# 《面向对象的软件构造实践》

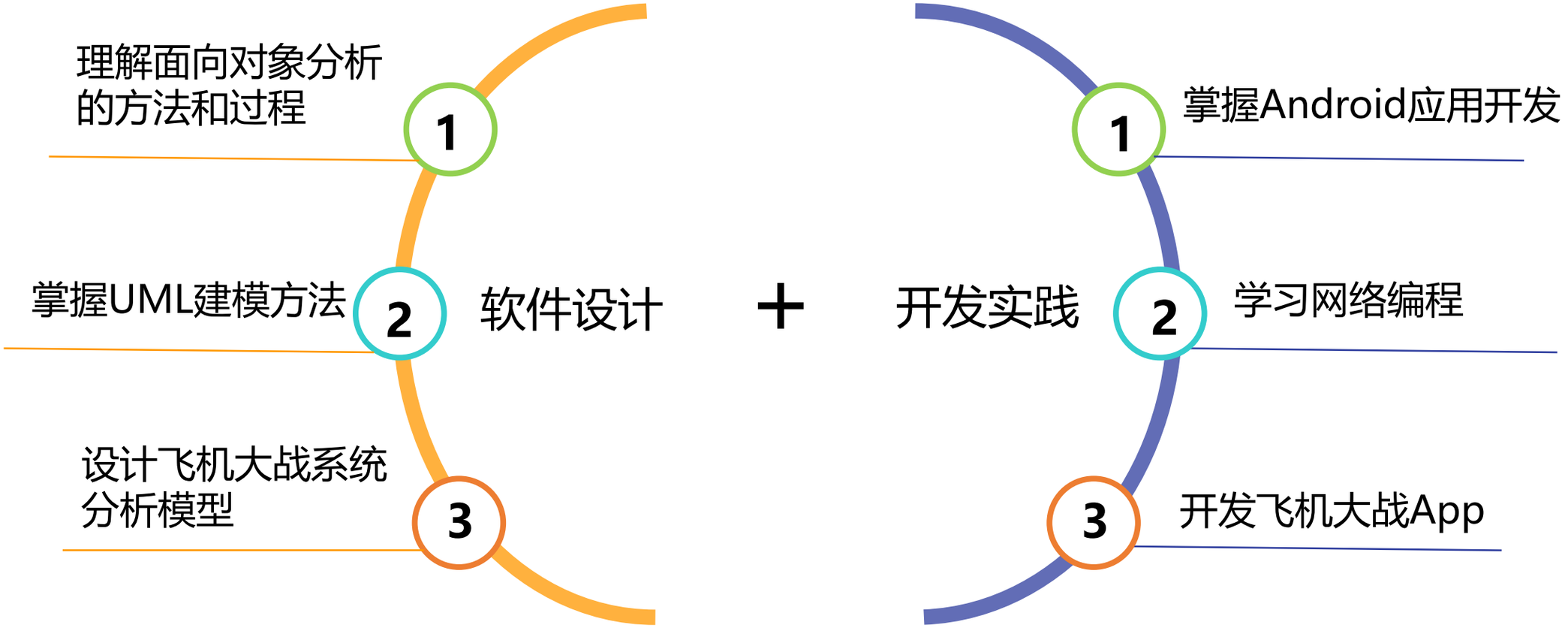
## Android开发技术基础



HITSZ 实验与创新实践教育中心

Education Center of Experiments and Innovations, HITSZ

# 课程安排


理解面向对象分析的方法和过程
1
掌握UML建模方法
2
设计飞机大战系统分析模型
3
软件设计
+
开发实践
1
掌握Android应用开发
2
学习网络编程
3
开发飞机大战App

# 课程回顾



## 面向对象的软件构造导论：


- Java语言
- 面向对象思想
- 设计模式
- 多线程、泛型
- Swing、网络编程
- PC客户端飞机大战

## 面向对象的软件构造实践：


- Android 飞机大战
- Android平台架构
- Android开发技术基础
- Android开发实践
- 调试、打包、发布等

# 课程内容





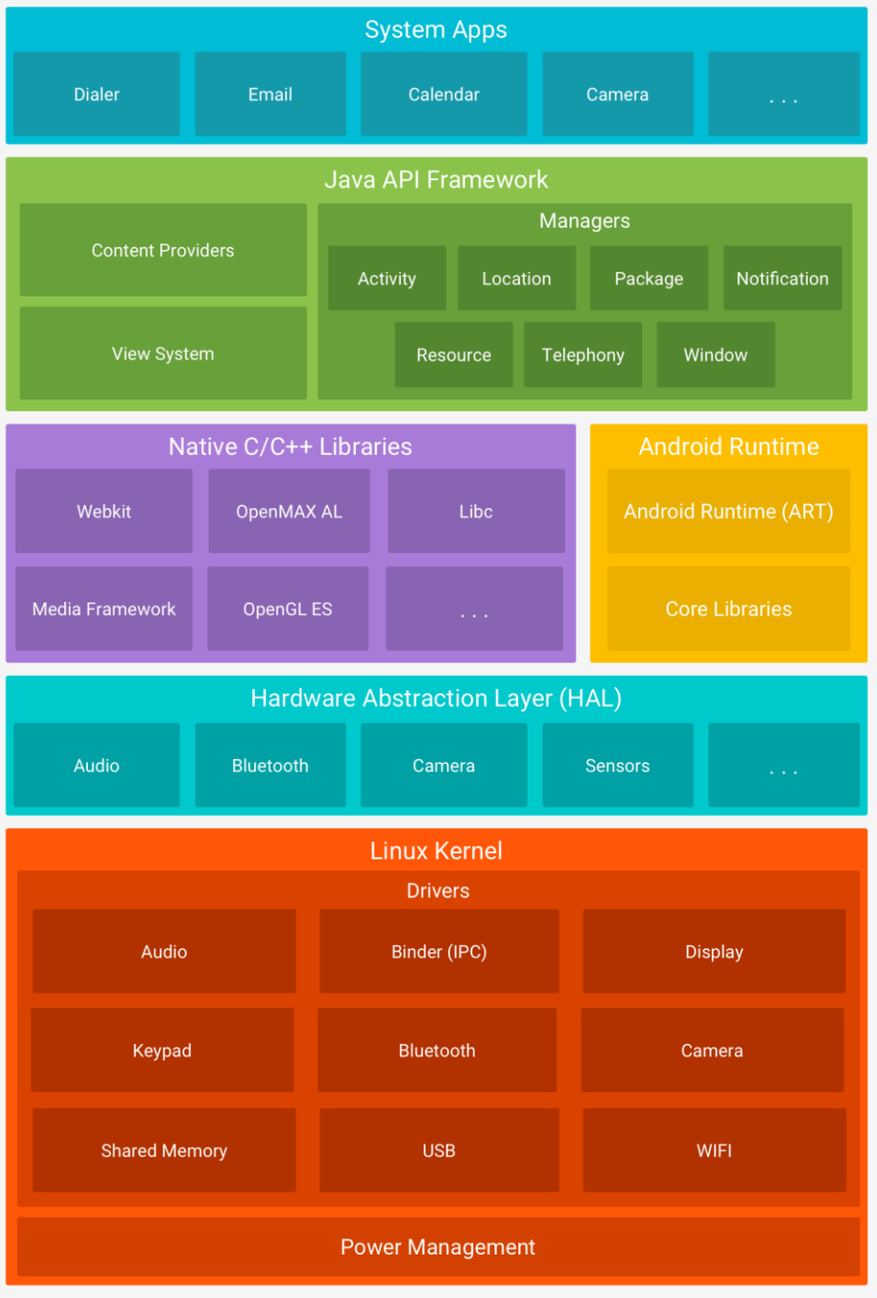
一、 Android平台架构



二、 Android应用开发技术基础

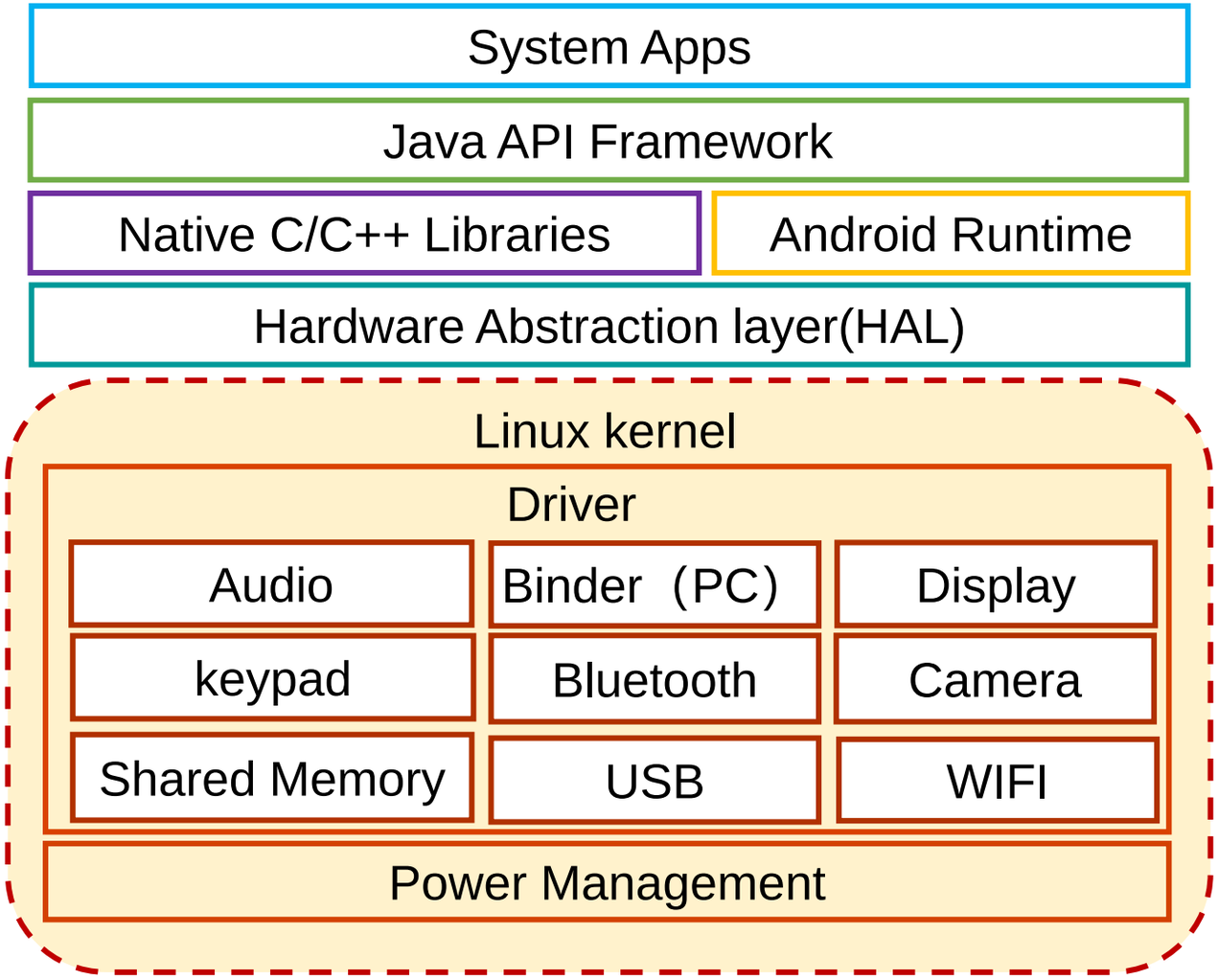
# Android平台架构





# Android平台架构






Android系统是Google开发的基于Linux平台的开源手机操作系统。

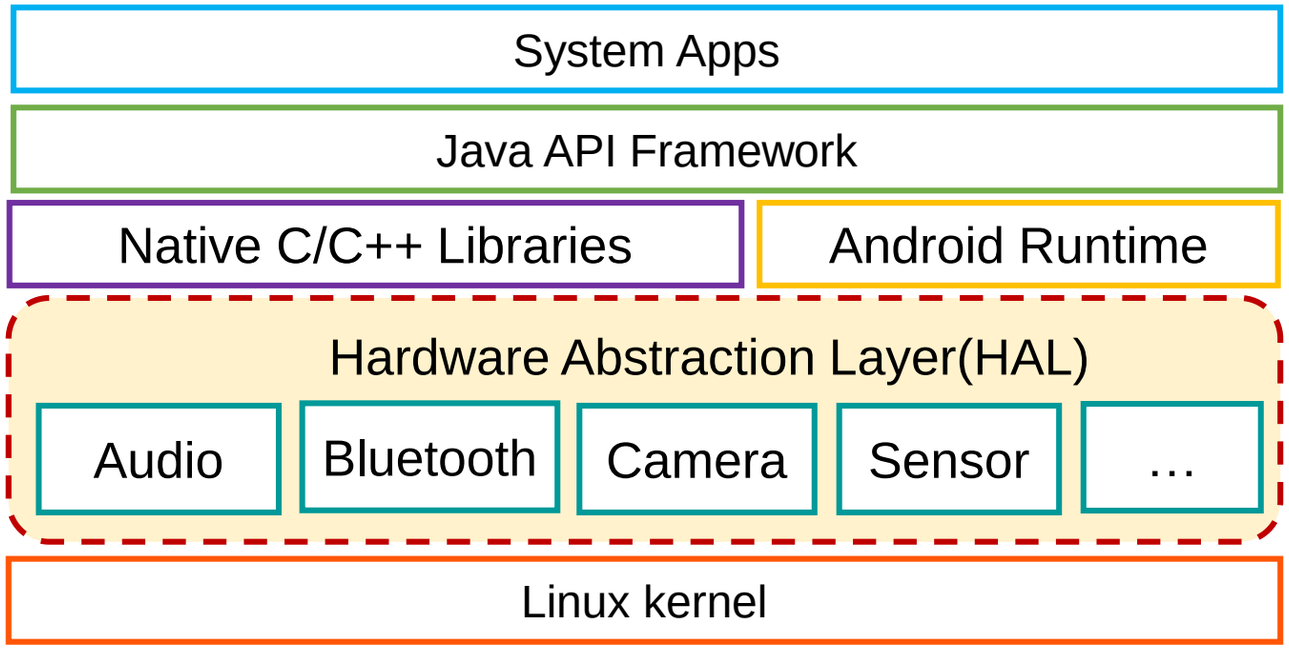
## 1. Linux 内核

通过借助Linux内核服务实现一些核心功能：

- 硬件设备驱动
- 进程和内存管理
- 网络协议栈
- 电源管理
- 无线通信等。

# Android平台架构






2. 硬件抽象层 (HAL)

对Linux内核驱动程序的封装，将硬件抽象化，屏蔽掉了底层的实现细节。

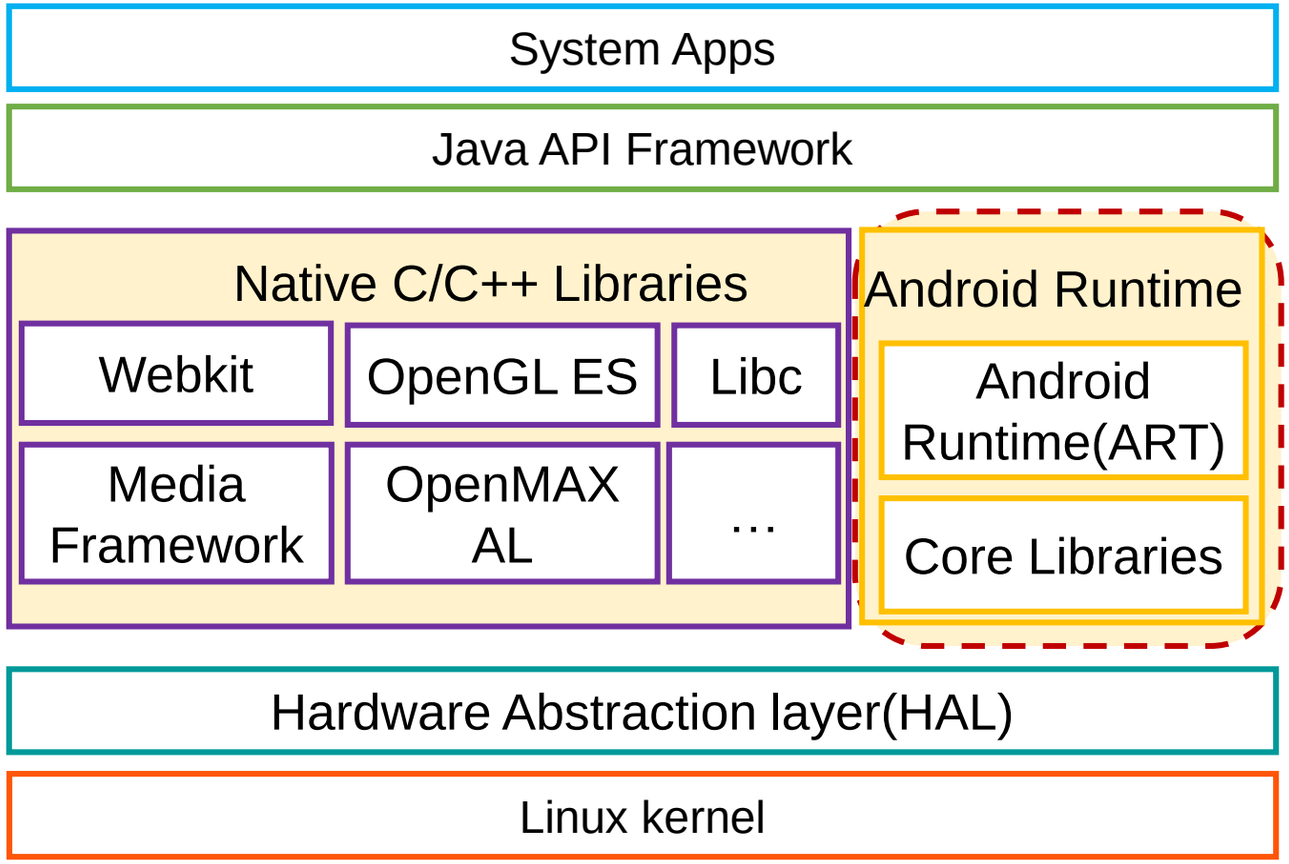
HAL规定了一套应用层对硬件层读写和配置的统一接口;

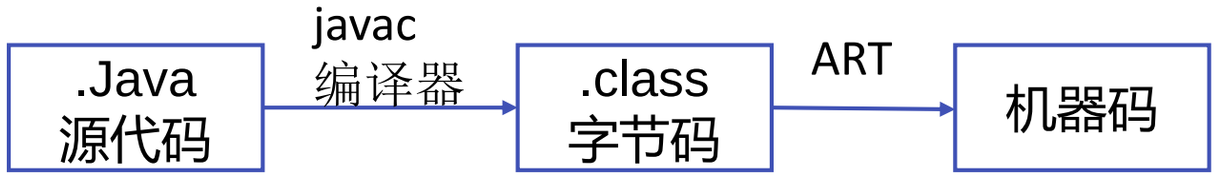
**Android系统架构：**

https://baijiahao.baidu.com/s?id=1661393081435125580&wfr=spider&for=pc

# Android平台架构






## 3. Android Runtime

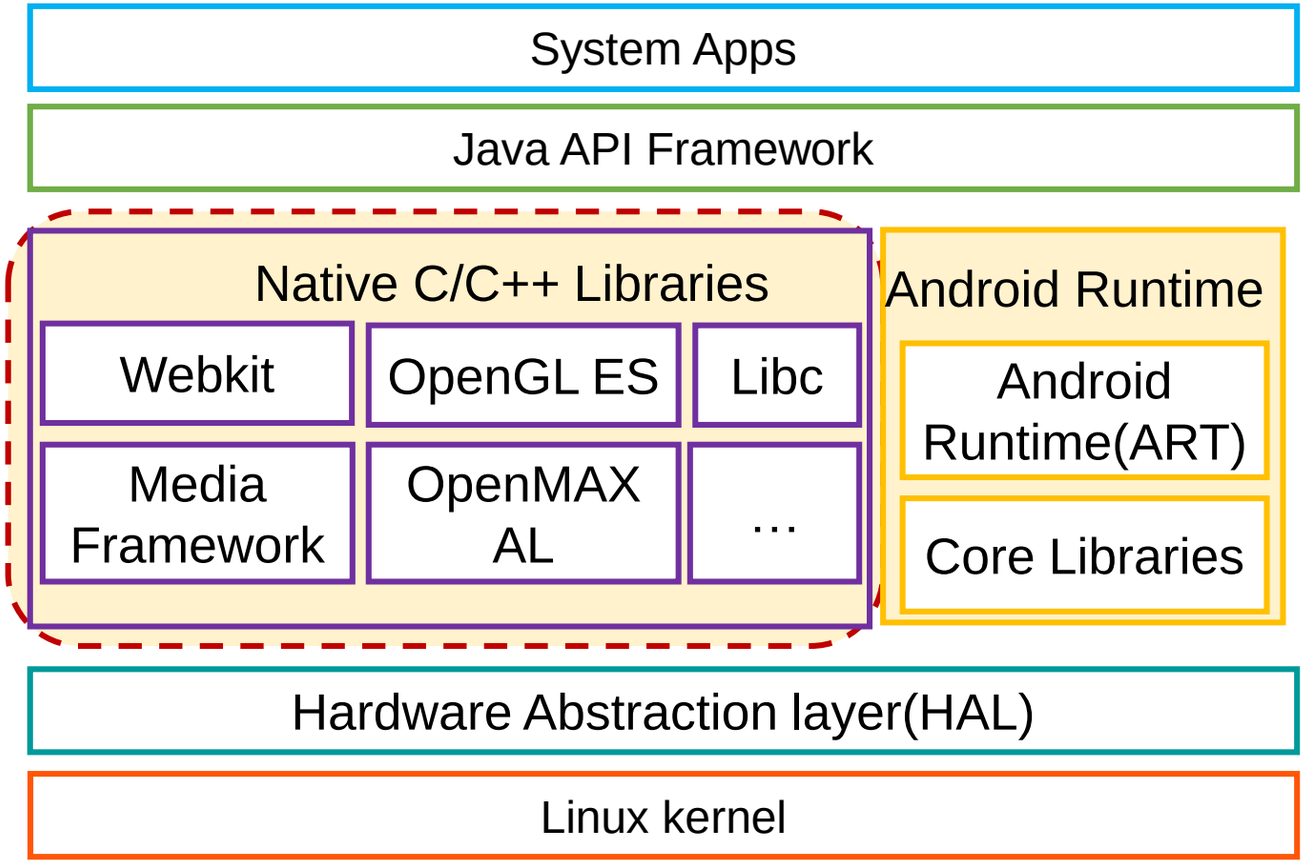
Android Runtime（缩写为ART），是一种在Android系统上的运行环境。

- Android应用层和Framework是Java，Java代码被编译打包成字节码的dex或者odex文件；
- 而操作系统运行时需要的是机器码。

作用：Java代码转换成 CPU处理器能够理解的机器码。

# Android平台架构





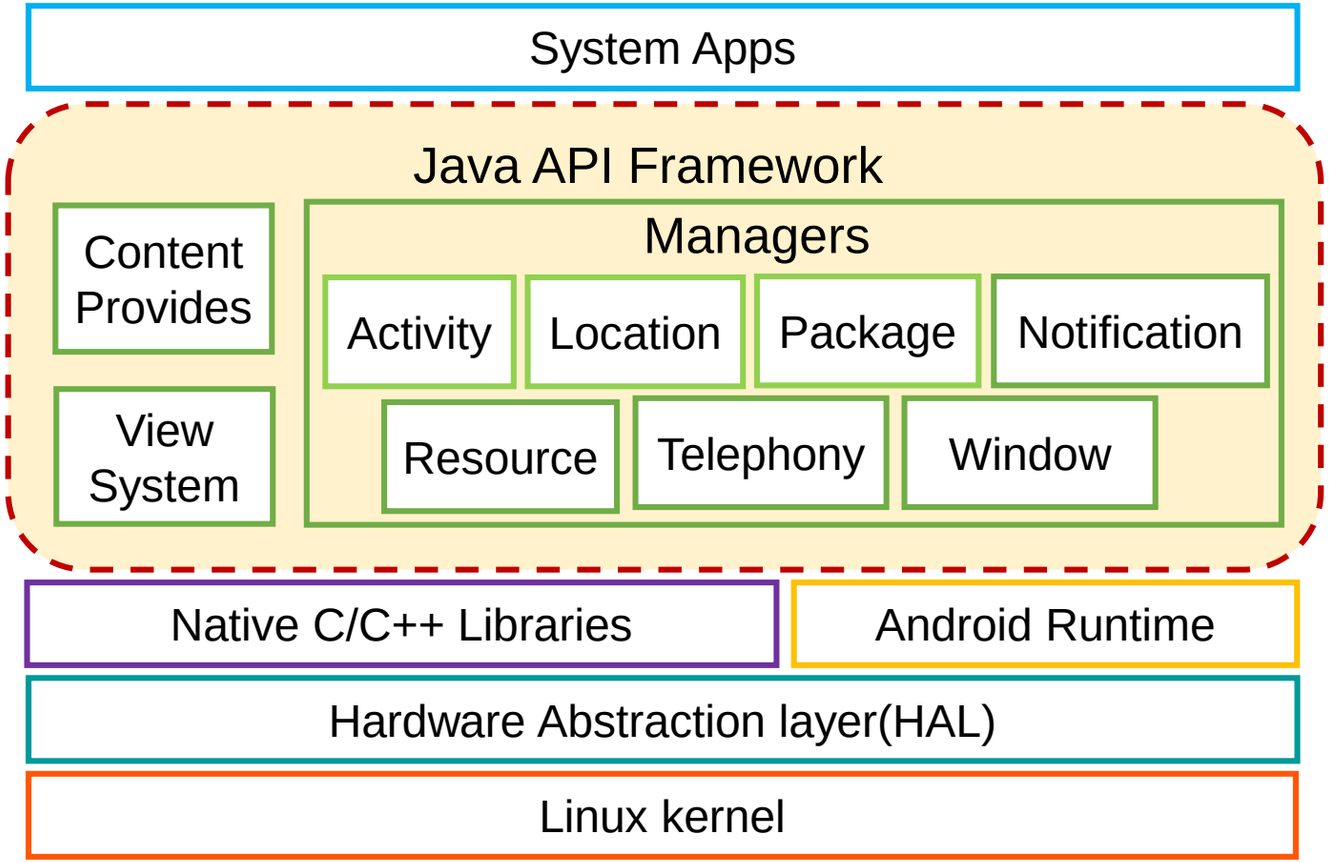

## 4. 原生C/C++库

- ✓ Android系统类库大部分由C/C++编写
- ✓ 通过Java Framework层以Java API 的形式向应用层显示其中部分原生库的功能。

- ➢ Webkit：Web浏览器的软件引擎;
- ➢ OpenGL：开放式图形库，是用于渲染2D、3D矢量图形的跨语言、跨平台的应用程序编程接口（API）
- ➢ SQLite：本地小型关系数据库;
- ➢ Media Framework：多媒体库，支持多种常用的音频和视频格式的录制和回放，所支持的编码格式包括MPEG4，MP3，H264，AAC，ARM

# Android平台架构





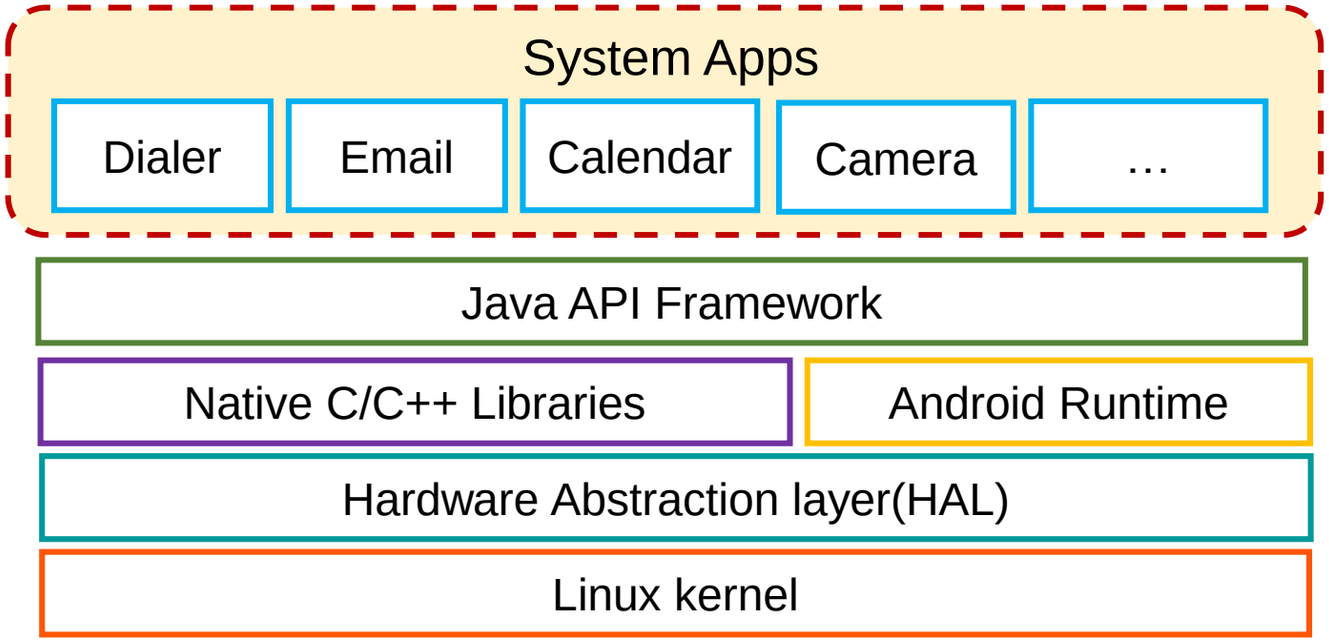

## 5. Java API FrameWork

由Java语言编写，提供了Android 系统整个功能集的API集合；

例如：
内容提供程序
视图系统
资源管理器
通知管理器
Activity管理器等；

# Android平台架构






## 6. 系统应用

① 各类与用户直接交互的应用程序;

② 运行于后台的服务程序。

电子邮件客户端
SMS程序
日历
地图
浏览器
联系人
其他设置等;

# Android平台架构



1. 了解Android系统架构分层的意义：

① 从宏观上认识了Android系统，同时也给我们的学习与实践指明了方向；

A. 若是从事Android应用开发，那应该研究Android的应用框架层和应用程序层；

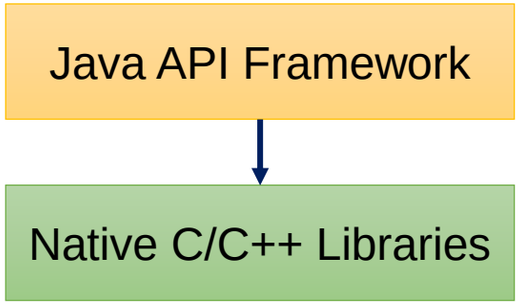
B. 若是从事Android系统开发，那应该研究Android的系统库和Android运行时。

C. 若是从事Android驱动开发，那应该研究Android的Linux内核。

② 采用分层架构的思想，架构清晰，层次分明，协同工作；

# Android平台架构






2. 为什么需要Native?

① 执行速度快。NativeC/C++代码执行速度比Java快。

② 方便移植。C/C++ 开发的库可以直接移植到Native层。

3. Java如何调用Nativie接口?

① JNI：Java Native Interface，提供了API实现Java和其他语言的通信（主要是C&C++）。

② NDK：Native Development Kit，快速开发C、C++动态库，并将so和应用一起打包成APK。

| | JNI | NDK |
|---|---|---|
| 定义 | Java中的接口 | Android中的工具开发包 |
| 作用 | 用于Java和本地语言（如C、C+）的交互 | 快速开发C、C++动态库，并将so和应用一起打包成APK |
| 关系 | JNI是实现的目的，NDK是在Android中实现JNI的手段，即在Android的开发环境中（Android Studio），通过NDK从而实现JNI的功能 | |

# 课程内容





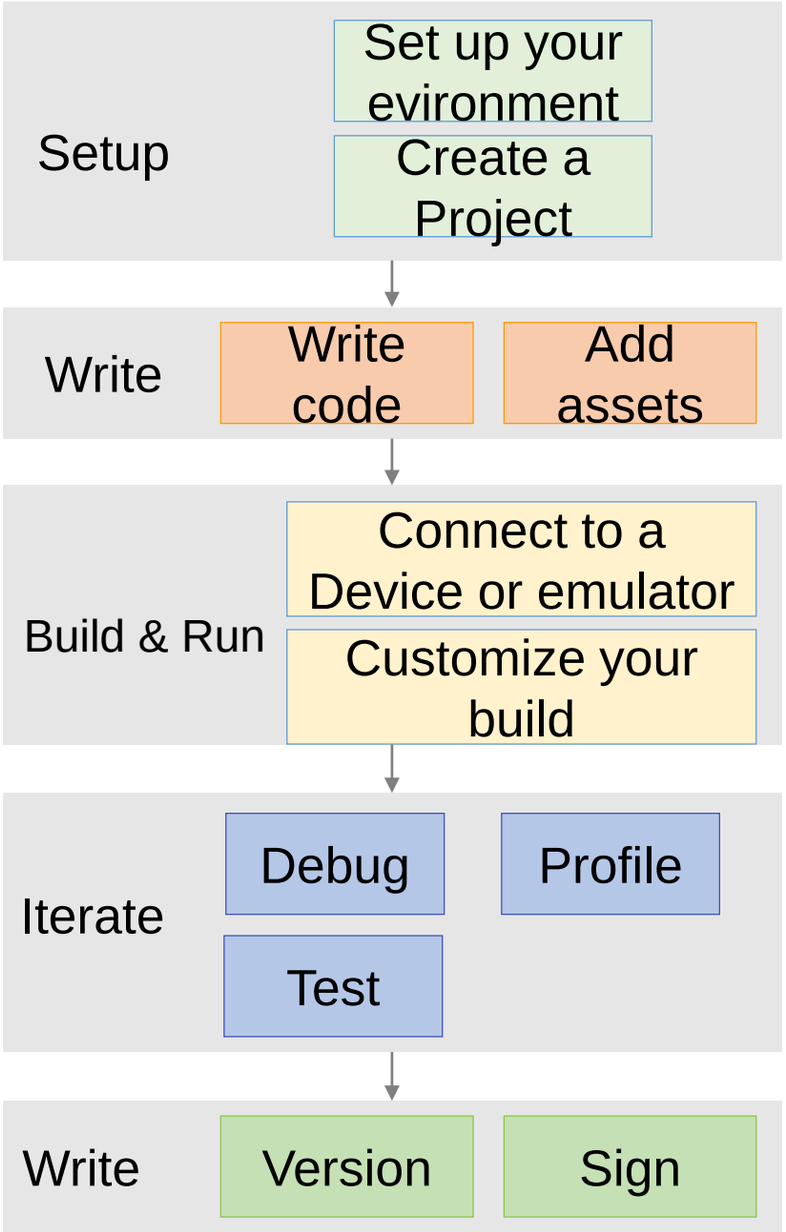
一、 Android平台架构



二、 Android应用开发技术基础

# Android应用开发基本流程






1. 设置工作区

安装 Android Studio 并创建项目。

2. 编写应用

设计应用UI、交互、完成网络请求、数据处理等业务逻辑编写。

3. 构建并运行

编译运行，将项目构建成一个可调试的 APK 软件包，以便在模拟器或 Android 设备上安装和运行。

4. 调试、剖析和测试

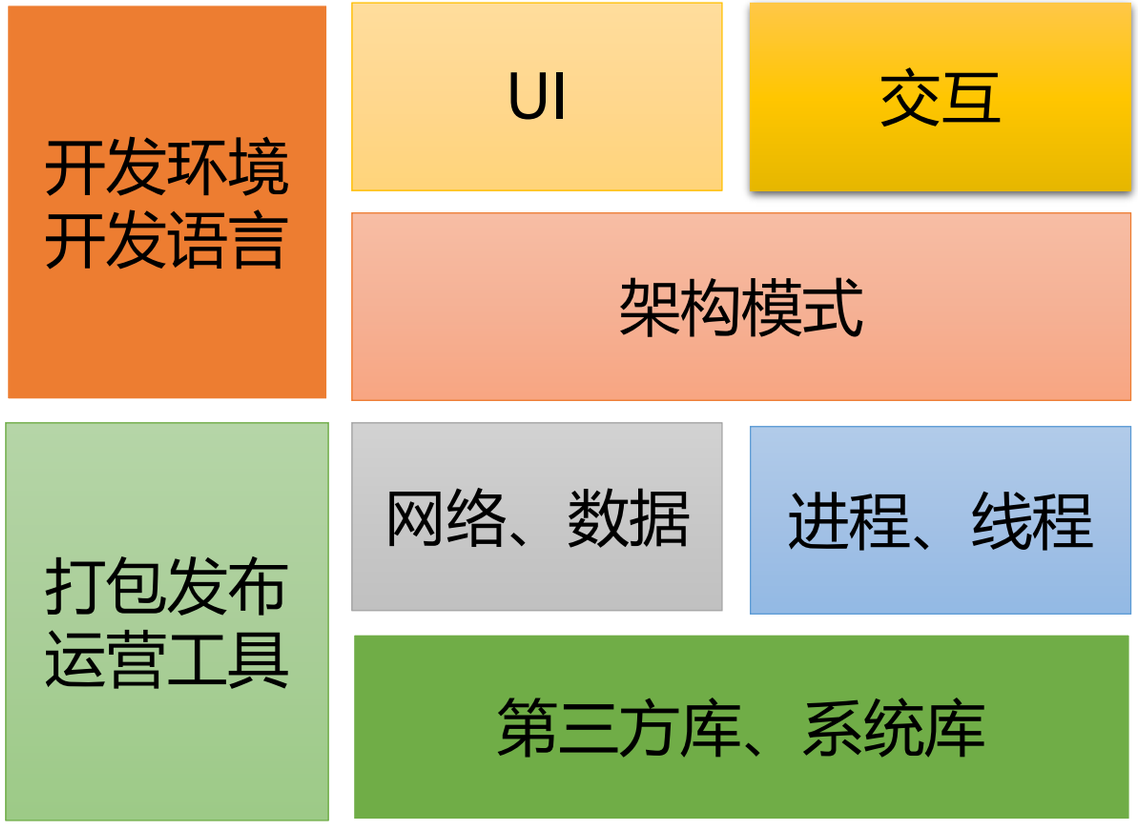
消除错误并优化应用性能。查看和分析各种性能指标（如内存使用情况、网络流量、CPU占用等）。

5. 发布

当准备向用户发布应用时，需要考虑管理应用版本、运营、构建APK包、使用密钥为应用签名。

# Android应用开发技术总览






1. 开发语言、开发环境
2. UI(User Interface )开发
3. 交互开发
4. 架构模式
5. 网络编程、数据存储
6. 进程、线程基础
7. 第三方库
8. 打包发布运营

# 1 开发语言、开发工具





IntelliJ IDEA：Java集成开发环境，支持Android开发，还支持JavaSE，Groovy，Scale，HTML，CSS，PHP等语言的开发，各类版本工具(git、svn等)、JUnit。


推荐使用

Android Studio：是谷歌开发的一款专注Android开发的集成开发环境；
安装简洁，功能强大，整合了Gradle便于工程和代码组件管理，启动速度快，内存占用低。

Android Studio安装步骤和注意事项请参考实验指导书！

# 2 UI开发





UI：User Interface

UI交互设计：是指对软件的人机交互、操作逻辑、界面美观的整体设计。

飞机大战UI元素：

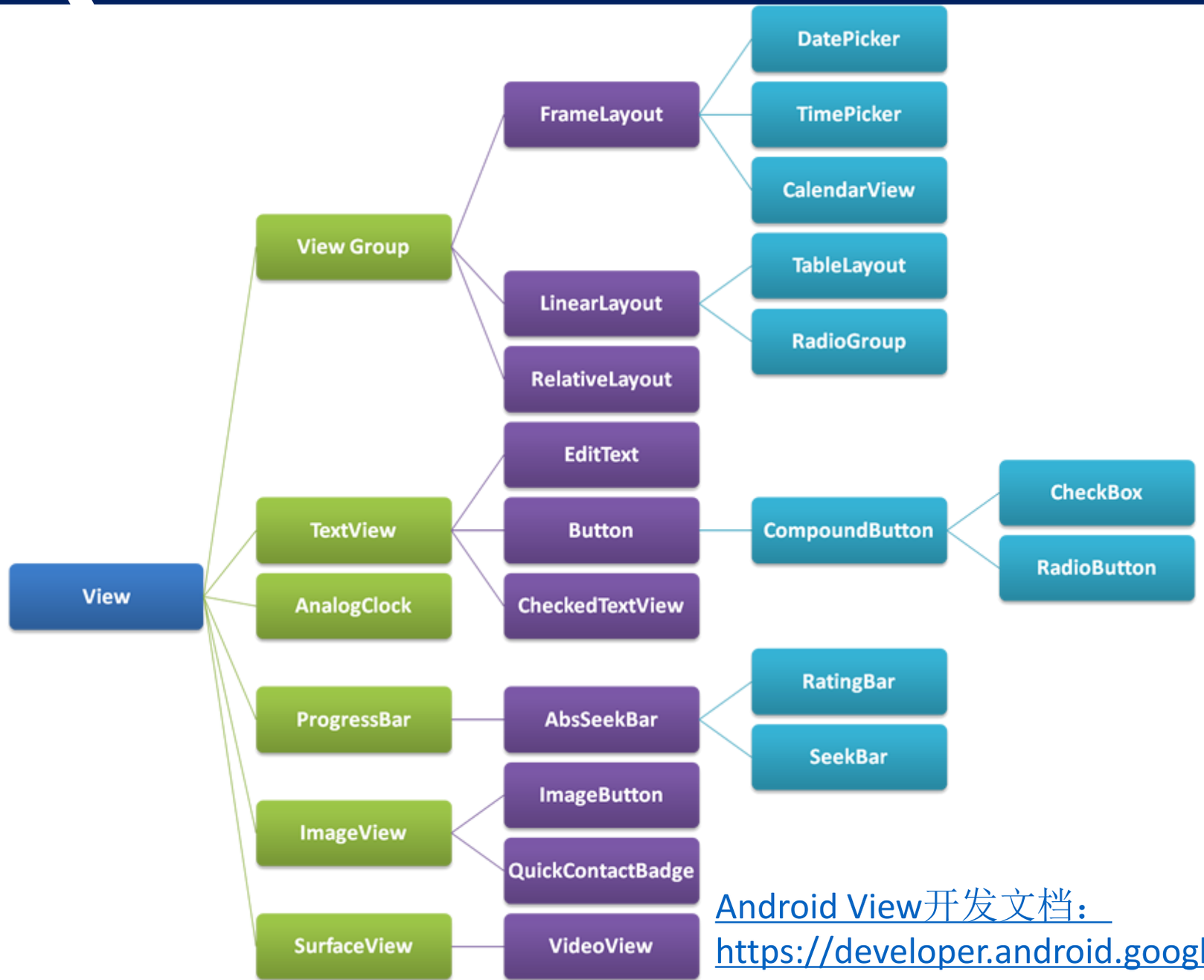
① View：TextView、IamgeView、Button…

② ViewGroup：LinearLayout、RelativeLayout…

③ Activity、Fragment…

# Android View






Android View开发文档：
https://developer.android.google.cn/reference/android/view/View?hl=en

# SurfaceView



**Android游戏开发常用的三种视图：**View、SurfaceView和GLSurfaceView。

**View** → **SurfaceView** → **GLSurfaceView**

- 显示视图，内置画布，图形绘制函数、触屏事件、按键事件函数等；
- 必须在UI主线程内更新画面，速度较慢。

- View类的子类；
- 使用双缓冲机制；
- 在新的子线程中更新画面，刷新界面速度比View快。
- 适用于2D游戏的开发。

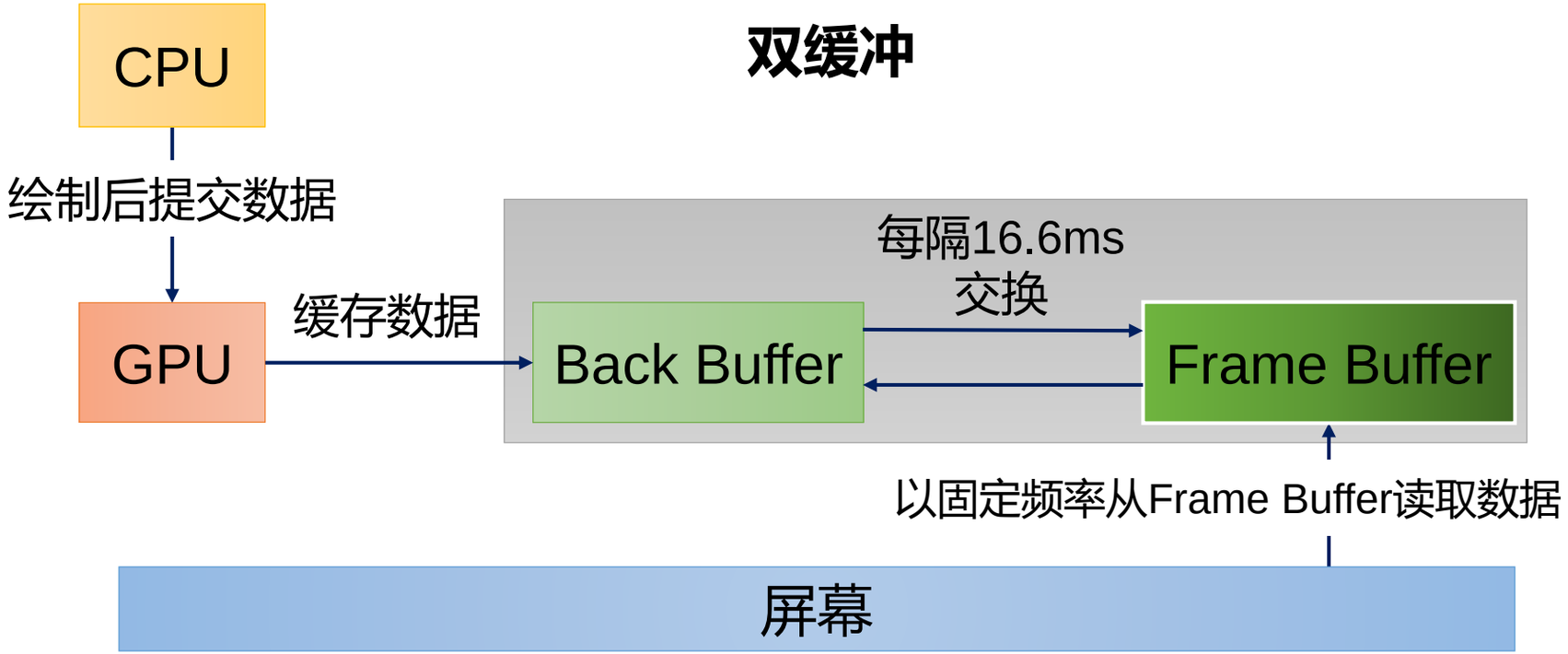
- 基于SurfaceView视图再次进行拓展的视图类，专用于3D游戏开发的视图；
- 是SurfaceView的子类，OpenGL专用。

UI的主线程中更新画面可能会引发问题：

假如更新画面的时间过长，导致主UI线程会被正在画的函数阻塞，从而无法响应按键，触屏等消息。

# SurfaceView 双缓冲






- 双缓冲优点

① 绘制过程中不会出现闪烁花屏现象；

② 高效，将图像一次性绘制到屏幕上比一次绘制一部分要高效的多。

- 双缓冲缺点

图片过大时会严重消耗内存。

# SurfaceView案例演示



参考附件代码：MySurfaceView.zip

# 3 交互





① 按钮点击；

② 手势、移动鼠标、点击屏幕；

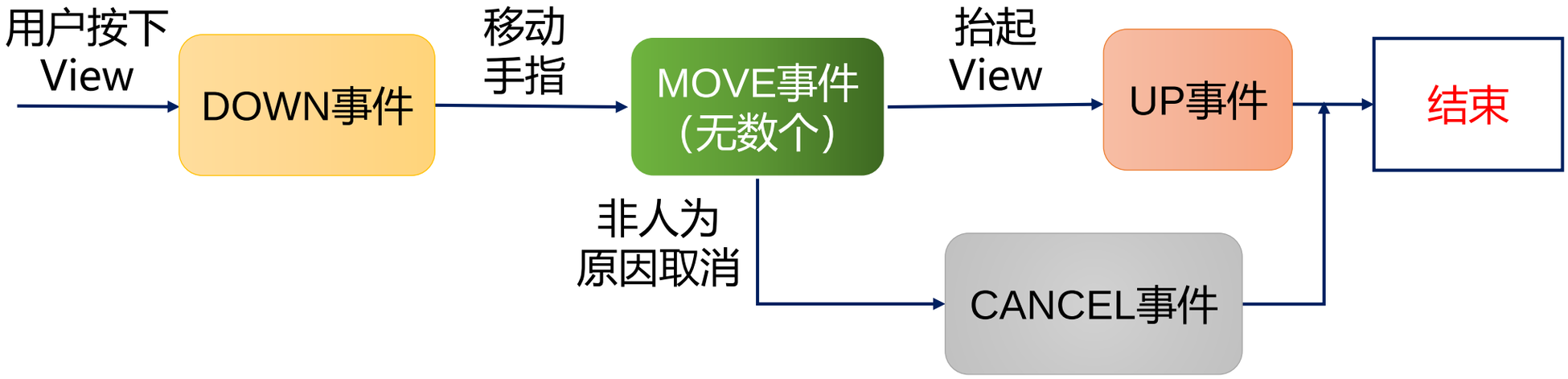
③ 点击系统Home键、点击系统back键……

……

# 交互事件模型



交互事件:




- Android两种事件处理模型

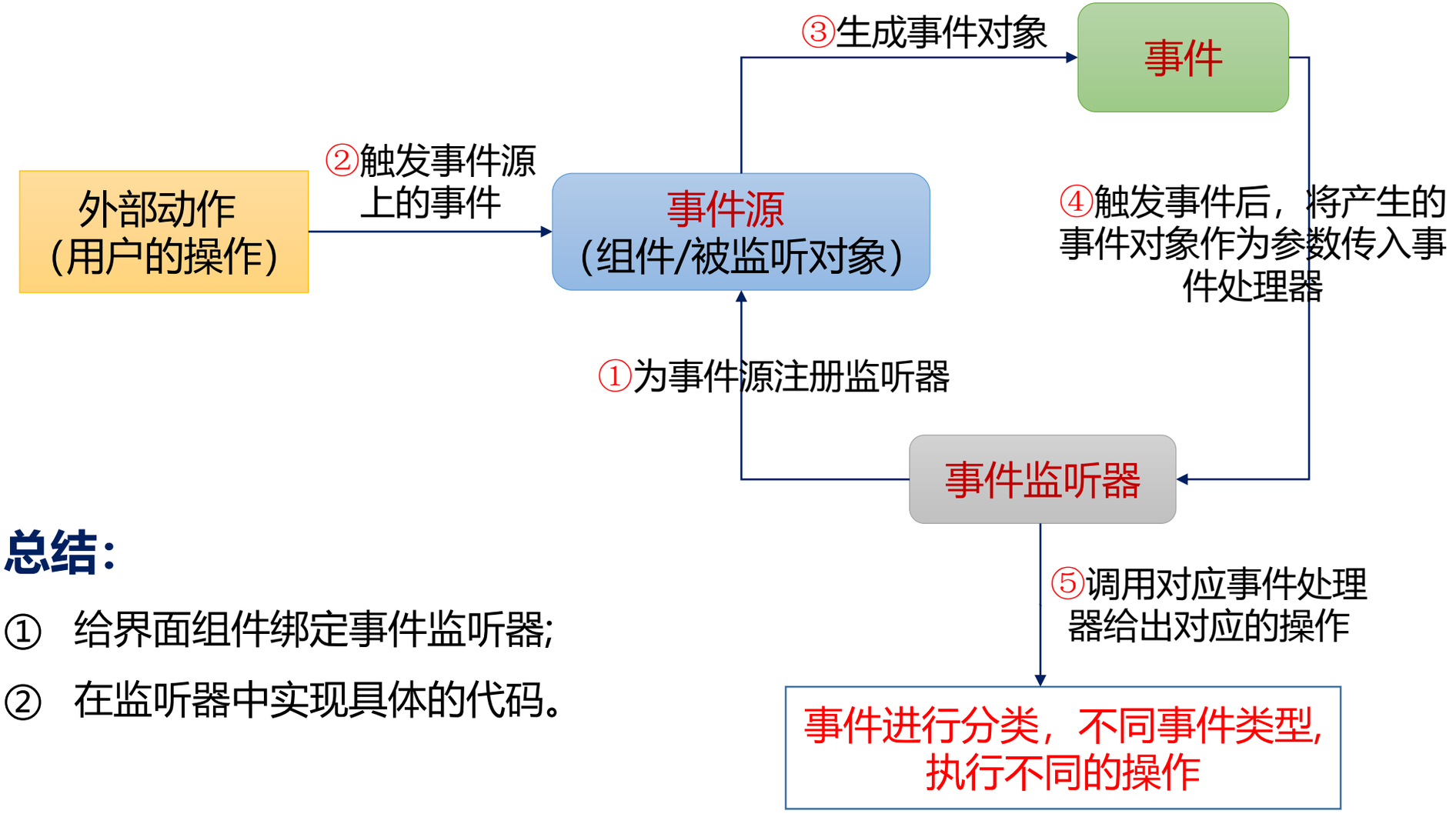
① 基于监听的事件处理;

② 基于回调的事件处理;

# 基于监听的事件处理



## 1. 基于监听的事件处理机制——流程模型图




**总结：**

① 给界面组件绑定事件监听器;

② 在监听器中实现具体的代码。

# 基于回调的事件处理



## 2. 基于回调机制的事件处理

① 无事件监听器；

② 用户在组件上触发某个事件时，组件自己特定的方法负责处理该事件。

③ 为了实现回调机制的事件处理，Android为所有的GUI组件都提供了回调方法；

④ 以View为例来说明，可以通过重写View中的这些回调方法来实现需要的响应事件。

应用场景：自定义View控件

*boolean onKeyDown(int keyCode , KeyEventevent);用户在该组件上按下时触发的

*boolean onKeyLongPress(int keyCode ,keyEvent event);用户在该组件上长按时触发的

*boolean onkeyShorcut(int keyCode ,keyEvent event);当一个键盘快捷键事件发生时触发该方法

*boolean onkeyUp(int keyCode , keyEventevent);用户松开按键的时候触发的事件

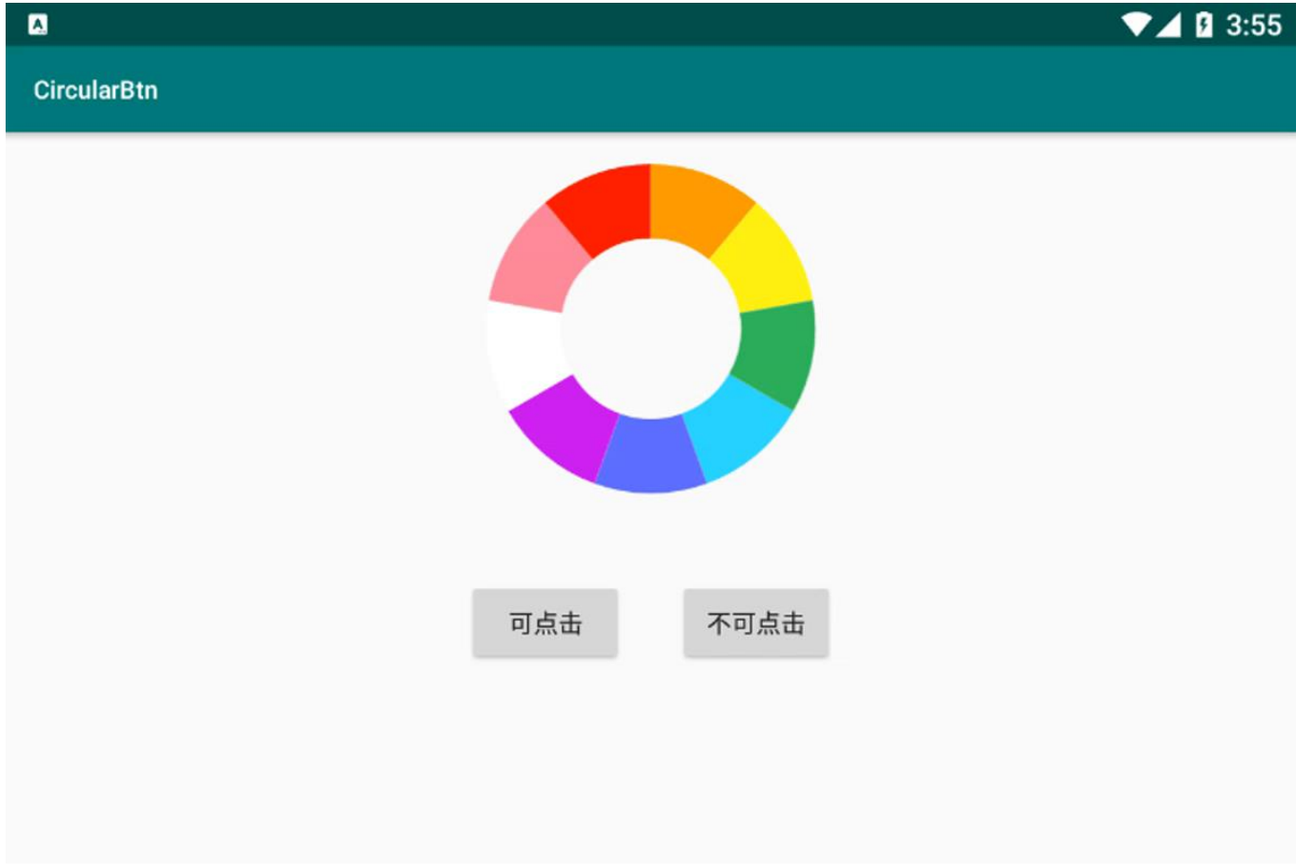
*boolean onTouchEvent(MotionEvent event);用户在该组件上触发触摸屏时触发该方法

*boolean onTrackballEvent(MotionEventevent);用户在该组件商触发轨迹球时触发该方法

# 案例



参考附件代码：CircularBtn-master.zip

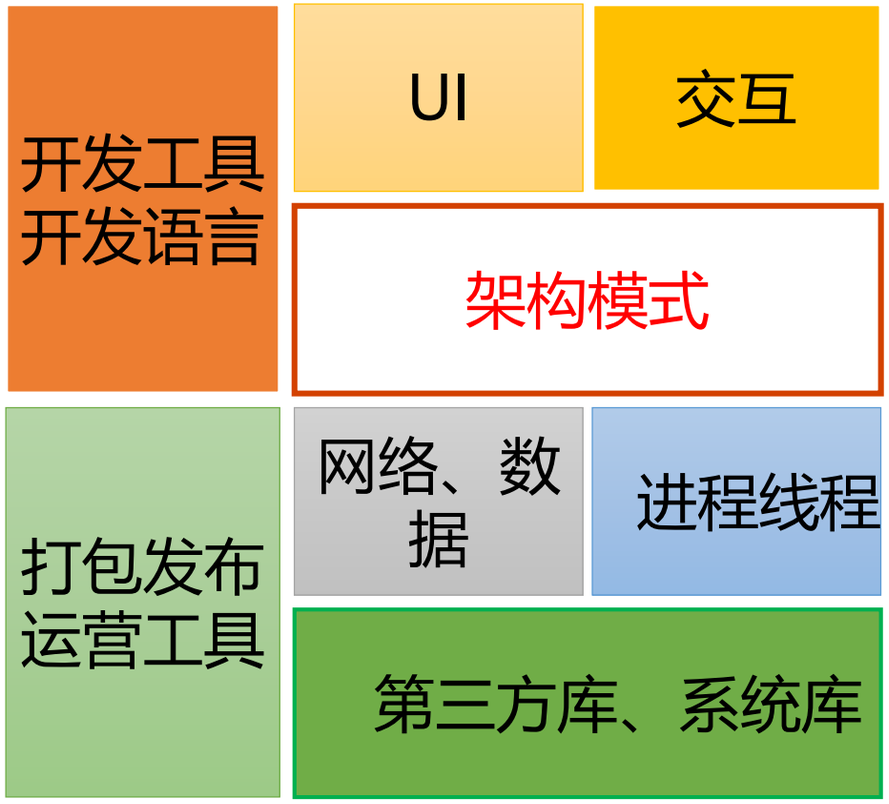



1、CirculBtn继承View。

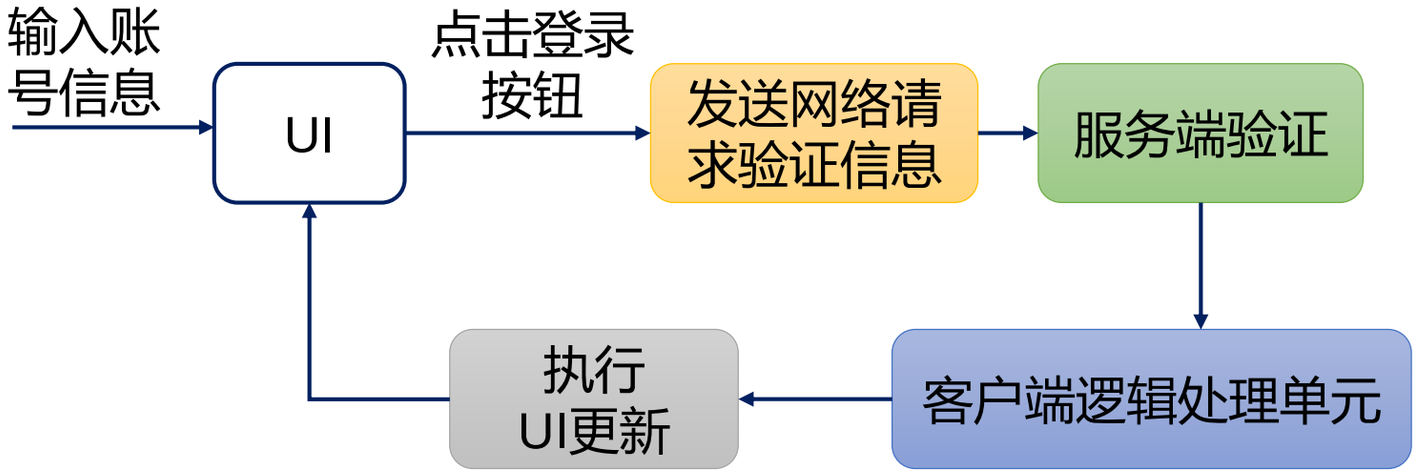
2、重写了OnChangeSized、OnDraw、OnTouchEvent方法。

# 4 架构模式






场景：登录




Android应用常见的开发架构模式有MVC、MVP和MVVM；

手段：通过软件分层、接口隔离等；

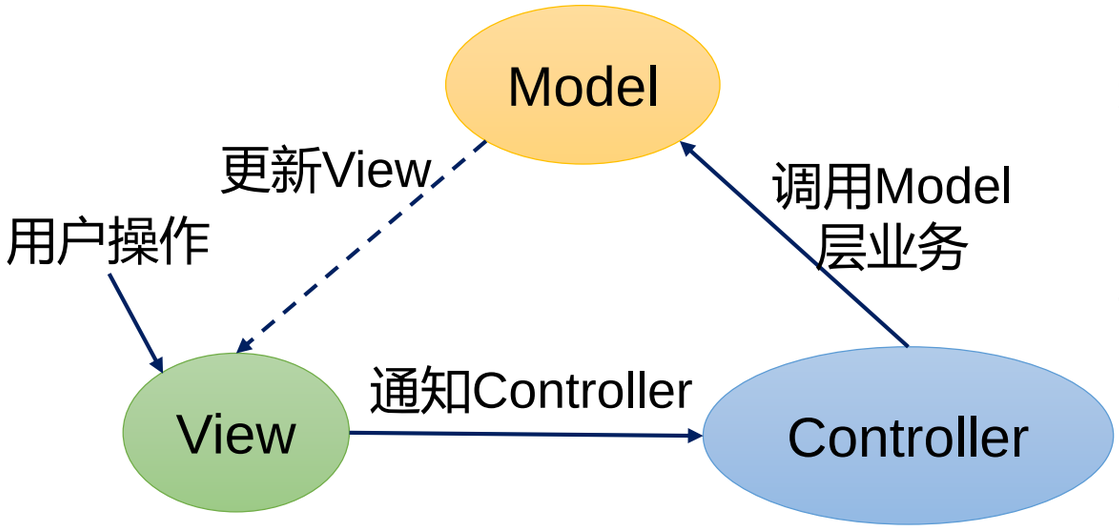
目的：降低耦合，减少迭代过程的改动，增强稳定性，减少项目维护成本。

# MVC模型



MVC： (模型(Model) - 视图(View) - 控制器(Controller)

一种业务逻辑、数据、界面显示分离的代码组织方法。




View：UI视图层，显示Model层的数据结果。

Model ：模型层，处理业务逻辑数据（网络请求、数据库存取、算法操作等）。

Controller：在 Android 中一般为Activity/Fragment，获取View层的数据，并向Model层发起请求。

MVC流程：

- View接收用户的操作
- View将用户的操作交给Controller
- Controller向Model发起请求
- Model数据处理后通知View更新

# MVC分析

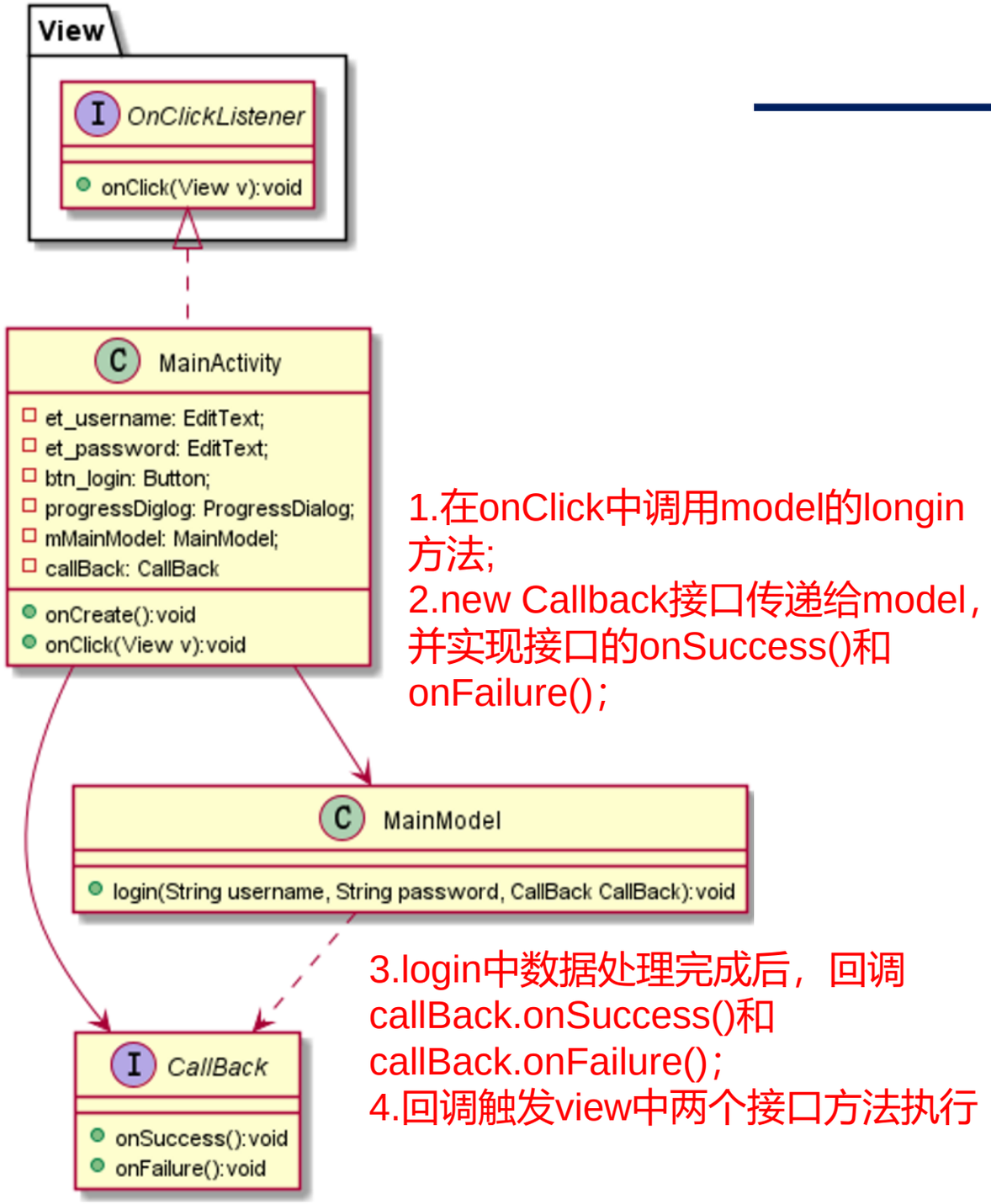
优缺点分析：

优点：

简单易上手，一定程度降低了代码间的耦合性；

缺点：

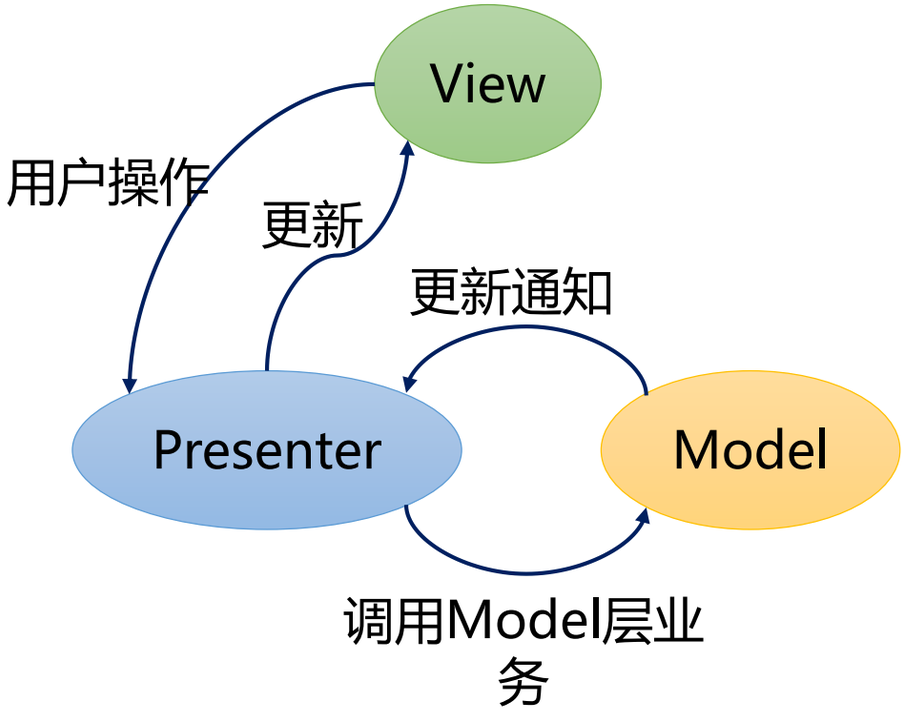
- Activity/Fragment既有View的性质，也具有Controller的性质，导致这个类的职责不断增加，以致变得庞大臃肿。
- View会与Model直接交互，所以Activity/Fragment与Model的耦合性很高。



# MVP模型



MVP：Model（模型层）-View（视图层）-Presenter（协调器/主持者）




MVP优点：

① Model层与View层完全分离；

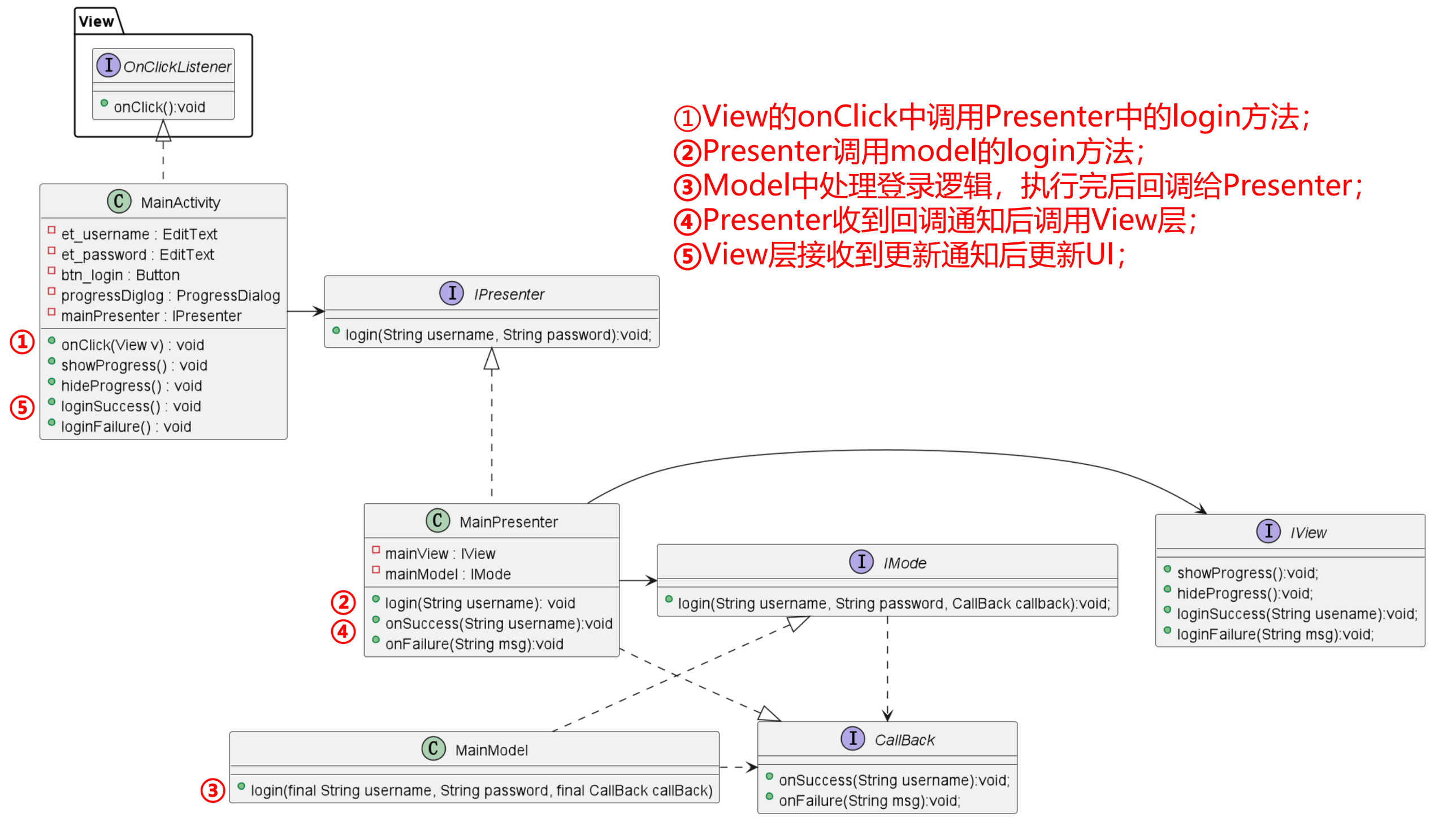
② 可以将大量的逻辑操作放到Presenter中，避免Activity的臃肿；

缺点：

① MVP模式View和Presenter双向依赖，解耦依然不够彻底。

② 需要定义各层接口进行实现，接受Presenter回调的地方必须实现对应的接口。

③ 需要实现多余的模板方法，新增很多类，并且可读性不好。


View
OnClickListener
onClick():void
MainActivity
et_username : EditText
et_password : EditText
btn_login : Button
progressDiglog : ProgressDialog
mainPresenter : IPresenter
onClick(View v) : void
showProgress() : void
hideProgress() : void
loginSuccess() : void
loginFailure() : void
IPresenter
login(String username, String password):void;
MainPresenter
mainView : IView
mainModel : IMode
login(String username): void
onSuccess(String username):void
onFailure(String msg):void
IMode
login(String username, String password, CallBack callback):void;
IView
showProgress():void;
hideProgress():void;
loginSuccess(String usename):void;
loginFailure(String msg):void;
MainModel
login(final String username, String password, final CallBack callBack)
CallBack
onSuccess(String username):void;
onFailure(String msg):void;
①View的onClick中调用Presenter中的login方法;
②Presenter调用model的login方法;
③Model中处理登录逻辑，执行完后回调给Presenter;
④Presenter收到回调通知后调用View层;
⑤View层接收到更新通知后更新UI;

# MVC、MVP实例



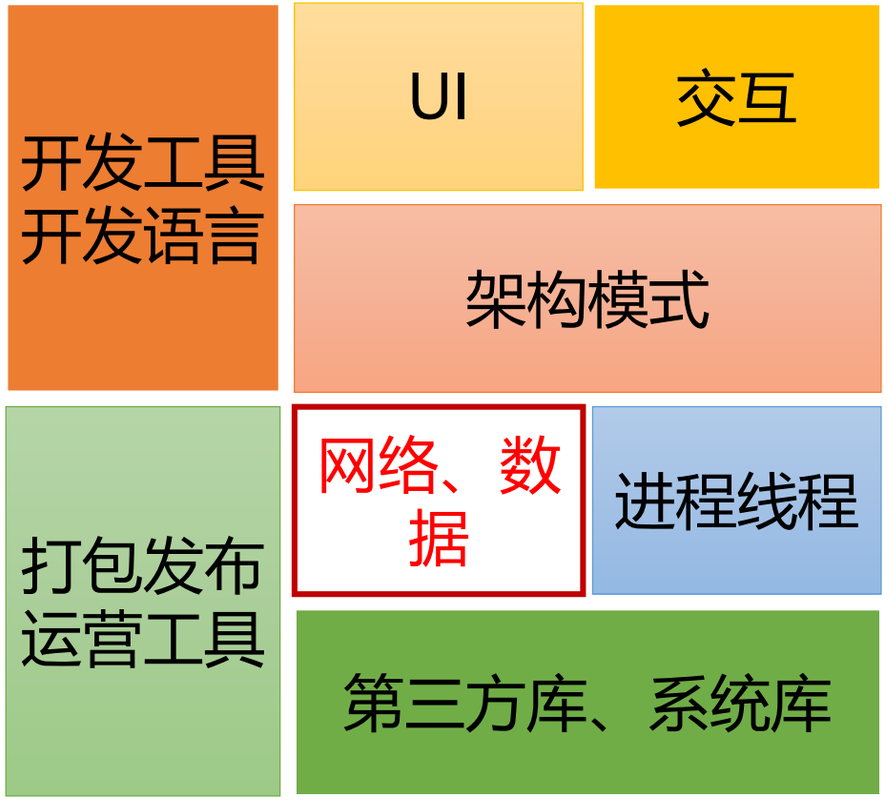
参考附件代码：MVCDemo.zip、MVPDemo.zip

Android开发框架模式(MVC、MVP、MVVM)实例解析学习：
https://www.cnblogs.com/aademeng/articles/6773518.html
https://blog.csdn.net/u012440207/article/details/82493518

关于回调函数的理解：
https://blog.csdn.net/xsf50717/article/details/50520462

# 5 网络&数据






网络业务场景：

- 账户登录时用户信息服务端校验
- 游戏得分云同步、双人对战得分时实时显示
- 商品道具列表

数据存储业务场景：

① 游戏模式设置：本次选择的游戏模式在下次启动时依然有效；

② 单机版登录账户信息本地存储；

③ 单机版得分排行榜记录；

④ 应用的配置信息

……

# HTTP协议



## Android与互联网交互的三种方式

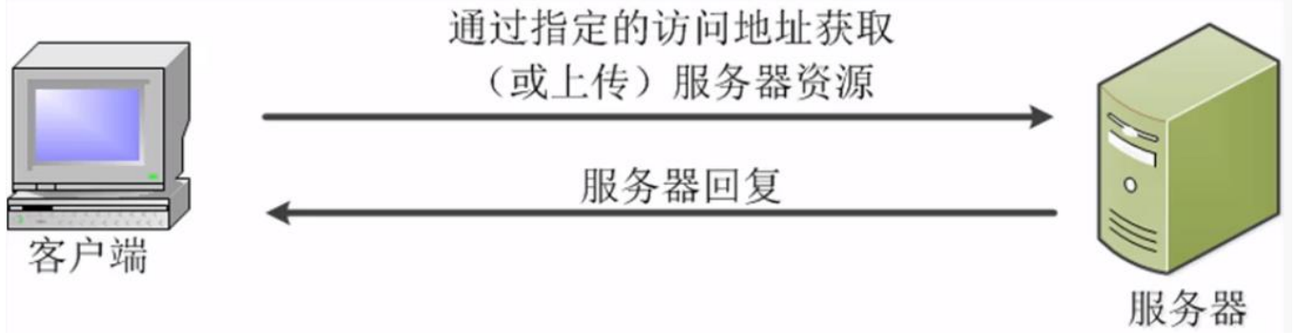
数据上传 → 使用Http Post方法上传图片、文本，格式化数据，音视频文件等；使用Socket上传大文件等

数据下载 → 使用Http Get方法从网络下载图片、代码文本、格式化数据、音频视频文件等

数据浏览 → 通过WebView浏览网络，通过JS调用Android手机里的东西

## 使用HTTP协议访问Web

Hypertext Transfer Protocol（超文本传输协议），是TCP/IP协议的一个应用层协议，用于定义Web浏览器与Web服务器之间交换数据的过程。



# HttpURLConnection



## HttpURLConnection的基本用法

步骤1
- 获得HttpURLConnection类的实例

步骤2
- 设置HTTP请求参数

步骤3
- 调用connect()连接远程资源

步骤4
- 利用getInputStream()访问资源（GET请求）
- 利用getOutputStream()传输资源（POST请求）

步骤5
- 关闭HttpURLConnection连接

# HttpURLConnection



## HttpURLConnection的基本用法

步骤1：获得HttpURLConnection类的实例

由于HttpURLConnection类是一个抽象类，不能直接实例化对象，因此需要使用URL的openConnection()方法创建具体的实例。

```
//1. 使用new关键字创建一个URL对象，并传入目标的网络地址
URL  url = new URL(“https://www.baidu.com”);

//2.调用openConnection()方法，创建HttpURLConnection类的实例
HttpURLConnection  connection = (HttpURLConnection)url.openConection();
```

# HttpURLConnection



## HttpURLConnection的基本用法

步骤2：设置HTTP请求参数

| 函数 | 说明 |
| --- | --- |
| **setRequestMethod()** | 设置请求参数，主要有两种方式：GET请求、POST请求 |
| **setConnectTimeOut()** | 设置连接超时时间 |
| **setReadTimeOut()** | 设置读取超时时间 |
| **setRequestProperty()** | 设置请求头参数，主要是添加HTTP请求HEAD中的一些参数 |
| **setDoOutput()** | 设置是否向HttpURLConnection输出，对于POST请求，参数要放在http正文中，因此需要设为true，默认情况下为false |
| **setDoInput()** | 设置是否从HttpURLConnection读入，默认情况下为true |

# HttpURLConnection



## HttpURLConnection的基本用法

步骤3：调用connect()连接远程资源

```
connection.connect();
```

服务器响应函数

| 函数 | 说明 |
| --- | --- |
| getResponseCode() | 获取服务器的响应代码 |
| getResponseMessage() | 获取服务器的响应消息 |

注意

往往在读取数据前，需要查询服务器的响应结果。

```
int responseCode = connection.getResponseCode();
if(responseCode == HttpURLConnection.HTTP_OK)
    {          读取数据的操作          }
```

# HttpURLConnection



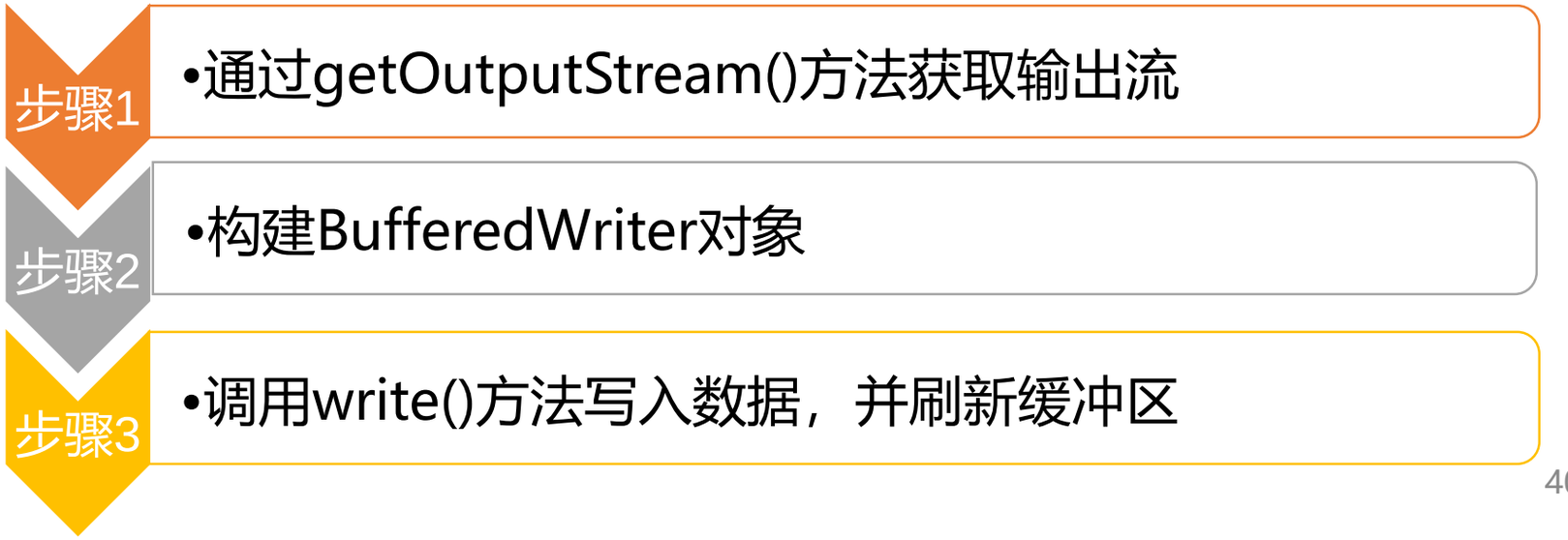
## HttpURLConnection的基本用法

步骤4：利用getOutputStream()传输POST消息（POST请求）

使用getOutputStream()方法用来传输POST消息，该方法得到的是一个输出流，该输出流中保存的是发送给服务器端的数据。


```
connection.setDoOutput(true);        //允许写出
```

使用BufferedWriter缓冲流往输出流中写入数据



## HttpURLConnection的基本用法

步骤5：关闭HttpURLConnection连接

所有的操作全部完成后，就可以调用disconnect()方法将这个HTTP连接关闭掉。

```
if(connection != null)
    connection.disconnect();
```


（1）声明网络权限：

```
<uses-permission android:name="android.permission.INTERNET"/>
<android:usesCleartextTraffic= “true “/> (API级别28或更高级别时需要)
```

（2）网络请求，需要单独开辟一个子线程，然后等到数据返回成功后回到主线程进行UI操作。

# HttpURLConnection案例



客户端代码：

```
public void appachNetworkTest(){
    try {
        Log.i( tag: "ChrisDuanXXX", msg: "1111111");
        URL url = new URL( spec: "http://10.0.2.2:8080/app/login?id=amey&pwd=123");
        // URL url = new URL("http://10.250.205.90/:8080/app/login?id=amey&pwd=amey567");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.setConnectTimeout(2 * 1000);
        conn.setReadTimeout(2 * 1000);
        if (conn.getResponseCode() != 200) {
            Log.i(TAG,  msg: "connect error responseCode:" + conn.getResponseCode());
        } else {
            InputStream in = conn.getInputStream();
            ByteArrayOutputStream outStream = new ByteArrayOutputStream();

            byte[] buffer = new byte[1024];
            int len = 0;

            while((len = in.read(buffer)) != -1){
                outStream.write(buffer, off: 0,len);
            }
            in.close();
```

```
package com.example.demo1;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class HelloController {
    @GetMapping("/app/login")
    public String login(String id,String pwd){
        if (id==null || id.equals("")){
            return "please input valid id";
        }
        if (pwd==null || pwd.equals("")){
            return "please input valid pwd";
        }
        return "id:" + id + " pwd:" + pwd;
    }
}
```

服务端程序

```
Emulator Pixel_XL_API_30 Andro   com.hit.networkappach (31348)   Verbose   MainActivity
2022-04-13 15:22:21.776 31348-31373/com.hit.networkappach I/MainActivity: get respone string: id:amey pwd:123
```

参考附件代码：NetworkAppach.zip

# Android三大存储



1 SharedPreferences 存储

2 文件 存储

3 SQLite 数据库存储




| _id | title | content | imgurl | curtime |
|---|---|---|---|---|
| 1 | i课程 | 用户名：hudie0011@163.com 密码：198199Sun. | /storage/emulated/0/img_20200219094038.jpg | 2020/2/19 |
| 2 | 计算机教育杂志社 | 账号：hudie0011@163.com 密码：198199 | /storage/emulated/0/img_20200219094220.jpg | 2020/2/19 |
| 3 | 家里Wifi | 账号：21-501 密码：wjq198218 | /storage/emulated/0/img_20200219094336.jpg | 2020/2/19 |
| 4 | 慕课网 | 网址：www.imooc.com 用户名：15006258956 密码：198199 | /storage/emulated/0/img_20200219094453.jpg | 2020/2/19 |
| 5 | 实验楼205路由器 | 用户名：TP-LINK 205 密码：szitu1234. | /storage/emulated/0/img_20200219094657.jpg | 2020/2/19 |
| 6 | 智慧苏信微信公众号 | 用户名：szxxxy@szitu.cn 密码：szituxueyuan | /storage/emulated/0/img_20200219094848.jpg | 2020/2/19 |
| 7 | 2月23日最跌情况 | 软件184： 刘权菠请假（回家） 孙鹏请假（脚受伤） 其余全齐。 软件185: 宋维荣迟至 | /storage/emulated/0/img_20200222113237.jpg | 2020/2/22 |
| 8 | 上海开发者大会 | 会议内容： 1 利用TF自动判断 2包含多种方言的智能音响 3自己组装一个小鸭子跳舞 4 | /storage/emulated/0/img_20200222113718.jpg | 2020/2/22 |
| 9 | 开学时间 | 线上上课时间 2020.2.24日 | /storage/emulated/0/img_20200222113912.jpg | 2020/2/22 |
| 11 | 阿斯顿 | 去旅有趣吧 | /storage/emulated/0/img_20200219092252.jpg | 2020/2/22 |
| 12 | 发达法 | 当中的幸存者上传这些 | /storage/emulated/0/img_20200222085034.jpg | 2020/2/22 |
| 13 | 大赛的萨芬大赛 | 手动阀福星的发送 | /storage/emulated/0/img_20200222085217.jpg | 2020/2/22 |
| 14 | 是短发是 | 阿萨德发 | /storage/emulated/0/img_20200219094220.jpg | 2020/2/22 |

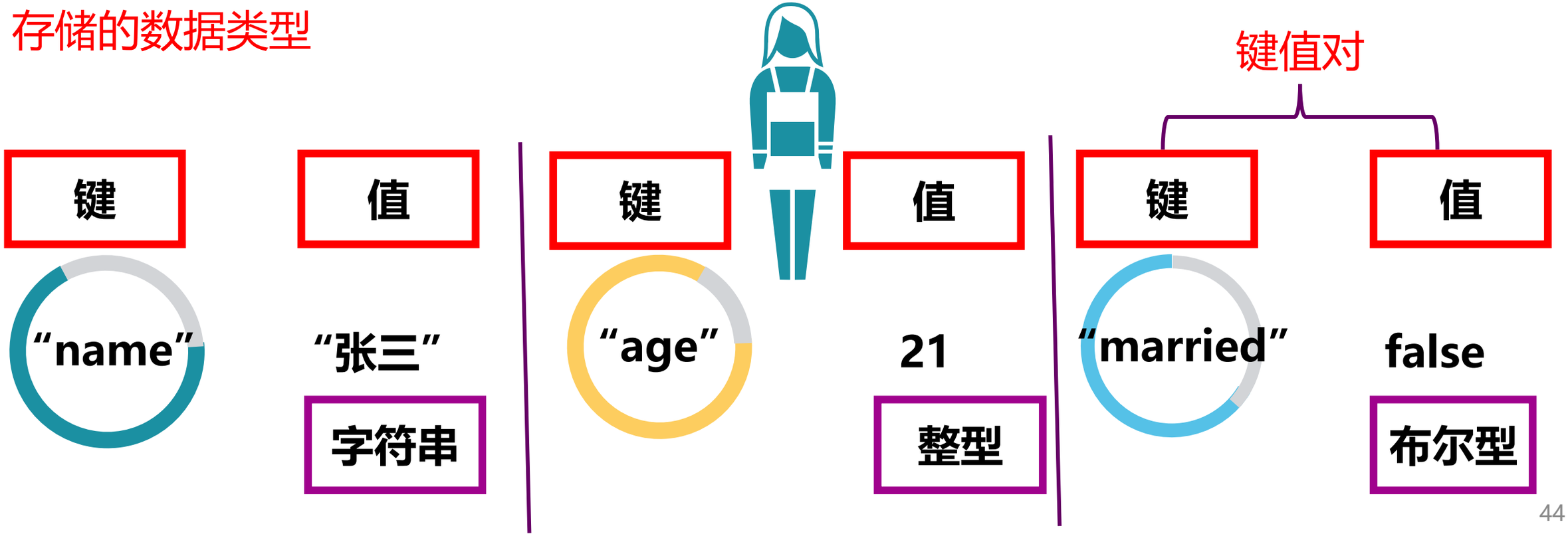
# SharedPreferences存储



## 一、简介

SharedPreferences是使用键值的方式来存储数据的。也就是说，当保存一条数据的时候，需要给这条数据提供一个对应的键，这样在读取数据的时候就可以通过这个键把相应的值取出来。

存储的数据类型



# SharedPreferences存储



## 二、使用方法

由于SharedPreferences是一个接口，而且在这个接口里没有提供写入数据和读取数据的能力。但其内部有一个Editor内部接口，Editor接口有一系列方法来操作SharedPreference的用法：

文件名

访问权限
覆盖模式

| 方法 | 作用 | 示例 |
| --- | --- | --- |
| edit( ) | 获得SharedPreferences.Edit对象 | getSharedPreferences("**myfile**",**0**).edit( ) |
| putString( )<br>putInt( )<br>putBoolean( ) | 向对象中添加数据 | editor.putString( “name”, “张三");<br>editor.putInt( “age”, 21);<br>editor.putBoolean("married",true); |
| commit( ) | 提交数据，完成数据存储操作 | editor.commit( ); |
| getString( )<br>getInt( )<br>getBoolean( ) | 从文件中读取数据 | .getString ("name", "");<br>.getInt ( “age", **0**);<br>.getBoolean ( “married", **false**); |

**默认值**
找不到值，就取默认值

# SharedPreferences案例演示




写入SP的数据：

```xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="st" value="false" />
    <string name="name">hitsz</string>
    <string name="pwd">hitsz123456</string>
</map>
```

参考附件代码：ZSShardPreferenct.zip

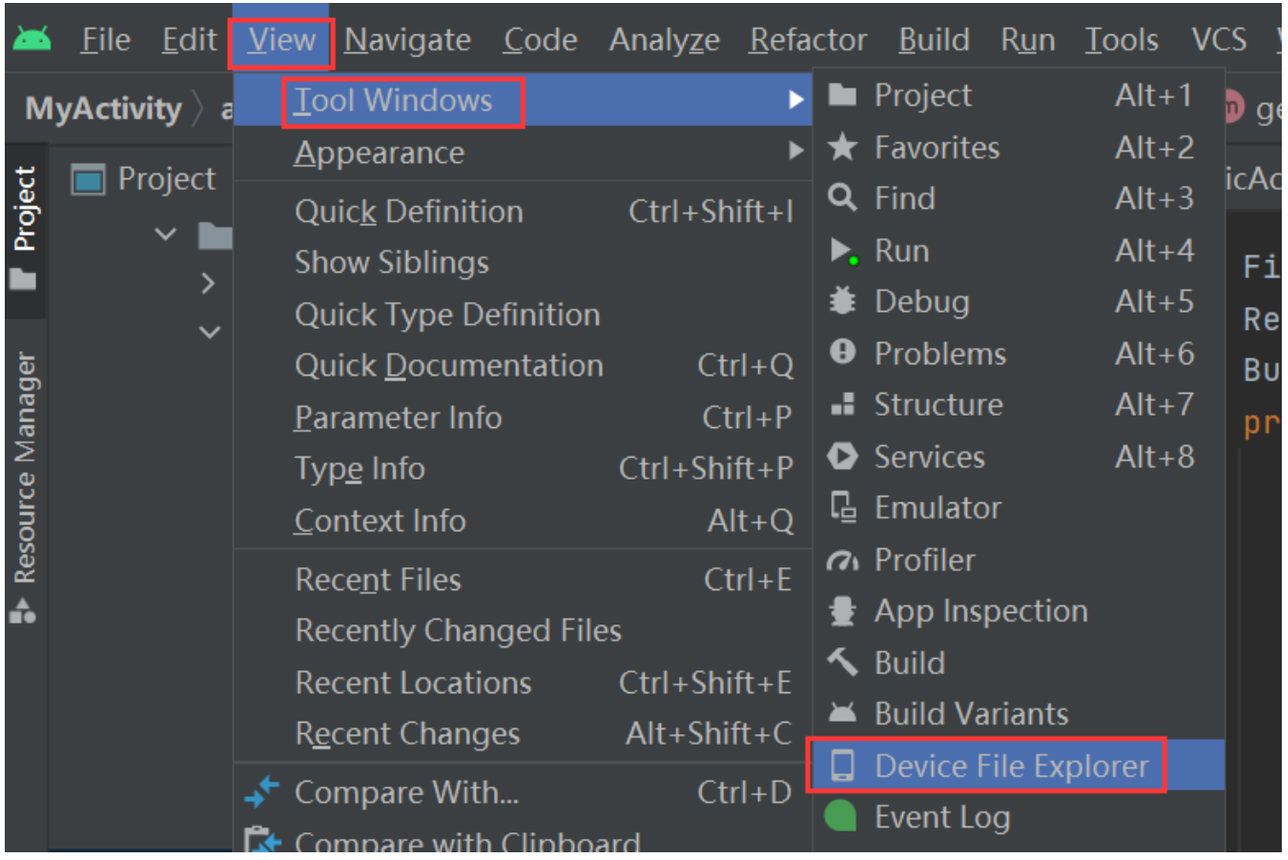
# 存储数据文件查看

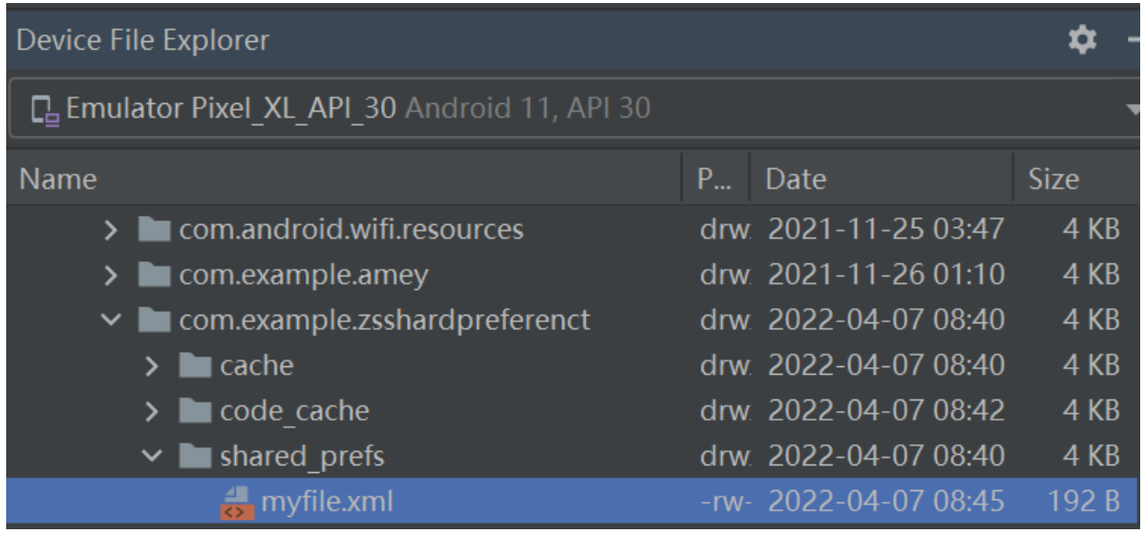


查看写入数据的文件信息：

① 点击Android Studio开发工具菜单栏View – toolWindows– Device File Explorer（如图一）；

② 打开Device File Explorer窗口，默认存储路径： /data/data/<PackageName>/files，找到存储的文件名，如myfile.txt;

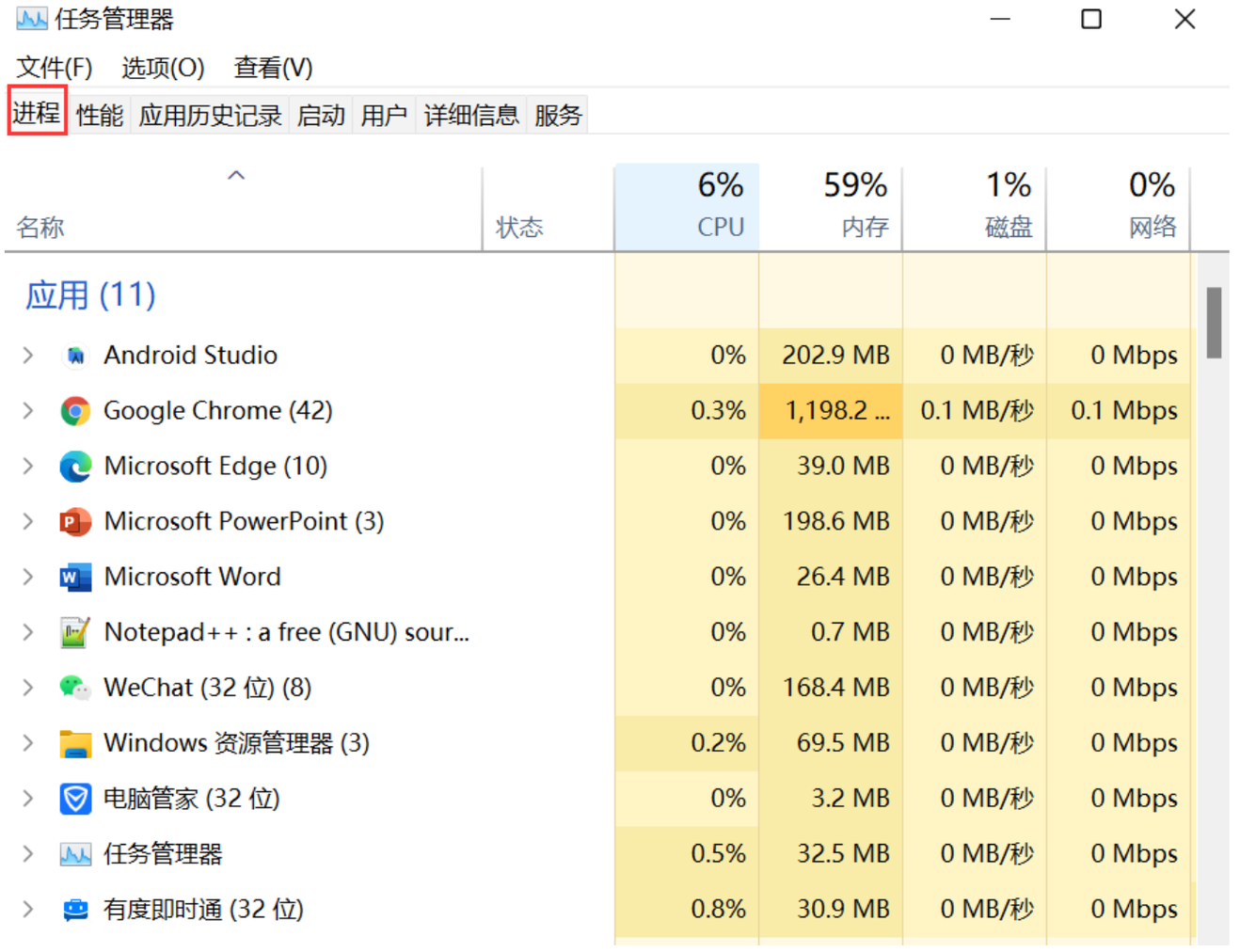


# 6 进程&线程




开发工具
开发语言
UI
交互
架构模式
打包发布
运营工具
网络、数据
进程线程
第三方库、系统库


任务管理器
文件(F) 选项(O) 查看(V)
进程 性能 应用历史记录 启动 用户 详细信息 服务
名称 状态 6% CPU 59% 内存 1% 磁盘 0% 网络
应用 (11)
Android Studio 0% 202.9 MB 0 MB/秒 0 Mbps
Google Chrome (42) 0.3% 1,198.2 ... 0.1 MB/秒 0.1 Mbps
Microsoft Edge (10) 0% 39.0 MB 0 MB/秒 0 Mbps
Microsoft PowerPoint (3) 0% 198.6 MB 0 MB/秒 0 Mbps
Microsoft Word 0% 26.4 MB 0 MB/秒 0 Mbps
Notepad++ : a free (GNU) sour... 0% 0.7 MB 0 MB/秒 0 Mbps
WeChat (32 位) (8) 0% 168.4 MB 0 MB/秒 0 Mbps
Windows 资源管理器 (3) 0.2% 69.5 MB 0 MB/秒 0 Mbps
电脑管家 (32 位) 0% 3.2 MB 0 MB/秒 0 Mbps
任务管理器 0.5% 32.5 MB 0 MB/秒 0 Mbps
有度即时通 (32 位) 0.8% 30.9 MB 0 MB/秒 0 Mbps

## Android应用进程

```
generic_x86:/ $ ps -ef | grep com
root             36     2 0 05:38:36 ?     00:00:00 [kcompactd0]
root             99     2 0 05:38:36 ?     00:00:00 [krfcommd]
system          307     1 2 05:38:55 ?     00:01:45 android.hardware.graphics.composer@
bluetooth       682   289 0 05:39:07 ?     00:00:11 com.android.bluetooth
u0_a112         706   289 0 05:39:07 ?     00:00:32 com.android.systemui
network_stack   842   289 0 05:39:08 ?     00:00:03 com.android.networkstack.process
secure_element  880   289 0 05:39:08 ?     00:00:00 com.android.se
radio           914   289 0 05:39:08 ?     00:00:26 com.android.phone
u0_a107        1060   289 0 05:39:09 ?     00:00:12 com.android.launcher3
u0_a87         1176   289 0 05:39:10 ?     00:00:17 com.google.android.gms.persistent
u0_a103        1265   289 0 05:39:11 ?     00:00:00 com.android.inputmethod.latin
u0_a117        1341   289 0 05:39:12 ?     00:00:02 com.android.providers.media.module
u0_a87         1369   289 0 05:39:12 ?     00:00:10 com.google.android.gms
radio          1399   289 0 05:39:12 ?     00:00:00 com.android.ims.rcsservice
system         1428   289 0 05:39:12 ?     00:00:00 com.android.emulator.multidisplay
u0_a87         1525   289 0 05:39:12 ?     00:00:01 com.google.process.gservices
system         6575   289 0 07:12:20 ?     00:00:00 com.android.keychain
u0_a87         6612   289 0 07:12:20 ?     00:00:00 com.google.android.gms.ui
u0_a114        6669   289 0 07:12:21 ?     00:00:00 com.android.permissioncontroller
u0_a123        6860     1 0 07:14:45 ?     00:00:00 install_server-6a9a542f com.hitsz
u0_a123        6879   289 0 07:14:46 ?     00:00:04 com.hitsz
u0_a87         7342   289 0 07:27:08 ?     00:00:00 com.google.android.gms.unstable
u0_a124        7448   289 30 07:31:29 ?    00:00:06 com.tencent.qqlive:services
u0_a124        7502   289 12 07:31:32 ?    00:00:02 com.tencent.qqlive:pmservice
u0_a124        7553   289 11 07:31:32 ?    00:00:01 com.tencent.qqlive:xg_vip_service
u0_a124        7706   289 22 07:31:33 ?    00:00:03 com.tencent.qqlive:cache
```

- 进程的启动：

启动App，Android会启动一个Linux进程和一个主线程。

- 进程的退出：

① 被动结束进程：根据进程中运行的组件类别以及组件的状态来判断该进程的优先级，Android会首先停止那些优先级低的进程。

② 主动结束进程：主动结束程序运行。

- 进程优先级：

前台进程>可见进程》服务进程》后台进程》空进程

# 线程

## Android线程

主线程：进程拥有的默认线程，即UI线程；

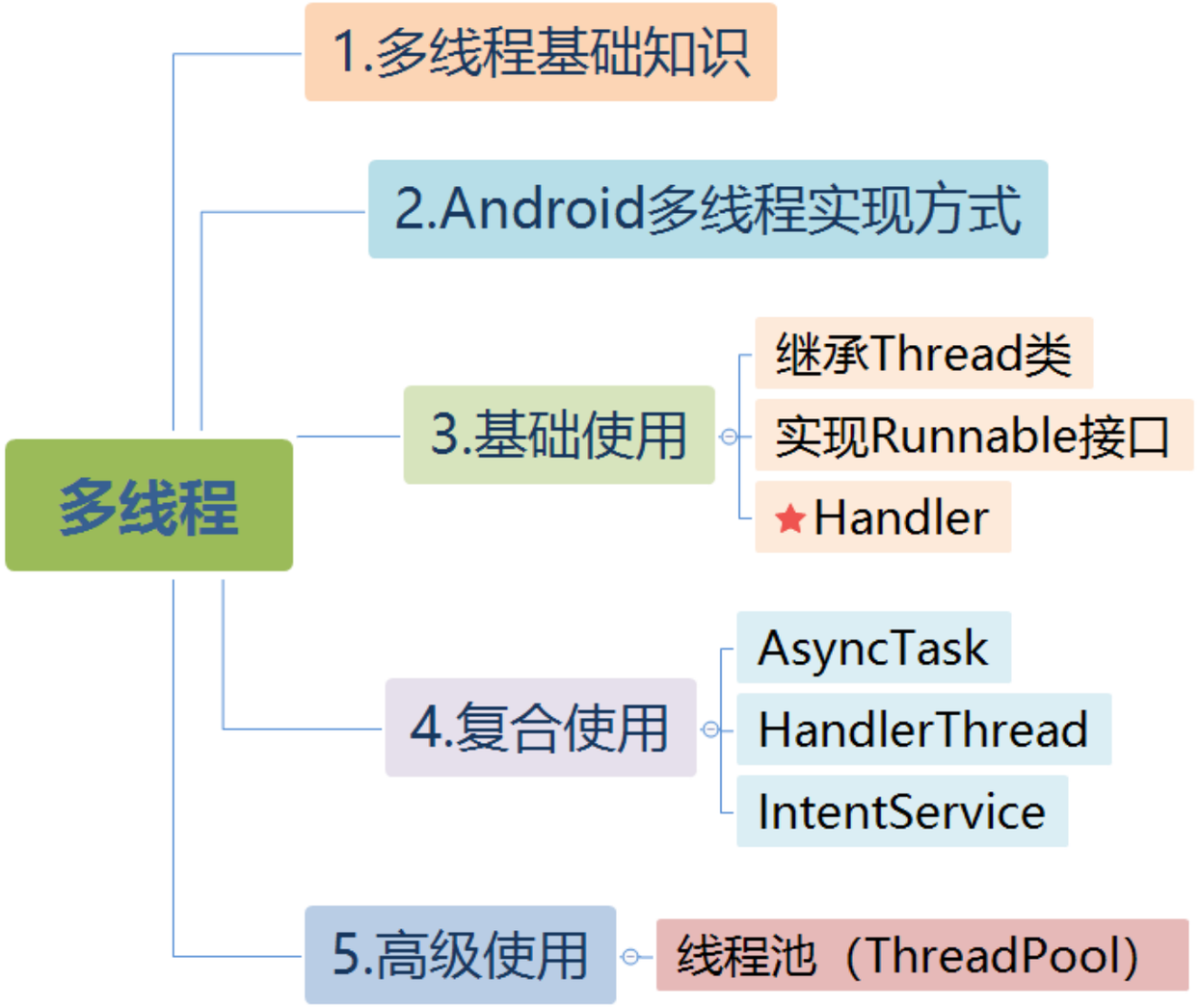
子线程：进程中手动创建的线程，用于执行耗时任务。

## 多线程业务场景

网络请求：访问网络获取商品列表，用户信息校验；

数据下载：从web下载音乐视频文件等。

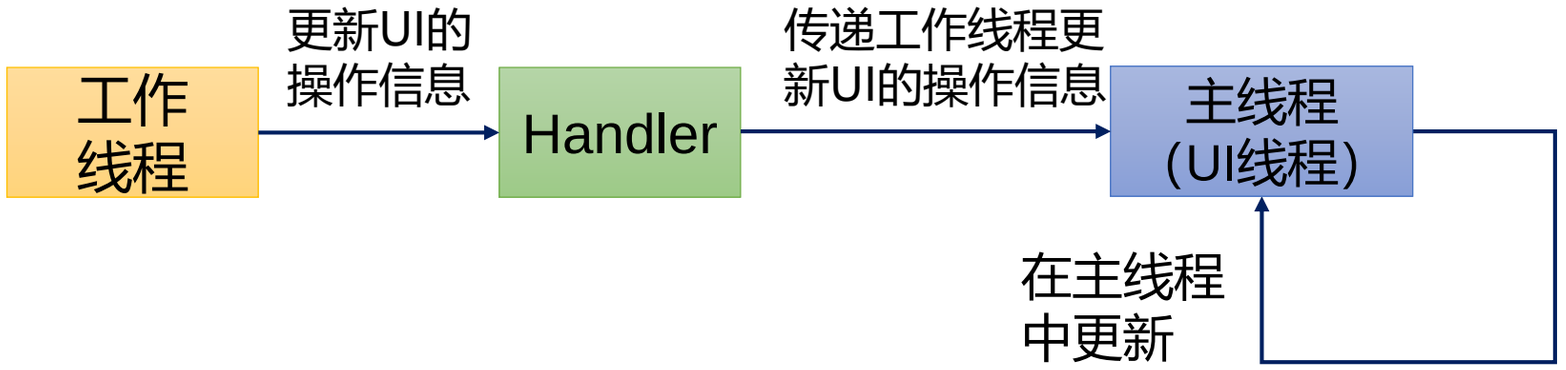
数据库读写等；



# 线程



Android Handler使用：




- 定义

一套Android消息传递机制。

- 作用

在多线程的应用中将工作线程中需更新UI的操作信息传递到UI主线程，实现主线程对UI的更新处理，最终实现异步消息的处理。

- 解决问题

➢ Android的UI访问是没有加锁的，所以多个线程访问UI是不安全的。

➢ 规定只能在UI线程中访问UI，故而主线程的响应速度不应该受到影响；

➢ 所以所有耗时操作都应该放置到子线程中进行。

UI线程阻塞超过5s会发生ANR（Application Not Response）错误。


参考文档：
https://www.jianshu.com/p/e172a2d58905
https://www.jianshu.com/p/9fe944ee02f7

# Handler使用案例



参考附件代码：HandlerDemo.zip

```java
// 采用继承Thread类实现多线程演示
new Thread() {
    @Override
    public void run() {
        try {
            Thread.sleep( millis: 4000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // 步骤3：创建所需的消息对象
        Message msg = Message.obtain();
        msg.what = 1; // 消息标识
        msg.obj = "A"; // 消息内存存放

        // 步骤4：在工作线程中 通过Handler发送消息到消息队列中
        mHandler.sendMessage(msg);
    }
}.start();
```

```java
// 步骤1：（自定义）新创建Handler子类(继承Handler类) & 复写handleMessage()
class Mhandler extends Handler {

    // 通过复写handlerMessage() 从而确定更新UI的操作
    @Override
    public void handleMessage(Message msg) {
        // 根据不同线程发送过来的消息，执行不同的UI操作
        // 根据 Message对象的what属性 标识不同的消息
        switch (msg.what) {
            case 1:
                mTextView.setText("执行了线程1的UI操作");
                break;
            case 2:
                mTextView.setText("执行了线程2的UI操作");
                break;
        }
    }
}
```

# Android Service



## 什么是Service

Service是Android中实现程序后台运行的解决方案，适用于执行不需要和用户交互而且长期运行的任务，运行在主线程。

## Service的适用场景应该具备条件

不依赖于用户可视的UI界面；

具有较长时间的运行特性。

## 飞机大战游戏Service使用场景

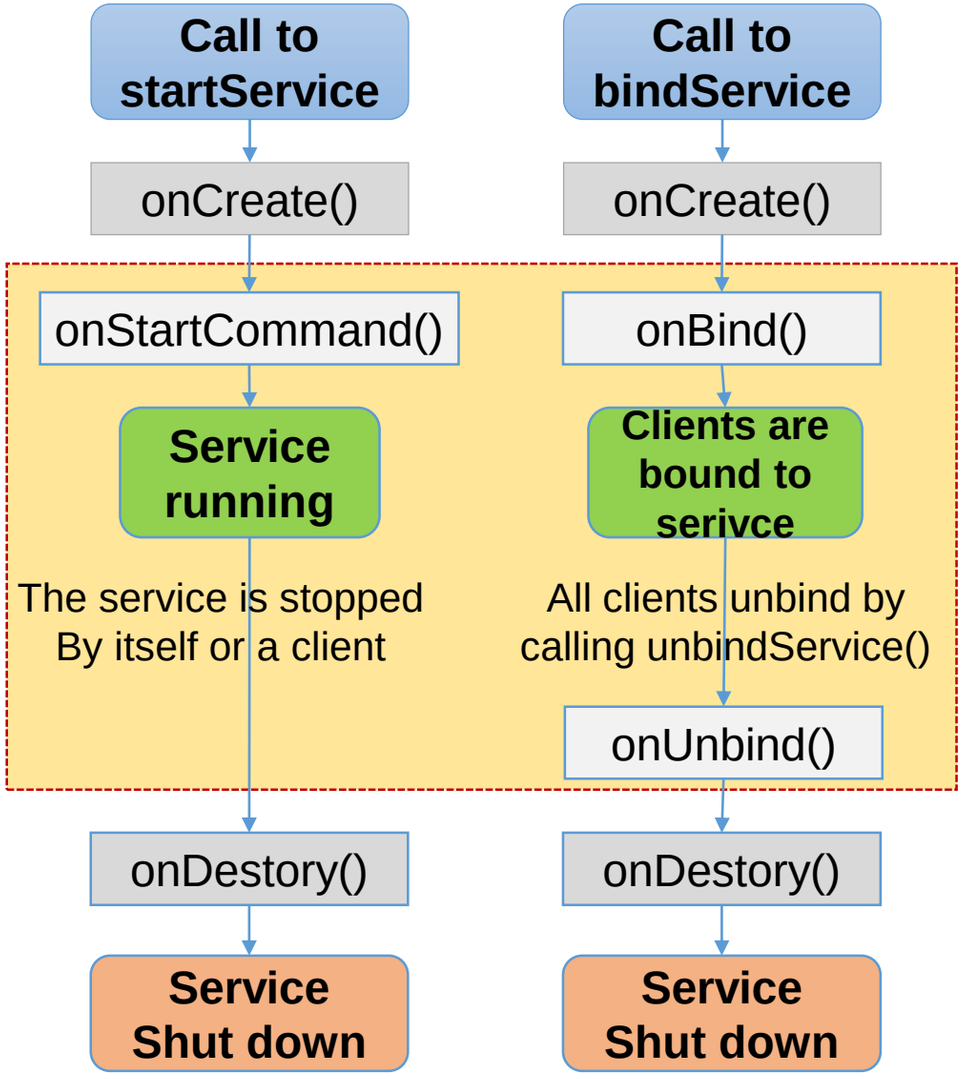
背景音乐播放

| 不同 | Thread | Service |
| --- | --- | --- |
| 概念不同 | 程序执行的最小单元，分配CPU的基本单位 | Android的一种机制，没有UI的后台组件 |
| 执行任务 | 可执行耗时任务 | 运行在主线程，不可执行耗时任务；如果需要，在内部启动Thread |
| 使用场景 | 执行可能会影响UI主线程的耗时任务 | 不需要UI的长时间后台任务，如播放音乐、下载 |

# Android Service基本用法



## Service的生命周期




Service使用方法：

① 新建TestService继承Service,重写父类的onCreate(),onStartCommand(),onDestroy()的方法。

② 在AndroidManifest中配置Service（Android Studio新版本新建Service类时会自动添加）；

③ 执行Service

④ context.startService()开启服务流程：

context.startService()-->onCreate()-->onStartCommand()--->context.stopService()-->onDestroy()-->Service Stop

# Android Service案例



参考附件代码：MusicService.zip

参考文档：https://www.jianshu.com/p/cd9df908b361

# 7 第三方库




- 系统库

Android系统本身也提供了丰富的系统库，比如访问Camera、音乐播放器、图片处理、视频播放组件库等;

- 第三方库

Logger日志库，Lottie动画库，

Universal Image Loader图片加载库……;

- 第三方SDK

米推送：push信息推送

areSDK Mob：平台的社会化分享和登录

盟统计：运营数据统计

云服务

ash堆栈信息上报，分析统计，设备、版本聚集等。

# 8 打包发布




- 构建调试版本包

  作用：

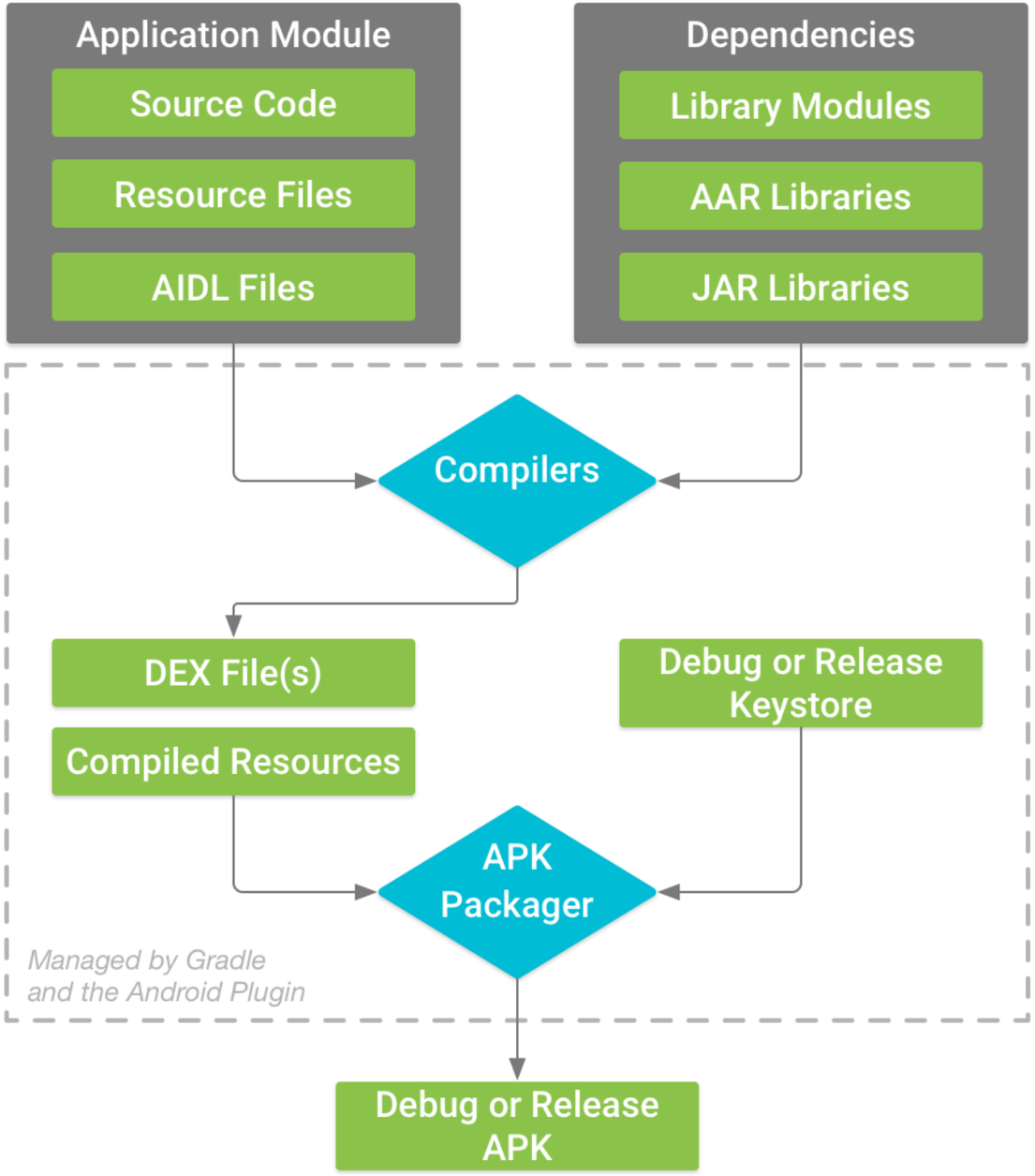
  本地调试，打印调试日志

- 构建发布版本包

  发布到应用市场

# 打包发布操作流程






典型Android 应用模块的构建流程

① 编译器将源代码转换成 DEX 文件；

② 打包器将 DEX 文件和编译后的资源组合成 APK ；

③ 必须先为 APK签名后才能将应用安装到Android设备，打包器使用调试或发布密钥库为 APK签名；

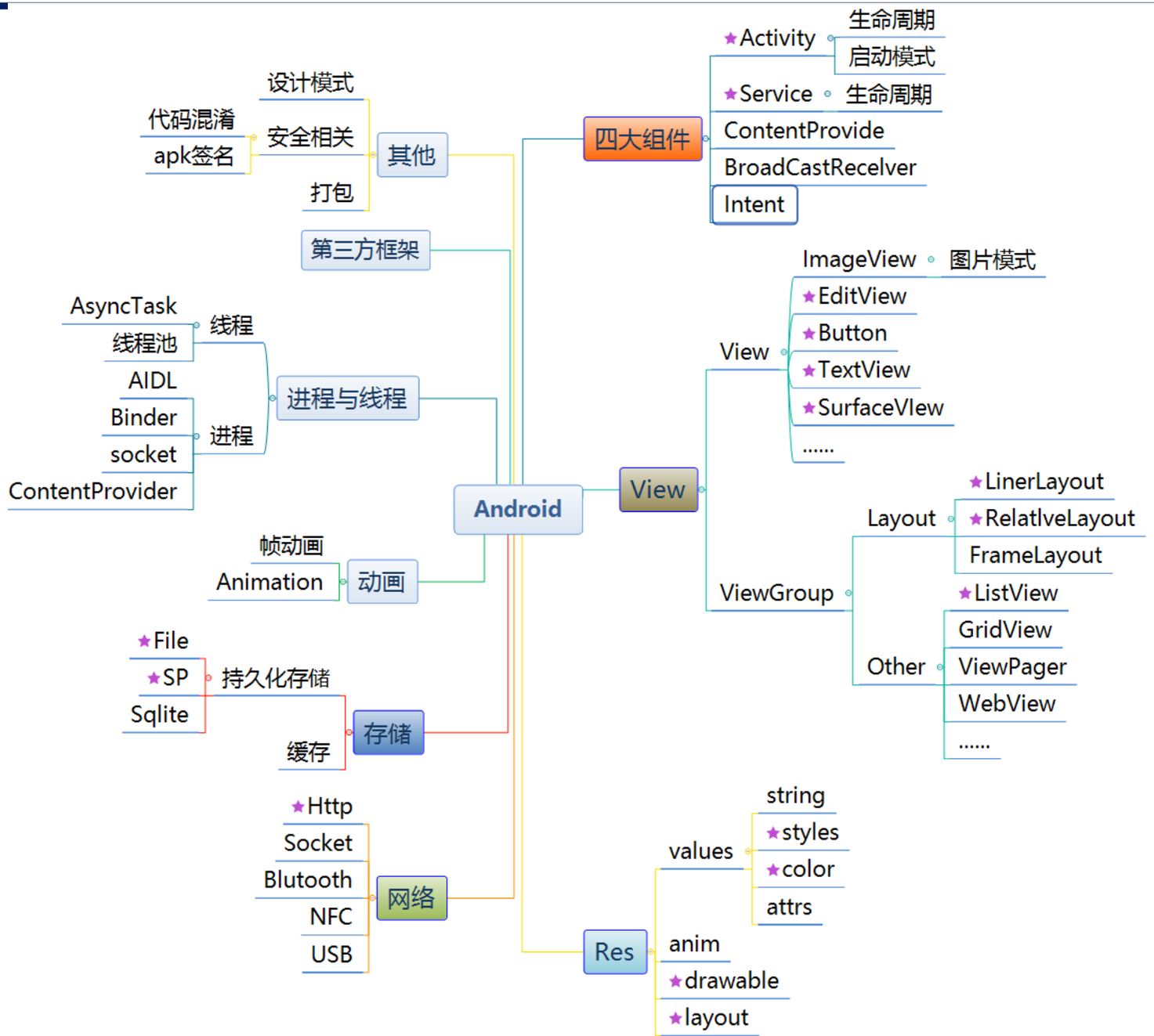
④ 在生成最终 APK 之前，打包器会使用 zipalign 工具对应用进行优化，以减少其在设备上运行时所占用的内存。

密钥说明：

- 如果构建的是调试版应用，会自动使用调试密钥库为应用签名。
- 如果构建的是发布版应用，打包器会使用发布密钥库（需要进行配置）为应用签名,请参阅在 Android Studio 中为应用签名

# Android开发实践-初阶





# 总结



1. 认识Android系统平台架构
2. 了解了Android开发基本流程


3. 学习Android开发基础知识（UI、交互、架构模式、网络访问、数据存储、进程线程、第三方库、打包发布）
4. 开始Android开发实践

# 谢谢！