
                .observeOn(Schedulers.single()) // db线程执行下面的
                .doOnSuccess(new Consumer<TodayGanks>() {
                            @Override
                            public void accept(TodayGanks todayGanks) throws Exception {
                                List willInsert = (List) (new ArrayList());

                                willInsert.addAll(todayGanks.getFuli());
                                willInsert.addAll(todayGanks.getAndroid());
                                willInsert.addAll(todayGanks.getFrondEnd());
                                willInsert.addAll(todayGanks.getVideo());
                                dao.nukeThenInsert(willInsert);// 执行成功, 写入db
                            }
                        }
                );
    }
```

```
    /**
     * 从网络获取今日数据
     */
    @SuppressLint("CheckResult")
    Single<TodayGanks> getTodayGanksFromWeb() {
        return Single.zip( // 这里是一次性发所有请求, 也可以一条一条发
                    service.getByCategory(GankService.CATEGORY_FULI, 1, 1).subscribeOn(Schedulers.io()), // 福利
                    service.getByCategory(GankService.CATEGORY_ANDROID, 20, 1).subscribeOn(Schedulers.io()), // Android
                    service.getByCategory(GankService.CATEGORY_FRONT_END, 20, 1).subscribeOn(Schedulers.io()), // 前端
                    service.getByCategory(GankService.CATEGORY_VIDEO, 1, 1).subscribeOn(Schedulers.io()), // 视频
                    this::mergeResults)
//                  .subscribeOn(Schedulers.io()) // 网络线程执行上面的(没必要)
                .observeOn(Schedulers.single()) // db线程执行下面的
                .doOnSuccess(this::replaceDbEntity);
    }

    private TodayGanks mergeResults(ResultJson t1, ResultJson t2, ResultJson t3, ResultJson t4) {
        // ...
    }

    private void replaceDbEntity(TodayGanks todayGanks) {
        // ...
    }
```

```java
/**
 * 从db获取缓存数据
 */
private Single<TodayGanks> getTodayGankFromDb() {
    return dao.getAll()
            .map(this::listToTodayGanks)
            .subscribeOn(Schedulers.single()); // db线程处理
}

/**
 * 优先从从db获取, 如果失败则从网络获取
 */
public Single<TodayGanks> getTodayGanks() {
    return getTodayGankFromDb()
            .flatMap(todayGanks ->
                    (todayGanks == null)
                            ? getTodayGanksFromWeb()
                            : Single.just(todayGanks)); // 如果数据为空则总网络获取
}
```

ViewModel

```
public class MainFragmentViewModel extends BaseAndroidViewModel {

}
```

```
public class MainFragmentViewModel extends BaseAndroidViewModel {

    public MutableLiveData<TodayGanks> todayGanks;
    public MutableLiveData<Throwable> error;
    public MutableLiveData<Boolean> isLoading;
    private GankRepository repo;

}
```

```java
public class MainFragmentViewModel extends BaseAndroidViewModel {

    private static final String TAG = "DemoViewModel";

    public MutableLiveData<TodayGanks> todayGanks;
    public MutableLiveData<Throwable> error;
    public MutableLiveData<Boolean> isLoading;
    private GankRepository repo;

    public MainFragmentViewModel(@NonNull Application app) {
        super(app);
        this.todayGanks = new MutableLiveData();
        this.error = new MutableLiveData();
        this.isLoading = new MutableLiveData();
        this.repo = new GankRepository();
        initLoad();
    }

}
```

```
/**
 * 首次加载
 */
@SuppressLint("CheckResult")
@MainThread
private void initLoad() {
    isLoading.postValue(true);
    repo.getTodayGanks()
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(it -> {
                isLoading.postValue(false);
                todayGanks.postValue(it);
            }, throwable -> {
                isLoading.postValue(false);
                error.postValue(throwable);
            });
}
```

```
/**
 * 更新
 */
@SuppressLint("CheckResult")
@MainThread
public void update() {
    if (!isLoading.getValue()) { // 避免重入
        isLoading.postValue(true);
        repo.getTodayGanksFromWeb()
                .subscribeOn(AndroidSchedulers.mainThread())
                .subscribe(it -> {
                    isLoading.postValue(false);
                    todayGanks.postValue(it);
                }, throwable -> {
                    isLoading.postValue(false);
                    error.postValue(throwable);
                });
    }
}
```

View(UI)

```xml
<?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">
    <data>
        <variable name="viewModel" type="com.oppo.mvvm_demo.main.viewmodel.MainFragmentViewModel"/>
    </data>

    <androidx.swiperefreshlayout.widget.SwipeRefreshLayout
            android:id="@+id/swipe_refresh"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:refreshing="@{viewModel.isLoading}"
            app:onRefreshListener="@{() -> viewModel.update()}">
        <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/gank_list"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager">
        </androidx.recyclerview.widget.RecyclerView>
    </androidx.swiperefreshlayout.widget.SwipeRefreshLayout>
</layout>
```

```
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    viewModel = ViewModelProviders.of(this).get(MainFragmentViewModel.class);
}

@Nullable
public View onCreateView(@NotNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle                    savedInstanceState) {
    binding = DataBindingUtil.inflate(inflater, R.layout.main_fragment, container, false);
    binding.setLifecycleOwner(this);// 必须设置这个, 不然viewModel中的livedata改变不会收到通知
    adapter = new GankRecyclerAdapter(new ItemClickCallback() {
        public void onClick(Gank gank) {
            onGankClick(gank);
        }
    });

    binding.setViewModel(viewModel);
    binding.gankList.setAdapter(adapter);
    viewModel.error.observe(this, new Observer<Throwable>() {
        @Override
        public void onChanged(Throwable it) {
            onError(it);
        }
    });

    viewModel.todayGanks.observe((LifecycleOwner) this,
            new Observer<TodayGanks>() {
                @Override
                public void onChanged(TodayGanks todayGanks) {
                    adapter.setTodayGanks(todayGanks.toList());
                }
            }
    );

    return binding.getRoot();
}
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<!--福利-->
<layout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto">
    <data>
        <variable name="gank" type="com.oppo.mvvm_demo.main.pojo.Gank"/>
    </data>
    <androidx.cardview.widget.CardView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:foreground="?android:attr/selectableItemBackground"
            app:cardUseCompatPadding="true"
            app:cardCornerRadius="12dp"
            app:cardElevation="5dp">
        <ImageView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:adjustViewBounds="true"
                app:glideUrl="@{gank.url}"
                tools:ignore="ContentDescription">
        </ImageView>
    </androidx.cardview.widget.CardView>
</layout>
```

```
@Override
public RecyclerView.ViewHolder onCreateViewHolder(@NotNull ViewGroup parent, int viewType) {

    RecyclerView.ViewHolder viewHolder;

    switch (viewType) {
        case Gank.GROUP_FULI:
        case Gank.GROUP_ANDROID:
        case Gank.GROUP_FRONT_END:
        case Gank.GROUP_VIDEO:
            viewHolder = new GroupVH((TextView)(View.inflate(parent.getContext(), R.layout.group_item, null)));
            break;
        case Gank.FULI:
            viewHolder = new FuliVH(
                    DataBindingUtil.inflate(
                            LayoutInflater.from(parent.getContext()),
                            R.layout.gank_fuli_item,
                            parent,
                            false
                    )
            );
            break;
        case Gank.ANDROID:
        case Gank.FRONT_END:
        case Gank.VIDEO:
            viewHolder = new DefaultVH(
                    DataBindingUtil.inflate(
                            LayoutInflater.from(parent.getContext()),
                            R.layout.gank_item,
                            parent,
                            false
                    )
            );
            break;
        default:
            viewHolder = null;
            // impossible
    }
    return viewHolder;
}
```

```java
@Override
public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {

    switch (holder.getItemViewType()) {
        case Gank.GROUP_FULI:
            ((GroupVH) holder).getTitleView().setText((CharSequence) "福利");
            break;
        case Gank.GROUP_ANDROID:
            ((GroupVH) holder).getTitleView().setText((CharSequence) "Android");
            break;
        case Gank.GROUP_FRONT_END:
            ((GroupVH) holder).getTitleView().setText((CharSequence) "前端");
            break;
        case Gank.GROUP_VIDEO:
            ((GroupVH) holder).getTitleView().setText((CharSequence) "休息视频");
            break;
        case Gank.FULI:
            GankFuliItemBindingImpl gankFuliItemBindingImpl = ((FuliVH) holder).getBinding();
            gankFuliItemBindingImpl.setGank(this.getItem(position));
            gankFuliItemBindingImpl.executePendingBindings();
            break;

        case Gank.ANDROID:
        case Gank.FRONT_END:
        case Gank.VIDEO:
            GankItemBindingImpl gankItemBindingImpl = ((DefaultVH) holder).getBinding();
            gankItemBindingImpl.setGank(this.getItem(position));
            gankItemBindingImpl.setOnClick(this.clickCallback);
            gankItemBindingImpl.executePendingBindings();
            break;
    }
}
```

分离关注点原则

- 每一层有固定职责
- 方便测试

Mock和断言

Mock:

在测试过程中，对于某些不容易构造或者不容易获取的对象，用一个虚拟的对象来创建以便测试的测试方法。

断言(Assertion):

用于判断测试结果是否符合预期

各层测试思路和方法

UI: 使用Espresso , Mock ViewModel

ViewModel: 只需mock Repository, 可以直接使用Junit test执行(实际上很难)

Repository: 需要mock Dao和WebService, 验证以下行为

- 是否正确发起了网络请求
- 数据是否正确地保存到DB中
- 请求顺序是否如预期(例: 本地有缓存则不发起网络请求)

Dao: 使用Room自带的InMemoryDataBase进行测试, 不要使用真实DB测试

WebService: 不建议测试, 原因后面说

单元几个原则

- 不要测试有随机性/有状态的事物，如：
  - 网络
  - 真实数据库
- 非真机测试(如Robolectric)，不要测试性能
- 尽量避免测试多线程，使用一些工具转成单线程
- 慎用new Thread() / GlobalScope(kotlin)，这两种在实际情况中都很难写单元测试.
- 模块之间不要强依赖!!

代码演示，推荐使用kotlin

```kotlin
class GankDaoTest {

    // 将RxJava的异步线程转成同步执行
    @Rule
    @JvmField
    val instantRunRule = RxJavaInstantRunRule()

    private lateinit var dao: GankDao
    private lateinit var db: AppDatabase

    @Before
    fun setUp() {
        val ctx = ApplicationProvider.getApplicationContext<Context>()
        db = Room
            .inMemoryDatabaseBuilder(ctx, AppDatabase::class.java)
            .allowMainThreadQueries() /允许主线程执行数据库操作, 由于单元测试是在主线程执行,因此要打开这个开关, 避免room抛异常
            .build()
        dao = db.gankDao()
    }

    @After
    fun tearDown() {
        db.close()
    }
}
```

```
class GankDaoTest {

    // 将RxJava的异步线程转成同步执行
    @Rule
    @JvmField
    val instantRunRule = RxJavaInstantRunRule()

    private lateinit var dao: GankDao
    private lateinit var db: AppDatabase

    @Before
    fun setUp() {
        val ctx = ApplicationProvider.getApplicationContext<Context>()
        db = Room
            .inMemoryDatabaseBuilder(ctx, AppDatabase::class.java)
            .allowMainThreadQueries() /允许主线程执行数据库操作, 由于单元测试是在主线程执行,因此要打开这个开关, 避免room抛异常
            .build()
        dao = db.gankDao()
    }

    @After
    fun tearDown() {
        db.close()
    }
}
```

```kotlin
@Test
fun insertGank() {
    val testData = createTestData()
    dao.insertGank(testData)
    val dbDatas = getAllSync()

    //Junit 断言
    assertTrue(dbDatas.size == 2)
    assertEquals(dbDatas[0], testData[0])
    assertEquals(dbDatas[1], testData[1])

    // Hamcrest 方式
    assertThat(dbDatas.size, Matchers.`is`(Matchers.equalTo(2)))
    // 静态import后, 可以这么写
    assertThat(dbDatas.size, `is`(equalsTo(2)))
    assertThat(dbDatas,contains(*testData.toTypedArray()))
}
```

```kotlin
/**
 * 断言两个对象是否相等(equals)，可以传null
 */
infix fun <T> T?.shouldBe(that: T?) {
    if(that === null) {
        shouldBeNull()
    } else {
        assertThat(this, equalTo(that))
    }
}

/**
 * 顺序和内容完全相同
 */
inline infix fun <reified T> Iterable<T>.shouldContains(items: Iterable<T>) {
    assertThat(this, Matchers.contains(*items.toList().toTypedArray()))
}
```

```
// 静态import后, 可以这么写
assertThat(dbDatas.size, `is`(equalsTo(2)))
assertThat(dbDatas,contains(*testData.toTypedArray()))

dbDatas.size shouldBe 2
dbDatas shouldContains testData
```

```
// 静态import后, 可以这么写
assertThat(dbDatas.size, `is`(equalsTo(2)))
assertThat(dbDatas,contains(*testData.toTypedArray()))

dbDatas.size shouldBe 2
dbDatas shouldContains testData
```

```kotlin
class GankRepositoryTest {

    // 将RxJava的异步线程转成同步执行
    @Rule
    @JvmField
    val instantRunRule = RxJavaInstantRunRule()

    @MockK
    private lateinit var dao: GankDao
    @MockK
    private lateinit var service: GankService

    private lateinit var repo: GankRepository

    @Before
    fun setUp() {
        MockKAnnotations.init(this, relaxed = true)
        repo = GankRepository(dao, service)
    }

    @After
    fun tearDown() {}
}
```

```kotlin
@Test
fun getTodayGanksFromWeb() {
    every { service.getByCategory(any(), any(), any()) } answers {
        val type = firstArg() as String
        val cnt = secondArg() as Int
        val id = when(type){
            GankService.CATEGORY_ANDROID -> 0
            GankService.CATEGORY_FULI -> 1000
            GankService.CATEGORY_FRONT_END -> 2000
            GankService.CATEGORY_VIDEO -> 3000
            else -> 9000
        }
        createMockResult(type, cnt, id, false)
    }
    val result = repo
            .getTodayGanksFromWeb()
            .blockingGet()
    // 验证调用
    verify(exactly = 1) {
        service.getByCategory(GankService.CATEGORY_FULI, 1, 1)
        service.getByCategory(GankService.CATEGORY_ANDROID, 20, 1)
        service.getByCategory(GankService.CATEGORY_FRONT_END, 20, 1)
        service.getByCategory(GankService.CATEGORY_VIDEO, 1, 1)

        dao.nukeThenInsert(any())
    }

    // 验证输出结果是否正确
    result.fuli shouldHasSize 1
    result.android shouldHasSize 20
    result.frondEnd shouldHasSize 20
    result.video shouldHasSize 1
}
```

```
@Test
fun newInstance() {
    every { repo.getTodayGanks() } returns Single.just(TodayGanks())
    val vm = MainFragmentViewModel(ApplicationProvider.getApplicationContext(), repo)

    // 验证执行了方法
    verify { repo.getTodayGanks() }
    vm.todayGanks.value!!.isEmpty() shouldBe true
    vm.error.value shouldBe null
    vm.isLoading.value shouldBe false
}

@Test
fun updateWithException() {
    val vm = createVm()
    every { repo.refreshTodayGanks() } returns Single.fromCallable { throw RuntimeException() }
    vm.update()
    vm.error.value shouldBeInstanceOf RuntimeException::class.java
}
```

SharedViewModel

```kotlin
@Shared // 标记可以多个Activity/ViewModel共享
class MainFragmentViewModel : BaseAndroidViewModel

// 标记可以多个Activity/ViewModel共享, 没有Activity/Fragment使用的话, 3s后销毁
@Shared(maintenanceTime = 3000)
class MainFragmentViewModel : BaseAndroidViewModel

// 永不销毁, 可以用来存全局变量
@Shared(type = Shared.SharedType.SINGLETON)
class MainFragmentViewModel : BaseAndroidViewModel

// 使用方法
viewModel = HeytapViewModelProviders.of(this)[MainFragmentViewModel::class.java]
```

网络层封装

- 内部的pb协议解包
- 环境切换(正式/测试/开发…etc，http/https)
- 内容产品(大部分)登录逻辑，包括:
  - Session过期后自动续期后重发请求
  - Token失效切换匿名登录

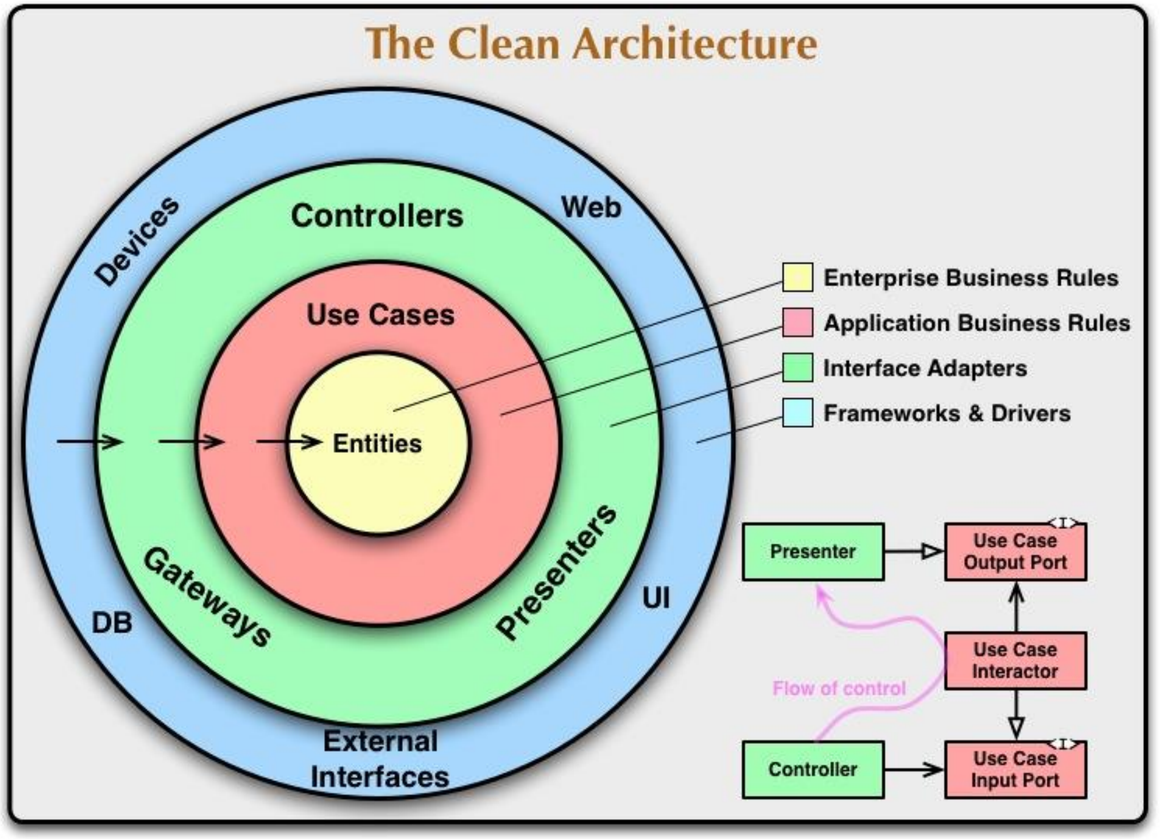
```
public final class YoliDomains {

    private static final String TAG = "YoliDomains";
    public volatile static int ENV = WebServiceFactory.NORMAL;

    // 主url配置
    @BaseUrl(value = {DomainConfig.RELEASE_HOST_HTTPS, DomainConfig.RELEASE_HOST_HTTP},
            pre_rls = {DomainConfig.PRE_RELEASE_HOST_HTTPS, DomainConfig.PRE_RELEASE_HOST_HTTP},
            dev = {DomainConfig.DEV_HOST,DomainConfig.DEV_HOST},
            test = {DomainConfig.TEST_HOST, DomainConfig.TEST_HOST})
    @CallAdapter(RxJava2)
    @ConvertFactory(O_Pb)
    class MainURL {
    }

    @BaseUrl(value = {DomainConfig.RELEASE_HOST_HTTPS, DomainConfig.RELEASE_HOST_HTTP},
            pre_rls = {DomainConfig.PRE_RELEASE_HOST_HTTPS, DomainConfig.PRE_RELEASE_HOST_HTTP},
            dev = {DomainConfig.DEV_HOST,DomainConfig.DEV_HOST},
            test = {DomainConfig.TEST_HOST, DomainConfig.TEST_HOST})
    @CallAdapter(RxJava2)
    @ConvertFactory({O_Pb, Gson})
    class NewMainURL {
    }

    public static void setENV(int ENV) {
        if (ActivityManager.isUserAMonkey()) {
            Log.i(TAG, "monkey setENV");
            YoliDomains.ENV = TEST;
        } else {
            Log.i(TAG, "setEN " + ENV);
            YoliDomains.ENV = ENV;
        }
    }
}
```

未来

2019年7月，谷歌发布了**Android Architecture Blueprints v2**，以下是master上展示的架构

- Kotlin Coroutines for background operations.
- A single-activity architecture, using the Navigation component to manage fragment operations.
- A presentation layer that contains a fragment (View) and a **ViewModel** per screen (or feature).
- Reactive UIs using **LiveData** observables and **Data Binding**.
- A **data layer** with a repository and two data sources (local using Room and remote) that are queried with one-shot operations (no listeners or data streams).
- Two **product flavors**, mock and prod, to ease development and testing (except in the Dagger branch).
- A collection of unit, integration and e2e **tests**, including "shared" tests that can be run on emulator/device or Robolectric.

未来


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port

-- Uncle Bob