1. Activity 和 Fragment 的生命周期

Activity.

启动 Activity：onCreate() -> onStart() -> onResume()

退居后台：onpause() -> onStop()

重返前台：onStart() -> onResume()

后台运行且内存不足： onPause() -> onStop() -> Kill Process

重启该 Activity：onCreate() -> onStart() -> onResume()

启动新 Activity,并把旧 Activity finish()掉 :旧 Activity 的 onPause() -> 新 Activity 的 onCreate() -> onStart() -> onResume() -> 旧 Activity 的 onStop() -> onDestory(), 因此不要在 onPause() 做耗时的逻辑操作。避免 ANR 异常。

锁屏：onPause() -> onStop()

解屏：onStart() -> onResume()

Fragment.

onAttach() -> onCreate() -> onCreateView() -> onActivityCreate() -> onStart() ->onResume() -> onPause() -> onStop() -> onDestoryView() –> onDestory() ->onDetach()

2. Activity 的四种启动模式及其特点

1. Standard：标准模式，每次创建 Activity 都会压进任务栈栈顶。不管该 Activity 是否存在于任务栈内。 完整生命周期流程
2. SingleTop：栈顶复用模式，每次创建 Activity 时都会检查该 Activity 和栈顶 Activity 是否相同，如果是同一个 Activity 实例则复用该栈顶 Activity，不在重新创建 Activity 实例和执行生命周期方法。
3. SingleTask：栈内复用模式，每次 创建 Activity 实例时都会检测任务栈内是否存在相同 Activity 实例，如果有则复用该 Activity 实例并执行 onNewIntent 方法，并且将该 Activity 实例之上的 Activity 移除出栈。没有则跟 standard。
4. SingleInstance：加强版的 SingleTask 模式，每次创建一个 Activity 实例都会重新创建一个新的任务栈来维护，实例独占任务栈。其他实例则在标准任务栈上维护。

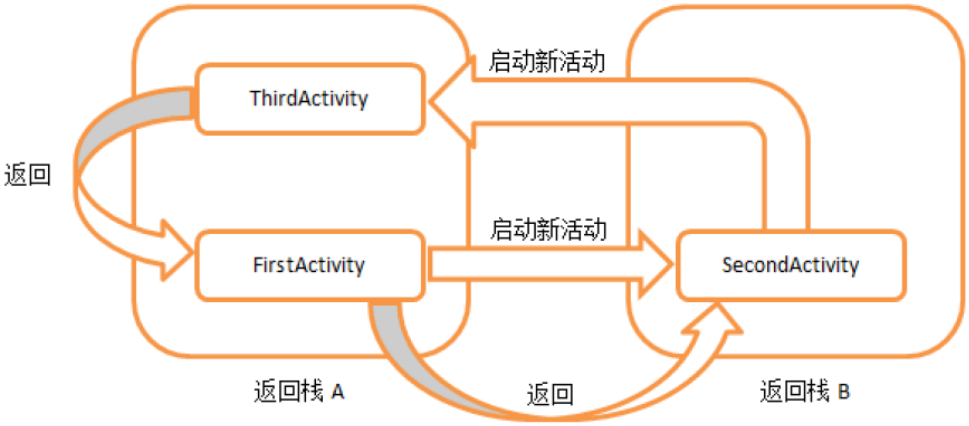


图 2.38

3. Activity 的缓存方法

A、B 两个 Activity，A 进入 B 后 A 实例可能会被内存回收（系统强行回收），导致返回 A 时不掉用 onRestart 方法而是调用 onCreate 方法。导致原有数据丢失。

** onSaveInstanceState()

这时，我们可以重写 onSaveInstanceState()方法保存临时数据、View 的状态。--当 Activity 可能被销毁的情况：

--按 Home 键回主界面
--长按 Home 键跳转 App
--按下电源键
--启动新 Activity
--横竖屏切换（未指定 configChange 的情况）

```
@Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        outState.putString("anAnt","Android");
    }
```

super.onSaveInstanceState()已帮我们实现部分 UI 状态存储。
onSaveInstantState()尽量用来保存临时信息、UI 状态。。不要用来做持久化存储。

** onRestoreInstanceState()

onSaveInstateState 和 onRestoreInstateState()不一定成对触发。
onRestoreIntanceState()的 buddle 也会传入 onCreate 方法中。我们也可以在 onCreate 方法上进行数据还原。　　还有 onRestoreInstateState()方法在 onStart()之后执行。

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putBoolean("MyBoolean", true);
        savedInstanceState.putDouble("myDouble", 1.9);
        savedInstanceState.putInt("MyInt", 1);
        savedInstanceState.putString("MyString", "Welcome back to Android");
        // etc.
        super.onSaveInstanceState(savedInstanceState);
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);

        boolean myBoolean = savedInstanceState.getBoolean("MyBoolean");
        double myDouble = savedInstanceState.getDouble("myDouble");
        int myInt = savedInstanceState.getInt("MyInt");
        String myString = savedInstanceState.getString("MyString");
}
```

4. Service 的生命周期，启动的两种方法

在上下文环境下使用

1. Context.bindService(Intent service, ServiceConnection conn, int flag)
2. Context.StartService(intent Service)

bindService -- Service 生命周期与 Context 绑定，Context 销毁，Service 就销毁。

StartService --Service 生命周期不与 Context 关联，Service 不会随 Context 销毁而销毁。

但无论哪种方式启动，都需要在 manifest 清单文件上注册/声明：

```
<service
        android:name=".packnameName.youServiceName"
        android:enabled="true" />
```

5. 怎样保证 Service 不被杀死

1. Service 的 onStartCommand 方法上返回值设置成 **START_STICKY。**
   kill 后会被重启（等待 5 秒左右），重传 Intent，保持与重启前一样
2. 提升 Service 进程的优先级。
   在 startCommand 上调用 startForeGround()使服务状态变为前台服务。在 onDestory()上调用 stopForeGround()。像手机播放器一样不管手机如何休眠，运行在后台依然不停播放。
3. 在 onDestory()上重启 Service
   在 onDestory 方法上再 startService。
4. 通过广播监听 Service 状态。
   通过 service + brocast 的方式当 service 执行 onDestory 时，发送自定义广播，Recever 接收到广播后重新启动 Service。
5. 放一个像素在前台。（手机 QQ）

6. 广播的两种注册方法，区别？

静态注册：Manifest 清单文件上对广播进行注册声明。

动态注册：在 Activity 组件上用代码进行动态注册。

区别：静态注册在 App 退出时 Receiver 仍然能够接收广播。

动态注册当 App 退出时，广播就无法被接收。

7. Intent 的使用方法，能传递什么？

Intent 是组件间进行交互的重要方式。通过 Intent 机制，创建 Activity、Service、BrocastReceiver 组件，还可以在组件间传递数据。

显式 intent：new intent 时就指定想要打开的组件。

隐士 intent：通过一系列的匹配动作，action、category 来匹配过滤出相应的组件。

隐式 intent 还可以打开其他应用的 Activity 组件，比如浏览器、拨号。。

常用方法：

```
it.putExtras("bd","xxx");
intent.getExtra("bd")
```

传递类型：

基本数据类型及其数组、

String/CharSeq 及其数组、
Parcelable\Serializable 序列化对象。

8. ContentProvider 使用方法

内容提供者是用来供给其他应用程序访问当前应用的数据的一个组件。实现不同的应用程序之间进行数据共享的功能。

使用：

在当前应用上创建 ContentProvider 内容提供者，并实现他的 增删改查和 onCreate、getType 方法，并且通过 uriMatach 添加他的 Uri 匹配路径。

在其他应用上通过 Context.getContentResolver 获取内容解析者，调用他的增删改查方法并传入匹配的 Uri 和匹配参数即可操作其他应用的数据。

9. Thread、AsycTask、IntentService 的使用场景及特点

一般开发中执行一些耗时操作或异步操作（加载网络、访问数据库）时，通常需要使用 Thread 类创建一个新线程来执行。

IntentService 是本质上是 Service，基于 Service+Handler 只是在 Service 上又做了一些异步的多线程的封装。我们可以在 onHandleIntent 上面执行耗时操作，执行完毕后自动销毁 Service。

AsyncTask 基于 Thread 池 和 Handler，帮助我们进行了异步多线程的封装。能够避免程序线程过多导致的性能开销过大。主要方法：onPreExecute()-主 doInBackground()-子 onPostExecute()-主 onProgressUpdate() –主，需要在 doInBackgrond()上调用 publishProgress()..

10. 五大布局：FrameLayout、LinearLayout、AbsoluteLayout、RelativeLayout、TableLayout（ConstraintLayout 加强可视化操作、可视化编写界面）

FrameLayout：帧布局，子控件默认叠加在左上角。我们可以通过 layout_gravity 控制子控件的相对位置。

Linearlayout：线性布局，以一行或一列为一个控件独占的布局，需要 Orientation 指定具体排列方式。

RelativeLayout：相对布局，布局内部可放置多个控件，每个控件间的相对位置可以相对排列。常用：layout_centerInparent/alignParentRight..

AbsoulteLayout：绝对布局，放置多个空间，每个控件自定义 x，y 坐标。会有屏幕适配问题。

TableLayout：表格布局，将子控件分配到行或列上，一个 TableLayout 有多个 TableRow 组成。

ConstraintLayout：约束布局，android 新加入的布局，用于强化图形界面手动绘制界面，加强可视化操作。 （强化可视化操作，通过图形界面去绘制我们的布局我们的界面元素）

11. Android 数据存储形式

SharePreference：轻量级的数据存储，本质是一个 xml 文件，通过键值对存储数据。

File 文件存储：也就是我问说文件 IO，用于存储大量数据，需要解决数据同步更新问题。

Sqlite 数据库：Andoroid 提供轻量级的数据库数据存储，支持常用的 Sql 语句。Android 提供 SqliteDatabase 这个类，封装了一些 API，让我们更方便的操作数据库。

ContentProvider：内容提供者存储数据，用于提供给其他应用来进行的数据操作。暴露 URI。

12. Sqlite 的基本操作

一般我们使用 Sqlite 数据库，都会自己写一个类 MySQLiteHelper 继承 SQLiteOpenHelper 并实现抽象方法：onCreate() --创建表 onUpgrade() –版本号提升，更新数据库.

后期通过 helper.getReadableDatabse()/getWritableDatabase() 获取数据库 db，增删改查!

```
public class StudentDatabaseHelper extends SQLiteOpenHelper {
    private Context mContext;
    private static final String CREATE_STU = "create table if not exists stu(_id integer
primary key autoincrement," +
            "name text, sex text, class text, born text, address text, phone text)";

    public StudentDatabaseHelper(Context context, String name,
                                 SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
        mContext = context;
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_STU);
        Toast.makeText(mContext, "Create STU success!~", Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
}
```

```
public boolean onCreate() {
    dbHelper = new StudentDatabaseHelper(getContext(), "stu.db", null, 1);
    return true;
}

public Cursor query(@NonNull Uri uri, @Nullable String[] projection, @Nullable String
selection, @Nullable String[] selectionArgs, @Nullable String sortOrder) {
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    Cursor cursor = db.query("stu",projection,selection,selectionArgs,null,null,sortOrder);
    return cursor;
}
```

13. Android 中的 MVC

View 视图层

Controller 控制层

Model 模型层

接收用户操作，将数据指令传递给 Controller 层。然后 Controller 完成业务逻辑后，请求 Model 层状态改变，Model 层将新数据发送到 View 层，用户界面得到反馈。

14. Include、merge、viewstub 的作用

布局优化!

-- <include>:包含布局，当我们写界面写得比较大时，我们可以将界面划分出几个独立的子布局，然后再通过 include 标签将这些布局包含进来。

-- <merge>:用来减少我们布局层级嵌套的问题，比如我们在一个界面上 include 上另一个布局，这时候 include 布局上面的顶层的 Linearlayout、RelativeLayout 就无意义了，而且他还会消耗系统资源。这时我们可以使用 Merge 标签，减少视图层级。

--<viewstub>:布局需要时使用，通过 ViewStub 标签包含的内容初始化时不会被加载，可达到节省内存的目的，后期通过代码可以动态令布局内容显示。

```
<ViewStub
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout="@layout/progressbar"
            android:id="@+id/viewStub"/>
```

代码动态显示：

```
1.  ((ViewStub) findViewById(R.id.stub_import)).setVisibility(View.VISIBLE);
2.  // or
3.  View importPanel = ((ViewStub) findViewById(R.id.stub_import)).inflate();
```

15. Json 的优势

首先 Json 是一种轻量级的数据交换格式，数据以键值对来存储。

Json 优势：

1. 数据格式简单、易于读写，没 xml 那么沉余多标签。
2. 数据格式带压缩，占用带宽小。
3. 支持多种语言，便于服务器端解析和处理。

缺点：

使用广度没 xml 那么高，兼容性没 xml 格式数据那么好。

16. 动画主要有哪几类？各有什么特点？

补间动画、帧动画、属性动画

补间动画：一般分为 4 种动画方式，alpha（透明渐变）、translate（平移）、scale（缩放）、rotate（旋转），还能组合 Set。通过指定 View 控件或布局的开始跟结束时状态和变化时间，就可以完成一系列动画效果。　　实现方式也有两种：Xml 文件 和 Java 代码

Xml：

```
<?xml version="1.0" encoding="utf-8"?>
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromXScale="0.0"
```

```
    android:fromYScale="0.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="1.0"
    android:toYScale="1.0"/>


    //MainActivity 加载
    Animation animation = AnimationUtils.loadAnimation(mContext, R.anim.alpha_anim);
    img = (ImageView) findViewById(R.id.img);
    img.startAnimation(animation);
```

Java 代码：

```
Button mButton = (Button) findViewById(R.id.Button);
       // 步骤 1:创建 需要设置动画的 视图 View
Animation rotateAnimation = new
     RotateAnimation(0,270,Animation.RELATIVE_TO_SELF,0.5f,Animation.RELATIVE_TO_SELF,0.5f);
       // 步骤 2：创建旋转动画的对象 & 设置动画效果：旋转动画对应的 Animation 子类为 RotateAnimation

rotateAnimation.setDuration(3000);
       // 固定属性的设置都是在其属性前加“set”，如 setDuration（）

mButton.startAnimation(rotateAnimation);
       // 步骤 3：播放动画
```

缺陷：

作用对象局限，只能作用 View 或布局，无法改变视图的某事属性，不如颜色。
没有真正改变 View 的属性，不会真正改变位置效果，只是视觉效果。
动画效果单一，只能是 4 种动画效果或跟他们的组合。

帧动画：

帧动画是 Android 动画中比较容易实现的一种动画，其核心是通过一张张图片形成帧从而形成连续播放的动画效果。类似于 gif 动画一样。（animation-list、AnimationDrawable）

实现：

```
    <?xml version="1.0" encoding="utf-8"?>
    <animation-list xmlns:android="http://schemas.android.com/apk/res/android">
        <item
            android:drawable="@drawable/a_0"
            android:duration="100" />
        <item
            android:drawable="@drawable/a_1"
            android:duration="100" />
        <item
            android:drawable="@drawable/a_2"
            android:duration="100" />
    </animation-list>


         //MainActivity 实现
    ImageView img= (ImageView) findViewById(R.id.animation1);
    img.setImageResource(R.drawable.frame_anim1);
```

```
AnimationDrawable animationDrawable1 = (AnimationDrawable) img.getDrawable();
animationDrawable1.start();
```

属性动画：

新 SDK 提供的一种动画类型，弥补帧动画和补间动画的不足。

原理：在一定时间间隔内，通过不断对值进行改变。不断将该值赋给对象属性，从而实现对象在该属性上的动画效果。核心类：ValueAnimation、ObjectAnimation。

17. Handler、Loop 消息队列模型，各部分作用。

讲到消息队列，就涉及到 MessageQueue 消息队列、Message 消息、Handler、Looper 轮询器。

之所以需要 Handler 消息机制，因为我们开发中一般都会遇到需要加载网络、加载数据库等等的一些耗时操作，而我们的主线程是不可以被阻塞的，也就是不能够进行耗时操作。这时候我们就需要开启子线程完成耗时操作，完成后需要更新 UI。这是候就用到了 Handler 消息机制。通过发送 Message 消息，将数据传递到主线程中让其更新 UI。

Message：消息，用来在线程间传递数据的工具。携带少量数据，在线程间进行数据交互。.arg1、.arg2（int）、.obj、.what（obj）来传递数据。

MessageQueue :消息队列（单链表维护），用来存放 handler 发送过来的消息，并等待 Looper 轮询器轮询并处理。值得注意，消息队列是线程私有的。

Handler：用于来送和处理消息。发送消息一般为：handler.sendMessage()方法。发送的消息一般都会到达 Handler 的 handleMessage（）方法中。

Looper：轮询器，是每个线程中 MessageQueue 的管家。调用 looper.loop()之后，looper 会一直检测一直轮询查看 MessageQueue 消息队列中是否存在消息，如果存在则取出消息，并传递到 handleMessage（）方法中。

18. 怎样退出终止 App

1. 容器式，在 Application 下创建一个容器（集合）来管理（存放）所有 Activity，需要退出应用时，将容器内所有的 Activity 全部 Finish 掉即可。
2. 广播式，在 BaseActivity 中注册广播，需要退出应用时发出广播，Activity 接收到后 finish()掉。
3. 通过 API 杀死进程：*android.os.Process.killProcess(android.os.Process.myPid());*
4. Receiver + SingleTask 式，SplashActivity –> HomeActivity(SingleTask 栈内唯一) +…。后期需要退出应用是只需打开 HomeActivity 并发送退出广播，应用就会退出。

19. Assest 目录和 Res 目录区别

Res 目录下的文件会被映射到 R.java 文件当中。后期我们需要用到资源文件就能够通过 R 文件访问。

Assest 目录下的文件则不会映射到 R.java 文件当中。需要用到资源文件时需要通过 AssestManager 来访问。

20. Android 如何加速启动 Activity

1. 避免在 onCreate()方法上进行大量逻辑或耗时操作。因为执行 onCreate 方法时，界面还没有显示出来，屏幕处于黑屏状态。一旦执行过多耗时操作，则黑屏时间过长，降低用户体验。
2. A 页面启动 B 页面时，不要在 onPause 方法上执行的耗时操作。因为 B 页面的生命周期必须要等到 A 页面的 onPause 方法执行后才执行。
3. 尽量减少在主线程的阻塞时间。耗时操作放在子线程上执行。
4. 优化布局文件，减少布局层次结构。布局层次结构过大消耗系统性能。
5. 提高 ListView、RecyclerView、Adapter 的效率：复用 converView、使用 ViewHolder、缓存 item 数据、数据分段分页显示。

21. Android 内存优化方法：ListView 优化、及时关闭资源、图片缓存…

1. 图片内存的优化：
   -- 在 Bitmap 对象上做处理，添加软引用或弱引用。
   -- 使用三级缓存机制。图片加载进内存时使用 LRUCache 的缓存机制来保存 Bitmap 对象。
   -- 大图片需要进行压缩处理（BitmapFactory.Options（）表示要缩小原来的几分之一）
2. 其他的要注意：
   -- 及时关闭资源对象，比如 IO 流数据库 Cursor..
   -- ListView、RecycleView 复用 converView、使用 ViewHolder、缓存每个 item 图片、数据。。

22. Android 中弱引用与软引用的应用场景。

-- 软引用：持有软引用的对象，一般 jvm 虚拟机都不会对他进行垃圾回收，除非系统内存严重不足时才对该对象进行回收。
-- 弱引用 :仅仅持有弱引用的对象，jvm 虚拟机都会在下次垃圾回收并扫描到该对象存在就会对其进行回收处理，不管当前系统内存是否足够。
-- 软引用我们一般会使用在图片缓存上面，Bitmap 对象进行软引用处理，系统内存不足是就能够回收图片这部分占用的内存，从而避免内存溢出的发生。
-- 弱引用我们一般会使用在内部类对外部类的引用持有上。例如我们在 Activity 内部创建了 Handler 的内部类，内部类当中需要持有外部类引用，我们就可以对外部类的 Activity 进行弱引用处理，避免后期 Activity 销毁后无法进行内存回收的处境。

23. Bitmap 的 4 种属性，没中属性队形的大小

-- 宽、高、单位像素所占用的字节数。
-- Bitmap 对象用 BitmapFactory 进行创建。
-- 可以使用 BitmapFactory.option()来对原图片进行压缩（传入 options.inSampleSize）。
Options.inJustDecodeBounds = true，表示该方法只获取图片宽高等信息，不获取完整图片流。
然后针对 Options .outWidth/height 对自定义控件描绘。

24. View 与 ViewGroup 分类自定义 View 过程：onMeasure()、onLayout()、onDraw()

自定义控件流程：

1. **自定义属性**
   a) 分析完成控件显示所需属性值。
   b) 在 res/values/attrs.xml 上定义属性声明。
   c) 在 layout 布局中使用
   d) 在 View 的构造函数中去获取

2. **重写方法**
    a) onMeasure() 测量自定义 View 宽高
    b) onLayout() 布局控件在屏幕中的位置
    c) onDraw() 绘制控件
3. **处理 Touch 触摸事件**
    a) onInterceptTouchEvent 拦截事件
    b) onTouchEvent() 处理用户触摸、滑动、点击事件。
4. View 的状态保存与恢复

25. Touch 事件的分发机制

a) dispatchTouchEvent(MotionEvent ev)　　分发事件

b) onInterceptTouchEvent(MotionEvent ev) 拦截事件
--True -> 拦截调用 onTouchEvent。
--Flase -> 向下传递事件调用下层 dispatchTouchEvent()

c) onTouchEvent(MotionEvent ev) 处理事件
--True -> 消费事件，不再向上传递。
--Flase –> 不消费事件，事件继续向上传递。

26. Android 长连接，怎么处理心跳机制

移动运营商为了减轻负荷，在客户端和服务器端一段时间没有通信时，就会中断通信服务。因此，客户端应用不得不定时发送一个心跳报文到服务器端，以确保连接的有效性 一般我们用来维护推送的长连接。（长连接比较耗电）

iOS 推送：iOS 的推送是通过系统来维护，也就是说 iOS 在系统级别上维护了一个客户端应用到苹果服务器的长连接。然后再将需要的推送的消息先推送到苹果服务器，再通过系统长连接推送到手机应用当中。

好处：

1. 稳定，因为长连接是系统进程，不会被杀死。
2. 省电，系统只需维护一个长连接即可。不用每个应用都各自维护链接。
3. 安全，只有苹果开发者才能够进行消息推送。

Android 推送：android 长连接是由各自应用来维护的。Google 也有推出过推送框架，但在国内使用不了。而最近工信部也联合各大手机厂商（华为、ov、小米）和推送厂商（个推、极光）等等联合来推出一套 android 的统一推送服务。目的在于管理国内应用的消息推送问题。统一由终端厂商提供系统推送服务各厂商实现推送接口的统一。允许开发者接入，降低学习成本，提高用户体验。

27. Zygote 的启动过程

1. Loader 层：按下电源键开始，出事化硬件信息，将引导程序加载进内存，进入 Linux Kernel 层。

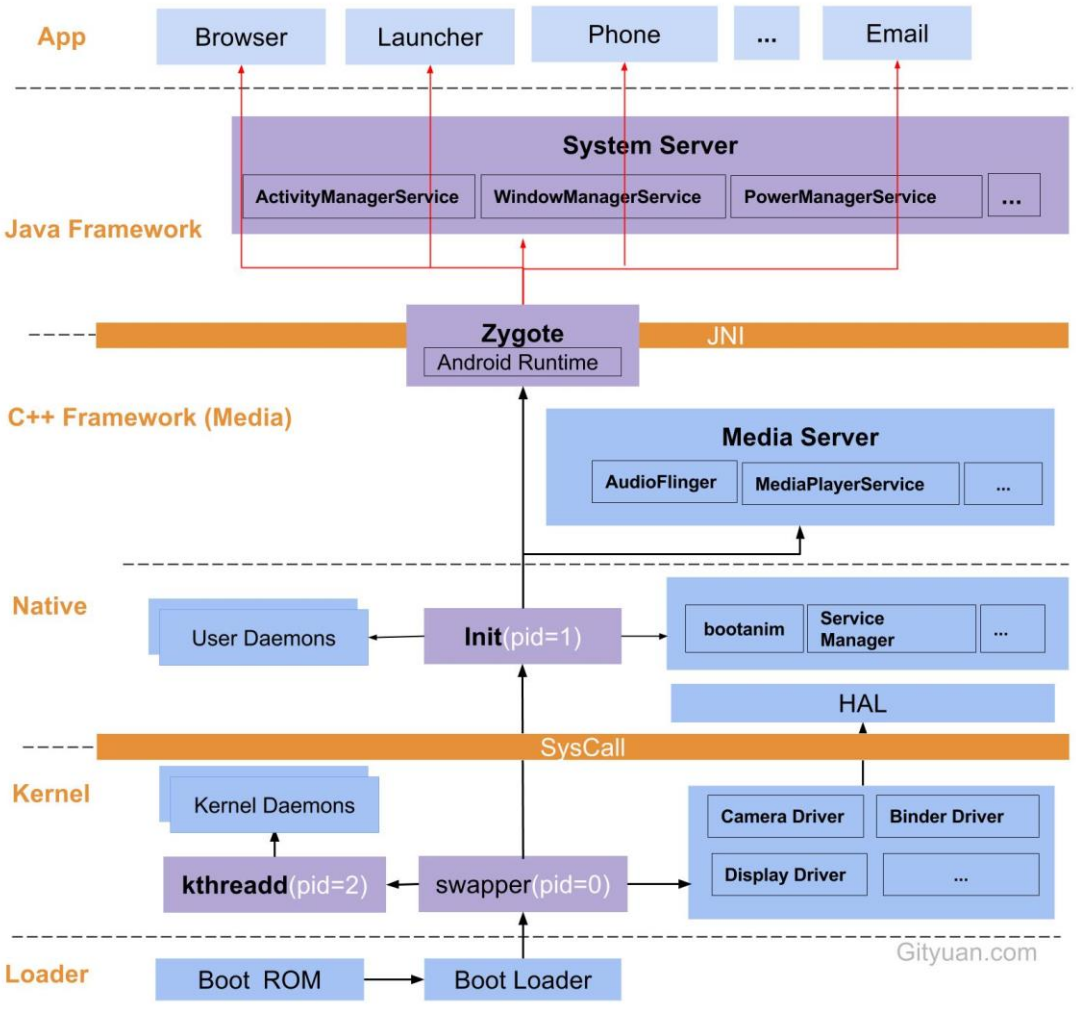
2. Linux Kernel 层：正式开始进入 Android 系统，创建系统第一个进程 –Swapper/idle

进程。这是系统的第一个进程，用于初始化进程管理、内存管理和加载各种驱动。(并且 Kernel 还会创建一个系统内核进程 Kthreadd)，进入 Native 层

3. Native 层：Native 层会孵化出 init 进程，该进程是用户空间守护进程，是所有用户进程都由 init 进程直接或间接的孵化而来。Init 进程主要负责启动 serviceManager（管理 binder IPC 通信的服务）、bootanim（开机动画）、等等的一些底层服务。并且孵化 logd、installd、adbd 等用户守护进程。 最最最重要的是，init 进程通过解析 init.rc 脚本文件后孵化出了 Zygote 进程！-- Android 系统的第一个 Java 虚拟机进程。
进入 Framework 层。

4. Framework 层：在 Zygote 进程中加载 JVM 虚拟机，创建 System Server，System Server 负责启动和管理整个 JavaFramework，包括 ActivityManager、PowerManager 等服务。

5. APP 层 :Zygote 创建的第一个应用层进程 Launcher，就是我们开机后看到的桌面。Zygote 还会创建浏览器、拨号、短信。。等等的一些系统应用。每个应用一般都会运行的单独的一个进程中。还有其他的应用进程都是由 Zygote 进程创建的！



28. Android IPC Binder 原理

Binder 是 Android 中的一种跨进程间通信的方式。

Binder 是采用客户端-服务器的通信方式。Binder 进程通信涉及到 4 个主要角色 :Client、Server、ServerManager、Binder 驱动。其中 Client、Server、ServerManager 是定义在用户空间上的，而 Binder 驱动则定义在内核空间中。

ServerManager（Native 层 c++）用于管理系统中的各种服务。是 Binder 通信机制的大管家，是 Binder 机制上的守护进程，用来给我们的 Client、Server 提供服务接口。Server 进程要先在 ServerManager 上注册，然后 Client 向 ServerManager 获取 Server 服务。

Binder 驱动：当 Server 端在 ServerManager 上注册了 Binder 对象后，Binder 驱动就会相应的创建一个 mRemote 对象。该对象也是 Binder 类，Client 客户端可以获取这个 mRemote 对象来访问远程方法。

-- Client、Server、ServerManager 定义在用户空间；Binder 驱动定义在内核空间。

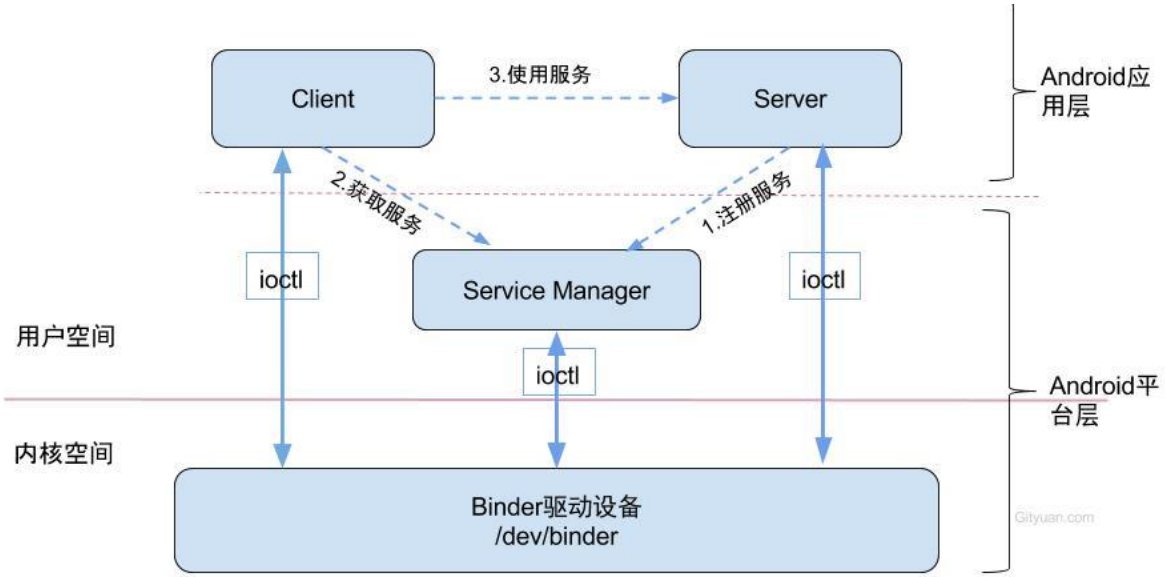
--Binder 驱动、ServerManager 在 Android 平台已经实现好，开发者只需实现 Client 和 Server。

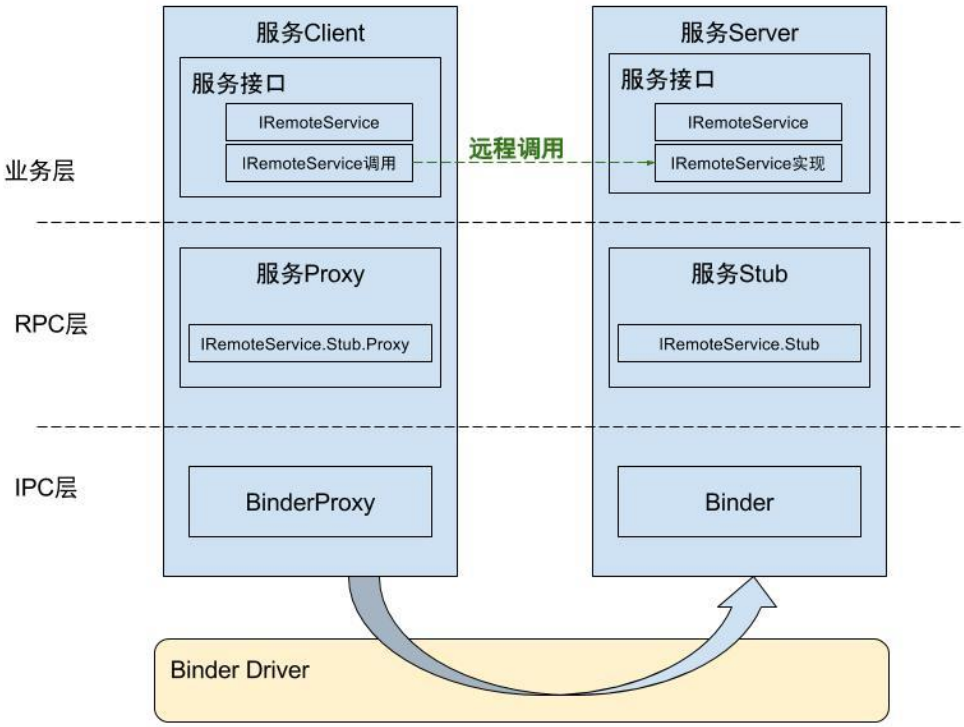
--Binder 驱动程序通过设备文件/dev/binder 与用户空间进行通信。Client、Server、ServerManager 通过 open、ioctl 文件操作函数与 Binder 进行通信。

--Client 与 Server 之间的通信是由 Binder 驱动间接完成的。

AIDL 主要为了方便我们开发者实现 Binder 进程通信，系统为我们生成底层代码。

1. 用 aidl 语言定义需要调用的接口函数。
2. Server 端实现这些方法。
3. Client 端获取 Server 端 Binder 对象 并调用这些方法。





29. 使用过什么框架、是否读过源码？底层实现原理

https://segmentfault.com/a/1190000005073746

缓存：DiskLruCache

图片加载：Glide

图片处理：Glide-transformations

网络请求：

- OkHttp -> 一个 http 与 http/2 的客户端
- Volley -> Google 推出的 Android 异步网络请求框架和图片下载框架
- Retrofit -> 类型安全的 Http 客户端

网络解析：

- Gson -> Json 和 Java bean 的转化
- Jackson -> Json、Xml、Java bean 的相互转换。

数据库：

- OrmLite

依赖注入：

- ButterKnife
- Dagger2

响应式编程：

RxJava

RxAndroid

RxBinding

性能优化：Studio 上的 Memeory Monitor

事件总线：

测试框架：

调试框架：

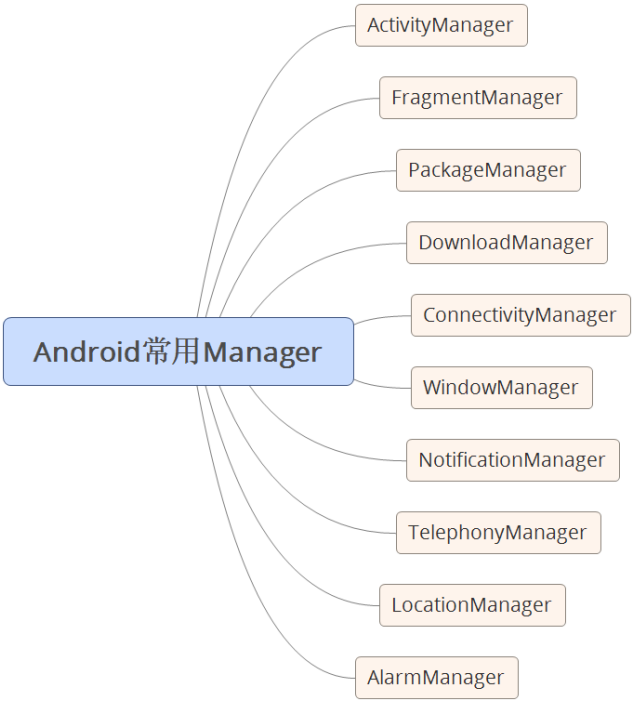
30. 如何实现一个网络框架(参考 Volley)

1.缓存队列,以 url 为 key 缓存内容可以参考 Bitmap 的处理方式，这里单独开启一个线程。

2.网络请求队列，使用线程池进行请求。

3.提供各种不同类型的返回值的解析如 String，Json，图片等等。

31. Android 常用 Manager



32. Android5.0、Android6.0 新特性

Android5.0：

--新增 MaterialDesign 设计风格

--支持 64 位 ART 虚拟机（新虚拟机，预编译技术，安装时就将字节码编译成机器码）

Android6.0：

--支持快速充电的切换

--曾强应用权限管理

--新增大量过度动画，用户体验更流畅。

Android7.0：

--支持分屏多任务

--增加 VR 支持

--新增 Instant Apps（即时应用）功能

--支持 Java8 语言的新特性

Android8.0：

--画中画模式

--官方支持 Kotlin 编程语言。

--加强后台 App 管理

--值得注意的是，对于 App 的加速完全是基于系统层面，运用到的技术包括代码本地化、并发垃圾收集压缩等等，而 App 本身则没有任何变化。

-- Android O 对于续航和硬件资源的优化，新系统通过更加智能的后台任务管理逻辑，限制了一些大量消耗资源的后台服务长期运行，借此在提升电池续航的同时也释放了更多内存资源。

你是如何自学 Android

首先是去图书馆或者上京东、亚马逊买些评价比较好的入门书去看。然后翻墙上 Google 看看 Android 的官方文档跟 Api。再上网找些视频看看，结合书本跟视频上的例子不断的去敲代码，加深印象。然后看别人的博客，看看被人对一些技术的看法和理解。平时有空做一些项目，向 github 提交代码，在看看别人的一些开源框架，开源项目。多学习学习。觉得自己基础掌握的不错之后，在去开始看进阶的书，像设计模式、别人项目框架、代码封装、第三方类库，以及看源码，看完源码学习到一些思想，然后再尝试着写点代码做点小东西，想想代码的提升。