

使用Android Architecture Component 开发应用（附demo）



Chuckiefan (/u/191a96d00530) [+ 关注](#)

2017.11.27 11:01* 字数 3050 阅读 447 评论 4 喜欢 9

(/u/191a96d00530)

image

相关文章：

- **【翻译】** 安卓架构组件(1)-App架构指导 ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6\(1\)-App%E6%9E%B6%E6%9E%84%E6%8C%87%E5%AF%BC.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6(1)-App%E6%9E%B6%E6%9E%84%E6%8C%87%E5%AF%BC.html))
- **【翻译】** 安卓架构组件(2)-添加组件到你的项目中 ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6\(2\)-](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6(2)-)



- %E6%B7%BB%E5%8A%A0%E7%BB%84%E4%BB%B6%E5%88%B0%E4%BD%A0%E7%9A%84%E9%A1%B9%E7%9B%AE%E4%B8%AD.html)
- 【翻译】 安卓架构组件(3)-处理生命周期 ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6\(3\)-%E5%A4%84%E7%90%86%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6(3)-%E5%A4%84%E7%90%86%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F.html))
 - 【翻译】 安卓架构组件(4)-LiveData ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6\(4\)-LiveData.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6(4)-LiveData.html))
 - 【翻译】 安卓架构组件(5)-ViewModel ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6\(5\)-ViewModel.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6(5)-ViewModel.html))
 - 【翻译】 安卓架构组件(6)-Room持久化类库 ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6\(6\)-Room%E6%8C%81%E4%B9%85%E5%8C%96%E7%B1%BB%E5%BA%93.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6(6)-Room%E6%8C%81%E4%B9%85%E5%8C%96%E7%B1%BB%E5%BA%93.html))
 - 【翻译】 安卓架构组件(7)-分页库 ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6\(7\)-%E5%88%86%E9%A1%B5%E5%BA%93.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6(7)-%E5%88%86%E9%A1%B5%E5%BA%93.html))
 - 安装包下载地址 (<https://link.jianshu.com?t=http://chuckiefan.com/2017/11/27/%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6Demo%E4%B8%8B%E8%BD%BD.html>)

今年的Google I/O大会发布了一系列的类库，统称为架构组件（Architecture Component），旨在帮助开发者构建健壮、易于测试和易于维护的应用。从我目前掌握的情况来看，这一系列类库为开发者指明了比较清晰的架构思路，确实可以显著提高应用的开发效率且对质量有一定的保证。于是我尽快翻译了官方所有相关的文档（即上面的1-6篇）。我最初接触时这些类库还处于alpha-3版本，时至今日已经发布了1.0版本，趋于稳定。同时我发现Google又发布了分页的类库，于是有了第七篇翻译。在翻译的文章中，有朋友询问能否提供一份Demo，以提供一个较为清晰直观的印象，于是就有了这篇文章。在文章的最后我会提供app的演示，安装包下载以及将项目开源至github。

本文并不会详细介绍每个类库，相关的内容还请上面的文章中查阅。本文的主要目的是使用这些类库来开发一款应用，并介绍整个的开发过程，从而看到这一系列类库的用法以及特点。



此外在整个应用的构建中我们不会涉及到所有的用法（根据我在实际项目的使用情况，本文所介绍的内容足以应付日常的需求），如果需要深入了解，请自行研究源码。

0.准备

要构建一款什么样的应用？阅读本文的大量读者都是Android应用的开发者，涉及服务端研发的并不多，为了避免增加不必要的学习曲线，我认为自行开发服务端接口是没有意义的。所以经过考虑，我选中了豆瓣的电影API (<https://link.jianshu.com?t=https://developers.douban.com/wiki/?title=guide>)，因此该应用的所有数据源皆获取自豆瓣。

所以本文会介绍一个电影信息app的开发过程。

关于本文开发中所需要了解的技术如下：

- 基本的Android开发经验，如RecyclerView之类控件的使用等
- RxJava
- Kotlin
- Retrofit2

我相信对于阅读本文的读者，这些要求并不是什么问题。在响应式编程如此火热的现在，RxJava和Retrofit已经成了很多项目的必备基础技术，而Kotlin已经成为官方宣布支持的语言。

好了，现在让我们开始。

本文会分成两个阶段编写，在第一个阶段我们会使用Android架构组件编写三个电影列表，即下文中的正在上映，即将上映和Top250。为了整个demo的完善性，电影详情界面的编写会在后面完成，截止本文发布时，只完成了第一个阶段。

1.基本的界面编写

这一部分没有什么可值得介绍的地方，但是需要说明一下我们的界面。我们会选取三个列表进行展示：

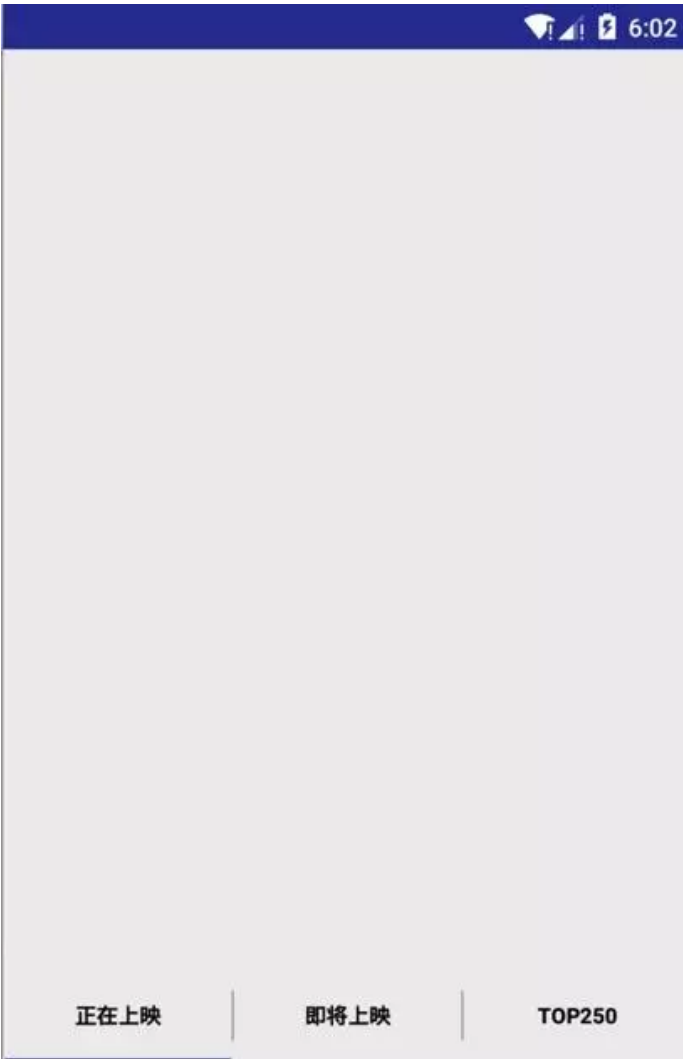
- 正在上映
- 即将上映
- Top250

正在上映是正在院线上映的电影列表，即将上映是即将在院线上映的电影列表，Top250是指评分最高的250部电影。



需要说明的是，“正在上映”和“即将上映”和具体的城市绑定，为了简化该部分对本文核心内容的影响，我们在接口数据的请求时会使用默认的城市，即北京。

界面如下：



电影列表

你可以使用任何你熟悉的类库来完成这三个Fragment,需要实现下拉加载和上拉更新等操作^[1]。

2.架构综述

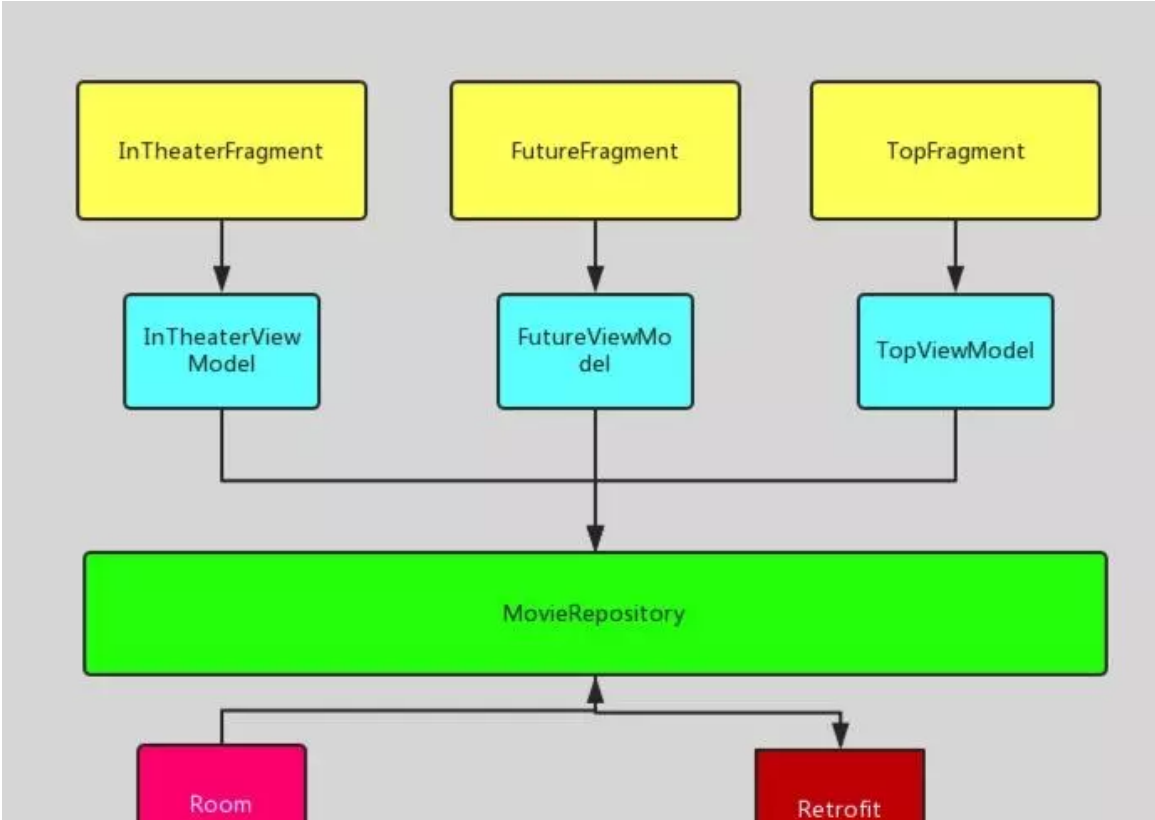
界面的编写不是什么困难的工作，真正需要我们关心的是整个应用的架构是怎样的。回顾一下第一篇文章 ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6\(1\)-App%E6%9E%B6%E6%9E%84%E6%8C%87%E5%AF%BC.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6(1)-App%E6%9E%B6%E6%9E%84%E6%8C%87%E5%AF%BC.html))，我们在这里给出我们的架构图示。

^

+

🔖

🔗



架构图示

三个Fragment会通过各自的ViewModel获取数据，而所有的ViewModel都会从MovieRepository拿到数据。在MovieRepository中，我们通过Retrofit从豆瓣API获得数据，存储在Room中，而MovieRepository则从Room获取数据。

3.网络层

现在我们开始编写这个应用的主要部分，首先从网络请求入手，即架构图示中的Retrofit部分。

3.1 豆瓣API

首先我们需要查阅豆瓣的API文档

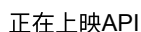
正在上映：

^

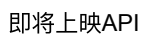
+

🔖

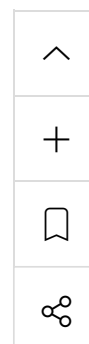
🔗



即将上映:



Top250:



Top250API

3.2接口编写

于是有了API接口的编写：

```
interface DoubanApi {

    /**
     * @param city 表示院线所在城市, 可为空, 如果为空则默认为北京市
     *
     * @return [MoviesResp]
     */
    @GET("v2/movie/in_theaters")
    fun retrieveInTheaters(@Query("city") city: String?): Observable<MoviesResp>

    /**
     * 即将上映的电影
     * @param start 开始, 默认为0
     * @param count 每次请求数量, 默认为20
     *
     * @return [MoviesResp]
     */
    @GET("v2/movie/coming_soon")
    fun retrieveComingSoon(@Query("start") start: Int?, @Query("count") count: Int?): Observable<MoviesResp>

    /**
     * Top250 评分最高的电影
     * @param start 开始, 默认为0
     * @param count 每次请求数量, 默认为20
     *
     * @return [MoviesResp]
     */
    @GET("v2/movie/top250")
    fun retrieveTop250(@Query("start") start: Int?, @Query("count") count: Int?): Observable<MoviesResp>

}
```

需要说明的是，这里的 `MoviesResp` 是豆瓣API接口返回json数据所对应的实体类，详细的内容可在demo中查看。

3.3 Retrofit编写

接下来是Retrofit的编写工作，由于本文不是讲解Retrofit的内容，因此在这里直接给出代码。

```
class DoubanRetrofit {
    private val TAG: String = this.javaClass.simpleName
    private val PAGESIZE = Constant.PAGESIZE//20
    companion object {
        private val API = buildAPI()

        private fun buildAPI(): DoubanApi {
            return Retrofit.Builder()
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
                .addConverterFactory(JacksonConverterFactory.create())
                .baseUrl(Constant.url)// "http://api.douban.com/"
                .build()
                .create(DoubanApi::class.java)
        }
        private var instance: DoubanRetrofit? = null

        @Synchronized
        fun get(): DoubanRetrofit {
            if (null == instance) {
                instance = DoubanRetrofit()
            }
            return instance!!
        }
    }

    /**
     * 正在上映
     */
    fun inTheaterMovies(): Observable<MoviesResp>{

        return API.retrieveInTheaters(null)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(Schedulers.io())
    }

    /**
     * 即将上映
     * @param start 开始位置
     */
    fun comingSoonMovies(start: Int): Observable<MoviesResp>{

        return API.retrieveComingSoon(start, PAGESIZE)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(Schedulers.io())
    }

    /**
     * 评分top250电影
     * @param start 开始位置
     */
    fun top250Movies(start: Int): Observable<MoviesResp>{
        return API.retrieveTop250(start, PAGESIZE)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(Schedulers.io())
    }
}
```



上面的代码并不是最终版本，我们会在后面进行修改

正如在架构图示所描述的那样，网络请求层获取的数据会通过ROOM存储在本地数据库，所以接下来我们要先完成持久层的编写。

4.持久层

我们已经完成了网络层的编写，接下来需要进行持久层的编写。

4.1 实体类

我们需要定义电影的实体类。由于我们应用的当前版本只展示了电影列表，那么不妨先从这个角度来看界面是怎样的：

列表项界面

于是我们有了实体类：

```
/**
 * 项目：ArchitectureComponentDemo
 * 作者：Chuckifan
 * 时间：2017/11/26 16:40
 * 内容：电影列表项，用于持久化
 * @param id 电影id
 * @param avatar 电影图片
 * @param title 电影名称
 * @param rating 电影评分
 * @param director 导演
 * @param casts 主演
 * @param genres 类型
 * @param year 年份
 * @param isInTheater true 正在上映
 * @param isComming true 即将上映
 * @param isTop250 true top250
 */
data class Movie(var id: Long = 0,
                 var avatar:String?="",
                 var title: String? = "未知", var rating: Float? = 0f, var ratingStr: String? = "未知",
                 var casts: String? = "未知", var genres: String? = "未知", var year: Int? = 0,
                 var isInTheater: Boolean = false, var isComming:Boolean = false, var isTop250:Boolean = false)
```



上面的注释已经很清晰地说明了每个成员变量的用途。

4.2 ROOM

4.2.1 实体类

如果仅仅是定义了这样的一个类，那么它和API返回数据所对应的实体类就没有任何区别了，我们接下来需要使用ROOM对这个实体类进行标记，使其可以映射为关系数据库中的数据元素：

```
/**
 * 项目 : ArchitectureComponentDemo
 * 作者 : Chuckifan
 * 时间 : 2017/11/26 16:40
 * 内容 : 电影列表项, 用于持久化
 * @param id 电影id
 * @param avatar 电影图片
 * @param title 电影名称
 * @param rating 电影评分
 * @param director 导演
 * @param casts 主演
 * @param genres 类型
 * @param year 年份
 * @param isInTheater true 正在上映
 * @param isComming true 即将上映
 * @param isTop250 true top250
 */
@Entity
data class Movie(@PrimaryKey val id: Long = 0,
                 var avatar:String?="",
                 var title: String? = "未知", var rating: Float? = 0f, var ratingStr: String? = "未知",
                 var casts: String? = "未知", var genres: String? = "未知", var year: Int? = 0,
                 var isInTheater: Boolean = false, var isComming:Boolean = false, var isTop250: Boolean = false)
```

这里插入一点和本应用编写无关的内容。我个人主张不直接使用远程服务所返回的数据直接持久化到数据库，这是为了将持久层与网络层解耦，如果我们不想使用豆瓣的数据接口还可以使用其他服务接口，比如IMDb。我们唯一需要做的工作是修改从接口数据实体类到持久化类的转化方法。

但是本文最重要的目的是介绍一个用于熟悉安卓架构组件的demo。我在本文之前曾经用Java写过一个版本，在那个版本中为了熟悉这套类库以及节省时间，我直接使用了豆瓣返回的数据，那么就涉及到实体类的嵌套，以及列表数据的映射处理，关于这部分内容的介绍请参阅这里 ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6\(6\)-Room%E6%8C%81%E4%B9%85%E5%8C%96%E7%B1%BB%E5%BA%93.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6(6)-Room%E6%8C%81%E4%B9%85%E5%8C%96%E7%B1%BB%E5%BA%93.html))。在本文不会对这部分内容进行介绍，仅仅会给出相关代码，当然这部分代码不会在最后的应用出现：

Subject:电影条目

Subject:电影条目



```
@Entity
public class Subject {
    @Embedded
    private Rating rating;
    private List<String> genres;
    private String title;
    private List<Cast> casts;
    private int collect_count;
    private String original_title;
    private String subtype;
    private List<Director> directors;
    private String year;
    @Embedded
    private Image images;
    private String alt;
    @PrimaryKey
    private String id;

    public Rating getRating() {
        return rating;
    }

    public void setRating(Rating rating) {
        this.rating = rating;
    }

    public List<String> getGenres() {
        return genres;
    }

    public void setGenres(List<String> genres) {
        this.genres = genres;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public List<Cast> getCasts() {
        return casts;
    }

    public void setCasts(List<Cast> casts) {
        this.casts = casts;
    }

    public int getCollect_count() {
        return collect_count;
    }

    public void setCollect_count(int collect_count) {
        this.collect_count = collect_count;
    }

    public String getOriginal_title() {
        return original_title;
    }

    public void setOriginal_title(String original_title) {
        this.original_title = original_title;
    }

    public String getSubtype() {
        return subtype;
    }
}
```



```
}

public void setSubtype(String subtype) {
    this.subtype = subtype;
}

public List<Director> getDirectors() {
    return directors;
}

public void setDirectors(List<Director> directors) {
    this.directors = directors;
}

public String getYear() {
    return year;
}

public void setYear(String year) {
    this.year = year;
}

public Image getImages() {
    return images;
}

public void setImages(Image images) {
    this.images = images;
}

public String getAlt() {
    return alt;
}

public void setAlt(String alt) {
    this.alt = alt;
}

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}
}
```

Rating:评分



```
public class Rating {  
  
    private int max;  
    private double average;  
    private String stars;  
    private int min;  
    public void setMax(int max) {  
        this.max = max;  
    }  
    public int getMax() {  
        return max;  
    }  
  
    public void setAverage(double average) {  
        this.average = average;  
    }  
    public double getAverage() {  
        return average;  
    }  
  
    public void setStars(String stars) {  
        this.stars = stars;  
    }  
    public String getStars() {  
        return stars;  
    }  
  
    public void setMin(int min) {  
        this.min = min;  
    }  
    public int getMin() {  
        return min;  
    }  
  
}
```

Cast:主演，最多可获得4个，数据结构为影人的简化描述，



```
public class Cast {  
  
    private String alt;  
    @Embedded  
    private Avatar avatars;  
    private String name;  
    private String id;  
    public void setAlt(String alt) {  
        this.alt = alt;  
    }  
    public String getAlt() {  
        return alt;  
    }  
  
    public Avatar getAvatars() {  
        return avatars;  
    }  
  
    public void setAvatars(Avatar avatars) {  
        this.avatars = avatars;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getId() {  
        return id;  
    }  
  
    public void setId(String id) {  
        this.id = id;  
    }  
}
```

Director:导演，数据结构为影人的简化描述



```
public class Director {  
  
    private String alt;  
    @Embedded  
    private Avatar avatars;  
    private String name;  
    private String id;  
  
    public String getAlt() {  
        return alt;  
    }  
  
    public void setAlt(String alt) {  
        this.alt = alt;  
    }  
  
    public Avatar getAvatars() {  
        return avatars;  
    }  
  
    public void setAvatars(Avatar avatars) {  
        this.avatars = avatars;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getId() {  
        return id;  
    }  
  
    public void setId(String id) {  
        this.id = id;  
    }  
}
```

Image:电影海报图，分别提供288px x 465px(大)， 96px x 155px(中) 64px x 103px(小)尺寸



```
public class Image {  
  
    private String small;  
    private String large;  
    private String medium;  
    public void setSmall(String small) {  
        this.small = small;  
    }  
    public String getSmall() {  
        return small;  
    }  
  
    public void setLarge(String large) {  
        this.large = large;  
    }  
    public String getLarge() {  
        return large;  
    }  
  
    public void setMedium(String medium) {  
        this.medium = medium;  
    }  
    public String getMedium() {  
        return medium;  
    }  
  
}
```

Avatar:影人头像，分别提供420px x 600px(大), 140px x 200px(中) 70px x 100px(小)尺寸

```
public class Avatar{  
  
    private String small;  
    private String large;  
    private String medium;  
    public void setSmall(String small) {  
        this.small = small;  
    }  
    public String getSmall() {  
        return small;  
    }  
  
    public void setLarge(String large) {  
        this.large = large;  
    }  
    public String getLarge() {  
        return large;  
    }  
  
    public void setMedium(String medium) {  
        this.medium = medium;  
    }  
    public String getMedium() {  
        return medium;  
    }  
  
}
```

以及Converter:




```
public class ListConverter {

    private static final ObjectMapper mapper = new ObjectMapper();

    @TypeConverter
    public static String strList2Json(List<String> value) {
        String result = null;
        try {
            result = mapper.writeValueAsString(value);
        } catch (IOException e) {
            LogUtils.e(e);
        }

        return result;
    }

    @TypeConverter
    public static List<String> json2StrList(String json) {
        List<String> result = null;
        try {

            JavaType javaType = getCollectionType(ArrayList.class, String.class);
            result = mapper.readValue(json, javaType);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }

    @TypeConverter
    public static String castList2Json(List<Cast> value) {
        String result = null;
        try {
            result = mapper.writeValueAsString(value);
        } catch (IOException e) {
            LogUtils.e(e);
        }

        return result;
    }

    @TypeConverter
    public static List<Cast> json2CastList(String json) {
        List<Cast> result = null;
        try {

            JavaType javaType = getCollectionType(ArrayList.class, Cast.class);
            result = mapper.readValue(json, javaType);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }

    @TypeConverter
    public static String directorList2Json(List<Director> value) {
        String result = null;
        try {
            result = mapper.writeValueAsString(value);
        } catch (IOException e) {
            LogUtils.e(e);
        }

        return result;
    }

    @TypeConverter
    public static List<Director> json2DirectorList(String json) {
```



```

List<Director> result = null;
try {
    JavaType javaType = getCollectionType(ArrayList.class, Director.class);
    result = mapper.readValue(json, javaType);
} catch (IOException e) {
    e.printStackTrace();
}
return result;
}

private static JavaType getCollectionType(Class<?> collectionClass, Class<?>...
    return mapper.getTypeFactory().constructParametricType(collectionClass, elemen
}
}

```

转换类的使用在文章 ([https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%B6%B6\(6\)-Room%E6%8C%81%E4%B9%85%E5%8C%96%E7%B1%BB%E5%BA%93.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/06/07/%E7%BF%BB%E8%AF%91-%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%B6%B6(6)-Room%E6%8C%81%E4%B9%85%E5%8C%96%E7%B1%BB%E5%BA%93.html))

中有介绍，这里不再赘述。

Room%E6%8C%81%E4%B9%85%E5%8C%96%E7%B1%BB%E5%BA%93.html)中有介绍，这里不再赘述。

Room%E6%8C%81%E4%B9%85%E5%8C%96%E7%B1%BB%E5%BA%93.html)中有介绍，这里不再赘述。

4.2.2 DAO

在使用分页库之前我们无法完整的介绍DAO的代码，这里暂时只给出写入和删除的方法：

```

@Dao
interface MovieDao {

    /**
     * 保存电影，冲突时替换
     */
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun save(movies: List<Movie>)

    /**
     * 删除正在上映的电影
     */
    @Query("DELETE FROM Movie WHERE isInTheater")
    fun removeIntheaterMovies()

    /**
     * 删除即将上映的电影
     */
    @Query("DELETE FROM Movie WHERE isComming")
    fun removeCommingSoonMovies()

    /**
     * 删除TOP250的电影
     */
    @Query("DELETE FROM Movie WHERE isTop250")
    fun removeTop250Movies()
}

```

4.2.3 Database



接下来是数据库类的编写：

```
@Database(entities = arrayOf(Movie::class),version = BuildConfig.VERSION_CODE)
abstract class MovieDatabase : RoomDatabase(){

    abstract fun movieDao():MovieDao

    companion object {
        private var instance: MovieDatabase? = null
        @Synchronized
        fun get(context: Context): MovieDatabase {
            if (instance == null) {
                instance = Room.databaseBuilder(context.applicationContext,
                    MovieDatabase::class.java, "MovieDB")
                    .build()
            }
            return instance!!
        }
    }
}
```

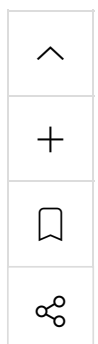
至此，持久层的编写可以暂时告一段落。

5. Repository和分页

在这一部分，我们要完成Repository的编写。正如架构图示多描述的那样，Repository层的数量往往和DAO相对应。我们需要在Repository层完成接口数据的获取以及转换为持久化数据，同时需要完成持久化数据的完成。

5.1 完善DAO

首先完善DAO中获取电影数据的部分：



```
@Dao
interface MovieDao {

    /**
     * 保存电影，冲突时替换
     */
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun save(movies: List<Movie>)

    /**
     * 删除正在上映的电影
     */
    @Query("DELETE FROM Movie WHERE isInTheater")
    fun removeInTheaterMovies()

    /**
     * 删除即将上映的电影
     */
    @Query("DELETE FROM Movie WHERE isComming")
    fun removeCommingSoonMovies()

    /**
     * 删除TOP250的电影
     */
    @Query("DELETE FROM Movie WHERE isTop250")
    fun removeTop250Movies()

    /**
     * 获取正在上映的电影
     */
    @Query("SELECT * FROM Movie WHERE isInTheater")
    fun queryInTheaterMovies():LivePagedListProvider<Int,Movie>

    /**
     * 获取正在上映的电影
     */
    @Query("SELECT * FROM Movie WHERE isComming")
    fun queryCommingsoonMovies():LivePagedListProvider<Int,Movie>

    /**
     * 获取top250的电影
     */
    @Query("SELECT * FROM Movie WHERE isTop250 ORDER BY rating DESC")
    fun queryTOP250Movies():LivePagedListProvider<Int,Movie>
}
```

截止目前，这篇文章不会涉及到LiveData的使用，我会尽快完善电影详情部分，到时候会补充LiveData的内容。

5.2完善Retrofit

接下来我们可以完善Retrofit的部分，使得从豆瓣获取的数据可以转化为我们持久层的实体类：



```

class DoubanRetrofit {
    private val TAG: String = this.javaClass.simpleName
    private val PAGESIZE = Constant.PAGESIZE//20

    companion object {
        private val API = buildAPI()

        private fun buildAPI(): DoubanApi {
            return Retrofit.Builder()
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
                .addConverterFactory(JacksonConverterFactory.create())
                .baseUrl(Constant.url)// "http://api.douban.com/"
                .build()
                .create(DoubanApi::class.java)
        }

        private var instance: DoubanRetrofit? = null

        @Synchronized
        fun get(): DoubanRetrofit {
            if (null == instance) {
                instance = DoubanRetrofit()
            }
            return instance!!
        }
    }

    /**
     * 正在上映
     */
    fun inTheaterMovies(): Observable<List<Movie>> {

        return API.retrieveInTheaters(null)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(Schedulers.io())
            .concatMap({ origin ->
                Observable.just(origin.subjects)
            }).map({ subjects ->
                convert(subjects, true, false, false)
            })
    }

    /**
     * 即将上映
     * @param start 开始位置
     */
    fun comingSoonMovies(start: Int): Observable<List<Movie>> {

        return API.retrieveComingSoon(start, PAGESIZE)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(Schedulers.io())
            .concatMap({ origin ->
                Observable.just(origin.subjects)
            }).map({ subjects ->
                convert(subjects, false, true, false)
            })
    }

    /**
     * 评分top250电影
     * @param start 开始位置
     */
    fun top250Movies(start: Int): Observable<List<Movie>> {
        return API.retrieveTop250(start, PAGESIZE)
    }
}

```



```

        .observeOn(AndroidSchedulers.mainThread())
        .subscribeOn(Schedulers.io())
        .concatMap({ origin ->
            Observable.just(origin.subjects)
        }).map({ subjects ->
            convert(subjects, false, false, true)
        })
    }

/**
 * [Subject]列表转为[Movie]列表
 */
private fun convert(subjects: List<Subject>, isInTheater: Boolean, isComming: Boolean, isTop250: Boolean): List<Movie> {
    val movies: List<Movie> = subjects.map { item ->
        //评分
        val rating: Float
        val ratingStr: String
        if (null == item.rating) {
            ratingStr = "(0.0)"
            rating = 0.0f
        } else {
            rating = (item.rating.average / 2).toFloat()
            ratingStr = "(${item.rating.average})"
        }

        //导演
        val directors: String = if (null == item.directors) {
            ""
        } else {
            item.directors.joinToString("/", "", "", -1, "...", {it ->
                it.name
            })
        }

        //主演
        val casts = if (null == item.casts) {
            ""
        } else {
            item.casts.joinToString("/", "", "", -1, "...", {it ->
                it.name
            })
        }

        //类型
        val genres = if (null == item.genres) {
            ""
        } else {
            item.genres.joinToString("/", "", "", -1, "...")
        }

        val movie = Movie(item.id.toLong()
            , item.images.medium, item.title, rating, ratingStr, directors,
            item.year, isInTheater, isComming, isTop250)

        movie
    }

    return movies
}

```

5.4 Repository



完善了DAO和Retrofit的编写以后，我们就可以开始编写Repository层的代码。在Repository层，我们需要完成以下两个内容：

1. 将Retrofit层获取到的数据持久化在数据库中
2. 从数据库中获取数据

就是说，Repository层连接了Retrofit和ROOM，就像架构图示所描述的那样。



```
class MovieRepository(context: Context) {
    private val dao: MovieDao = MovieDatabase.get(context).movieDao()
    private val inTheaterMovies = dao.queryInTheaterMovies()
    private val commingsoonMovies = dao.queryCommingsoonMovies()
    private val top250Movies = dao.queryTOP250Movies()

    private val retrofit = DoubanRetrofit.get()

    /**
     * 获取正在上映的电影
     */
    fun getInTheaterMovies() = inTheaterMovies

    /**
     * 获取即将上映的电影
     */
    fun getCommingsoonMovies() = commingsoonMovies

    /**
     * 获取TOP250的电影
     */
    fun getTop250Movies() = top250Movies

    /**
     * 刷新正在上映的电影，并删除之前的数据
     */
    fun refreshInTheaterMovies(view: RefreshView){
        retrofit.inTheaterMovies()
            .subscribe({movies->
                ioThread {
                    dao.removeInTheaterMovies()
                    dao.save(movies)
                }
            }, { error ->
                view.onError(error)
                view.onRefreshCompleted()
            }, {
                view.onRefreshCompleted()
            })
    }

    /**
     * 刷新即将上映的电影，并删除之前的数据
     */
    fun refreshCommingsoonMovies(view: RefreshView){
        retrofit.commingSoonMovies(0)
            .subscribe({movies->
                ioThread {
                    dao.removeCommingSoonMovies()
                    dao.save(movies)
                }
            }, { error ->
                view.onError(error)
                view.onRefreshCompleted()
            }, {
                view.onRefreshCompleted()
            })
    }

    /**
     * 刷新TOP250的电影，并删除之前的数据
     */
    fun refreshTop250Movies(view: RefreshView){
        retrofit.top250Movies(0)
            .subscribe({movies->
```




```

        ioThread {
            dao.removeTop250Movies()
            dao.save(movies)
        }
    }, { error ->
        view.onError(error)
        view.onRefreshCompleted()
    }, {

        view.onRefreshCompleted()
    })
}

/**
 * 加载更多即将上映的电影
 */
fun loadMoreCommingSoonMovies(start:Int, view: RefreshView){
    retrofit.commingSoonMovies(start)
        .subscribe({movies->
            ioThread {
                dao.save(movies)
            }
        }, { error ->
            view.onError(error)
            view.onLoadMoreCompleted()
        }, {

            view.onLoadMoreCompleted()
        })
}

/**
 * 加载更多TOP250的电影
 */
fun loadMoreTop250Movies(start: Int, view:RefreshView){
    retrofit.top250Movies(start)
        .subscribe({movies->
            ioThread {
                dao.save(movies)
            }
        }, { error ->
            view.onError(error)
            view.onLoadMoreCompleted()
        }, {

            view.onLoadMoreCompleted()
        })
}

}

```

至此，Repository完成。

6. ViewModel

三个ViewModel层的编写比较相似，区别在于正在上映的ViewModel没有分页加载、每个ViewModel所调用Repository的方法不同。这里我们选取即将上映的ViewModel进行介绍：



```
class CommingsoonViewModel(app: Application) : AndroidViewModel(app) {
    private val repo = MovieRepository(app)
    private val datas = repo.getCommingsoonMovies()

    private var start:Int = 0

    fun getData() = datas.create(0, PagedList.Config.Builder()
        .setPageSize(Constant.PAGESIZE)
        .setEnablePlaceholders(Constant.ENABLE_PLACEHOLDERS)
        .build())

    fun refresh(view: RefreshView){
        start = 0
        repo.refreshCommingsoonMovies(view)
    }

    fun loadmore(view:RefreshView){
        start += Constant.PAGESIZE

        repo.loadMoreCommingsoonMovies(start,view)
    }
}
```

回顾一下架构图示，ViewModel层调用Repository的方法。我们在这里使用PagedList对分页数据进行配置。

7. UI补遗

最后我们再来看看UI方面有哪些需要补充的地方。

7.1 item_movie

首先是列表项的layout，item_movie:



```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:orientation="horizontal"
    >

    <ImageView android:id="@+id/picture" android:layout_width="80dp"
        android:layout_height="80dp"
        android:layout_gravity="center_vertical"

        android:layout_marginLeft="16dp"

        android:layout_marginRight="16dp"
        android:contentDescription="@string/movie_face"
        android:src="@mipmap/ic_launcher"/>

    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:orientation="vertical"
    >
        <TextView android:id="@+id/title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="@dimen/title_text_size"
            tools:text="肖申克的救赎 The Shawshank Redemption"/>
        <LinearLayout android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
        >
            <RatingBar android:id="@+id/rating_bar"
                style="@style/Base.Widget.AppCompat.RatingBar.Small"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center_vertical"
                android:numStars="5"

            />
            <TextView android:id="@+id/rating_des" android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center_vertical"
                tools:text=" (5.2)"/>
        </LinearLayout>

        <LinearLayout android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
        >
            <TextView android:id="@+id/textView" android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/director"/>

            <TextView android:id="@+id/director" android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                tools:text="弗兰克·德拉邦特"/>
        </LinearLayout>

        <LinearLayout android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
        >
            <TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
                android:text="@string/actor"
            />
        </LinearLayout>
    </LinearLayout>

```



```
        <TextView android:id="@+id/actors" android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            tools:text="蒂姆·罗宾斯 / 摩根·弗里曼 / 鲍勃·冈顿" />
    </LinearLayout>

    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
    >
        <TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:text="@string/type"
        />

        <TextView android:id="@+id/type" android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            tools:text="犯罪/剧情" />
    </LinearLayout>

    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
    >
        <TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:text="@string/year"
        />

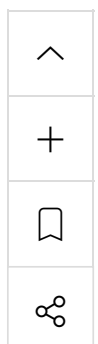
        <TextView android:id="@+id/year" android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            tools:text="1994" />
    </LinearLayout>

</LinearLayout>

</LinearLayout>
```

7.2 MovieAdapter

其次是用于RecyclerView的MovieAdapter：



```

class MovieAdapter : PagedListAdapter<Movie, MovieViewHolder>(diffCallback) {
    override fun onBindViewHolder(holder: MovieViewHolder, position: Int) {
        holder.bindTo(getItem(position))
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MovieViewHolder {
        return MovieViewHolder(parent)
    }

    companion object {
        /**
         * This diff callback informs the PagedListAdapter how to compute list differences
         * when PagedList arrives.
         * <p>
         * When you add a Cheese with the 'Add' button, the PagedListAdapter uses diff
         * to detect there's only a single item difference from before, so it only needs to
         * rebind a single view.
         *
         * @see android.support.v7.util.DiffUtil
         */
        private val diffCallback = object : DiffCallback<Movie>() {
            override fun areItemsTheSame(oldItem: Movie, newItem: Movie): Boolean {
                return oldItem.id == newItem.id
            }

            /**
             * Note that in kotlin, == checking on data classes compares all content
             * typically you'll implement Object#equals, and use it to compare objects.
             */
            override fun areContentsTheSame(oldItem: Movie, newItem: Movie): Boolean {
                return oldItem == newItem
            }
        }
    }
}

```

7.3 Fragment

最后是我们Fragment中的关键代码片段：

```

override fun onActivityCreated(savedInstanceState: Bundle?) {
    super.onActivityCreated(savedInstanceState)

    viewModel = ViewModelProviders.of(this).get(CommingsoonViewModel::class.java)
    viewModel.getData().observeForever(Observer(adapter::setList))
    onRefresh()
}

```

8. 总结（附demo）

app的演示：点击跳转 ([https://link.jianshu.com?](https://link.jianshu.com?t=http://player.youku.com/embed/XMzE4NzM2MDcyOA==)

[t=http://player.youku.com/embed/XMzE4NzM2MDcyOA==](https://link.jianshu.com?t=http://player.youku.com/embed/XMzE4NzM2MDcyOA==))

安装包下载地址以及开源代码地址：点击跳转 ([https://link.jianshu.com?](https://link.jianshu.com?t=http://chuckiefan.com/2017/11/27/%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6Demo%E4%B8%8B%E8%BD%BD.html)

[t=http://chuckiefan.com/2017/11/27/%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6Demo%E4%B8%8B%E8%BD%BD.html](https://link.jianshu.com?t=http://chuckiefan.com/2017/11/27/%E5%AE%89%E5%8D%93%E6%9E%B6%E6%9E%84%E7%BB%84%E4%BB%B6Demo%E4%B8%8B%E8%BD%BD.html))



本文介绍了使用安卓架构组件构建一个简单应用的过程。诚然，目前为止我们没有介绍到所有相关类库的基本使用，我会在后面尽量补充LiveData等相关内容的介绍。

但是从另一个角度来看，正如谷歌自己所说，这套类库只是提供一个架构设计的思路和参考，如果有更好的选择，完全可以不用关心这套组件。而这套组件中的类库也是可选择的，比如，你在持久层有更好的选择，就不需要使用Room。

本文书写以及代码的编写是在业余时间完成的，难免仓促和疏忽。如果您有任何问题和建议欢迎在文章下方评论，或者在github上提issue，我会及时回复并定期整理在文章中。

另外，欢迎您为文章点赞以及在github项目中点击star，这些是对我最大的回报和动力，谢谢。

1. 这里的下拉是指手指从下向上滑动，相反上拉是指从上向下滑动。↩

不需要大家进行打赏，如果我的文章能够对您有所帮助，希望您可以点击下方喜欢按钮，谢谢。

赞赏支持

评论 (/nb/17262983) 举报文章 © 著作权归作者所有




Chuckiefan (/u/191a96d00530)

写了 57711 字，被 281 人关注，获得了 440 个喜欢 (/u/191a96d00530)

+ 关注

喜欢 | 9





更多分享

被以下专题收入，发现更多相似内容

- + 收入我的专题
- 

Android开发 (/c/d1591c322c89?utm_source=desktop&utm_medium=notes-included-collection)
- 

程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-

- collection)
-  android... (/c/9699216a22f7?utm_source=desktop&utm_medium=notes-included-collection)
-  首页投稿（暂停... (/c/bDHhpK?utm_source=desktop&utm_medium=notes-included-collection)
-  Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)
-  Android... (/c/ee9f47f6d778?utm_source=desktop&utm_medium=notes-included-collection)


Android - 收藏集 (/p/dad51f6c9c4d?utm_campaign=maleskine&utm_c...

用两张图告诉你，为什么你的 App 会卡顿？ - Android - 掘金 Cover 有什么料？ 从这篇文章中你能获得这些料： 知道setContentView()之后发生了什么？ ... Android 获取 View 宽高的常用正确方式，避免为零 - 掘金...

 passiontim (/u/e946d18f163c?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommend

掘金 Android 文章精选合集 (/p/5ad013eb5364?utm_campaign=maleski...

用两张图告诉你，为什么你的 App 会卡顿？ - Android - 掘金Cover 有什么料？ 从这篇文章中你能获得这些料： 知道setContentView()之后发生了什么？ ... Android 获取 View 宽高的常用正确方式，避免为零 - 掘金...

 掘金官方 (/u/5fc9b6410f4f?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommend

(/p/b6d91bffca4?

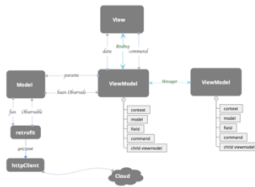


utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommend
【翻译】 Architecture Components - Android App应用架构组件指南 (/p/...

应用架构组件指南 原文请看： https://developer.android.com/topic/libraries/architecture/guide.html 本指南适用于具有构建安卓APP基础知识的开发人员，现在想知道最佳实践和建议的架构，以构建强大的生产级安...

 dengyin2000 (/u/efa51344ce61?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommend

(/p/2fc41a310f79?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommend
如何构建Android MVVM应用程序 (/p/2fc41a310f79?utm_campaign=mal...

1、概述 Databinding 是一种框架，MVVM是一种模式，两者的概念是不一样的。我的理解DataBinding是一

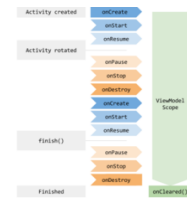
个实现数据和UI绑定的框架，只是一个实现MVVM模式的工具。ViewModel和View可以通过DataBinding来...



Kelin (/u/942d45d61f59?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommend

(/p/7d3d2ca11411?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommend

Android 应用架构组件（Architecture Components）实践 (/p/7d3d2ca11...

Architecture Components 是在 2017 年 Google I/O 大会上，Google 官方推出的一个构建 Android 应用架构的库。它可以帮你避免在 Android 应用开发中常见的一些问题，比如：内存泄露，管理组件生命周期等等...



lijiankun24 (/u/1abe21b7ff5f?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommend

(/p/523b57d3dfcb?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommend

以记忆为界 (/p/523b57d3dfcb?utm_campaign=maleskine&utm_content=...

我与左耳的缘分挺深的。从小学看饶雪漫的第一本书《咱们班》到初一看的左耳，就开始疯狂的阅读饶雪漫的书，看了这么多年的饶雪漫，或许跟其他作家比起来，饶雪漫的市场偏于学生，但在青春岁月中饶雪...



未央素年 (/u/1f15f7f3c847?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommend

你向我走来 (/p/90cec218ba56?utm_campaign=maleskine&utm_content=...

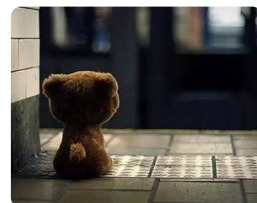
你和光一起 向我走来 在我肩上 轻声耳语 你和爱一起 向我走来 在我身旁 微笑着听我念诗 ...



爱做白日梦的哈罗德 (/u/afd62b1f581b?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommend

(/p/325ef6cfbdc7?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommend

放弃一个暗恋很久的人是什么感觉？ (/p/325ef6cfbdc7?utm_campaign=m...

▼ 1 你问我什么感觉 你拔过牙吗 你戒过毒吗 ▼ 2 我从一个有心事儿的单身狗 变成了一个没有心事儿的单身狗 ▼ 3 我是那么多人心目中的好女孩儿 可我没那个运气成为你喜欢的那个人 ▼ 4 从始至终没有拥有过 却...




小小星球 (/u/884a0da63886?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommend

日念一好9月24日 (/p/e82b9caa9989?utm_campaign=maleskine&utm_c...

日念亲人一处好，加持美好享幸福！ 第53天(2017.9.24) 1.念老公好：一早又去婆婆家帮忙了，装灯、接水，辛苦一天! 2.念孩子好:大宝上午的美术作品一节课就完成了!他说老师表扬他是“高手”! 小宝上午在超市玩具...

 轩轩妈慧子 (/u/59091bd76469?

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommend

^

+

🔖

🔗