应用市场笔记

## 服务器搭建

服务器：tomcat7

War包：GooglePlayServer.war 本质：压缩包，与APK文件类似（自动解压）

相对路径：apache-tomcat-XXXX\webapps\

(war包放入 tomcat 后，浏览器访问时，tomcat自动解压war包，生成同名的文件夹 GooglePlayServer )

应用集资源配置：

配置文件路径：webapps\GooglePlayServer\WEB-INF\classes\ system.properties

配置信息为WebInfos所在位置（自己配置：就是把自己的WebInfos 所在的上级路径配置好， 注意：路径名要双斜杠”\\”代替单斜杠）。

注意：需要使用最新的WebInfos，一些应用的信息进行了调整。

## 项目展示

## 应用如何分层

引入点：应用如何分包

1、按模块分包：account、cart等

2、按层次分包：bean、ui(view)等

### MVC——Android原生分层方式

model (模型)：数据

view (视图)：界面

controller (控制器)：简单的业务处理，控制数据传递

如：

1、加载联系人列表数据：ListView（界面）、Adapter（控制器）、Cursor（数据Contact ）

2、layout（界面）、Activity（控制器）、数据（db、xml、net…）

Android中MLA（model、layout、activity）

### MVP——业务复杂项目的分层方式

model (模型)：数据

view (视图)：界面

presenter (调度)：复杂的业务处理，控制数据传递

### Android中的MVVM

model (模型)：数据

view (视图)：界面

VM(view model): View Model负责提供方法、属性来操作VIew的状态，组装model作为View动作的结果，并且触发view自己的事件。

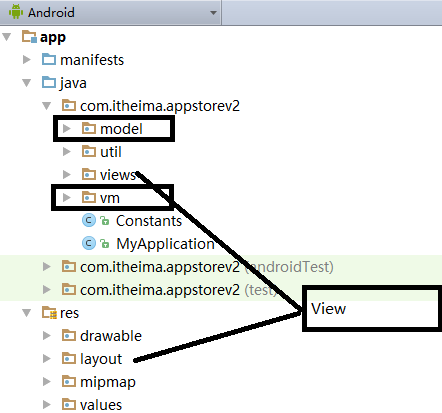
Android DataBinding已经集成到AndroidStudio中。

### 分层总结

无论以什么方式进行分层，其实都在围绕这如下几项内容：

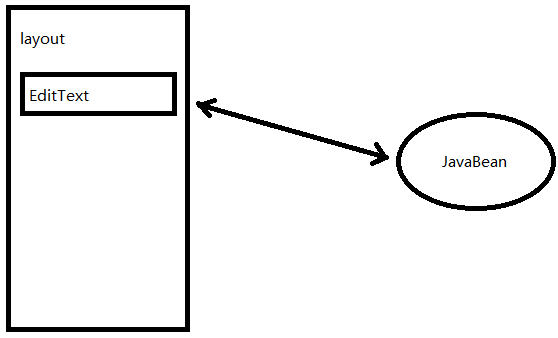
1、数据存储；2、数据展示；3、数据处理；4、数据传输

## 包结构划分



## Android databinding使用

为什么使用databinding——解决数据传输问题



常见实现方式：

1. findViewById找到控件，调用控件的设置文本方法将实体中数据设置到控件上；
2. 使用三方工具，如ButterKnife，配置注解，帮助我们完成findViewById的操作。如同第一种一样设置数据；

如果用户输入新内容了，我们还需要在1和2的基础上调用获取内容的方法，并将新内容设置给实体。

1. DataBinding如何处理？

### 使用步骤：

#### 1、开启DataBinding

由于AndroidStudio已经默认集成了DataBinding，所以我们只需要将开关打开即可

在应用的build.grandle中添加开启配置

android {  
    ....  
    dataBinding {  
        enabled = true  
    }  
}

#### 2、创建实体，并在Layout中绑定数据

public class User {  
   public String name;  
   public User(String name) {  
       this.name = name;   
   }  
}

准备好bean后我们可以通过下面的表达是将控件和属性值绑定。

<EditText android:layout\_width="wrap\_content"  
           android:layout\_height="wrap\_content"  
           android:text="@{user.name}"/>

绑定好后我们需要解决user这个对象从那里来。

#### 3、关联实体和控件

   <data>  
       <variable name="user" type="com.example.User"/>  
   </data>

此时就相当于我们代码中声明了一个User。

数据和控件都已经准备好了，我们需要将两者放到一起。此时，大家可以把data标签看成是全局变量，而原来的布局文件看成是方法，我们需要一个类将两者组装起来。

#### 4、使用layout标签将界面和data标签组装到一起

<?xml version="1.0" encoding="utf-8"?>  
<layout xmlns:android="http://schemas.android.com/apk/res/android">  
   <data>  
       <variable name="user" type="com.example.User"/>  
   </data>  
   <LinearLayout  
       android:orientation="vertical"  
       android:layout\_width="match\_parent"  
       android:layout\_height="match\_parent">  
       <TextView android:layout\_width="wrap\_content"  
           android:layout\_height="wrap\_content"  
           android:text="@{user.name}"/>  
   </LinearLayout>  
</layout>

类有了之后，全局变量还没有赋值，我们可以通过变量的setter方法进行赋值，即调用setUser方法设置User数据。

#### 5、实体对象的创建及数据设置s

处理完上述4步工作后需要rebuild一下工程，工具会根据layout标签生成对应的类，类名与该布局文件这篇文档介绍了如何使用 Data Binding Library 来编写声明式的布局，这样可以减少应用中逻辑代码和布局之间所需要的“胶水代码”。  
  
Data Binding Library 提供了非常好的灵活性与兼容性 － 它是一个 support library，可以在 Android 2.1(API level 7+)及其以上的平台使用。  
  
要使用 data binding，Android 的 Gradle 插件必须是 1.5.0-alpha1 或更高版本。  
  
## 搭建构建环境  
1. 在 Android SDK Manager 中下载最新的 Support Library。  
2. 在 app module 的 build.gradle 文件中添加 dataBinding 元素，如下：  
```java  
android {  
    ....  
    dataBinding {  
        enabled = true  
    }  
}  
```  
这样DataBinding插件就会在你的项目内添加编译和运行时必需的依赖配置。  
  
如果你的 app module 依赖了一个使用 data binding 的库，那么你的 app module 的 build.gradle 也必须配置 data binding  
  
此外，还要确定您使用的 Android Studio 支持 DataBinding 特性的。在 Android Studio 1.3 以及之后的版本提供了 data binding 的支持，详见 [Android Studio Support for Data Binding](http://developer.android.com/intl/zh-cn/tools/data-binding/guide.html#studio\_support)。  
  
## Data Binding 中的布局文件入门  
### 1. 编写 data binding 表达式  
DataBinding 的布局文件与以前的布局文件有一点不同。它以一个 layout 标签作为根节点，里面包含一个 data 标签与 view 标签。view 标签的内容就是不使用 data binding 时的普通布局文件内容。例子如下：  
```xml  
<?xml version="1.0" encoding="utf-8"?>  
<layout xmlns:android="http://schemas.android.com/apk/res/android">  
   <data>  
       <variable name="user" type="com.example.User"/>  
   </data>  
   <LinearLayout  
       android:orientation="vertical"  
       android:layout\_width="match\_parent"  
       android:layout\_height="match\_parent">  
       <TextView android:layout\_width="wrap\_content"  
           android:layout\_height="wrap\_content"  
           android:text="@{user.firstName}"/>  
       <TextView android:layout\_width="wrap\_content"  
           android:layout\_height="wrap\_content"  
           android:text="@{user.lastName}"/>  
   </LinearLayout>  
</layout>  
```  
在 data 标签中定义的 user 变量描述了一个可以在布局中使用的属性  
```xml  
<variable name="user" type="com.example.User"/>  
```  
在布局文件中写在属性值里的表达式使用 “@{}” 的语法。在这里，TextView 的文本被设置为 user 中的 firstName 属性。  
```xml  
<TextView android:layout\_width="wrap\_content"  
          android:layout\_height="wrap\_content"  
          android:text="@{user.firstName}"/>  
```  
  
### 2. 数据对象  
假设你有一个 plain-old Java object(POJO) 的 User 对象。  
```java  
public class User {  
   public final String firstName;  
   public final String lastName;  
   public User(String firstName, String lastName) {  
       this.firstName = firstName;  
       this.lastName = lastName;  
   }  
}  
```  
这个类型的对象拥有不可改变的数据(immutable)。在应用中，写一次之后永不变动数据的对象很常见。这里也可以使用 JavaBeans 对象：  
```java  
public class User {  
   private final String firstName;  
   private final String lastName;  
   public User(String firstName, String lastName) {  
       this.firstName = firstName;  
       this.lastName = lastName;  
   }  
   public String getFirstName() {  
       return this.firstName;  
   }  
   public String getLastName() {  
       return this.lastName;  
   }  
}  
```  
从 data binding 的角度看，这两个类是等价的。用于 TextView 的`android:text`属性的表达式`@{user.firstName}`，对于 POJO 对象会读取 firstName 字段，对于 JavaBeans 对象会调用 getFirstName() 方法。此外，如果 user 中有 firstName() 方法存在的话，也可以使用@{user.firstName}表达式调用。  
  
### 3. 绑定数据  
3.1 数据绑定工具在编译时会基于布局文件生成一个 Binding 类。默认情况下，这个类的名字将基于布局文件的名字产生，把布局文件的名字转换成帕斯卡命名形式并在名字后面接上”Binding”。例如，上面的那个布局文件叫 main\_activity.xml，所以会生成一个 MainActivityBinding 类。这个类包含了布局文件中所有的绑定关系（user 变量和user表达式），并且会根据绑定表达式给布局文件中的 View 赋值。  
在 inflate 一个布局的时候创建 binding 的方法如下：  
```java  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
   super.onCreate(savedInstanceState);  
   MainActivityBinding binding = DataBindingUtil.setContentView(this, R.layout.main\_activity);  
   User user = new User("Test", "User");  
   binding.setUser(user);  
}  
```  
就这么简单！运行应用，你会发现测试信息已经显示在界面中了。  
  
3.2 你也可以通过以下这种方式绑定view：  
```java  
MainActivityBinding binding = MainActivityBinding.inflate(getLayoutInflater());  
```  
MainActivityBinding.inflate() 会填充 MainActivityBinding 对应的布局，并创建 MainActivityBinding 对象，把布局和MainActivityBinding对象绑定起来。  
  
3.3 如果在 ListView 或者 RecyclerView 的 adapter 中使用 data binding，可以这样写：  
```java  
ListItemBinding binding = ListItemBinding.inflate(layoutInflater, viewGroup, false);  
//or  
ListItemBinding binding = DataBindingUtil.inflate(layoutInflater, R.layout.list\_item, viewGroup, false);  
```  
  
### 4. 事件处理  
数据绑定允许你编写表达式来处理view分发的事件（比如 onClick）。事件属性名字取决于监听器方法名字。例如[View.OnLongClickListener](https://developer.android.com/reference/android/view/View.OnLongClickListener.html)有[onLongClick()](https://developer.android.com/reference/android/view/View.OnLongClickListener.html#onLongClick(android.view.View))的方法，因此这个事件的属性是android:onLongClick  
。处理事件有两种方法：  
- 方法引用绑定：在您的表达式中，您可以引用符合监听器方法签名的方法。当表达式的值为方法引用时，Data Binding会创建一个侦听器中封装方法引用和方法所有者对象，并在目标视图上设置该侦听器。如果表达式的值为null，数据绑定则不会创建侦听器，而是设置一个空侦听器。  
- Lisenter 绑定：如果事件处理表达式中包含lambda表达式。数据绑定一直会创建一个监听器，设置给视图。当事件分发时，侦听器才会计算lambda表达式的值。  
  
#### 4.1. 方法引用绑定  
事件可以直接绑定到处理器的方法上，类似于`android：onClick`可以分配一个Activity中的方法。与`View＃onClick`属性相比，方法绑定一个主要优点是表达式在编译时就处理了，因此如果该方法不存在或其签名不正确，您会收到一个编译时错误。   
  
方法引用绑定和监听器绑定之间的主要区别是，包裹方法引用的监听器是在数据绑定时创建的，包裹监听器绑定是在触发事件时创建的。如果您喜欢在事件发生时执行表达式，则应使用监听器绑定。   
  
要将事件分配给其处理程序，使用正常的绑定表达式，该值是要调用的方法名称。例如，如果数据对象有两个方法：  
```java  
public class MyHandlers {  
    public void onClickFriend(View view) { ... }  
}  
```  
像下面这样，绑定表达式可以为视图分配一个点击监听器：  
```java  
<?xml version="1.0" encoding="utf-8"?>  
<layout xmlns:android="http://schemas.android.com/apk/res/android">  
   <data>  
       <variable name="handlers" type="com.example.Handlers"/>  
       <variable name="user" type="com.example.User"/>  
   </data>  
   <LinearLayout  
       android:orientation="vertical"  
       android:layout\_width="match\_parent"  
       android:layout\_height="match\_parent">  
       <TextView android:layout\_width="wrap\_content"  
           android:layout\_height="wrap\_content"  
           android:text="@{user.firstName}"  
           android:onClick="@{handlers::onClickFriend}"/>  
   </LinearLayout>  
</layout>  
```  
请注意，表达式中的方法签名必须与监听器对象中的方法签名完全匹配。  
  
#### 4.2. Lisenter 绑定  
Lisenter 绑定是在应用程序运行中事件发生时才绑定表达式。它们和方法引用绑定类似，但Listener 绑定允许运行时的任意数据绑定表达式。此功能适用于版本2.0及更高版本的Android Gradle插件。 在方法引用绑定中，方法的参数必须与事件侦听器的参数匹配。在Listener 绑定中，只要返回值必须与Lisenter的预期返回值匹配（除非它期望void）。  
  
例如，您可以有一个presenter类，它具有以下方法：  
```java  
public class Presenter {  
    public void onSaveClick(Task task){}  
}  
```  
然后，您可以将点击事件绑定到您的类中，如下所示：  
```xml  
<?xml version="1.0" encoding="utf-8"?>  
<layout xmlns:android="http://schemas.android.com/apk/res/android">  
  <data>  
      <variable name="task" type="com.android.example.Task" />  
      <variable name="presenter" type="com.android.example.Presenter" />  
  </data>  
  <LinearLayout   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent">  
      <Button   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"  
          android:onClick="@{() -> presenter.onSaveClick(task)}" />  
  </LinearLayout>  
</layout>  
```  
侦听器由只允许由作为表达式的根元素的lambda表达式表示。当在表达式中使用回调时，数据绑定自动创建必要的侦听器并且为事件注册。当视图触发事件时，数据绑定将执行给定的表达式。与在正则绑定表达式中一样，在执行这些侦听器表达式时，仍然会获得数据绑定的空值和线程安全性。  
注意，在上面的示例中，我们没有在ambda表达式中定义传递到 onClick（android.view.View）的视图参数。侦听器绑定为侦听器参数提供两个选择：忽略方法的所有参数和命名所有参数。如果您喜欢命名参数，可以在表达式中使用它们。例如，上面的表达式可以写成：  
```xml  
android:onClick="@{(view) -> presenter.onSaveClick(task)}"  
```  
如果你想要使用表达式中的参数，可以在这样使用：  
```java  
public class Presenter {  
    public void onSaveClick(View view, Task task){}  
}  
```  
```xml  
android:onClick="@{(theView) -> presenter.onSaveClick(theView, task)}"  
```  
您也可以使用具有多个参数的lambda表达式：  
```java  
public class Presenter {  
    public void onCompletedChanged(Task task, boolean completed){}  
}  
```  
```xml  
<CheckBox   
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"  
    android:onCheckedChanged="@{(cb, isChecked) -> presenter.completeChanged(task, isChecked)}" />  
```  
如果正在侦听的事件返回类型不是void的值，则表达式也必须返回相同类型的值。例如，如果你想监听长点击事件，你的表达式应该返回布尔值。  
```java  
public class Presenter {  
    public boolean onLongClick(View view, Task task){}  
}  
```  
```xml  
android:onLongClick="@{(theView) -> presenter.onLongClick(theView, task)}"  
```  
如果由于空对象而无法计算表达式，数据绑定将返回该类型的默认Java值。例如，引用类型为null，int为0，boolean为false等。 如果需要使用带谓词（例如三元）的表达式，则可以使用void作为符号。  
```xml  
android:onClick="@{(v) -> v.isVisible() ? doSomething() : void}"  
```  
#### 4.3. 避免复杂侦听器  
Listener表达式非常强大，可以使您的代码非常容易阅读。另一方面，包含复杂表达式的 Listener 又会使您的布局难以阅读和难以维护。这些表达式应该像从UI传递可用数据到回调方法一样简单。您应该在从侦听器表达式调用的回调方法中实现任何业务逻辑。   
  
一些专门的点击事件处理程序存在，他们需要一个属性，而不是android：onClick避免冲突。已创建以下属性以避免此类冲突。  
  
|    Class        |    Listener Setter    |    Attribute    |  
|    :---:        |    :---:            |    :----:        |  
|SearchView        |setOnSearchClickListener(View.OnClickListener)        |android:onSearchClick|  
|ZoomControls    |setOnZoomInClickListener(View.OnClickListener)        |android:onZoomIn|  
|ZoomControls    |setOnZoomOutClickListener(View.OnClickListener)    |android:onZoomOut|  
  
##  Data Binding 中的布局文件进阶  
### 1. import  
1.1 data标签内可以有0个或者多个 import 标签。你可以在布局文件中像使用 Java 一样导入引用。  
```xml  
<data>  
    <import type="android.view.View"/>  
</data>  
```  
导入的类有两种用处，一种是用来定义变量，一种是在表达式中访问类中的静态方法和属性  
现在 View 可以被在表达式中这样引用：  
```xml  
<TextView  
   android:text="@{user.lastName}"  
   android:layout\_width="wrap\_content"  
   android:layout\_height="wrap\_content"  
   android:visibility="@{user.isAdult ? View.VISIBLE : View.GONE}"/>  
```  
1.2 当类名发生冲突时，可以使用 alias：  
```xml  
<import type="android.view.View"/>  
<import type="com.example.real.estate.View"  
        alias="Vista"/>  
```  
现在，Vista 可以用来引用 com.example.real.estate.View ，与 View 在布局文件中同时使用。  
  
1.3 导入的类型可以用于变量的类型引用和表达式中：  
```xml  
<data>  
    <import type="com.example.User"/>  
    <import type="java.util.List"/>  
    <variable name="user" type="User"/>  
    <variable name="userList" type="List<User>"/>  
</data>  
```  
> 注意：Android Studio 还没有对导入提供自动补全的支持。你的应用还是可以被正常编译，要解决这个问题，你可以在变量定义中使用完整的包名。  
  
```xml  
<TextView  
   android:text="@{((User)(user.connection)).lastName}"  
   android:layout\_width="wrap\_content"  
   android:layout\_height="wrap\_content"/>  
```  
1.4 导入的类也可以在表达式中使用静态属性/方法:  
```xml  
<data>  
    <import type="com.example.MyStringUtils"/>  
    <variable name="user" type="com.example.User"/>  
</data>  
…  
<TextView  
   android:text="@{MyStringUtils.capitalize(user.lastName)}"  
   android:layout\_width="wrap\_content"  
   android:layout\_height="wrap\_content"/>  
```  
1.5 和 Java 一样，java.lang.\* 会被自动导入。  
  
### 2. 变量  
  
### 3. 自定义 Binding 类名  
  
### 4. Includes  
  
### 5. 表达式语言  
  
## 数据对象  
### 1. Observable 对象  
  
### 2. Observable 属性  
  
### 3. Observable 容器类  
  
## 生成绑定  
### 1. 创建  
  
### 2. 带有 ID 的 View  
  
### 3. 变量  
  
### 4. ViewStub  
  
### 5. 高级绑定  
  
#### 1. 动态变量  
  
#### 2. 直接 binding  
  
#### 3. 后台线程问题  
  
  
## 属性 Setter  
  
### 1. 自动 Setter  
  
### 2. 重命名 Setter  
  
### 3. 自定义 Setter  
  
## 转换器  
  
### 1. 对象转换  
  
### 2. 自定义转换  
  
## Android Studio 对 Data binding 的支持的名称相关。

@Override  
protected void onCreate(Bundle savedInstanceState) {  
   super.onCreate(savedInstanceState);  
   ActivityMainBinding binding = DataBindingUtil.setContentView(this, R.layout.activity\_main);  
   User user = new User("Test", "User");  
   binding.setUser(user);  
}

命名规范

布局文件：R.layout.activity\_main

类名：ActivityMainBinding

#### 总结：

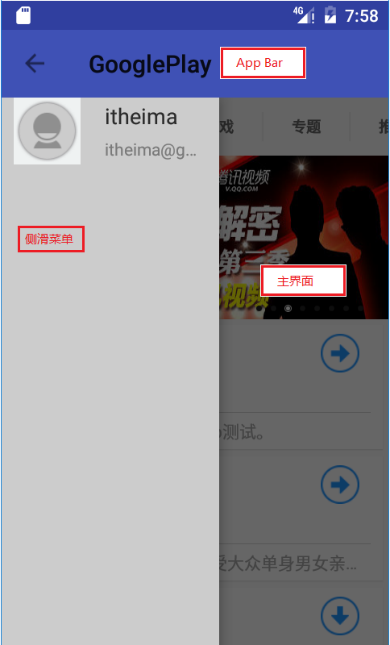
把布局文件看成是一个类，有属性，有方法。属性值需要通过set进来。

## 导入常用工具类

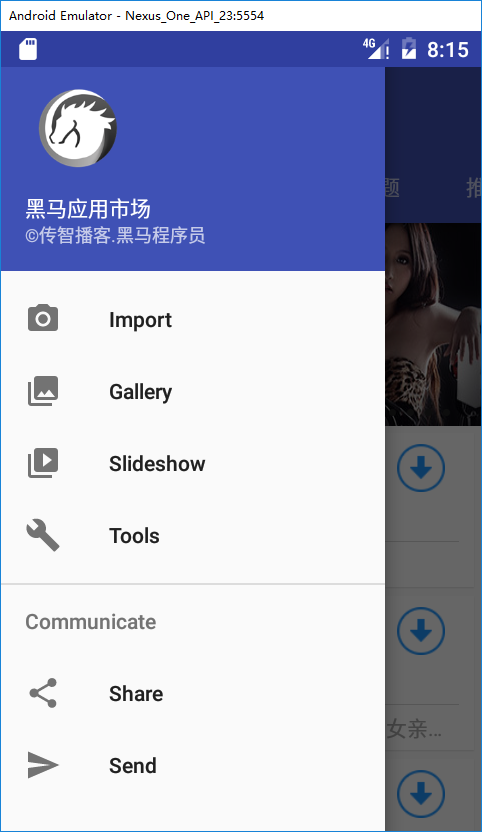
[素材\utils](素材/utils)

## 界面分析

示例：



工具条一家独大，菜单和界面一个容器。

从图2可以看出：菜单弹出后居于整个界面之上，即菜单与内容是同级别存在。

从图3可以看出：多条目展示容器滚动时，？Bar与Tabs一同滚动，直到？Bar消失。

## ToolBar使用

Toolbar是在 Android 5.0 开始推出的一个 Material Design 风格的导航控件 ，Google 非常推荐大家使用 Toolbar 来作为Android客户端的导航栏，以此来取代之前的 Actionbar 。与 Actionbar 相比， Toolbar 明显要灵活的多。它不像 Actionbar 一样，一定要固定在Activity的顶部，而是可以放到界面的任意位置。

只需要在 文件中增加：

<**android.support.v7.widget.Toolbar  
 android:id="@+id/toolbar"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="@color/colorPrimary"/>**

在Activity中编写：

Toolbar toolbar = (Toolbar) findViewById(R.id.***toolbar***);  
*// 注意：需要在setSupportActionBar调用之前设置好Toolbar的配置  
  
// 使用toolbar代替actionbar*setSupportActionBar(toolbar);

易错点：

1、Toolbar会替换掉ActionBar，如果在样式中使用到XXX**ActionBar**

一定要进行替换，或者单独给当前界面配置新的样式。

2、toolbar的相关设置，最好在setSupportActionBar方法前操作。

## DrawerLayout使用

完整的名称：**android.support.v4.widget.DrawerLayout**

配置文件中引入DrawerLayout

<**android.support.v4.widget.DrawerLayout  
xmlns:android="http://schemas.android.com/apk/res/android"  
xmlns:app="http://schemas.android.com/apk/res-auto"  
android:id="@+id/main\_drawerLayout"  
android:layout\_height="match\_parent"  
android:layout\_width="match\_parent"**>

</**android.support.v4.widget.DrawerLayout**>

菜单：

**android.support.design.widget.NavigationView**

<**android.support.design.widget.NavigationView  
 android:id="@+id/nav\_view"  
 android:layout\_height="match\_parent"  
 android:layout\_width="wrap\_content"  
 android:layout\_gravity="start"  
 android:fitsSystemWindows="true"  
 app:headerLayout="@layout/nav\_header"  
 app:menu="@menu/activity\_main\_drawer"**/>

代码：

*// 初始化DrawerLayout  
// 将Toolbar与DrawerLayout整合***drawerLayout** = (DrawerLayout) findViewById(R.id.***main\_drawerLayout***);  
ActionBarDrawerToggle barDrawerToggle=**new** ActionBarDrawerToggle(**this**, **drawerLayout**,toolbar,R.string.***open***,R.string.***close***);  
barDrawerToggle.syncState();  
*// 如果想看到菜单滑动过程中，Toolbar的一些细节变化（菜单按钮）***drawerLayout**.addDrawerListener(barDrawerToggle);

*// 处理菜单项点击*NavigationView navigationView= (NavigationView) findViewById(R.id.***nav\_view***);  
navigationView.setNavigationItemSelectedListener(**this**);  
@Override  
**public boolean** onNavigationItemSelected(@NonNull MenuItem item) {  
 item.setChecked(**true**);  
 **drawerLayout**.closeDrawers();  
 **return false**;  
}

## 主界面内容部分联动

联动的处理：

按照现有的知识，我们可以给垂直滚动容器增加滚动监听，将滚动的距离传递给ToolBar所在的容器，做出一个联动的效果，不过这个联动是有延时的，我们是先获取到垂直滚动容器滚动的距离，然后将距离设置给其他容器。——伪联动

需求：如果需要同步滚动，但是滚动的内容不能放到同一个容器里该如何操作。

思路：将需要一起滚动的内容统一放到一个更大的容器，让该容器帮我们统一处理联动操作。

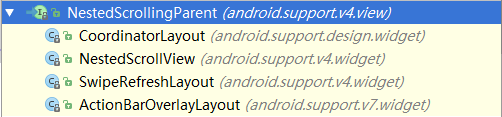
#### 处理联动的容器

**android.support.design.widget.CoordinatorLayout**

系统提供给我们专门处理联动的容器。

为什么说这家伙就能够处理联动，看一个标识：NestedScrollingParent

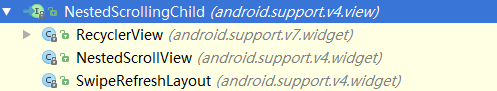
**public class** CoordinatorLayout **extends** ViewGroup **implements** NestedScrollingParent

实现这个接口的容器就是处理联动的父容器。  


父容器存在了，子控件就位后就可以处理联动操作了。

#### 可以嵌入的联动控件

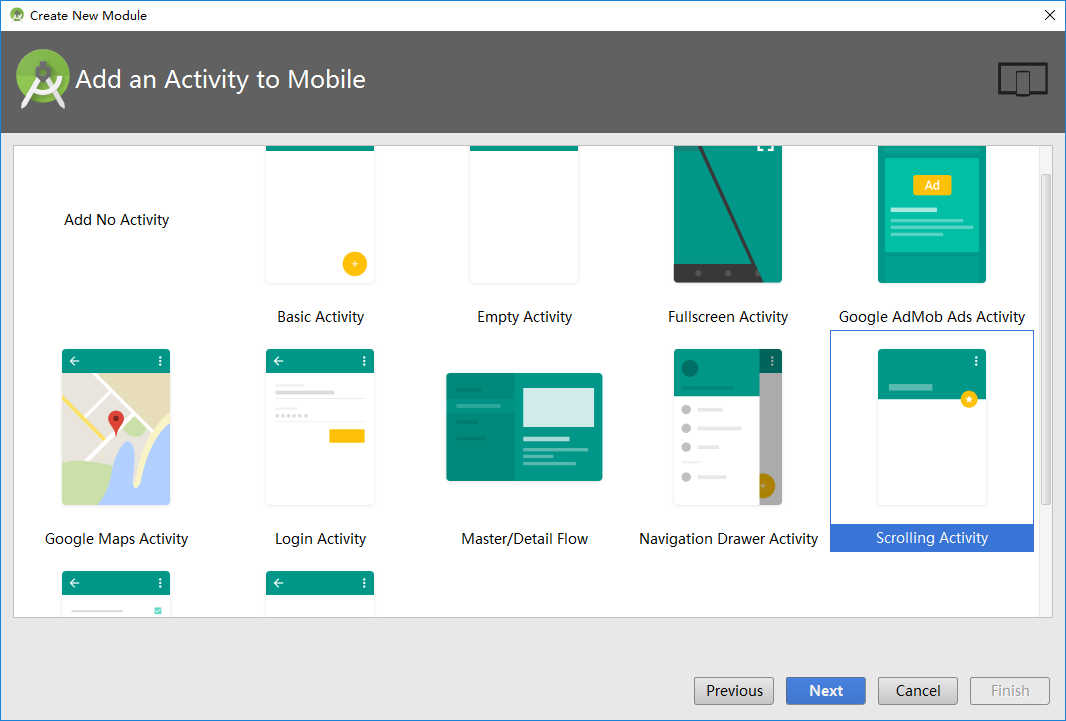
接口：NestedScrollingChild



#### 学习方式

我们可以通过官方提供的案例进行学习。

在创建Moudle时Activity创建选择Scrolling Activity

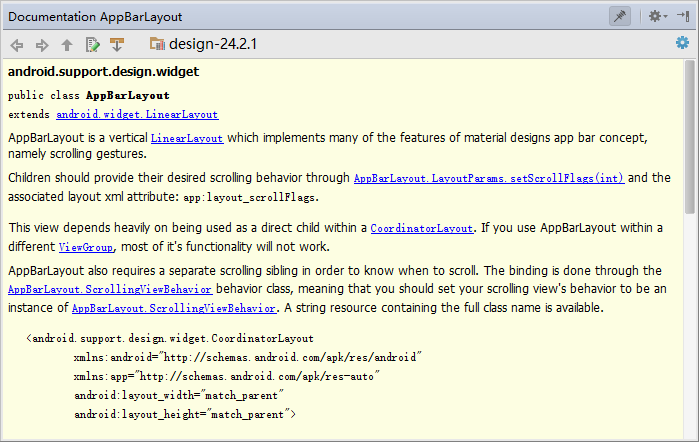


生成了一个layout布局：

*<?***xml version="1.0" encoding="utf-8"***?>*<**android.support.design.widget.CoordinatorLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:fitsSystemWindows="true"**>  
  
 <**android.support.design.widget.AppBarLayout  
 android:id="@+id/app\_bar"  
 android:layout\_width="match\_parent"  
 android:layout\_height="@dimen/app\_bar\_height"  
 android:fitsSystemWindows="true"  
 android:theme="@style/AppTheme.AppBarOverlay"**>  
  
 <**android.support.design.widget.CollapsingToolbarLayout  
 android:id="@+id/toolbar\_layout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:fitsSystemWindows="true"  
 app:contentScrim="?attr/colorPrimary"  
 app:layout\_scrollFlags="scroll|exitUntilCollapsed"**>  
  
 <**android.support.v7.widget.Toolbar  
 android:id="@+id/toolbar"  
 android:layout\_width="match\_parent"  
 android:layout\_height="?attr/actionBarSize"  
 app:layout\_collapseMode="pin"  
 app:popupTheme="@style/AppTheme.PopupOverlay"** />  
  
 </**android.support.design.widget.CollapsingToolbarLayout**>  
 </**android.support.design.widget.AppBarLayout**>  
  
 <**android.support.v4.widget.NestedScrollView   
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"**>  
 ……  
</**android.support.v4.widget.NestedScrollView**>  
 ……  
</**android.support.design.widget.CoordinatorLayout>**

#### AppBarLayout

官方文档如下：





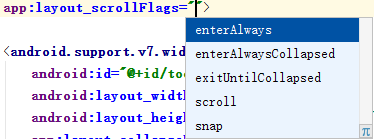
可以得出如下信息：

1. 垂直的线性布局，识别滚动（app bar）
2. 子控件设置滚动时的表现方式有两种：

代码方式：AppBarLayout.LayoutParams.setScrollFlags(int)

属性配置：app:layout\_scrollFlags

方式包括：



EnterAlways，无论怎么滚动，向上到全部隐藏，向下到全部显示。

1. AppBarLayout必须作为CoordinatorLayout的直接子节点，否则多数功能无法实现。
2. 为了让AppBarLayout起作用，CoordinatorLayout中需要有控件是滚动控件，并且该控件必须实现NestedScrollingChild接口，并且通过声明：app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"告知AppBarLayout什么时候滚动。

## TabLayout使用

通过查看官方文档，我们可以知道，TabLayout是android.widget.HorizontalScrollView子类。

常用API：

1. 创建Tab项：newTab()
2. 设置文本和图片：setText(int)、setIcon(int)
3. 手动添加项：tabLayout.addTab(tabLayout.newTab().setText("Tab 1"));

4、与ViewPager整合：tabs.setupWithViewPager(viewPager);

代码如下：

*//TabLayout处理  
// 1、手动添加  
// 2、与ViewPager绑定，通过Pageadapter的getPageTitle(int position)可以获取到选项卡  
  
// 1、手动添加*TabLayout tabs = (TabLayout) findViewById(R.id.***tabs***);  
tabs.addTab(tabs.newTab().setText(**"Tab 1"**));  
tabs.addTab(tabs.newTab().setText(**"Tab 2"**));  
tabs.addTab(tabs.newTab().setText(**"Tab 3"**));  
  
*// 2、与ViewPager绑定，通过Pageadapter的getPageTitle(int position)可以获取到选项卡*ViewPager viewPager = (ViewPager) findViewById(R.id.***viewpager***);  
tabs.setupWithViewPager(viewPager);

## 测试界面联调

目的：判断界面是否能够达到整体联动的效果。

为什么判断？

主要原因是NestedScrollingChild（RecyclerView）被套在了ViewPager的Fragment中了。同时也想验证ListView是否可以触发联动。

步骤：

1. 完成ViewPager的Adapter创建；
2. 添加两个Fragment实例，其中一个使用RecyclerView，一个使用ListView；
3. 给RecyclerView和ListView设置测试数据。

问题：

ViewPager的Adapter选择:

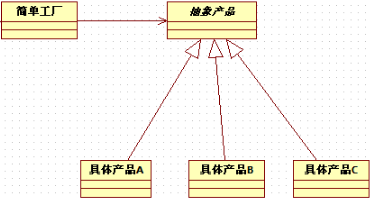
FragmentPagerAdapter VS FragmentStatePagerAdapter

区别：是否长时间持有Fragment引用。

## 使用简单工厂创建Fragment

目的：创建一组同类对象（条件不同时，生成的对象不同）

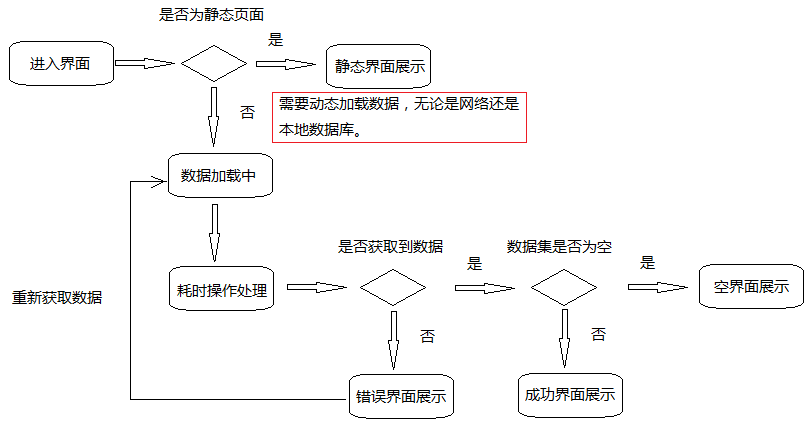
类图：



ViewPager展示的都是Fragment子类，唯一区别是不同的位置上显示的Fragment不同。我们可以使用工厂创建Fragment，条件就是位置信息。

## 加载界面处理

应用中界面处理的通用流程如下：



需要准备的界面如下：

1. 是否是静态界面
   1. 是：静态界面（单独准备）
   2. 否：流程继续
2. 是否读取到数据
   1. 读取数据是耗时操作，所以读取之前使用加载数据界面（通用界面），读取结果知道后隐藏即可。
   2. 否：显示错误界面（通用界面），需要有重试的按钮。
   3. 是：流程继续
3. 是否为空数据集
   1. 是：显示空界面（通用界面）
   2. 否：显示成功界面（单独准备）

代码抽取的流程：

1. 从HomeFragment和AppFragment中将公共代码抽取到BaseFragment；
2. 从BaseFragment中将公共代码抽取到CommonPager中，满足Fragment和Activity的使用需求。

## 首页界面流程代码处理

## 应用界面流程代码处理

## 流程代码抽取

存在问题：

1. 应用界面的加载数据方法写到了onResume中，但实际情况是，点击应用选项时，界面内容已经加载完成了，我们希望达到的目标是：点击应用过界面，然后从头开始走流程。
2. 频繁切换Fragment时，流程都会从头开始，如果耗时操作已经开始了，我们不希望再次开启，如果不是错误界面展示，我们不希望再次走完流程。
3. 流程的开启有两种情况：第一次加载，数据读取出错

在切换Fragment时，如果完成了第一次加载，以后只有当数据读取出错时才会走流程

1. 如何让Activity与Fragment通用

我们把两个Fragment“通用”的内容放到了BaseFragment中，如果想让Activity与Fragment通用，我们只需要在现有的BaseFragment中剔除Fragment元素

1. 关于Handler的子线程中创建问题，如果使用CommonPager的开发人员，不小心在子线程创建了CommonPager那么我们的Handler就会运行在子线程中。

如何确保Handler在主线程中被创建？

在Activity、Fragment或其他确保在主线程...，我们自己定义的Application

如何判断：CommonPager会在很多Activity或Fragment中使用，其中的Handler会一直使用。

## 首页联网获取数据

### 协议介绍



### 利用okhttp获取联网数据

#### 简介

1. OkHttp是一个高效的HTTP库（地址：https://github.com/square/okhttp）

* 支持SPDY ，共享同一个Socket来处理同一个服务器的所有请求
* 如果SPDY不可用，则通过连接池来减少请求延时
* 无缝的支持GZIP来减少数据流量
* 缓存响应数据来减少重复的网络请求

1. SPDY:

SPDY（读作“SPeeDY”）是[Google](http://baike.baidu.com/view/105.htm)开发的基于[TCP](http://baike.baidu.com/subview/32754/8048820.htm)的应用层协议，用以最小化网络延迟，提升网络速度，优化用户的网络使用体验。SPDY并不是一种用于替代HTTP的协议，而是对[HTTP](http://baike.baidu.com/view/9472.htm)协议的增强。新协议的功能包括数据流的多路复用、请求优先级以及HTTP报头压缩。谷歌表示，引入SPDY协议后，在实验室测试中页面加载速度比原先快64%。

#### 添加依赖

两种方式：

1. 网络状态好的情况下直接通过:project structure设置dependencies即可
2. 网络条件不好：添加离线依赖[素材\libs](素材/libs)

找离线的依赖

maven仓库:http://mvnrepository.com/

jcenter仓库:https://bintray.com/bintray/jcenter

maven的国内镜像:http://maven.oschina.net

#### 使用步骤

1. 创建OkHttpClient对象
2. 创建请求对象（Request）
3. 发起请求

### 利用Gson解析数据

## 首页数据展示

#### Databinding处理RecyclerView数据绑定

#### 常用图片加载框架比较

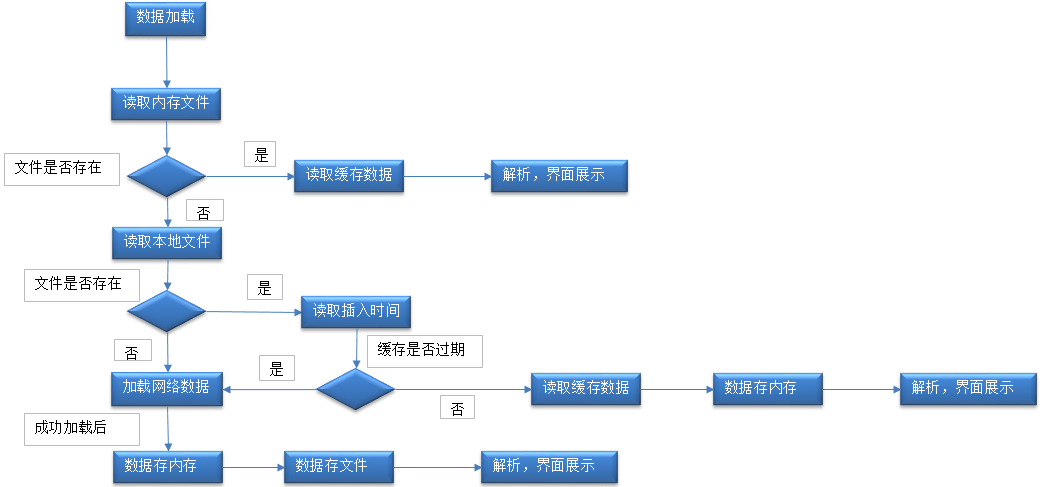
#### 使用Glide加载图片

## 快速完成应用界面数据加载和界面展示

## 抽取公共部分代码

## 三级缓存流程分析

文件的三级缓存流程分析：



## 协议缓存处理

### 内存缓存

### 本地缓存

#### 普通文件

#### 数据库

## 首页轮播

从界面设置上尽量避免使用Viewpager嵌套

第三方框架：AndroidImageSlider

GITHub：https://github.com/daimajia/AndroidImageSlider

aar包：比普通的jar包多包含了一些内容。

不仅仅包含了代码，也包含了对应的资源。

aar包导入步骤：

1. 在project的build.gradle文件中增加，具体位置：，具体位置：allprojects🡪repositories下面

flatDir {  
 dirs **'libs'**}

1. aar包拷贝到libs中
2. 在Module中的build.gradle文件中增加依赖

compile( **name**: **'xxx'**, **ext**: **'aar'**)

AndroidImageSlider使用步骤：

1. layout中引入控件

<com.daimajia.slider.library.SliderLayout  
 android:id="@+id/slider"  
 android:layout\_width="match\_parent"  
 android:layout\_height="200dp"/>

1. 初始化控件

SliderLayout sliderShow = (SliderLayout) findViewById(R.id.slider);

1. 添加图片项

TextSliderView textSliderView = new TextSliderView(this);  
textSliderView  
 .description("描述信息")  
 .image("图片资源url或者本地图片的id");  
sliderShow.addSlider(textSliderView);

1. 相关属性设置

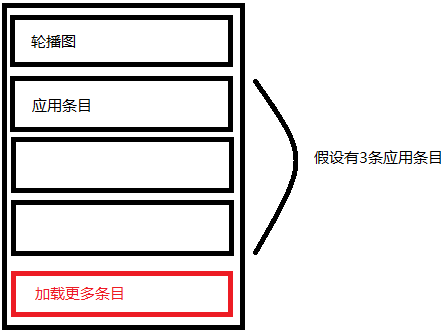
// 设置过渡效果  
sliderShow.setPresetTransformer(SliderLayout.Transformer.Accordion);  
// 设置默认指示器位置(默认指示器白色,位置在sliderlayout底部)  
|sliderShow.setPresetIndicator(SliderLayout.PresetIndicators.Center\_Bottom);  
// 设置持续时间  
sliderShow.setDuration(2000);

## 上拉加载

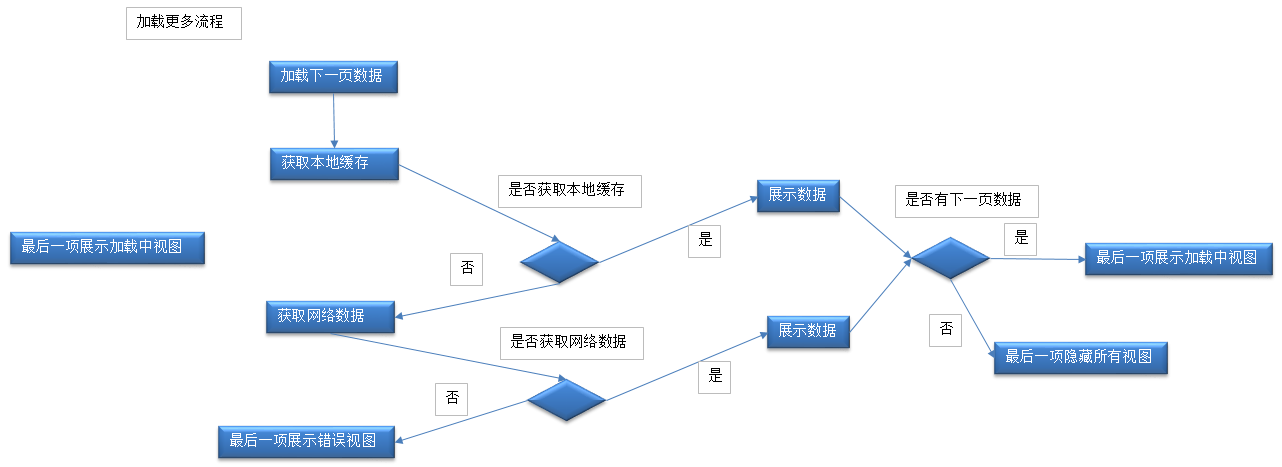
在新闻项目中已经详细分析过上拉和下拉的内容了，我们在这个项目采用另一个思路来完成该功能。

处理思路：

加载数据的时候默认最后一项就是加载数据的条目，当最后一项展示出来的时候标识已经滚动到条目的底部了，我们可以在加载最后一条时向服务器发送请求（获取下一页）。如图：



### 加载下一页流程



## 详情页面

### 折叠动画（fold）：

1. 数据加载完成后每个控件都有自己的完整高度，展示给用户的是折叠之后的高度。
   1. 高度值：完整、折叠
2. 动画处理：
   1. 展开
      1. 由折叠高度变换成完整高度，高度变换动画；
      2. 箭头由向下状态旋转180度变为向下，旋转动画；
   2. 合并
      1. 由完整高度变换成折叠高度，高度变换动画；
      2. 箭头由向上状态旋转180度变为向上，旋转动画；

### 使用ValueAnimator实现动画的步骤

一般使用ValueAnimator实现动画分为以下七个步骤：

      1. 调用ValueAnimation类中的ofInt(int...values)、ofFloat(String propertyName,float...values)等静态方法实例化ValueAnimator对象，并设置目标属性的属性名、初始值或结束值等值;

      2.调用addUpdateListener(AnimatorUpdateListener mListener)方法为ValueAnimator对象设置属性变化的监听器;

      3.创建自定义的Interpolator，调用setInterpolator(TimeInterpolator value)为ValueAniamtor设置自定义的Interpolator;(可选，不设置默认为缺省值)

      4.创建自定义的TypeEvaluator,调用setEvaluator(TypeEvaluator value)为ValueAnimator设置自定义的TypeEvaluator;(可选，不设置默认为缺省值)

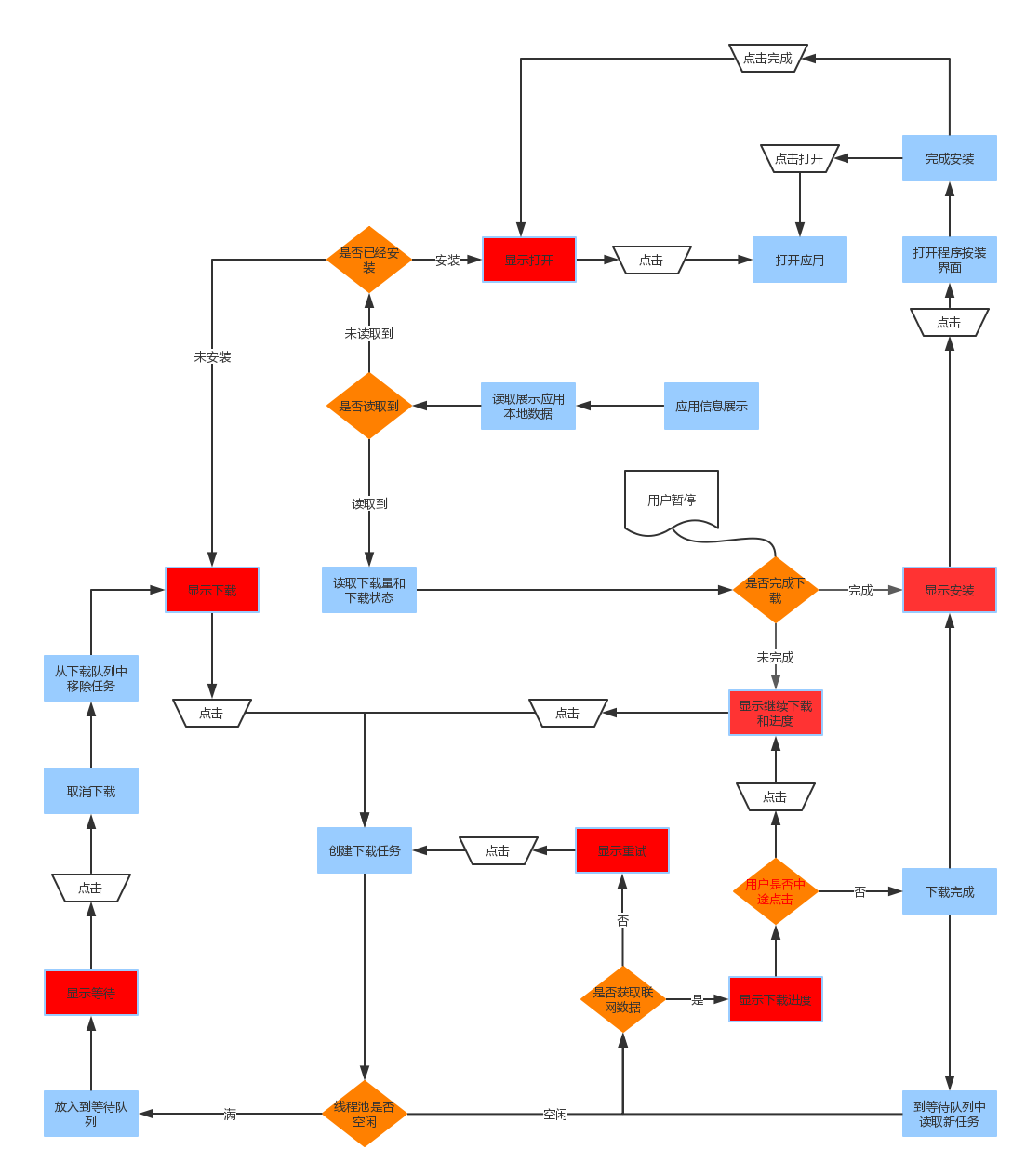
      5.在AnimatorUpdateListener 中的实现方法为目标对象的属性设置计算好的属性值。

      6.设置动画的持续时间、是否重复及重复次数等属性;

      7.为ValueAnimator设置目标对象并开始执行动画。

## 下载

流程图：



### 下载流程处理

分为三个阶段处理：

1. 应用信息展示
   1. 使用数据库记录应用信息状态
   2. 读取数据
   3. 展示数据
2. 下载管理
   1. 线程池管理
      1. 下载线程
      2. 等待队列
   2. 新应用下载
   3. 断点下载
3. 安装和卸载
   1. 注册应用安装和卸载广播
   2. 安装管理：
      1. 保持信息进数据库
      2. 更新应用状态和界面
   3. 卸载管理：
      1. 删除数据库记录
      2. 删除已经下载apk文件
      3. 更新应用状态和界面

### 应用信息展示

* 使用数据库记录应用信息状态
  + 数据库表设计

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 含义 | 类型 | 备注 |
| Id | 主键 | long | 工具维护的自增主键 |
| app\_id | 应用标识 | long |  |
| package\_name | 应用包名 | 字符串 |  |
| size | 文件大小 | long |  |
| download\_size | 已经下载大小 | long |  |
| state | 应用状态 | int | 已经安装  未下载（不记录）  下载完成  下载等待  下载暂停  下载中  下载过程中出现异常 |

* + Sugar轻量级数据库工具使用

官方链接：http://satyan.github.io/sugar/getting-started.html

使用步骤：

1. 配置（添加依赖）

compile 'com.github.satyan:sugar:1.4'

1. 使用
   1. AndroidManifest.xml配置application节点
      1. android:name="com.orm.SugarApp"
      2. 配置meta-data
         1. DATABASE：数据库文件名
         2. VERSION：版本号
         3. QUERY\_LOG：日志管理
         4. DOMAIN\_PACKAGE\_NAME：DOMAIN实体Bean，实体Bean所在包的名字
      3. 创建实体Bean，作为SugarRecord子类（eg. public class Book extends SugarRecord）

* 读取数据
* 展示数据
  + 状态和展示文本对应关系

|  |  |  |  |
| --- | --- | --- | --- |
| 已经安装 | 打开 | INSTALL\_ALREADY | 1 |
| 未下载 | 下载 | DOWNLOAD\_NOT | 2 |
| 下载完成 | 安装 | DOWNLOAD\_COMPLETED | 3 |
| 下载等待 | 等待 | DOWNLOAD\_WAIT | 4 |
| 下载暂停 | 继续 | DOWNLOAD\_STOP | 5 |
| 下载中 | 进度展示 | DOWNLOADING | 0 |
| 下载过程中出现异常 | 重试 | DOWNLOAD\_ERROR | -1 |
|  |  |  |  |

* + 使用TextView展示
  + 使用自定义控件展示，为下载管理做准备

### 下载管理

* 线程池管理
  + 下载队列：使用Executors创建固定大小的线程池，控制执行线程的数量，超出部分放到等待队列，当执行任务完成后，需要检查一下等待队列。
  + 等待队列：添加、获取任务，添加放到队列的尾部，获取从队列的头部。
* 新应用下载
  + 下载任务
    - 联网下载数据
    - 封装下载中需要使用到的数据
    - 存在问题
      * 下载任务与界面条目的对应关系：给DownloadInfo类增加一个应用标识的字段。
      * 每次条目设置数据的时候，都会从本地数据库中读取数据：对读取出来的数据进行缓存，当用户向上滚动屏幕，不需要再次从数据库中查询。
      * 更新问题
        + 思路：在下载的过程中会修改DownloadInfo中信息，当DownloadInfo信息修改的时候希望界面能够做出更新。
        + 解决步骤：

必须持有界面的引用，用于是使用RecyclerView展示一组条目，我们需要将条目存储到容器中；

当Download数据更新的时候，调用该容器做循环的操作，发现当前展示的是下载的应用，做更新界面的动作

* + - 自定义控件处理状态和进度展示
      * 进度显示
        + 需要知道圆心、半径、圆环的宽度（灰色）
        + 绘制进度（红色），外界可以设置进度
      * 状态对应的文字信息绘制（蓝色）
        + 如何将文字绘制到圆环中心
        + 外界可以设置文字
  + 点击时间处理
    - 不同状态对应的点击动作

|  |  |  |
| --- | --- | --- |
| 状态 | 显示文本 | 点击动作 |
| 已经安装 | 打开 | 打开应用 |
| 未下载 | 下载 | 创建下载任务进行下载 |
| 下载完成 | 安装 | 应用安装 |
| 下载等待 | 等待 | 删除任务 |
| 下载暂停 | 继续 | 创建下载任务进行下载 |
| 下载中 | 进度展示 | 暂停 |
| 下载过程中出现异常 | 重试 | 创建下载任务进行下载 |

### 安装和卸载

* 注册应用安装和卸载广播

<**receiver android:name=".receiver.AppChange"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.PACKAGE\_ADDED"** />  
 <**action android:name="android.intent.action.PACKAGE\_REMOVED"** />  
 <**data android:scheme="package"** />  
 </**intent-filter**>  
</**receiver**>

了解：

ACTION\_PACKAGE\_ADDED 一个新应用包已经安装在设备上，数据包括包名（最新安装的包程序不能接收到这个广播）  
ACTION\_PACKAGE\_REPLACED 一个新版本的应用安装到设备，替换之前已经存在的版本  
ACTION\_PACKAGE\_CHANGED 一个已存在的应用程序包已经改变，包括包名  
ACTION\_PACKAGE\_REMOVED 一个已存在的应用程序包已经从设备上移除，包括包名（正在被安装的包程序不能接收到这个广播）  
ACTION\_PACKAGE\_RESTARTED 用户重新开始一个包，包的所有进程将被杀死，所有与其联系的运行时间状态应该被移除，包括包名（重新开始包程序不能接收到这个广播）  
ACTION\_PACKAGE\_DATA\_CLEARED 用户已经清楚一个包的数据，包括包名（清除包程序不能接收到这个广播）

* 安装管理：
  + 保持信息进数据库
  + 更新应用状态和界面
* 卸载管理：
  + 删除数据库记录
  + 删除已经下载apk文件
  + 更新应用状态和界面

### 应用信息入库

入库信息：

1. 下载信息：从下载开始进行信息记录，主要记录下载进度，和状态。状态包括：暂停、完成、下载错误。
2. 安装和卸载信息：安装信息入库，卸载后删除数据库记录和apk文件。