《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

前言

历时半年，我们整理了这份市面上最全面的安卓面试题解析大全

包含了腾讯、百度、小米、阿里、乐视、美团、58、猎豹、360、新浪、搜狐等一线互联网公司面试被问到的题目。熟悉本文中列出的知识点会大大增加通过前两轮技术面试的几率。

特意推荐给安卓开发的朋友们，定价友好只需要 19.9 元。请前往支持正版。

如何使用它？

1. 可以通过目录索引直接翻看需要的知识点，查漏补缺。
2. 五角星数表示面试问到的频率，代表重要推荐指数

目录

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

第一章：Android 基础

一、 Activity(更新 2019.5.5)

1、什么是 Activity?

四大组件之一,一般的,一个用户交互界面对应一个 activity
setContentView() ,// 要显示的布局
button.setOnclickLinstener{
}, activity 是 Context 的子类,同时实现了 window.callback 和 keyevent.callback, 可以处理与窗体用户交互的事件.
我开发常用的的有
FragmentActivitiyListActivity ,PreferenceActivity ,TabAcitivty 等...

2、请描述一下 Activity 生命周期

Activity 从创建到销毁有多种状态，从一种状态到另一种状态时会激发相应的回

调方法，这些回调方法包括：onCreate onStart onResume onPause onStop onDestroy

其实这些方法都是两两对应的，onCreate 创建与 onDestroy 销毁；

onStart 可见与 onStop 不可见；onResume 可编辑（即焦点）与 onPause；

如果界面有共同的特点或者功能的时候,还会自己定义一个 BaseActivity.
进度对话框的显示与销毁

3、常见的 Activity 类型有 FragmentActivitiy，ListActivity，TabAcitivty 等。请描述一下 Activity 生命周期

Activity 从创建到销毁有多种状态，从一种状态到另一种状态时会激发相应的回调方法，这些回调方法包括：onCreate onStart onResume onPause onStop onDestroy

其实这些方法都是两两对应的，onCreate 创建与 onDestroy 销毁；

onStart 可见与 onStop 不可见；onResume 可编辑（即焦点）与 onPause。

4、如何保存 Activity 的状态或者(Activiy 重启怎么保存数据？)

Activity 的状态通常情况下系统会自动保存的，只有当我们需要保存额外的

数据时才需要使用到这样的功能。

一般来说，调用 onPause()和 onStop()方法后的 activity 实例仍然存在于内存中，activity 的所有信息和状态数据不会消失，当 activity 重新回到前台之后，所有的改变都会得到保留。

但是当系统内存不足时，调用 onPause()和 onStop()方法后的 activity 可能会被系统摧毁，此时内存中就不会存有该 activity 的实例对象了。如果之后这个 activity 重新回到前台，之前所作的改变就会消失。为了避免此种情况的发生，我们可以覆写 onSaveInstanceState()方法。onSaveInstanceState()方法接受一个 Bundle 类型的参数，开发者可以将状态数据存储到这个 Bundle 对象中，这样即使 activity 被系统摧毁，当用户重新启动这个 activity 而调用它的 onCreate()方法时，上述的 Bundle 对象会作为实参传递给 onCreate()方法，开发者可以从 Bundle 对象中取出保存的数据，然后利用这些数据将 activity 恢复到被摧毁之前的状态。

需要注意的是，onSaveInstanceState()方法并不是一定会被调用的，因为有些场景是不需要保存状态数据的. 比如用户按下 BACK 键退出 activity 时，用户显然想要关闭这个 activity，此时是没有必要保存数据以供下次恢复的，也就是 onSaveInstanceState()方法不会被调用. 如果调用 onSaveInstanceState()方法，调用将发生在 onPause()或 onStop()方法之前。

```
@Override
    protected void onSaveInstanceState(Bundle outState) {
        // TODO Auto-generated method stub
        super.onSaveInstanceState(outState);
    }
```

5、 两个 Activity 之间跳转时必然会执行的是哪几个方法？

一般情况下比如说有两个 activity,分别叫 A,B,当在 A 里面激活 B 组件的时候,A 会调用 onPause()方法,然后 B 调用 onCreate() ,onStart(), onResume()。

这个时候 B 覆盖了窗体, A 会调用 onStop()方法. 如果 B 是个透明的,或者是对话框的样式, 就不会调用 A 的 onStop()方法。

6、 横竖屏切换时 Activity 的生命周期（★★★★）

此时的生命周期跟清单文件里的配置有关系。

1．不设置 Activity 的 android:configChanges 时，切屏会重新调用各个生命周期默认首先销毁当前 activity,然后重新加载。

2．设置 Activity android:configChanges="orientation|keyboardHidden|screenSize"时，切屏不会重新调用各个生命周期，只会执行 onConfigurationChanged 方法。

通常在游戏开发，屏幕的朝向都是写死的。

7、 如何将一个 Activity 设置成窗口的样式

只需要给我们的 Activity 配置如下属性即可。

android:theme="@android:style/Theme.Dialog"

8、 如何退出 Activity？如何安全退出已调用多个 Activity 的 Application？（★★★★）

1、通常情况用户退出一个 Activity 只需按返回键，我们写代码想退出 activity 直接调用 finish()方法就行。

2、记录打开的 Activity：

每打开一个 Activity，就记录下来。在需要退出时，关闭每一个 Activity 即可。

```
//伪代码
List<Activity> lists ;// 在 application 全局的变量里面
lists = new ArrayList<Activity>();
lists.add(this);
for(Activity activity: lists)
{
   activity.finish();
}
lists.remove(this);
```

3、发送特定广播：

在需要结束应用时，发送一个特定的广播，每个 Activity 收到广播后，关闭即可。

//给某个 activity 注册接受接受广播的意图

registerReceiver(receiver, filter)

//如果过接受到的是 关闭 activity 的广播 就调用 finish()方法 把当前的 activity finish()掉

4、递归退出

在打开新的 Activity 时使用 startActivityForResult，然后自己加标志，在 onActivityResult 中处理，递归关闭。

5、其实 也可以通过 intent 的 flag 来实现

intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)激活一个新的 activity。此时如果该任务栈中已经有该 Activity，那么系统会把这个 Activity 上面的所有 Activity 干掉。其实相当于给 Activity 配置的启动模式为 SingleTop。

9、 Activity 的四种启动模式，singletop 和 singletask 区别是什么？一般书签的使用模式是 singletop，那为什么不使用 singletask?

singleTop 跟 standard 模式比较类似。唯一的区别就是，当跳转的对象是位于栈顶的 activity（应该可以理解为用户眼前所 看到的 activity）时，程序将不会生成一个新的 activity 实例，而是直接跳到现存于栈顶的那个 activity 实例。拿上面的例子来说，当 Act1 为 singleTop 模式时，执行跳转后栈里面依旧只有一个实例，如果现在按返回键程序将直接退出。

singleTask 模式和 singleInstance 模式都是只创建一个实例的。在这种模式下，无论跳转的对象是不是位于栈顶的 activity，程序都不会生成一个新的实例(当然前提是栈里面已经有这个实例)。这种模式相当有用 ,在以后的多 activity 开发中，常会因为跳转的关系导致同个页面生成多个实例，这个在用户体验上始终有点不好，而如果你将对应的 activity 声明为 singleTask 模式，这种问题将不复存在。在主页的 Activity 很常用

10、Android 中的 Context, Activity，Appliction 有什么区别？（★★）

相同：Activity 和 Application 都是 Context 的子类。

Context 从字面上理解就是上下文的意思 ,在实际应用中它也确实是起到了管理上下文环境中各个参数和变量的总用，方便我们可以简单的访问到各种资源。

不同：维护的生命周期不同。 Context 维护的是当前的 Activity 的生命周期，Application 维护的是整个项目的生命周期。

使用 context 的时候，小心内存泄露，防止内存泄露，注意一下几个方面：

1. 不要让生命周期长的对象引用 activity context，即保证引用 activity 的对象要与 activity 本身生命周期是一样的。

2. 对于生命周期长的对象，可以使用 application，context。

3. 避免非静态的内部类，尽量使用静态类，避免生命周期问题，注意内部类对外部对象引用导致的生命周期变化。

11、两个 Activity 之间传递数据，除了 intent，广播接收者，content provider 还有啥？

1）利用 static 静态数据，public static 成员变量

2）利用外部存储的传输，

例如 File 文件存储

SharedPreferences 首选项

Sqlite 数据库

12、Context 是什么？，一个应用有多少个 Context(2019.5.5）（★★）

1、它描述的是一个应用程序环境的信息，即上下文。

2、该类是一个抽象(abstract class)类，Android 提供了该抽象类的具体实现类（ContextIml）。

3、通过它我们可以获取应用程序的资源和类，也包括一些应用级别操作，例如：启动一个 Activity，发送广播，接受 Intent，信息，等。

4、一个应用 Context 数量=Activity 个数+service 个数+1 个

13.activity 之间还有 activity 和 service 之间如何传值，可以传图片吗？ (2019.5.5）（上海）

Activity 之间的传值：startActivity，通过 Intent 对象的各种 putExtra 方法来进行传递。在第二个 Activity 对象中，可以通过 getIntent() 方法来得到跳转到这个 Activity 的 Intent 对象，然后通过 Intent 对象的各种 getXXExtra 方法来得到我们的传过来的值。

Activity 和 service 之间传值：

1、在 activity 中通过 startService(intent)即可 ,同样 intent.putStringExtra()，然后再 service 中的 onStart 函数中获取该值,this.getIntent(),intent.getString()

在这里我们需要在 Mainfeist 文件中注册这个 service

<service Android:enabled="true" android:name=".Service">

</service>

2、service 可以从 public int onStartCommand(Intent intent, int flags, int startId)中取出从 activity 中传过来的值。intent.getExtra()获得 bundle 对象，可从中取值。

3、activity 也可以用 bindService(intent, conn,BIND_AUTO_CREATE);传值，把要传的值绑定在 intent 里，在 service 的 public IBinder onBind(Intent

intent) 方法里取得 intent。

4、同时也可以在 reseiver 里面注册一个广播 ,在 activity 里 sendbroadcast（intent）传值。

可以传递图片，用 Intent 把图片的地址或者把图片对象用 Intent 传过去，用 bitmap 对象。

二、 Service

1.Service 是否在 main thread 中执行, service 里面是否能执行耗时的操作?（★★）

默认情况,如果没有显示的指 servic 所运行的进程, Service 和 activity 是运行在当前 app 所在进程的 main thread(UI 主线程)里面。

service 里面不能执行耗时的操作(网络请求,拷贝数据库,大文件)

特殊情况 ,可以在清单文件配置 service 执行所在的进程 ,让 service 在另外的进程中执行

```
<service
    android:name="com.baidu.location.f"
    android:enabled="true"
    android:process=":remote" >
</service>
```

2.Activity 怎么和 Service 绑定，怎么在 Activity 中启动自己对应的 Service？

Activity 通过 bindService(Intent service, ServiceConnection conn, int flags)跟 Service 进行绑定，当绑定成功的时候 Service 会将代理对象通过回调的形式传给 conn，这样我们就拿到了 Service 提供的服务代理对象。

在 Activity 中可以通过 startService 和 bindService 方法启动 Service。一般情况下如果想获取 Service 的服务对象那么肯定需要通过 bindService（）方法，比如音乐播放器，第三方支付等。如果仅仅只是为了开启一个后台任务那么可以使用 startService（）方法。

3.请描述一下 Service 的生命周期（★★★）

Service 有绑定模式和非绑定模式，以及这两种模式的混合使用方式。不同的使用方法生命周期方法也不同。

非绑定模式：当第一次调用 startService 的时候执行的方法依次为 onCreate()、onStartCommand()，当 Service 关闭的时候调用 onDestory 方法。

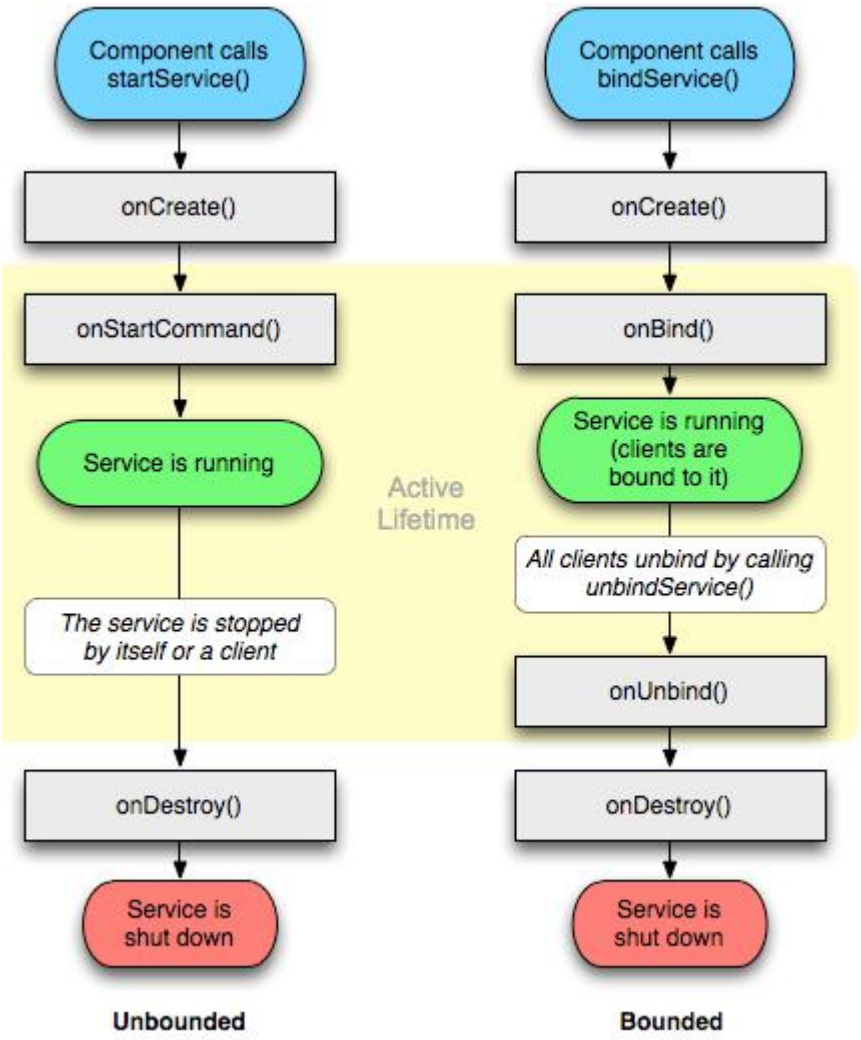
绑定模式：第一次 bindService（）的时候，执行的方法为 onCreate()、onBind()解除绑定的时候会执行 onUnbind()、onDestory()。

上面的两种生命周期是在相对单纯的模式下的情形。我们在开发的过程中还必须注意 Service 实例只会有一个，也就是说如果当前要启动的 Service 已经存

在了那么就不会再次创建该 Service 当然也不会调用 onCreate（ ）方法。

一个 Service 可以被多个客户进行绑定，只有所有的绑定对象都执行了 onBind（ ）方法后该 Service 才会销毁，不过如果有一个客户执行了 onStart()方法，那么这个时候如果所有的 bind 客户都执行了 unBind()该 Service 也不会销毁。

Service 的生命周期图如下所示，帮助大家记忆。



4.什么是 IntentService？有何优点？（★★★）

我们通常只会使用 Service，可能 IntentService 对大部分同学来说都是第一次听说。那么看了下面的介绍相信你就不再陌生了。如果你还是不了解那么在面试的时候你就坦诚说没用过或者不了解等。并不是所有的问题都需要回答上来的。

一、IntentService 简介

IntentService 是 Service 的子类，比普通的 Service 增加了额外的功能。先看 Service 本身存在两个问题：

Service 不会专门启动一条单独的进程，Service 与它所在应用位于同一个进程中；

Service 也不是专门一条新线程，因此不应该在 Service 中直接处理耗时的任务；

二、IntentService 特征

会创建独立的 worker 线程来处理所有的 Intent 请求；

会创建独立的 worker 线程来处理 onHandleIntent()方法实现的代码，无需处理多线程问题；

所有请求处理完成后，IntentService 会自动停止，无需调用 stopSelf()方法停止 Service；

为 Service 的 onBind()提供默认实现，返回 null；

为 Service 的 onStartCommand 提供默认实现 ,将请求 Intent 添加到队列中；

使用 IntentService

本人写了一个 IntentService 的使用例子供参考。该例子中一个 MainActivity 一个 MyIntentService，这两个类都是四大组件当然需要在清单文件中注册。这里只给出核心代码：

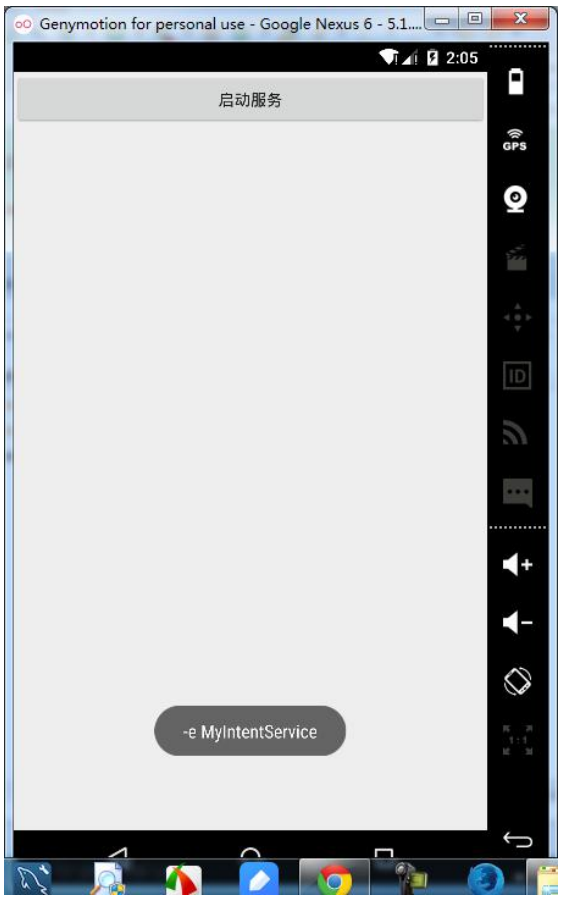
MainActivity.java:

```
public void click(View view){
    Intent intent = new Intent(this, MyIntentService.class);
    intent.putExtra("start", "MyIntentService");
    startService(intent);
}
```

MyIntentService.java

```
public class MyIntentService extends IntentService {

    private String ex = "";
    private Handler mHandler = new Handler() {

        public void handleMessage(android.os.Message msg) {
            Toast.makeText(MyIntentService.this, "-e " + ex,
Toast.LENGTH_LONG).show();
        }
    };

    public MyIntentService(){
        super("MyIntentService");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        ex = intent.getStringExtra("start");
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        /**
         * 模拟执行耗时任务
         * 该方法是在子线程中执行的，因此需要用到 handler 跟主线程进行通信
         */
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        mHandler.sendEmptyMessage(0);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

运行后效果如下：



5. 说说 Activity、Intent、Service 是什么关系

他们都是 Android 开发中使用频率最高的类。其中 Activity 和 Service 都是 Android 四大组件之一。他俩都是 Context 类的子类 ContextWrapper 的子类，因此他俩可以算是兄弟关系吧。不过兄弟俩各有各自的本领，Activity 负责用户界面的显示和交互，Service 负责后台任务的处理。Activity 和 Service 之间可以通过 Intent 传递数据，因此可以把 Intent 看作是通信使者。

6. Service 和 Activity 在同一个线程吗（★）

对于同一 app 来说默认情况下是在同一个线程中的，main Thread （UI Thread）。

7. Service 里面可以弹吐司么（★）

可以的。弹吐司有个条件就是得有一个 Context 上下文，而 Service 本身就是 Context 的子类，因此在 Service 里面弹吐司是完全可以的。比如我们在 Service 中完成下载任务后可以弹一个吐司通知用户。

8. 什么是 Service 以及描述下它的生命周期。Service 有哪些启动方法，有什么区别，怎样停用 Service？（★★★）

在 Service 的生命周期中，被回调的方法比 Activity 少一些，只有 onCreate, onStart, onDestroy,
onBind 和 onUnbind。
通常有两种方式启动一个 Service,他们对 Service 生命周期的影响是不一样的。

1． 通过 startService

Service 会经历 onCreate 到 onStart，然后处于运行状态，stopService 的时候调用 onDestroy 方法。

如果是调用者自己直接退出而没有调用 stopService 的话，Service 会一直在后台运行。

2． 通过 bindService

Service 会运行 onCreate，然后是调用 onBind，这个时候调用者和 Service 绑定在一起。调用者退出了，Srevice 就会调用 onUnbind->onDestroyed 方法。

所谓绑定在一起就共存亡了。调用者也可以通过调用 unbindService 方法来

停止服务，这时候 Srevice 就会调用 onUnbind->onDestroyed 方法。
需要注意的是如果这几个方法交织在一起的话，会出现什么情况呢？
一个原则是 Service 的 onCreate 的方法只会被调用一次，就是你无论多少次的 startService 又 bindService，Service 只被创建一次。

如果先是 bind 了，那么 start 的时候就直接运行 Service 的 onStart 方法，如果先是 start，那么 bind 的时候就直接运行 onBind 方法。
如果 service 运行期间调用了 bindService，这时候再调用 stopService 的话，service 是不会调用 onDestroy 方法的，service 就 stop 不掉了，只能调用 UnbindService, service 就会被销毁

如果一个 service 通过 startService 被 start 之后，多次调用 startService 的话，service 会多次调用 onStart 方法。多次调用 stopService 的话，service 只会调用一次 onDestroyed 方法。

如果一个 service 通过 bindService 被 start 之后，多次调用 bindService 的话，service 只会调用一次 onBind 方法。
多次调用 unbindService 的话会抛出异常。

9. 在 service 的生命周期方法 onstartConmand()可不可以执行网络操作？如何在 service 中执行网络操作？

可以直接在 Service 中执行网络操作,在 onStartCommand()方法中可以执行网络操作

三、 Broadcast Receiver

1.请描述一下 BroadcastReceiver（★★）

BroadCastReceiver 是 Android 四大组件之一，主要用于接收系统或者 app 发送的广播事件。

广播分两种：有序广播和无序广播。
内部通信实现机制：通过 Android 系统的 Binder 机制实现通信。
无序广播：完全异步，逻辑上可以被任何广播接收者接收到。优点是效率较高。缺点是一个接收者不能将处理结果传递给下一个接收者 ,并无法终止广播 intent 的传播。
有序广播：按照被接收者的优先级顺序，在被接收者中依次传播。比如有三个广播接收者 A，B，C，优先级是 A > B > C。那这个消息先传给 A，再传给 B，最后传给 C。每个接收者有权终止广播，比如 B 终止广播，C 就无法接收到。此外 A 接收到广播后可以对结果对象进行操作，当广播传给 B 时，B 可以从结果对象中取得 A 存入的数据。
在通过 Context.sendOrderedBroadcast(intent, receiverPermission, resultReceiver, scheduler, initialCode, initialData, initialExtras)时我们可以指定 resultReceiver 广播接收者，这个接收者我们可以认为是最终接收者，通常情况下如果比他优先级更高的接收者如果没有终止广播 ,那么他的 onReceive 会被执行两次，第一次是正常的按照优先级顺序执行，第二次是作为最终接收者接收。如果比他优先级高的接收者终止了广播，那么他依然能接收到广播。
在我们的项目中经常使用广播接收者接收系统通知，比如开机启动、sd 挂载、低电量、外播电话、锁屏等。
如果我们做的是播放器，那么监听到用户锁屏后我们应该将我们的播放之暂停等。

2.在 manifest 和代码中如何注册和使用 BroadcastReceiver

在清单文件中注册广播接收者称为静态注册，在代码中注册称为动态注册。静态注册的广播接收者只要 app 在系统中运行则一直可以接收到广播消息，动态注册的广播接收者当注册的 Activity 或者 Service 销毁了那么就接收不到广播了。

静态注册：在清单文件中进行如下配置

```
<receiver android:name=".BroadcastReceiver1" >
          <intent-filter>
              <action android:name="android.intent.action.CALL" >
              </action>
          </intent-filter>
</receiver>
```

动态注册：在代码中进行如下注册

```
         receiver = new BroadcastReceiver();
          IntentFilter intentFilter = new IntentFilter();
          intentFilter.addAction(CALL_ACTION);
          context.registerReceiver(receiver, intentFilter);
```

3.BroadCastReceiver 的生命周期

a. 广播接收者的生命周期非常短暂的，在接收到广播的时候创建，onReceive()方法结束之后销毁；

b. 广播接收者中不要做一些耗时的工作，否则会弹出 Application No Response 错误对话框；

c. 最好也不要在广播接收者中创建子线程做耗时的工作，因为广播接收者被销毁后进程就成为了空进程，很容易被系统杀掉；

d. 耗时的较长的工作最好放在服务中完成；

1、Android 引入广播机制的用意

a. 从 MVC 的角度考虑(应用程序内) 其实回答这个问题的时候还可以这样问，android 为什么要有那 4 大组件，现在的移动开发模型基本上也是照搬的 web 那一套 MVC 架构，只不过是改了点嫁妆而已。android 的四大组件本质上就是为了实现移动或者说嵌入式设备上的 MVC 架构，它们之间有时候是一种相互依存的关系，有时候又是一种补充关系，引入广播机制可以方便几大组件的信息和数据交互。
b. 程序间互通消息(例如在自己的应用程序内监听系统来电)
c. 效率上(参考 UDP 的广播协议在局域网的方便性)
d. 设计模式上(反转控制的一种应用，类似监听者模式)

四、 ContentProvider（更新 2019.6.14）

1.请介绍下 ContentProvider 是如何实现数据共享的(★★★)

在 Android 中如果想将自己应用的数据（一般多为数据库中的数据）提供给第三发应用，那么我们只能通过 ContentProvider 来实现了。

ContentProvider 是应用程序之间共享数据的接口。使用的时候首先自定义一个类继承 ContentProvider，然后覆写 query、insert、update、delete 等方法。因为其是四大组件之一因此必须在 AndroidManifest 文件中进行注册。把自己的数据通过 uri 的形式共享出去，android 系统下 不同程序 数据默认是不能共享访问，需要去实现一个类去继承 ContentProvider

```
public class PersonContentProvider extends ContentProvider{
        public boolean onCreate(){

                                }
    query(Uri, String[], String, String[], String)
```

```
        insert(Uri, ContentValues)
        update(Uri, ContentValues, String, String[])
        delete(Uri, String, String[])
}
```

```
<provider
android:exported="true"
android:name="com.itheima.contenProvider.provider.PersonContentPro
vider"android:authorities="com.itheima.person" />
```

第三方可以通过 ContentResolver 来访问该 Provider。

2.为什么要用 ContentProvider？它和 sql 的实现上有什么差别？（★★★）

ContentProvider 屏蔽了数据存储的细节,内部实现对用户完全透明,用户只需要关心操作数据的 uri 就可以了，ContentProvider 可以实现不同 app 之间共享。

Sql 也有增删改查的方法，但是 sql 只能查询本应用下的数据库。而 ContentProvider 还可以去增删改查本地文件. xml 文件的读取等。

3.说说 ContentProvider、ContentResolver、ContentObserver 之间的关系（★★★★）

a. ContentProvider 内容提供者，用于对外提供数据

b. ContentResolver.notifyChange(uri)发出消息

c. ContentResolver 内容解析者，用于获取内容提供者提供的数据

d. ContentObserver 内容监听器，可以监听数据的改变状态

e. ContentResolver.registerContentObserver()监听消息。

4.使用 contentProvider 获取本地所有的音频文件(上海)(更新 2019.5.5) (★★★)

Android 中，系统为多媒体类型的文件（比如图片、音频、视频等）建立了数据库(sqlite 数据库)，将文件的名称、大小等信息存储起来，从而完成多媒体数据的维护工作；所以我们需要实现建立一个实体类。

可以根据 ContentResover 获取到一个 Cursor，然后根据这个游标，遍历所有的歌曲的信息，设置给实体类，得到你想要的音频文件。因为是从本地读取数据，所以需要添加权限

<uses-permission

android:name="android.permission.READ_EXTERNAL_STORAGE"/>

五、 ListView

1.ListView 如何提高其效率？ (★★)

当 convertView 为空时，用 setTag()方法为每个 View 绑定一个存放控件的 ViewHolder 对象。当 convertView 不为空 ,重复利用已经创建的 view 的时候，

使用 getTag()方法获取绑定的 ViewHolder 对象，这样就避免了 findViewById 对控件的层层查询，而是快速定位到控件。

① 复用 ConvertView，使用历史的 view，提升效率 200%

② 自定义静态类 ViewHolder，减少 findViewById 的次数。提升效率 50%

③ 异步加载数据，分页加载数据。

④ 使用 WeakRefrence 引用 ImageView 对象

2.当 ListView 数据集改变后，如何更新 ListView

使用该 ListView 的 adapter 的 notifyDataSetChanged()方法。该方法会使 ListView 重新绘制。

3.ListView 如何实现分页加载（★★★★★）

① 设置 ListView 的滚动监听器：setOnScrollListener(new OnScrollListener{....})

在监听器中有两个方法： 滚动状态发生变化的方法(onScrollStateChanged)和 listView 被滚动时调用的方法(onScroll)

② 在滚动状态发生改变的方法中，有三种状态：

手指按下移动的状态：SCROLL_STATE_TOUCH_SCROLL: // 触摸滑动

惯性滚动（滑翔（flgin）状态）：SCROLL_STATE_FLING: // 滑翔

静止状态： SCROLL_STATE_IDLE: // 静止

对不同的状态进行处理：

分批加载数据，只关心静止状态：关心最后一个可见的条目，如果最后一个可见条目就是数据适配器（集合）里的最后一个，此时可加载更多的数据。在每次加载的时候，计算出滚动的数量，当滚动的数量大于等于总数量的时候，可以提示用户无更多数据了。

4.ListView 可以显示多种类型的条目吗（★★）

这个当然可以的，ListView 显示的每个条目都是通过 baseAdapter 的 getView(int position, View convertView, ViewGroup parent)来展示的，理论上我们完全可以让每个条目都是不同类型的 view。

比如：从服务器拿回一个标识为 id=1,那么当 id=1 的时候，我们就加载类型一的条目，当 id=2 的时候，加载类型二的条目。常见布局在资讯类客户端中可以经常看到。

除此之外 adapter 还提供了 getViewTypeCount（）和 getItemViewType(int position)两个方法。在 getView 方法中我们可以根据不同的 viewtype 加载不同的布局文件。

5.ListView 如何定位到指定位置

可以通过 ListView 提供的 lv.setSelection(listView.getPosition());方法。

6.如何在 ScrollView 中如何嵌入 ListView（★★★★）

通常情况下我们不会在 ScrollView 中嵌套 ListView，但是如果面试官非让我嵌套的话也是可以的。

在 ScrollView 添加一个 ListView 会导致 listview 控件显示不全，通常只会显示一条，这是因为两个控件的滚动事件冲突导致。所以需要通过 listview 中的 item 数量去计算 listview 的显示高度，从而使其完整展示，如下提供一个方法供大家参考。

```
        lv = (ListView) findViewById(R.id.lv);
        adapter = new MyAdapter();
        lv.setAdapter(adapter);
        setListViewHeightBasedOnChildren(lv);
-----------------------------------------------------
public void setListViewHeightBasedOnChildren(ListView listView) {
        ListAdapter listAdapter = listView.getAdapter();
        if (listAdapter == null) {
            return;
        }
        int totalHeight = 0;
        for (int i = 0; i < listAdapter.getCount(); i++) {
            View listItem = listAdapter.getView(i, null, listView);
            listItem.measure(0, 0);
            totalHeight += listItem.getMeasuredHeight();
```

```
        }
        ViewGroup.LayoutParams params = listView.getLayoutParams();
        params.height = totalHeight + (listView.getDividerHeight() *
(listAdapter.getCount() - 1));
        params.height += 5;// if without this statement,the listview will be
                           // little short
        listView.setLayoutParams(params);
    }
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

现阶段最好的处理的方式是： 自定义 ListView，重载 onMeasure()方法，设置全部显示。

```
package com.meiya.ui;

import android.widget.ListView;

/**
*
* @Description: scrollview 中内嵌 listview 的简单实现
*
* @File: ScrollViewWithListView.java
*
*
* @Version
*/
public class ScrollViewWithListView extends ListView {

public ScrollViewWithListView(android.content.Context context,
android.util.AttributeSet attrs) {
super(context, attrs);
}

/**
* Integer.MAX_VALUE >> 2,如果不设置，系统默认设置是显示两条
*/
public void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
int expandSpec = MeasureSpec.makeMeasureSpec(Integer.MAX_VALUE >> 2,
MeasureSpec.AT_MOST);
super.onMeasure(widthMeasureSpec, expandSpec);

}

}
```

7.ListView 中如何优化图片（★★）

图片的优化策略比较多。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

1、处理图片的方式：

如果 ListView 中自定义的 Item 中有涉及到大量图片的，一定要对图片进行细心的处理，因为图片占的内存是 ListView 项中最头疼的，处理图片的方法大致有以下几种：

①、不要直接拿路径就去循环 BitmapFactory.decodeFile;使用 Options 保存图片大小、不要加载图片到内存去。

②、对图片一定要经过边界压缩尤其是比较大的图片，如果你的图片是后台服务器处理好的那就不需要了

③、在 ListView 中取图片时也不要直接拿个路径去取图片，而是以 WeakReference（使用 WeakReference 代替强引用。比如可以使用 WeakReference mContextRef）、SoftReference、WeakHashMap 等的来存储图片信息。

④、在 getView 中做图片转换时，产生的中间变量一定及时释放

2、异步加载图片基本思想：

1）、 先从内存缓存中获取图片显示（内存缓冲）

2）、获取不到的话从 SD 卡里获取（SD 卡缓冲）

3）、都获取不到的话从网络下载图片并保存到 SD 卡同时加入内存并显示（视情况看是否要显示）

原理：

优化一：先从内存中加载，没有则开启线程从 SD 卡或网络中获取，这里注

意从 SD 卡获取图片是放在子线程里执行的，否则快速滑屏的话会不够流畅。

优化二：于此同时，在 adapter 里有个 busy 变量，表示 listview 是否处于滑动状态，如果是滑动状态则仅从内存中获取图片，没有的话无需再开启线程去外存或网络获取图片。

优化三：ImageLoader 里的线程使用了线程池，从而避免了过多线程频繁创建和销毁，如果每次总是 new 一个线程去执行这是非常不可取的，好一点的用的 AsyncTask 类，其实内部也是用到了线程池。在从网络获取图片时，先是将其保存到 sd 卡，然后再加载到内存，这么做的好处是在加载到内存时可以做个压缩处理，以减少图片所占内存。

8.ListView 中图片错位的问题是如何产生的（★★★）

图片错位问题的本质源于我们的 listview 使用了缓存 convertView，假设一种场景，一个 listview 一屏显示九个 item，那么在拉出第十个 item 的时候，事实上该 item 是重复使用了第一个 item，也就是说在第一个 item 从网络中下载图片并最终要显示的时候，其实该 item 已经不在当前显示区域内了，此时显示的后果将可能在第十个 item 上输出图像，这就导致了图片错位的问题。所以解决之道在于可见则显示，不可见则不显示。

10.如何刷新 ListView 中单个 item 的数据，不刷新整个 ListView 的数据？

修改单个 Item 的数据,然后调用适配器的 notifyDataSetChanged()方法

六、Intent

1.Intent 传递数据时，可以传递哪些类型数据？

Intent 可以传递的数据类型非常的丰富，java 的基本数据类型和 String 以及他们的数组形式都可以，除此之外还可以传递实现了 Serializable 和 Parcelable 接口的对象。

2.Serializable 和 Parcelable 的区别（★★★）

在使用内存的时候，Parcelable 类比 Serializable 性能高，所以推荐使用 Parcelable 类。

1．Serializable 在序列化的时候会产生大量的临时变量，从而引起频繁的 GC。

2．Parcelable 不能使用在要将数据存储在磁盘上的情况。尽管 Serializable 效率低点，但在这种情况下，还是建议你用 Serializable 。

实现：

1．Serializable 的实现，只需要继承 Serializable 即可。这只是给对象打

了一个标记，系统会自动将其序列化。

2．Parcelabel 的实现，需要在类中添加一个静态成员变量 CREATOR，这个变量需要继承 Parcelable.Creator 接口。

```
public class MyParcelable implements Parcelable {
    private int mData;

    public int describeContents() {
        return 0;
    }

    public void writeToParcel(Parcel out, int flags) {
        out.writeInt(mData);
    }

    public static final Parcelable.Creator<MyParcelable> CREATOR
            = new Parcelable.Creator<MyParcelable>() {
        public MyParcelable createFromParcel(Parcel in) {
            return new MyParcelable(in);
        }

        public MyParcelable[] newArray(int size) {
            return new MyParcelable[size];
        }
    };

    private MyParcelable(Parcel in) {
        mData = in.readInt();
    }
}
```

3.请描述一下 Intent 和 IntentFilter（★★★）

Android 中通过 Intent 对象来表示一条消息，一个 Intent 对象不仅包含

有这个消息的目的地，还可以包含消息的内容，这好比一封 Email，其中不仅应该包含收件地址，还可以包含具体的内容。对于一个 Intent 对象，消息“目的地”是必须的，而内容则是可选项。

通过 Intent 可以实现各种系统组件的调用与激活.

IntentFilter: 可以理解为邮局或者是一个信笺的分拣系统...

这个分拣系统通过 3 个参数来识别

Action: 动作 view

Data: 数据 uri uri

Category：而外的附加信息

Action 匹配

Action 是一个用户定义的字符串，用于描述一个 Android 应用程序组件，一个 IntentFilter 可以包含多个 Action。在 AndroidManifest.xml 的 Activity 定义时可以在其 <intent-filter >节点指定一个 Action 列表用于标示 Activity 所能接受的“动作”，例如：

```
<intent-filter >
<action android:name="android.intent.action.MAIN" />
<action android:name="cn.itheima.action" />
.....
</intent-filter>
```

如果我们在启动一个 Activity 时使用这样的 Intent 对象：

```
Intent intent =new Intent();
intent.setAction("cn.itheima.action");
```

那么所有的 Action 列表中包含了“cn.itheima”的 Activity 都将会匹配成功。

Android 预定义了一系列的 Action 分别表示特定的系统动作。这些 Action 通过常量的方式定义在 android.content. Intent 中，以“ACTION_”开头。我们可以在 Android 提供的文档中找到它们的详细说明。

URI 数据匹配

一个 Intent 可以通过 URI 携带外部数据给目标组件。在 <intent-filter > 节点中，通过 <data/>节点匹配外部数据。

mimeType 属性指定携带外部数据的数据类型，scheme 指定协议，host、port、path 指定数据的位置、端口、和路径。如下：

```
<data android:mimeType="mimeType" android:scheme="scheme"
android:host="host" android:port="port" android:path="path"/>
```

电话的 uri　tel: 12345

http://www.baidu.com

自己定义的 uri　itcast://cn.itcast/person/10

如果在 Intent Filter 中指定了这些属性，那么只有所有的属性都匹配成功时 URI 数据匹配才会成功。

Category 类别匹配

<intent-filter >节点中可以为组件定义一个 Category 类别列表，当 Intent 中包含这个列表的所有项目时 Category 类别匹配才会成功。

4.intent 的意义

保证了组件之间的通信，在组件之间传递了数据。

七、Fragment

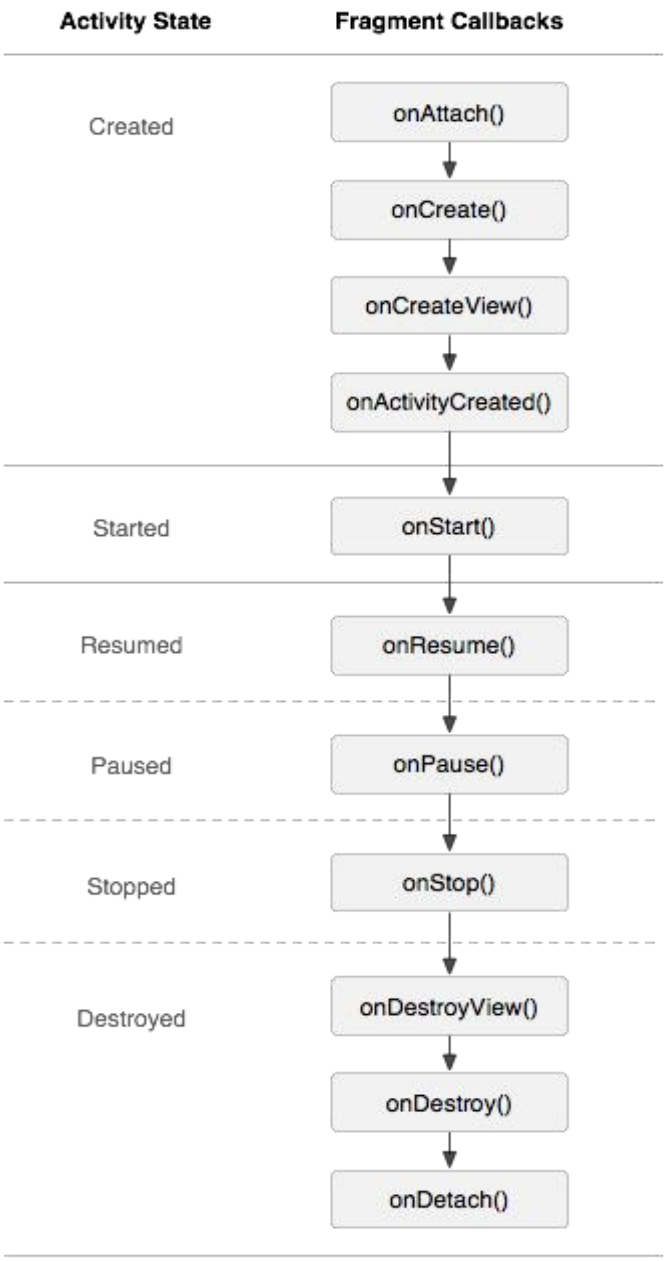
1.Fragment 跟 Activity 之间是如何传值的

当 Fragment 跟 Activity 绑定之后，在 Fragment 中可以直接通过 getActivity（）方法获取到其绑定的 Activity 对象，这样就可以调用 Activity 的方法了。在 Activity 中可以通过如下方法获取到 Fragment 实例

```
FragmentManager fragmentManager = getFragmentManager();
Fragment fragment = fragmentManager.findFragmentByTag(tag);
Fragment fragment = fragmentManager.findFragmentById(id);
```

获取到 Fragment 之后就可以调用 Fragment 的方法。也就实现了通信功能。

2.描述一下 Fragment 的生命周期



3.Fragment 的 replace 和 add 方法的区别

Fragment 本身并没有 replace 和 add 方法，这里的理解应该为使用

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

FragmentManager 的 replace 和 add 两种方法切换 Fragment 时有什么不同。

我们经常使用的一个架构就是通过 RadioGroup 切换 Fragment，每个 Fragment 就是一个功能模块。

```
                case R.id.rb_1:
                    rb_1.setBackgroundColor(Color.RED);
                    transaction.show(fragment1);
//                  transaction.replace(R.id.fl, fragment1, "Fragment1");
                    break;
                case R.id.rb_2:
                    rb_2.setBackgroundColor(Color.YELLOW);
//                  transaction.replace(R.id.fl, fragment2, "Fragment2");
                    transaction.show(fragment2);
                    break;
                case R.id.rb_3:
                    rb_3.setBackgroundColor(Color.BLUE);
//                  transaction.replace(R.id.fl, fragment3, "Fragment3");
                    transaction.show(fragment3);
                    break;
```

实现这个功能可以通过 replace 和 add 两种方法。

Fragment 的容器一个 FrameLayout，add 的时候是把所有的 Fragment 一层一层的叠加到了 FrameLayout 上了，而 replace 的话首先将该容器中的其他 Fragment 去除掉然后将当前 Fragment 添加到容器中。

一个 Fragment 容器中只能添加一个 Fragment 种类，如果多次添加则会报异常，导致程序终止，而 replace 则无所谓，随便切换。

因为通过 add 的方法添加的 Fragment，每个 Fragment 只能添加一次，因此如果要想达到切换效果需要通过 Fragment 的的 hide 和 show 方法结合者使用。将要显示的 show 出来，将其他 hide 起来。这个过程 Fragment 的生命周期没

有变化。

通过 replace 切换 Fragment，每次都会执行上一个 Fragment 的 onDestroyView ,新 Fragment 的 onCreateView、onStart、onResume 方法。基于以上不同的特点我们在使用的使用一定要结合着生命周期操作我们的视图和数据。

4.Fragment 如何实现类似 Activity 栈的压栈和出栈效果的？

Fragment 的事物管理器内部维持了一个双向链表结构，该结构可以记录我们每次 add 的 Fragment 和 replace 的 Fragment ,然后当我们点击 back 按钮的时候会自动帮我们实现退栈操作。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

Add this transaction to the back stack. This means that the transaction will be remembered after it is co
and will reverse its operation when later popped off the stack.

Parameters:
name An optional name for this back stack state, or null.

```
transaction.addToBackStack("name");

//实现源码 在 BackStackRecord 中
 public FragmentTransaction addToBackStack(String name) {
        if (!mAllowAddToBackStack) {
            throw new IllegalStateException(
                    "This FragmentTransaction is not allowed to be added to th
stack.");
        }
        mAddToBackStack = true;
        mName = name;
        return this;
    }
//上面的源码仅仅做了一个标记
```

除此之外因为我们要使用 FragmentManger 用的是 FragmentActivity ,因此 FragmentActivity 的 onBackPress 方法必定重新覆写了。打开看一下 , 发现确实如此。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
/**
     * Take care of popping the fragment back stack or finishing the activit
     * as appropriate.
     */
    public void onBackPressed() {
        if (!mFragments.popBackStackImmediate()) {
            finish();
        }
    }
  //mFragments 的原型是 FragmentManagerImpl，看看这个方法都干嘛了
```

```
  @Override
    public boolean popBackStackImmediate() {
        checkStateLoss();
        executePendingTransactions();
        return popBackStackState(mActivity.mHandler, null, -1, 0);
        }

//看看 popBackStackState 方法都干了啥，其实通过名称也能大概了解 只给几个片段吧，
了
                while (index >= 0) {
                    //从后退栈中取出当前记录对象
                   BackStackRecord bss = mBackStack.get(index);
                   if (name != null && name.equals(bss.getName())) {
                       break;
                   }
                   if (id >= 0 && id == bss.mIndex) {
                       break;
                   }
                   index--;
               }
```

5.Fragment 在你们项目中的使用

Fragment 是 android3.0 以后引入的的概念，做局部内容更新更方便，原

来为了到达这一点要把多个布局放到一个 activity 里面，现在可以用多 Fragment 来代替，只有在需要的时候才加载 Fragment，提高性能。

Fragment 的好处：

（1）Fragment 可以使你能够将 activity 分离成多个可重用的组件，每个都有它自己的生命周期和 UI。

（2）Fragment 可以轻松得创建动态灵活的 UI 设计，可以适应于不同的屏幕尺寸。从手机到平板电脑。

（3）Fragment 是一个独立的模块,紧紧地与 activity 绑定在一起。可以运行中动态地移除、加入、交换等。

（4）Fragment 提供一个新的方式让你在不同的安卓设备上统一你的 UI。

（5）Fragment 解决 Activity 间的切换不流畅，轻量切换。

（6）Fragment 替代 TabActivity 做导航，性能更好。

（7）Fragment 在 4.2.版本中新增嵌套 fragment 使用方法，能够生成更好的界面效果。

6.如何切换 fragement,不重新实例化

翻看了 Android 官方 Doc，和一些组件的源代码，发现 replace()这个方法只是在上一个 Fragment 不再需要时采用的简便方法.

正确的切换方式是 add() ,切换时 hide() ,add()另一个 Fragment ;再次切换时，只需 hide()当前，show()另一个。

这样就能做到多个 Fragment 切换不重新实例化：

```
public void switchContent(Fragment from, Fragment to) {


if (mContent != to) {


            mContent = to;


FragmentTransaction transaction
=mFragmentMan.beginTransaction().setCustomAnimations(android.R.anim.fad
e_in, R.anim.slide_out);


          if (!to.isAdded()) { // 先判断是否被 add 过
           transaction.hide(from).add(R.id.content_frame, to).commit(); // 隐藏
当前的 fragment，add 下一个到 Activity 中
      } else {
                transaction.hide(from).show(to).commit(); // 隐藏当前的
fragment，显示下一个
            }
        }
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
}
```

第二章：Android 高级

一、Android 性能优化（2019.7.5 更新）

1、如何对 Android 应用进行性能分析（★★★★）

一款 App 流畅与否安装在自己的真机里，玩几天就能有个大概的感性认识。不过通过专业的分析工具可以使我们更好的分析我们的应用。而在实际开发中，我们解决完当前应用所有 bug 后，就会开始考虑到新能的优化。

如果不考虑使用其他第三方性能分析工具的话，我们可以直接使用 ddms 中的工具，其实 ddms 工具已经非常的强大了。ddms 中有 traceview、heap、allocation tracker 等工具都可以帮助我们分析应用的方法执行时间效率和内存使用情况。

- traceview

（一）TraceView 简介

Traceview 是 Android 平台特有的数据采集和分析工具，它主要用于分析 Android 中应用程序的 hotspot（瓶颈）。Traceview 本身只是一个数据分析工具，而数据的采集则需要使用 Android SDK 中的 Debug 类或者利用 DDMS 工具。

二者的用法如下：

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

开发者在一些关键代码段开始前调用 Android SDK 中 Debug 类的 startMethodTracing 函数，并在关键代码段结束前调用 stopMethodTracing 函数。这两个函数运行过程中将采集运行时间内该应用所有线程（注意，只能是 Java 线程）的函数执行情况，并将采集数据保存到/mnt/sdcard/下的一个文件中。开发者然后需要利用 SDK 中的 Traceview 工具来分析这些数据。

借助 Android SDK 中的 DDMS 工具。DDMS 可采集系统中某个正在运行的进程的函数调用信息。对开发者而言，此方法适用于没有目标应用源代码的情况。DDMS 工具中 Traceview 的使用如下图所示。

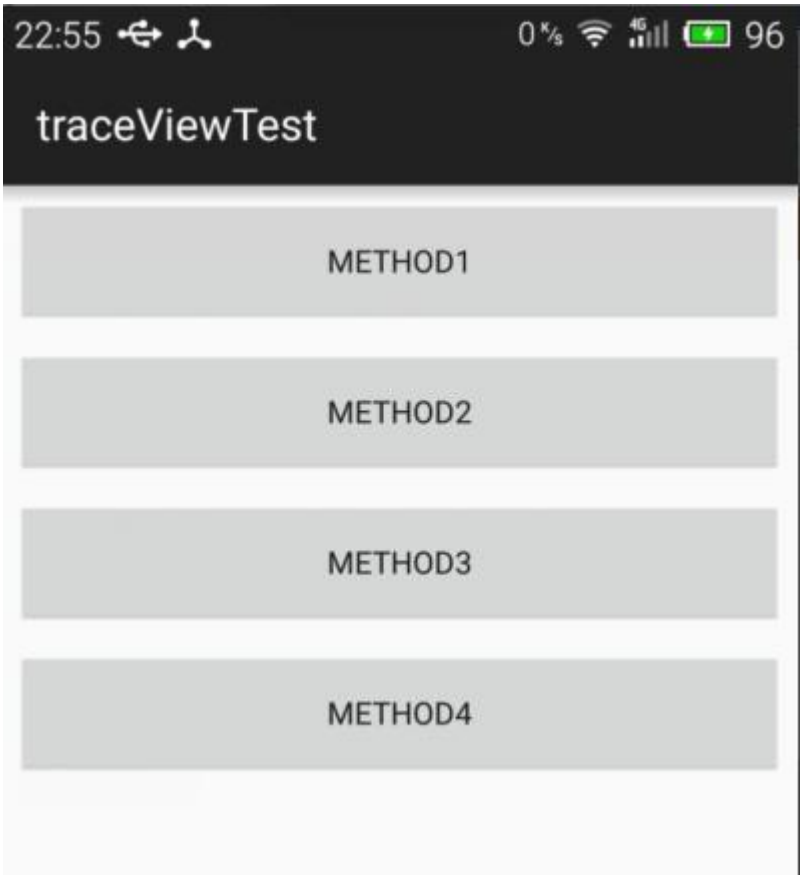
点击上图中所示按钮即可以采集目标进程的数据。当停止采集时，DDMS 会自动触发 Traceview 工具来浏览采集数据。

下面，我们通过一个示例程序介绍 Traceview 的使用。

实例程序如下图所示：界面有 4 个按钮，对应四个方法。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队



点击不同的方法会进行不同的耗时操作。

```
public class MainActivity extends ActionBarActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void method1(View view) {
        int result = jisuan();
        System.out.println(result);
    }
```

```
    private int jisuan() {
        for (int i = 0; i < 10000; i++) {
            System.out.println(i);
        }
        return 1;
    }

    public void method2(View view) {
        SystemClock.sleep(2000);
    }

    public void method3(View view) {
        int sum = 0;
        for (int i = 0; i < 1000; i++) {
            sum += i;
        }
        System.out.println("sum=" + sum);
    }

    public void method4(View view) {
        Toast.makeText(this, "" + new Date(), 0).show();
    }
}
```

我们分别点击按钮一次，要求找出最耗时的方法。点击前通过 DDMS 启动 Start Method Profiling 按钮。

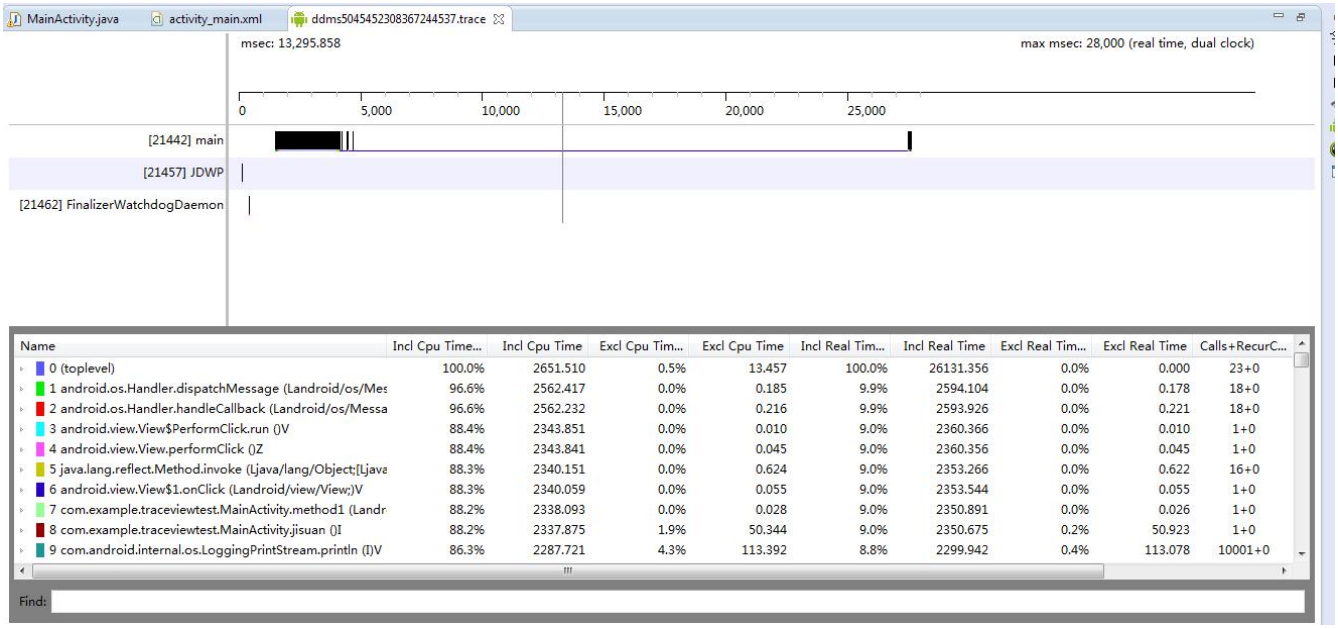
然后依次点击 4 个按钮，都执行后再次点击上图中红框中按钮，停止收集数据。

接下来我们开始对数据进行分析。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

当我们停止收集数据的时候会出现如下分析图表。该图表分为 2 大部分，上面分不同的行，每一行代表一个线程的执行耗时情况。main 线程对应行的的内容非常丰富，而其他线程在这段时间内干得工作则要少得多。图表的下半部分是具体的每个方法执行的时间情况。显示方法执行情况的前提是先选中某个线程。



我们主要是分析 main 线程。

上面方法指标参数所代表的意思如下：

列名	描述
Name	该线程运行过程中所调用的函数名
Incl Cpu Time	某函数占用的 CPU 时间，包含内部调用其它函数的 CPU 时间
Excl Cpu Time	某函数占用的 CPU 时间，但不含内部调用其它函数所占用的 CPU 时间
Incl Real Time	某函数运行的真实时间（以毫秒为单位），内含调用其它函数所占用的真实时间

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

Excl Real Time	某函数运行的真实时间（以毫秒为单位），不含调用其它函数所占用的真实时间
Call+Recur Calls/Total	某函数被调用次数以及递归调用占总调用次数的百分比
Cpu Time/Call	某函数调用 CPU 时间与调用次数的比。相当于该函数平均执行时间
Real Time/Call	同 CPU Time/Call 类似，只不过统计单位换成了真实时间

我们为了找到最耗时的操作，那么可以通过点击 Incl Cpu Time，让其按照时间的倒序排列。我点击后效果如下图：

Name	Incl Cpu Time...	Incl Cpu ...	Excl Cpu Tim...	Excl Cpu Time	Incl Real Tim...	Incl Real Time	Excl Real Tim...	Excl Real Time
0 (toplevel)	100.0%	2651.510	0.5%	13.457	100.0%	26131.356	0.0%	0.000
1 android.os.Handler.dispatchMessage (Landroid/os/Mes	96.6%	2562.417	0.0%	0.185	9.9%	2594.104	0.0%	0.178
2 android.os.Handler.handleCallback (Landroid/os/Messa	96.6%	2562.232	0.0%	0.216	9.9%	2593.926	0.0%	0.221
3 android.view.View$PerformClick.run ()V	88.4%	2343.851	0.0%	0.010	9.0%	2360.366	0.0%	0.010
4 android.view.View.performClick ()Z	88.4%	2343.841	0.0%	0.045	9.0%	2360.356	0.0%	0.045
5 java.lang.reflect.Method.invoke (Ljava/lang/Object;[Ljava	88.3%	2340.151	0.0%	0.624	9.0%	2353.266	0.0%	0.622
6 android.view.View$1.onClick (Landroid/view/View;)V	88.3%	2340.059	0.0%	0.055	9.0%	2353.544	0.0%	0.055
7 com.example.traceviewtest.MainActivity.method1 (Landr	88.2%	2338.093	0.0%	0.028	9.0%	2350.891	0.0%	0.026
8 com.example.traceviewtest.MainActivity.jisuan ()I	88.2%	2337.875	1.9%	50.344	9.0%	2350.675	0.2%	50.923
9 com.android.internal.os.LoggingPrintStream.println (I)V	86.3%	2287.721	4.3%	113.392	8.8%	2299.942	0.4%	113.078
10 com.android.internal.os.LoggingPrintStream.flush (Z)V	52.1%	1382.477	8.9%	234.993	5.3%	1388.300	0.9%	235.907
11 java.lang.StringBuilder.append (I)Ljava/lang/StringBuilc	29.9%	791.852	2.6%	69.036	3.1%	798.386	0.3%	69.077
12 java.lang.IntegralToString.appendInt (Ljava/lang/Abstra	27.3%	722.816	2.6%	67.797	2.8%	729.309	0.3%	67.760
13 java.lang.IntegralToString.convertInt (Ljava/lang/Abstra	24.7%	655.084	5.5%	146.132	2.5%	659.603	0.6%	147.898
14 java.lang.StringBuilder.substring (I)Ljava/lang/String;	13.7%	363.810	2.6%	70.150	1.4%	365.069	0.3%	69.746
15 java.lang.ThreadLocal.get ()Ljava/lang/Object;	11.6%	308.482	7.6%	201.578	1.2%	311.870	0.8%	203.362

通过分析发现：method1 最耗时，耗时 2338 毫秒。

那么有了上面的信息我们可以进入我们的 method1 方法查看分析我们的代码了。

heap

（二）heap 简介

heap 工具可以帮助我们检查代码中是否存在会造成内存泄漏的地方。

用 heap 监测应用进程使用内存情况的步骤如下：

1．启动 eclipse 后，切换到 DDMS 透视图，并确认 Devices 视图、Heap 视图都是打开的；

2．点击选中想要监测的进程，比如 system_process 进程；

3．点击选中 Devices 视图界面中最上方一排图标中的“Update Heap”图标；

4．点击 Heap 视图中的“Cause GC”按钮；

5．此时在 Heap 视图中就会看到当前选中的进程的内存使用量的详细情况。

说明：

a. 点击“Cause GC”按钮相当于向虚拟机请求了一次 gc 操作；

b. 当内存使用信息第一次显示以后，无须再不断的点击“Cause GC”，Heap 视图界面会定时刷新，在对应用的不断的操作过程中就可以看到内存使用的变化；

c. 内存使用信息的各项参数根据名称即可知道其意思，在此不再赘述。

如何才能知道我们的程序是否有内存泄漏的可能性呢？

- 这里需要注意一个值：Heap 视图中部有一个 Type 叫做 data object，即数据对象，也就是我们的程序中大量存在的类类型的对象。在 data object 一行中有一列是“Total Size”，其值就是当前进程中所有 Java 数据对象的内存总量，一般情况下，这个值的大小决定了是否会有内存泄漏。可以这样判

断：

- 不断的操作当前应用，同时注意观察 data object 的 Total Size 值；
- 正常情况下 Total Size 值都会稳定在一个有限的范围内，也就是说由于程序中的的代码良好，没有造成对象不被垃圾回收的情况，所以说虽然我们不断的操作会不断的生成很多对象，而在虚拟机不断的进行 GC 的过程中，这些对象都被回收了，内存占用量会会落到一个稳定的水平；
- 反之如果代码中存在没有释放对象引用的情况 则 data object 的 Total Size 值在每次 GC 后不会有明显的回落，随着操作次数的增多 Total Size 的值会越来越大，直到到达一个上限后导致进程被 kill 掉。
- 此处以 system_process 进程为例，在我的测试环境中 system_process 进程所占用的内存的 data object 的 Total Size 正常情况下会稳定在 2.2~2.8 之间，而当其值超过 3.55 后进程就会被 kill。

总之，使用 DDMS 的 Heap 视图工具可以很方便的确认我们的程序是否存在内存泄漏的可能性。

allocation tracker

（三）allocation tracker 简介

allocation tracker 是内存分配跟踪工具

步骤：

运行 DDMS，只需简单的选择应用进程并单击 Allocation tracker 标签，就会

打开一个新的窗口，单击“Start Tracing”按钮；

然后，让应用运行你想分析的代码。运行完毕后，单击“Get Allocations”按钮，一个已分配对象的列表就会出现第一个表格中。

单击第一个表格中的任何一项，在表格二中就会出现导致该内存分配的栈跟踪信息。通过 allocation tracker，不仅知道分配了哪类对象，还可以知道在哪个线程、哪个类、哪个文件的哪一行。

2、什么情况下会导致内存泄露（★★）

Android 的虚拟机是基于寄存器的 Dalvik，它的最大堆大小一般是 16M，有的机器为 24M。因此我们所能利用的内存空间是有限的。如果我们的内存占用超过了一定的水平就会出现 OutOfMemory 的错误。

内存溢出的几点原因：

（1）资源释放问题

程序代码的问题，长期保持某些资源，如 Context、Cursor、IO 流的引用，资源得不到释放造成内存泄露。

（2）对象内存过大问题

保存了多个耗用内存过大的对象（如 Bitmap、XML 文件），造成内存超出限制。

（3）static 关键字的使用问题

static 是 Java 中的一个关键字，当用它来修饰成员变量时，那么该变量就属于该类，而不是该类的实例。所以用 static 修饰的变量，它的生命周期是很长的 如果用它来引用一些资源耗费过多的实例(Context的情况最多)，这时就要谨慎对待了。

```
public class ClassName {
    private static Context mContext;
    //省略
}
```

以上的代码是很危险的，如果将 Activity 赋值到 mContext 的话。那么即使该 Activity 已经 onDestroy，但是由于仍有对象保存它的引用，因此该 Activity 依然不会被释放。

我们举 Android 文档中的一个例子。

```
    private static Drawable sBackground;
    @Override
    protected void onCreate(Bundle state) {
    super.onCreate(state);
    TextView label = new TextView(this);  //getApplicationContext
    label.setText("Leaks are bad");
    if (sBackground == null) {
        sBackground = getDrawable(R.drawable.large_bitmap);
    }
        label.setBackgroundDrawable(sBackground);
        setContentView(label);
}
```

sBackground 是一个静态的变量，但是我们发现，我们并没有显式的保存 Contex 的引用，但是，当 Drawable 与 View 连接之后，Drawable 就将 View 设置为一个回调，由于 View 中是包含 Context 的引用的，所以，实际上我们依然保存了 Context 的引用。这个引用链如下：

Drawable->TextView->Context

所以，最终该 Context 也没有得到释放，发生了内存泄露。

- 针对 static 的解决方案

1) 应该尽量避免 static 成员变量引用资源耗费过多的实例，比如 Context。
2) Context 尽量使用 ApplicationContext，因为 Application 的 Context 的生命周期比较长，引用它不会出现内存泄露的问题。
3) 使用 WeakReference 代替强引用。比如可以使用 WeakReference<Context> mContextRef;

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

1. 线程导致内存溢出

线程产生内存泄露的主要原因在于线程生命周期的不可控。我们来考虑下面一段代码。

```
public class MyActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        new MyThread().start();
    }
    private class MyThread extends Thread{
@Override
        public void run() {
            super.run();
            //do somthing  while(true)
        }
    }
}
```

这段代码很平常也很简单，是我们经常使用的形式。我们思考一个问题：假设 MyThread 的 run 函数是一个很费时的操作，当我们开启该线程后，将设备的横屏变为了竖屏，一 般情况下当屏幕转换时会重新创建 Activity，按照我们的想法，老的 Activity 应该会被销毁才对，然而事实上并非如此。

由于我们的线程是 Activity 的内部类，所以 MyThread 中保存了 Activity 的一个引用，当 MyThread 的 run 函数没有结束时，MyThread 是不会被销毁的 ,因此它所引用的老的 Activity 也不会被销毁 ,因此就出现了内存泄露的问题。

有些人喜欢用 Android 提供的 AsyncTask，但事实上 AsyncTask 的问题更

加严重，Thread 只有在 run 函数不结束时才出现这种内存泄露问题，然而 AsyncTask 内部的实现机制是运用了 ThreadPoolExcutor,该类产生的 Thread 对象的生命周期是不确定的，是应用程序无法控制的，因此如果 AsyncTask 作为 Activity 的内部类，就更容易出现内存泄露的问题。

针对这种线程导致的内存泄露问题的解决方案：

（一） 将线程的内部类，改为静态内部类（因为非静态内部类拥有外部类对象的强引用，而静态类则不拥有）。

（二） 在线程内部采用弱引用保存 Context 引用。

3、如何避免 OOM 异常（★★★★）

OOM 内存溢出，想要避免 OOM 异常首先我们要知道什么情况下会导致 OOM 异常。

（1）图片过大导致 OOM

Android 中用 bitmap 时很容易内存溢出，比如报如下错误：Java.lang.OutOfMemoryError : bitmap size exceeds VM budget。

解决方法：

方法 1： 等比例缩小图片

```
        BitmapFactory.Options options = new BitmapFactory.Options();
        options.inSampleSize = 2;
        //Options 只保存图片尺寸大小，不保存图片到内存
        BitmapFactory.Options opts = new BitmapFactory.Options();
         opts.inSampleSize = 2;
         Bitmap bmp = null;
        bmp = BitmapFactory.decodeResource(getResources(),
mImageIds[position],opts);
          //回收
          bmp.recycle();//
```

以上代码可以优化内存溢出，但它只是改变图片大小，并不能彻底解决内存溢出。

方法 2：对图片采用软引用，及时地进行 recyle()操作

```
SoftReference<Bitmap> bitmap = new SoftReference<Bitmap>(pBitmap);
if(bitmap != null){
    if(bitmap.get() != null && !bitmap.get().isRecycled()){
        bitmap.get().recycle();
        bitmap = null;
    }
}
```

方法 3：使用加载图片框架处理图片，如专业处理加载图片的 ImageLoader 图片加载框架。还有我们学的 XUtils 的 BitMapUtils 来做处理。

（2）界面切换导致 OOM

一般情况下，开发中都会禁止横屏的。因为如果是来回切换话，activity 的生命周期会重新销毁然后创建。

有时候我们会发现这样的问题，横竖屏切换 N 次后 OOM 了。

这种问题没有固定的解决方法，但是我们可以从以下几个方面下手分析。

1、看看页面布局当中有没有大的图片，比如背景图之类的。

去除 xml 中相关设置，改在程序中设置背景图（放在 onCreate()方法中）：

```
Drawable drawable = getResources().getDrawable(R.drawable.id);
ImageView imageView = new ImageView(this);
imageView.setBackgroundDrawable(drawable);
```

在 Activity destory 时注意，drawable.setCallback(**null**); 防止 Activity 得不到及时的释放。

2、跟上面方法相似，直接把 xml 配置文件加载成 view 再放到一个容器里，然后直接调用 this.setContentView(View view);方法，避免 xml 的重复加载。

3、 在页面切换时尽可能少地重复使用一些代码

比如：重复调用数据库，反复使用某些对象等等.....

（3）查询数据库没有关闭游标

程序中经常会进行查询数据库的操作，但是经常会有使用完毕 Cursor 后没有关闭的情况。如果我们的查询结果集比较小，对内存的消耗不容易被发现，只有在常时间大量操作的情况下才会出现内存问题，这样就会给以后的测试和问题排查带来困难和风险。

（4）构造 Adapter 时，没有使用缓存的 convertView

在使用 ListView 的时候通常会使用 Adapter，那么我们应该尽可能的使用 ConvertView。

为什么要使用 convertView?

当 convertView 为空时，用 setTag()方法为每个 View 绑定一个存放控件的 ViewHolder 对象。当 convertView 不为空，重复利用已经创建的 view 的时候，使用 getTag()方法获取绑定的 ViewHolder 对象，这样就避免了 findViewById 对控件的层层查询，而是快速定位到控件。

（5）Bitmap 对象不再使用时调用 recycle()释放内存

有时我们会手工的操作 Bitmap 对象，如果一个 Bitmap 对象比较占内存，当它不再被使用的时候，可以调用 Bitmap.recycle()方法回收此对象的像素所占用的内存，但这不是必须的，视情况而定。

（6）其他

Android 应用程序中最典型的需要注意释放资源的情况是在 Activity 的生命周期中，在 onPause()、onStop()、 onDestroy()方法中需要适当的释放资源的情况。使用广播没有注销也会产生 OOM。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

4、Android 中如何捕获未捕获的异常(2019.5.5)(★★★)

（一）UncaughtExceptionHandler

1、自定义一个 Application，比如叫 MyApplication 继承 Application 实现 UncaughtExceptionHandler。

2、覆写 UncaughtExceptionHandler 的 onCreate 和 uncaughtException 方法。

```
@Override
    public void onCreate() {
        super.onCreate();
        Thread.setDefaultUncaughtExceptionHandler(this);
    }

    @Override
    public void uncaughtException(final Thread thread, final Throwable ex) {
        new Thread(new Runnable() {

            @Override
            public void run() {
                Looper.prepare();
                System.out.println(Thread.currentThread());
                Toast.makeText(getApplicationContext(), "thread="+thread.getId(
ex="+ex.toString(), 1).show();
                Looper.loop();
            }
        }).start();
        SystemClock.sleep(3000);
        android.os.Process.killProcess(android.os.Process.myPid());
    }

}
```

注意：上面的代码只是简单的将异常打印出来。

在 onCreate 方法中我们给 Thread 类设置默认异常处理 handler，如果这句代码不执行则一切都是白搭。

在 uncaughtException 方法中我们必须新开辟个线程进行我们异常的收集工作，然后将系统给杀死。

3、在 AndroidManifest 中配置该 Application

```
<application
      android:name="com.example.uncatchexception.MyApplication"
```

（二）Bug 收集工具 Crashlytics

Crashlytics 是专门为移动应用开发者提供的保存和分析应用崩溃的工具。

国内主要使用的是友盟做数据统计。

Crashlytics 的好处：

1．Crashlytics 不会漏掉任何应用崩溃信息。

2．Crashlytics 可以象 Bug 管理工具那样，管理这些崩溃日志。

3．Crashlytics 可以每天和每周将崩溃信息汇总发到你的邮箱，所有信息一目了然。

使用步骤：

1.注册需要审核通过才能使用，国内同类产品顶多发个邮箱激活链接；

2.支持 Eclipse、Intellij IDEA 和 Android Studio 等三大 IDE；

3.Eclipse 插件是 iOS 主题风格 UI，跟其他 plugin 在一起简直是鹤立鸡群；

4.只要登录帐号并选择项目，会自动导入 jar 包并生成一个序列号，然后在 AndroidManifest.xml 和启动 Activity 的入口添加初始化代码，可以说是一键式操作，当然要使用除错误统计外的其他功能还是得自己添加代码；

5.不像友盟等国内同类产品，将固定的序列号直接写入 xml 文件，而是动态自动生成的；当然这个存放序列号的 xml 文件也是不能修改和提交到版本控制系统的；

6.后台可以设置邮件提醒，当然这个最好不要开启，Android 开发那数量惊人、千奇百怪的错误信息你懂的。

7.不仅能统计到 UncaughtException 这种未捕获的 Crash 异常信息，只要在 try/catch 代码块的 catch 中添加一行代码就能统计到任何异常；

```
try{    myMethodThatThrows(); }catch(Exception
e){    Crashlytics.logException(e);    //handle your exception here!    }
```

8.相当详细的错误信息，不仅仅是简单的打印 StackTrace 信息；并且能看到最近一次 crash 的机器可用内存等信息，而不仅仅是简单统计机型和版本号。

使用连接：http://blog.csdn.net/smking/article/details/39320695

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

5、 ANR 是什么？怎样避免和解决 ANR（★★★★★）

在 Android 上，如果你的应用程序有一段时间响应不够灵敏，系统会向用户显示一个对话框，这个对话框称作应用程序无响应（ANR：Application Not Responding）对话框。用户可以选择让程序继续运行，但是，他们在使用你的应用程序时，并不希望每次都要处理这个对话框。因此，在程序里对响应性能的设计很重要，这样，系统不会显示 ANR 给用户。

Activity 5 秒　broadcast10 秒

耗时的操作　worker thread 里面完成，handler message…AsynTask，intentservice.等…

ANR:Application Not Responding，即应用无响应

ANR 一般有三种类型：

1：KeyDispatchTimeout(5 seconds) --主要类型

按键或触摸事件在特定时间内无响应

2：BroadcastTimeout(10 seconds)

BroadcastReceiver 在特定时间内无法处理完成

3：ServiceTimeout(20 seconds) --小概率类型

Service 在特定的时间内无法处理完成

超时的原因一般有两种：

(1)当前的事件没有机会得到处理（UI 线程正在处理前一个事件没有及时完成或者 looper 被某种原因阻塞住）

(2)当前的事件正在处理，但没有及时完成

UI 线程尽量只做跟 UI 相关的工作，耗时的工作（数据库操作，I/O，连接网络或者其他可能阻碍 UI 线程的操作）放入单独的线程处理，尽量用 Handler 来处理 UI thread 和 thread 之间的交互。

UI 线程主要包括如下：

Activity:onCreate(), onResume(), onDestroy(), onKeyDown(), onClick()

AsyncTask: onPreExecute(), onProgressUpdate(), onPostExecute(), onCancel()

Mainthread handler: handleMessage(), post(runnable r)

查找 ANR 的方式： 1. 导出/data/data/anr/traces.txt，找出函数和调用过程，分析代码 2. 通过性能 LOG 人肉查找

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

6、Android 线程间通信有哪几种方式（★★★）

- 共享内存（变量）；
- 文件，数据库；
- Handler；
- Java 里的 wait()，notify()，notifyAll()

7、Devik 进程，linux 进程，线程的区别（★）

Dalvik 虚拟机运行在 Linux 操作系统之上。Linux 操作系统并没有纯粹的线程概念，只要两个进程共享一个地址空间，那么就可以认为它们是同一个进程的两个线程。Linux 系统提供了两个 fork 和 clone 调用，其中，前者是用来创建进程的，而后者是用来创建线程的。

一般来说，虚拟机的进程和线程都是和目标机器本地操作系统的进程和线程一一对应的，这样的好处是可以使本地操作系统来调度进程和线程。

每个 Android 应用程序进程都有一个 Dalvik 虚拟机实例。这样做得好处是 Android 应用程序进程之间不会互相影响，也就是说，一个 Android 应用程序进程的意外终止，不会影响到其他的应用程序进程的正常运行。

- 每个 Android 应用程序进程都是由一种称为 Zygote 的进程 fork 出来的。Zygote 进程是由 init 进程启动起来的，也就是在系统启动的时候启动的。

Zygnote 进程在启动的时候，会创建一个虚拟机实例，并且在这个虚拟机实例将所有的 Java 核心库都加载起来。每当 Zygote 进程需要创建一个 Android 应用程序进程的时候，它就通过复制自身来实现，也就是通过 fork 系统调用来实现。这些被 fork 出来的 Android 应用程序进程，一方面是复制了 Zygote 进程中的虚拟机实例，另外一方面是与 Zygote 进程共享了同一套 Java 核心库。这样不仅 Android 程序进程的创建很快，而且所有的应用程序都共享同一套 Java 核心库而节省了内存空间。

8、描述一下 android 的系统架构？

1. android 系统架构分从下往上为 linux 内核层、运行库、应用程序框架层、和应用程序层。
2. linuxkernel：负责硬件的驱动程序、网络、电源、系统安全以及内存管理等功能。
3. libraries 和 androidruntime：libraries：即 c/c++函数库部分，大多数都是开放源代码的函数库，例如 webkit，该函数库负责 android 网页浏览器的运行，例如标准的 c 函数库 libc、openssl、sqlite 等，当然也包括支持游戏开发 2dsgl 和 3dopengles，在多媒体方面有 mediaframework 框架来支持各种影音和图形文件的播放与显示，例如 mpeg4、h.264、mp3、 aac、amr、jpg 和 png 等众多的多媒体文件格式。android 的 runtime 负责解释和执行生成的 dalvik 格式的字节码。

4. applicationframework（应用软件架构），java 应用程序开发人员主要是使用该层封装好的 api 进行快速开发。
5. applications:该层是 java 的应用程序层，android 内置的 googlemaps、e-mail、即时通信工具、浏览器、mp3 播放 器等处于该层，java 开发人员开发的程序也处于该层，而且和内置的应用程序具有平等的位置，可以调用内置的应用程序，也可以替换内置的应用程序。

9、android 应用对内存是如何限制的?我们应该如何合理使用内存？（2019.01.24）（★★★★）

如何限制的？

Android 应用的开发语言为 Java，每个应用最大可使用的堆内存受到 Android 系统的限制

•Android 每一个应用的堆内存大小有限

•通常的情况为 16M-48M

•通过 ActivityManager 的 getMemoryClass()来查询可用堆内存限制

•3.0(HoneyComb)以上的版本可以通过 largeHeap=“true”来申请更多的堆内存

•NexueHeap 512

•如果试图申请的内存大于当前余下的堆内存就会引发 OutOfMemoryError()

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

•应用程序由于各方面的限制，需要注意减少内存占用，避免出现内存泄漏。

获取这个代码：

```java
mActivityManager = (ActivityManager) this.getSystemService(Context.ACTIVITY_SERVICE);
 mMaxMemory = mActivityManager.getMemoryClass();
```

如何合理使用内存？

1、注意资源回收，像数据库，输入输出流，定位操作这样的对象，要在使用完及时关闭流。

2、少使用静态变量，因为系统将静态变量的优先级设定的很高，会最后回收。所以可能因为静态变量导致该回收的没有回收。而回收了不该回收的内存。

3、注意大图片的缩放，如果载入的图片很大，要先经过自己程序的处理，降低分辨率等。最好设置多种分辨率格式的图片，以减少内存消耗。

4、动态注册监听，把一些只有显示的时候才使用到的监听放进程序内部，而不是放在 manifesat 中去。

5、减少使用动画，或者适当减少动画的帧数。

6、注意自己的程序逻辑，在该关闭自己程序的控件的时候，主动关闭，不要交给系统去决定。（这个要自己把握好，也不是说都自己搞定，只有那些自己确定需要关闭的对象，自己将其关闭。）

10、简述 android 应用程序结构是哪些？（2019.01.24）（★★★★）

Android 应用程序结构也就是讲我们的工程结构：

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

src 目录是源代码目录，所有允许用户修改的 java 文件和用户自己添加的 java 文件都保存在这个目录中

gen 目录是 1.5 版本新增的目录，用来保存 ADT 自动生成的 java 文件，例如 R.java 或 AIDL 文件

注意：R．java 文件（非常重要）

a) R.java 文件是 ADT 自动生成的文件 ,包含对 drawable、layout 和 values 目录内的资源的引用指针，Android 程序能够直接通过 R 类引用目录中的资源

b) R.java 文件不能手工修改，如果向资源目录中增加或删除了资源文件，则需要在工程名称上右击，选择 Refresh 来更新 R.java 文件中的代码

c) R 类包含的几个内部类，分别与资源类型相对应，资源 ID 便保存在这些内部类中，例如子类 drawable 表示图像资源，内部的静态变量 icon 表示资源名称，其资源 ID 为 0x7f020000。一般情况下，资源名称与资源文件名相同

android.jar 文件是 Android 程序所能引用的函数库文件，Android 通过平台所支持 API 都包含在这个文件中

assets 目录用来存放原始格式的文件，例如音频文件、视频文件等二进制格式文件。此目录中的资源不能被 R.java 文件索引。，所以只能以资截流的形式读取。一般情况下为空

layout 目录用来存放我们为每个界面写的布局文件

Strings.xml 文件是程序中的一些字符串的引用

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

AndroidManifest.xml 是 XML 格式的 Android 程序声明文件，包含了 Android 系统运行 Android 程序前所必须掌握的重要信息，这些信息包含应用程序名称、图标、包名称、模块组成、授权和 SDK 最低版本等 ,而且每个 Android 程序必须在根目录下包含一个 AndroidManifest.xml 文件

注：AndroidMainfest.xml 文件：

1) AndroidManifest.xml 文件的根元素是 manifest，包含了 xmlns:android、package、android:versionCode 和 android:versionName 共 4 个属性

2) xmlns:android 定义了 Android 的命名空间，值为 http://schemas.android.com/apk/res/android

3) package 定义了应用程序的包名称

4) android:versionCode 定义了应用程序的版本号，是一个整数值，数值越大说明版本越新，但仅在程序内部使用，并不提供给应用程序的使用者

5) android:versionName 定义了应用程序的版本名称，是一个字符串，仅限于为用户提供一个版本标识

6) manifest 元素仅能包含一个 application 元素，application 元素中能够声明 Android 程序中最重要的四个组成部分，包括 Activity、Service、BroadcastReceiver 和 ContentProvider，所定义的属性将影响所有组成部分

7) android:icon 定义了 Android 应用程序的图标 ,其中@drawable/icon 是一种资源引用方式，表示资源类型是图像，资源名称为 icon，对应的资源文件为 res/drawable 目录下的 icon.png

8) android:label 则定义了 Android 应用程序的标签名称

default.properties 文件记录 Android 工程的相关设置 ,该文件不能手动修改，需右键单击工程名称，选择“Properties”进行修改

11、请解释下 Android 程序运行时权限与文件系统权限的区别？（2019.01.24）（★★★）

apk 程序是运行在虚拟机上的,对应的是 Android 独特的权限机制，只有体现到文件系统上时才使用 linux 的权限设置。

（一）linux 文件系统上的权限

-rwxr-x--x system system 4156 2010-04-30 16:13 test.apk

代表的是相应的用户/用户组及其他人对此文件的访问权限，与此文件运行起来具有的权限完全不相关。比如上面的例子只能说明 system 用户拥有对此文件的读写执行权限；system 组的用户对此文件拥有读、执行权限；其他人对此文件只具有执行权限。而 test.apk 运行起来后可以干哪些事情，跟这个就不相关了。千万不要看 apk 文件系统上属于 system/system 用户及用户组，或者 root/root 用户及用户组，就认为 apk 具有 system 或 root 权限

（二）Android 的权限规则

（1)Android 中的 apk 必须签名

（2）基于 UserID 的进程级别的安全机制

（3）默认 apk 生成的数据对外是不可见的

（4）AndroidManifest.xml 中的显式权限声明

12、Framework 工作方式及原理，Activity 是如何生成一个 view 的，机制是什么？（2019.01.24）（★★）

所有的框架都是基于反射 和 配置文件（manifest）的。

普通的情况:

Activity 创建一个 view 是通过 ondraw 画出来的, 画这个 view 之前呢,还会调用 onmeasure 方法来计算显示的大小.

特殊情况：

Surfaceview 是直接操作硬件的，因为 或者视频播放对帧数有要求，onDraw 效率太低，不够使，Surfaceview 直接把数据写到显存。

13、多线程间通信和多进程之间通信有什么不同 ,分别怎么实现？（2019.01.24）（★★★★★）

一、进程间的通信方式

管道(pipe)：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

有名管道（namedpipe）： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

信号量(semophore）： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

消息队列(messagequeue）： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

信号（sinal）： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

共享内存(shared memory）：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

套接字(socket）： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

二、线程间的通信方式

锁机制：包括互斥锁、条件变量、读写锁

*互斥锁提供了以排他方式防止数据结构被并发修改的方法。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

*读写锁允许多个线程同时读共享数据，而对写操作是互斥的。

*条件变量可以以原子的方式阻塞进程，直到某个特定条件为真为止。对条件的测试是在互斥锁的保护下进行的。条件变量始终与互斥锁一起使用。

信号量机制(Semaphore)：包括无名线程信号量和命名线程信号量

信号机制(Signal)：类似进程间的信号处理

线程间的通信目的主要是用于线程同步，所以线程没有像进程通信中的用于数据交换的通信机制。

二、 Android 屏幕适配（更新 2019.5.5）

1、屏幕适配方式都有哪些（★★★★★）

1.1 适配方式之 dp

名词解释：

◆ 分辨率：eg：480*800,1280*720。表示物理屏幕区域内像素点的总和。(切记：跟屏幕适配没有任何关系)

因为我们既可以把 1280*720 的分辨率做到 4.0 的手机上面。我也可以把 1280*720 的分辨率做到 5.0 英寸的手机上面，如果分辨率相同，手机屏幕越小清晰。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

- px(pix)：像素，就是屏幕中最小的一个显示单元
- dpi（像素密度）：即每英寸屏幕所拥有的像素数，像素密度越大，显示画面细节就越丰富。

计算公式：像素密度=√{（长度像素数^2+宽度像素数^2）}/ 屏幕尺寸

注：屏幕尺寸单位为英寸 例：分辨率为 1280*720 屏幕宽度为 6 英寸 计算所得像素密度约等于 245，屏幕尺寸指屏幕对角线的长度。

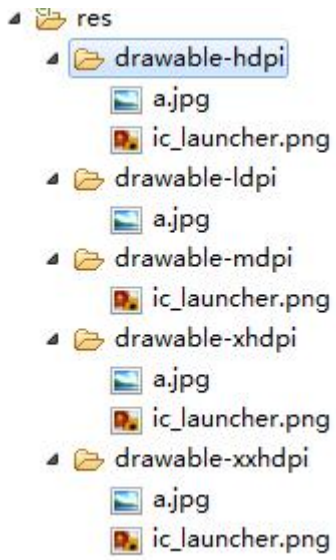
在 Android 手机中 dpi 分类：

ldpi	Resources for low-density (*ldpi*) screens (~120dpi).
mdpi	Resources for medium-density (*mdpi*) screens (~160dpi). (This is the baseline density.)
hdpi	Resources for high-density (*hdpi*) screens (~240dpi).
xhdpi	Resources for extra high-density (*xhdpi*) screens (~320dpi).

在我们的 Android 工程目录中有如下 drawable-*dpi 目录，这些目录是用来适配不同分辨率手机的。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队



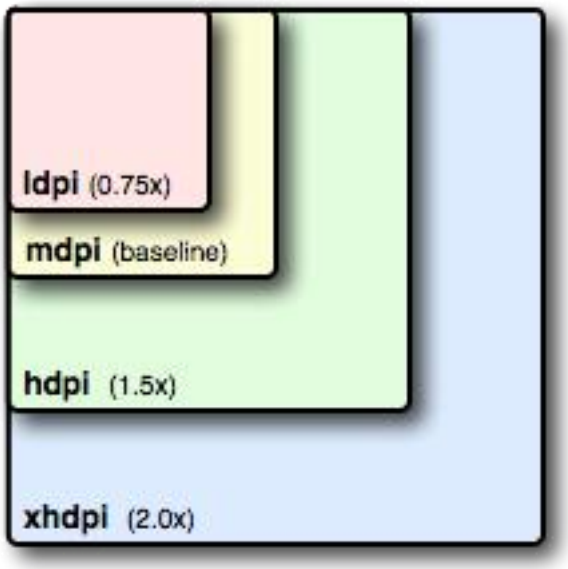
Android 应用在查找图片资源时会根据其分辨率自动从不同的文件目录下查找（这本身就是 Android 系统的适配策略），如果在低分辨的文件目录中比如 drawable-mdpi 中没有图片资源，其他目录中都有，当我们将该应用部署到 mdpi 分辨率的手机上时，那么该应用会查找分辨率较高目录下的资源文件，如果较高分辨率目录下也没有资源则只好找较低目录中的资源了。

常见手机屏幕像素及对应分别率级别：

- ldpi　320*240
- mdpi　480*320
- hdpi　800*480
- xhdpi　1280*720
- xxhdpi　1920*1080

dp 和 px 之间的简单换算关系：

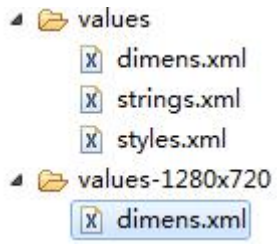
- ldpi 的手机 1dp=0.75px
- mdpi 的手机 1dp=1.0px
- hdpi 的手机 1dp=1.5px
- xhdpi 的手机 1dp=2.0px
- xxhdpi 的手机 1dp=3.0px



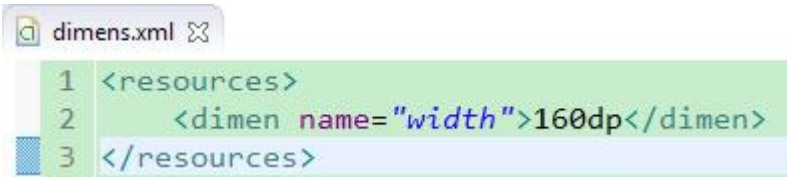
Tips：根据上面的描述我们得出如下结论，对于 mdpi 的手机，我们的布局通过 dp 单位可以达到适配效果。

1.2 适配方式之 dimens

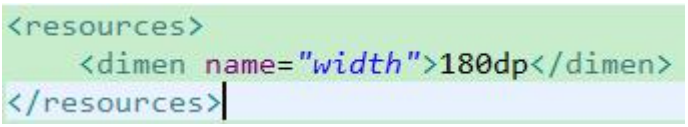
跟 drawable 目录类似的，在 Android 工程的 res 目录下有 values 目录，这个是默认的目录，同时为了适配不同尺寸手机我们可以创建一个 values-1280x720 的文件夹，同时将 dimens.xml 文件拷贝到该目录下。



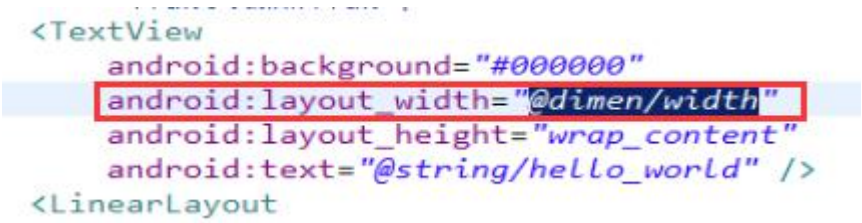
在 dimens.xml 中定义一个尺寸，如下图所示。



在 values-1280x720 目录中的 dimens.xml 中定义同样的尺寸名称，但是使用不同的尺寸，如下图所示。



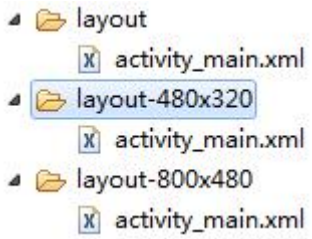
当我们在布局文件中使用长或者宽度单位时，比如下图所示，应该使用@dimen/width 来灵活的定义宽度。



Tips：在 values-1280x720 中，中间的是大写字母 X 的小写形式 x，而不是加减乘除的乘号。如果我们在 values-1280x720 中放置了 dimens 常量，一定记得也将该常量的对应值在 values 目录下的 dimens.xml 中放一份，因为该文件是默认配置，当用户的手机不是 1280*720 的情况下系统应用使用的是默认 values 目录中的 dimens.xml。

1.3 适配方式之 layout

跟 values 一样，在 Android 工程目录中 layout 目录也支持类似 values 目录一样的适配，在 layout 中我们可以针对不同手机的分辨率制定不同的布局，如下图所示。



1.4 适配方式之 java 代码适配

为了演示用 java 代码控制适配的效果，因此假设有这样的需求，让一个 TextView 控件的宽和高分别为屏幕的宽和高的一半。

我们新创建一个 Android 工程，修改 main_activity.xml，布局文件清单如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
```

```
    <!-- 当前控件宽高为屏幕宽度的各 50% -->
    <TextView
        android:id="@+id/tv"
        android:background="#000000"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</RelativeLayout>
```

在 MainActivity.java 类中完成用 java 代码控制 TextView 的布局效果，其代码清单如下：

```
public class MainActivity extends Activity {

    private static final String tag = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //去掉 title
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);
        //获取 TextView 控件
        TextView tv  = (TextView) findViewById(R.id.tv);
        //找到当前控件的夫控件(父控件上给当前的子控件去设定一个规则)
        DisplayMetrics metrics  = new DisplayMetrics();
        //给当前 metrics 去设置当前屏幕信息(宽(像素)高(像素))
        getWindowManager().getDefaultDisplay().getMetrics(metrics);
        //获取屏幕的高度和宽度
        Constant.srceenHeight = metrics.heightPixels;
        Constant.srceenWidth = metrics.widthPixels;
        //日志输出屏幕的高度和宽度
        Log.i(tag, "Constant.srceenHeight = "+Constant.srceenHeight);
        Log.i(tag, "Constant.srceenWidth = "+Constant.srceenWidth);
                //宽高各 50%
        RelativeLayout.LayoutParams layoutParams = new RelativeLayout.LayoutP
                //数学角度上 四舍五入
                (int)(Constant.srceenWidth*0.5+0.5),
                (int)(Constant.srceenHeight*0.5+0.5));
        //给 tv 控件设置布局参数
        tv.setLayoutParams(layoutParams);
    }
}
```

其中 Constant 类是一个常量类 ,很简单 ,只有两个常量用来记录屏幕的宽和高 ,

其代码清单如下：

```
public class Constant {
    public static int srceenHeight;
    public static int srceenWidth;
}
```

1.5 适配方式之 weight 权重适配

在控件中使用属性 android:layout_weight="1"可以起到适配效果 ,但是该属性的使用有如下规则：

只能用在线性控件中，比如 LinearLayout。

竖直方向上使用权重的控件高度必须为 0dp（Google 官方的推荐用法）

水平方向上使用权重的控件宽度必须为 0dp（Google 官方的推荐用法）

1.6 适配方式之百分比适配

2、屏幕适配的处理技巧都有哪些（★★★★）

手机自适应主要分为两种情况 :横屏和竖屏的切换 ,以及分辨率大小的不同。

2.1 横屏和竖屏的切换

1、Android 应用程序支持横竖屏幕的切换，Android 中每次屏幕的切换动会重启 Activity ,所以应该在 Activity 销毁(执行 onPause()方法和 onDestroy()

方法）前保存当前活动的状态；在 Activity 再次创建的时候载入配置，那样，进行中的游戏就不会自动重启了！有的程序适合从竖屏切换到横屏，或者反过来，这个时候怎么办呢？可以在配置 Activity 的地方进行如下的配置 android:screenOrientation="portrait"(landscape 是横向 ,portrait 是纵向)。这样就可以保证是竖屏总是竖屏了。

2、而有的程序是适合横竖屏切换的。如何处理呢？首先要在配置 Activity 的时候进行如下的配置：

android:configChanges="keyboardHidden|orientation" ,另外需要重写 Activity 的 onConfigurationChanged 方法。实现方式如下：

```
@Override
public void onConfigurationChanged(Configuration newConfig){
super.onConfigurationChanged(newConfig);
if(this.getResources().getConfiguration().orientation==Configuration.ORIENTA
LANDSCAPE){
//TODO
   }else
if(this.getResources().getConfiguration().orientation==Configuration.ORIENTA
PORTRAIT){
//TODO
   }
}
```

2.2 分辨率大小不同

对于分辨率问题，官方给的解决办法是创建不同的 layout 文件夹，这就需要对每种分辨率的手机都要写一个布局文件，虽然看似解决了分辨率的问题，但

是如果其中一处或多处有修改了，就要每个布局文件都要做出修改，这样就造成很大的麻烦。那么可以通过以下几种方式解决：

一）使用 layout_weight

目前最为推荐的 Android 多屏幕自适应解决方案。

该属性的作用是决定控件在其父布局中的显示权重，一般用于线性布局中。其值越小，则对应的 layout_width 或 layout_height 的优先级就越高（一般到 100 作用就不太明显了）；一般横向布局中，决定的是 layout_width 的优先级；纵向布局中，决定的是 layout_height 的优先级。

传统的 layout_weight 使用方法是将当前控件的 layout_width 和 layout_height 都设置成 fill_parent,这样就可以把控件的显示比例完全交给 layout_weight；这样使用的话，就出现了 layout_weight 越小，显示比例越大的情况（即权重越大，显示所占的效果越小）。不过对于 2 个控件还好，如果控件过多，且显示比例也不相同的时候，控制起来就比较麻烦了，毕竟反比不是那么好确定的。于是就有了现在最为流行的 0px 设值法。看似让人难以理解的 layout_height=0px 的写法，结合 layout_weight，却可以使控件成正比例显示，轻松解决了当前 Android 开发最为头疼的碎片化问题之一。

二）清单文件配置：【不建议使用这种方式，需要对不同的界面写不同的布局】

需要在 AndroidManifest.xml 文件的<manifest>元素如下添加子元素

<supports-screensandroid:largeScreens="true"

android:normalScreens="true"

android:anyDensity="true"

android:smallScreens="true"

android:xlargeScreens="true">

</supports-screens>

以上是为我们的屏幕设置多分辨率支持（更准确的说是适配大、中、小三种密度）。

Android:anyDensity="true"，这一句对整个的屏幕都起着十分重要的作用，值为 true，我们的应用程序当安装在不同密度的手机上时，程序会分别加载 hdpi,mdpi,ldpi 文件夹中的资源。相反，如果值设置为 false，即使我们在 hdpi,mdpi,ldpi，xdpi 文件夹下拥有同一种资源，那么应用也不会自动地去相应文件夹下寻找资源。而是会在大密度和小密度手机上加载中密度 mdpi 文件中的资源。

有时候会根据需要在代码中动态地设置某个值，可以在代码中为这几种密度分别设置偏移量,但是这种方法最好不要使用，最好的方式是在 xml 文件中不同密度的手机进行分别设置。这里地图的偏移量可以在 values-xpdi，values-hpdi,values-mdpi,values-ldpi 四种文件夹中的 dimens.xml 文件进行设置。

三）、其他：

说明：

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

在不同分辨率的手机模拟器下，控件显示的位置会稍有不同

通过在 layout 中定义的布局设置的参数，使用 dp（dip），会根据不同的屏幕分辨率进行适配

但是在代码中的各个参数值，都是使用的像素（px）为单位的

技巧：

1、尽量使用线性布局，相对布局，如果屏幕放不下了，可以使用 ScrollView（可以上下拖动）

ScrowView 使用的注意：

在不同的屏幕上显示内容不同的情况，其实这个问题我们往往是用滚动视图来解决的，也就是 ScrowView，需要注意的是 ScrowView 中使用 layout_weight 是无效的，既然使用 ScrowView 了，就把它里面的控件的大小都设成固定的吧。

2、指定宽高的时候，采用 dip 的单位，dp 单位动态匹配

3、由于 android 代码中写的单位都是像素，所有需要通过工具类进行转化

4、尽量使用 9-patch 图，可以自动的依据图片上面显示的内容被拉伸和收缩。其中在编辑的时候，灰色区域是被拉伸的，上下两个点控制水平方向的拉伸，左右两点控制垂直方向的拉伸

3、dp 和 px 之间的关系（★★）

dp：是 dip 的简写，指密度无关的像素。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

指一个抽象意义上的像素，程序用它来定义界面元素。一个与密度无关的，在逻辑尺寸上，与一个位于像素密度为 160dpi 的屏幕上的像素是一致的。要把密度无关像素转换为屏幕像素，可以用这样一个简单的公式：pixels=dips*(density/160)。举个例子，在 DPI 为 240 的屏幕上，1 个 DIP 等于 1.5 个物理像素。

布局时最好使用 dp 来定义我们程序的界面，因为这样可以保证我们的 UI 在各种分辨率的屏幕上都可以正常显示。

```
/**
 * 根据手机的分辨率从 px(像素) 的单位 转成为 dp
 */
public static int px2dip(Context context, float pxValue) {
    final float scale = context.getResources().getDisplayMetrics().density;
    return (int) (pxValue / scale + 0.5f);
}
/**
 * 根据手机的分辨率从 dip 的单位 转成为 px(像素)
 */
public static int dip2px(Context context, float dpValue) {
    final float scale = context.getResources().getDisplayMetrics().density;
    return (int) (dpValue * scale + 0.5f);
}
```

三、AIDL

1、什么是 AIDL 以及如何使用（★★★★）

①aidl 是 Android interface definition Language 的英文缩写，意思

Android 接口定义语言。

②使用 aidl 可以帮助我们发布以及调用远程服务，实现跨进程通信。

③将服务的 aidl 放到对应的 src 目录，工程的 gen 目录会生成相应的接口类

我们通过 bindService(Intent,ServiceConnect,int)方法绑定远程服务，在 bindService 中有一个 ServiceConnec 接口，我们需要覆写该类的 onServiceConnected(ComponentName,IBinder)方法，这个方法的第二个参数 IBinder 对象其实就是已经在 aidl 中定义的接口，因此我们可以将 IBinder 对象强制转换为 aidl 中的接口类。

我们通过 IBinder 获取到的对象(也就是 aidl 文件生成的接口)其实是系统产生的代理对象,该代理对象既可以跟我们的进程通信,又可以跟远程进程通信，作为一个中间的角色实现了进程间通信。

2、AIDL 的全称是什么?如何工作?能处理哪些类型的数据？(★★★)

AIDL 全称 Android Interface Definition Language（AndRoid 接口描述语言） 是一种接口描述语言; 编译器可以通过 aidl 文件生成一段代码，通过预先定义的接口达到两个进程内部通信进程跨界对象访问的目的。需要完成 2 件事情: 1. 引入 AIDL 的相关类.; 2. 调用 aidl 产生的 class.理论上，参数可以传递基本数

据类型和 String，还有就是 Bundle 的派生类，不过在 Eclipse 中,目前的 ADT 不支持 Bundle 做为参数。

四、Android 中的事件处理

1、Handler 机制（★★★★★）

Android 中主线程也叫 UI 线程，那么从名字上我们也知道主线程主要是用来创建、更新 UI 的，而其他耗时操作，比如网络访问，或者文件处理，多媒体处理等都需要在子线程中操作，之所以在子线程中操作是为了保证 UI 的流畅程度，手机显示的刷新频率是 60Hz，也就是一秒钟刷新 60 次，每 16.67 毫秒刷新一次，为了不丢帧，那么主线程处理代码最好不要超过 16 毫秒。当子线程处理完数据后，为了防止 UI 处理逻辑的混乱，Android 只允许主线程修改 UI，那么这时候就需要 Handler 来充当子线程和主线程之间的桥梁了。

我们通常将 Handler 声明在 Activity 中，然后覆写 Handler 中的 handleMessage 方法，当子线程调用 handler.sendMessage() 方法后 handleMessage 方法就会在主线程中执行。

这里面除了 Handler、Message 外还有隐藏的 Looper 和 MessageQueue 对象。

在主线程中 Android 默认已经调用了 Looper.preper()方法，调用该方法的

目的是在 Looper 中创建 MessageQueue 成员变量并把 Looper 对象绑定到当前线程中。当调用 Handler 的 sendMessage(对象)方法的时候就将 Message 对象添加到了 Looper 创建的 MessageQueue 队列中，同时给 Message 指定了 target 对象，其实这个 target 对象就是 Handler 对象。主线程默认执行了 Looper.looper（ ）方法，该方法从 Looper 的成员变量 MessageQueue 中取出 Message，然后调用 Message 的 target 对象的 handleMessage()方法。这样就完成了整个消息机制。

2、事件分发机制（★★★★★）

2.1 事件分发中的 onTouch 和 onTouchEvent 有什么区别，又该如何使用？

这两个方法都是在 View 的 dispatchTouchEvent 中调用的，onTouch 优先于 onTouchEvent 执行。如果在 onTouch 方法中通过返回 true 将事件消费掉，onTouchEvent 将不会再执行。

另外需要注意的是，onTouch 能够得到执行需要两个前提条件，第一 mOnTouchListener 的值不能为空，第二当前点击的控件必须是 enable 的。因此如果你有一个控件是非 enable 的，那么给它注册 onTouch 事件将永远得不到执行。对于这一类控件，如果我们想要监听它的 touch 事件，就必须通过在该控件中重写 onTouchEvent 方法来实现。

《Android 架构师面试题精编解析大全》

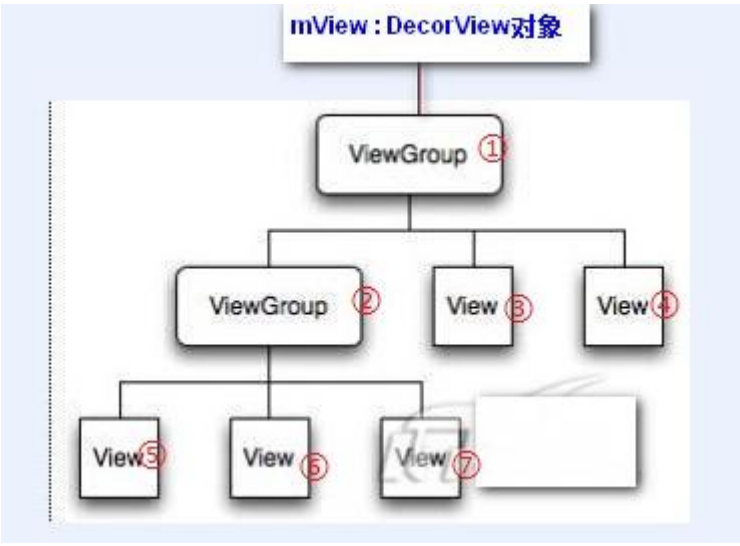
------腾讯享学课堂团队

2.2 请描述一下 Android 的事件分发机制

Android 的事件分发机制主要是 Touch 事件分发，有两个主角:ViewGroup 和 View。Activity 的 Touch 事件事实上是调用它内部的 ViewGroup 的 Touch 事件，可以直接当成 ViewGroup 处理。

View 在 ViewGroup 内，ViewGroup 也可以在其他 ViewGroup 内，这时候把内部的 ViewGroup 当成 View 来分析。

先分析 ViewGroup 的处理流程：首先得有个结构模型概念：ViewGroup 和 View 组成了一棵树形结构，最顶层为 Activity 的 ViewGroup，下面有若干的 ViewGroup 节点，每个节点之下又有若干的 ViewGroup 节点或者 View 节点，依次类推。如图：



当一个 Touch 事件(触摸事件为例)到达根节点，即 Acitivty 的 ViewGroup 时，它会依次下发，下发的过程是调用子 View(ViewGroup) 的 dispatchTouchEvent 方法实现的。简单来说，就是 ViewGroup 遍历它包含着

的子 View，调用每个 View 的 dispatchTouchEvent 方法，而当子 View 为 ViewGroup 时，又会通过调用 ViwGroup 的 dispatchTouchEvent 方法继续调用其内部的 View 的 dispatchTouchEvent 方法。上述例子中的消息下发顺序是这样的：①-②-⑤-⑥-⑦-③-④。dispatchTouchEvent 方法只负责事件的分发，它拥有 boolean 类型的返回值，当返回为 true 时，顺序下发会中断。在上述例子中如果⑤的 dispatchTouchEvent 返回结果为 true，那么⑥-⑦-③-④将都接收不到本次 Touch 事件。

1.Touch 事件分发中只有两个主角:ViewGroup 和 View。ViewGroup 包含 onInterceptTouchEvent、dispatchTouchEvent、onTouchEvent 三个相关事件。View 包含 dispatchTouchEvent、onTouchEvent 两个相关事件。其中 ViewGroup 又继承于 View。

2.ViewGroup 和 View 组成了一个树状结构 ,根节点为 Activity 内部包含的一个 ViwGroup。

3.触摸事件由 Action_Down、Action_Move、Aciton_UP 组成，其中一次完整的触摸事件中，Down 和 Up 都只有一个，Move 有若干个，可以为 0 个。

4.当 Acitivty 接收到 Touch 事件时 ,将遍历子 View 进行 Down 事件的分发。ViewGroup 的遍历可以看成是递归的。分发的目的是为了找到真正要处理本次完整触摸事件的 View，这个 View 会在 onTouchuEvent 结果返回 true。

5.当某个子 View 返回 true 时，会中止 Down 事件的分发，同时在 ViewGroup 中记录该子 View。接下去的 Move 和 Up 事件将由该子 View 直接

进行处理。由于子 View 是保存在 ViewGroup 中的，多层 ViewGroup 的节点结构时 ,上级 ViewGroup 保存的会是真实处理事件的 View 所在的 ViewGroup 对象:如 ViewGroup0-ViewGroup1-TextView 的结构中，TextView 返回了 true，它将被保存在 ViewGroup1 中，而 ViewGroup1 也会返回 true，被保存在 ViewGroup0 中。当 Move 和 UP 事件来时，会先从 ViewGroup0 传递至 ViewGroup1，再由 ViewGroup1 传递至 TextView。

6.当 ViewGroup 中所有子 View 都不捕获 Down 事件时，将触发 ViewGroup 自身的 onTouch 事件。触发的方式是调用 super.dispatchTouchEvent 函数，即父类 View 的 dispatchTouchEvent 方法。在所有子 View 都不处理的情况下，触发 Acitivity 的 onTouchEvent 方法。

7.onInterceptTouchEvent 有两个作用：1.拦截 Down 事件的分发。2.中止 Up 和 Move 事件向目标 View 传递，使得目标 View 所在的 ViewGroup 捕获 Up 和 Move 事件。

线程篇

Handler、Message、Looper、MessageQueue

一、相关概念的解释

主线程（UI 线程）

定义：当程序第一次启动时，Android 会同时启动一条主线程（Main Thread）

作用：主线程主要负责处理与 UI 相关的事件

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

Message（消息）

定义：Handler 接收和处理的消息对象（Bean 对象）

作用：通信时相关信息的存放和传递

ThreadLocal

定义：线程内部的数据存储类

作用：负责存储和获取本线程的 Looper

MessageQueue（消息队列）

定义：采用单链表的数据结构来存储消息列表

作用：用来存放通过 Handler 发过来的 Message，按照先进先出执行

Handler（处理者）

定义：Message 的主要处理者

作用：负责发送 Message 到消息队列&处理 Looper 分派过来的 Message

Looper（循环器）

定义：扮演 Message Queue 和 Handler 之间桥梁的角色

作用： 消息循环：循环取出 Message Queue 的 Message 消息派发：将取出的 Message 交付给相应的 Handler

2、自己画下图解

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

3、Handler 发送消息有哪几种方式？

一、sendMessage(Message msg) 二、post(Ruunable r)

4、Handler 处理消息有哪几种方式？

这个直接看 dispatchMessage()源码：

```
public void dispatchMessage(Message msg) {

    if (msg.callback != null) {

        //1. post()方法的处理方法

        handleCallback(msg);

    } else {

```

```
        if (mCallback != null) {

            if (mCallback.handleMessage(msg)) {

                return;

            }

        }

        //2. sendMessage()方法的处理方法

        handleMessage(msg);

    }
```

```
    }

    //1. post()方法的最终处理方法

    private static void handleCallback(Message message) {

        message.callback.run();

    }

    //2. sendMessage()方法的最终处理方法

    public void handleMessage(Message msg) {

    }
```

5.Message、Handler、MessageQueen、Looper 的之间的关系？

《Android 架构师面试题精编解析大全》

首先，是这个 MessageQueen，MessageQueen 是一个消息队列，它可以存储 Handler 发送过来的消息，其内部提供了进队和出队的方法来管理这个消息队列，其出队和进队的原理是采用单链表的数据结构进行插入和删除的，即 enqueueMessage()方法和 next()方法。这里提到的 Message，其实就是一个 Bean 对象，里面的属性用来记录 Message 的各种信息。

然后，是这个 Looper，Looper 是一个循环器，它可以循环的取出 MessageQueen 中的 Message，其内部提供了 Looper 的初始化和循环出去 Message 的方法 ,即 prepare()方法和 loop()方法。在 prepare()方法中 ,Looper 会关联一个 MessageQueen，而且将 Looper 存进一个 ThreadLocal 中，在 loop()方法中 ,通过 ThreadLocal 取出 Looper ,使用 MessageQueen 的 next()方法取出 Message 后，判断 Message 是否为空，如果是则 Looper 阻塞，如果不是，则通过 dispatchMessage()方法分发该 Message 到 Handler 中，而 Handler 执行 handlerMessage()方法，由于 handlerMessage()方法是个空方法 ,这也是为什么需要在 Handler 中重写 handlerMessage()方法的原因。这里要注意的是 Looper 只能在一个线程中只能存在一个。这里提到的 ThreadLocal，其实就是一个对象，用来在不同线程中存放对应线程的 Looper。

最后 ,是这个 Handler ,Handler 是 Looper 和 MessageQueen 的桥梁 ,Handler 内部提供了发送 Message 的一系列方法，最终会通过 MessageQueen 的 enqueueMessage()方法将 Message 存进 MessageQueen 中。我们平时可以

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

直接在主线程中使用 Handler，那是因为在应用程序启动时，在入口的 main 方法中已经默认为我们创建好了 Looper。

相关文章：blog.csdn.net/qq_30379689...

6.为什么在子线程中创建 Handler 会抛异常？

Handler 的工作是依赖于 Looper 的，而 Looper（与消息队列）又是属于某一个线程（ThreadLocal 是线程内部的数据存储类，通过它可以在指定线程中存储数据，其他线程则无法获取到），其他线程不能访问。因此 Handler 就是间接跟线程是绑定在一起了。因此要使用 Handler 必须要保证 Handler 所创建的线程中有 Looper 对象并且启动循环。因为子线程中默认是没有 Looper 的，所以会报错。 正确的使用方法是：

```
private final class WorkThread extends Thread {

        private Handler mHandler;



        public Handler getHandler() {
            return mHandler;
        }
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
        public void quit() {
            mHandler.getLooper().quit();
        }
        @Override
        public void run() {
            super.run();
            //创建该线程对应的 Looper，
            // 内部实现
            // 1。new Looper（）
            // 2。将 1 步中的 lopper 放在 ThreadLocal 里，ThreadLocal 是保存数据的，主要应用场景是：线程间数据互不影响的情况
            // 3。在 1 步中的 Looper 的构造函数中 new MessageQueue();
            //其实就是创建了该线程对用的 Looper，Looper 里创建 MessageQueue 来实现消息机制
            //对消息机制不懂得同学可以查阅资料，网上很多也讲的很不错。
            Looper.prepare();
            mHandler = new Handler() {
                @Override

                public void handleMessage(Message msg) {
                    super.handleMessage(msg);
```

```
                    Log.d("WorkThread", (Looper.getMainLooper() ==
Looper.myLooper()) + "," + msg.what);
```

```
                }
            };
            //开启消息的死循环处理即：dispatchMessage
            Looper.loop();
            //注意这 3 个的顺序不能颠倒
            Log.d("WorkThread", "end");
        }
    }
```

HandlerThread

1、HandlerThread 作用

当系统有多个耗时任务需要执行时 ,每个任务都会开启一个新线程去执行耗时任务，这样会导致系统多次创建和销毁线程，从而影响性能。为了解决这一问题，Google 提供了 HandlerThread ,HandlerThread 是在线程中创建一个 Looper 循环器，让 Looper 轮询消息队列，当有耗时任务进入队列时，则不需要开启新线程，在原有的线程中执行耗时任务即可，否则线程阻塞。

2、HanlderThread 的优缺点

HandlerThread 本质上是一个线程类，它继承了 Thread；

HandlerThread 有自己的内部 Looper 对象，可以进行 looper 循环；

通过获取 HandlerThread 的 looper 对象传递给 Handler 对象，可以在 `handleMessage()`方法中执行异步任务。

创建 HandlerThread 后必须先调用 `HandlerThread.start()`方法，Thread 会先调用 run 方法，创建 Looper 对象。

HandlerThread 优点是异步不会堵塞，减少对性能的消耗

HandlerThread 缺点是不能同时继续进行多任务处理，需要等待进行处理，处理效率较低

HandlerThread 与线程池不同，HandlerThread 是一个串行队列，背后只有一个线程。

相关文章：Android 多线程之 IntentService 完全详解

IntentService

它本质是一种特殊的 Service,继承自 Service 并且本身就是一个抽象类

它可以用于在后台执行耗时的异步任务，当任务完成后会自动停止

它拥有较高的优先级，不易被系统杀死（继承自 Service 的缘故），因此比较适合执行一些高优先级的异步任务 它内部通过 HandlerThread 和 Handler 实现异步操作

创建 IntentService 时，只需实现 onHandleIntent 和构造方法，onHandleIntent 为异步方法，可以执行耗时操作

即使我们多次启动 IntentService，但 IntentService 的实例只有一个，这跟传统的 Service 是一样的，最终 IntentService 会去调用 onHandleIntent 执行异步任务。

当任务完成后，IntentService 会自动停止，而不需要手动调用 `stopSelf()`。另外，可以多次启动 IntentService，每个耗时操作都会以工作队列的方式在 `IntentService` 中 `onHandlerIntent()`回调方法中执行，并且每次只会执行一个工作线程。

相关文章：Android 多线程之 IntentService 完全详解

AsyncTask

1、AsyncTask 是什么

AsyncTask 是一种轻量级的异步任务类，它可以在线程池中执行后台任务，然后把执行的进度和最终结果传递给主线程并主线程中更新 UI，通过 AsyncTask 可以更加方便执行后台任务以及在主线程中访问 UI，但是 AsyncTask 并不适合进行特别耗时的后台任务，对于特别耗时的任务来说，建议使用线程池。

2、AsyncTask 使用方法

三个参数 Params :表示后台任务执行时的参数类型,该参数会传给 AysncTask 的 doInBackground()方法 Progress：表示后台任务的执行进度的参数类型，该参数会作为 onProgressUpdate()方法的参数 Result：表示后台任务的返回结果的参数类型，该参数会作为 onPostExecute()方法的参数 五个方法 onPreExecute()：异步任务开启之前回调，在主线程中执行 doInBackground()：执行异步任务，在线程池中执行 onProgressUpdate()：当 doInBackground 中调用 publishProgress 时回调，在主线程中执行 onPostExecute() :在异步任务执行之后回调 ,在主线程中执行 onCancelled()：在异步任务被取消时回调

3、AsyncTask 引起的内存泄漏

原因：非静态内部类持有外部类的匿名引用，导致 Activity 无法释放 解决：AsyncTask 内部持有外部 Activity 的弱引用 AsyncTask 改为静态内部类 Activity 的 onDestory()中调用 AsyncTask.cancel()

4.结果丢失

屏幕旋转或 Activity 在后台被系统杀掉等情况会导致 Activity 的重新创建 ,之前运行的 AsyncTask 会持有一个之前 Activity 的引用，这个引用已经无效，这时调用 onPostExecute()再去更新界面将不再生效。

5、AsyncTask 并行 or 串行

AsyncTask 在 Android 2.3 之前默认采用并行执行任务 ,AsyncTask 在 Android

2.3 之后默认采用串行执行任务 如果需要在 Android 2.3 之后采用并行执行任务，可以调用 AsyncTask 的 executeOnExecutor();

6.AsyncTask 内部的线程池

```
private static volatile Executor sDefaultExecutor = SERIAL_EXECUTOR;
```

sDefaultExecutor 是 SerialExecutor 的一个实例，而且它是个静态变量。也就是说，一个进程里面所有 AsyncTask 对象都共享同一个 SerialExecutor 对象。

3、子线程发消息到主线程进行更新 UI，除了 handler 和 AsyncTask，还有什么？（★★★）

1、用 Activity 对象的 runOnUiThread 方法更新

在子线程中通过 runOnUiThread()方法更新 UI：

```
new Thread() {
            public void run() {
                //这儿是耗时操作，完成之后更新UI；
                runOnUiThread(new Runnable(){

                    @Override
                    public void run() {
                        //更新UI
                        imageView.setImageBitmap(bitmap);
                    }

                });
            }
        }.start();
```

如果在非上下文类中（Activity），可以通过传递上下文实现调用；

```
Activity activity = (Activity) imageView.getContext();
                activity.runOnUiThread(new Runnable() {

                    @Override
                    public void run() {
                        imageView.setImageBitmap(bitmap);
                    }
                });
```

2、用 View.post(Runnable r)方法更新 UI

```
imageView.post(new Runnable(){

                    @Override
                    public void run() {
                        imageView.setImageBitmap(bitmap);
                    }

                });
```

4、子线程中能不能 new handler？为什么？（★★★）

不能，如果在子线程中直接 new Handler() 会抛出异常 java.lang.RuntimeException: Can't create handler inside thread that has not called

在没有调用 Looper.prepare()的时候不能创建 Handler,因为在创建 Handler 的源码中做了如下操作

Handler 的构造方法中

```
public static Looper myLooper() {
        return sThreadLocal.get();
}
```

```
    mLooper = Looper.myLooper();
    if (mLooper == null) {
        throw new RuntimeException(
            "Can't create handler inside thread that has not called Looper.prepare()");
    }
```

五、Android 中的动画

1、Android 中的动画有哪几类，它们的特点和区别是什么（★★★）

Android 中动画分为两种，一种是 Tween 动画、还有一种是 Frame 动画。

Tween 动画，这种实现方式可以使视图组件移动、放大、缩小以及产生透明度的变化；

Frame 动画，传统的动画方法，通过顺序的播放排列好的图片来实现，类似电影。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

2、如何修改 Activity 进入和退出动画（★★）

可以通过两种方式，一是通过定义 Activity 的主题，二是通过覆写 Activity 的 overridePendingTransition 方法。

- 通过设置主题样式

在 styles.xml 中编辑如下代码：

```
<style name="AnimationActivity" parent="@android:style/Animation.Activity">
    <item name="android:activityOpenEnterAnimation">@anim/slide_in_left</ite
    <item name="android:activityOpenExitAnimation">@anim/slide_out_left</ite
    <item name="android:activityCloseEnterAnimation">@anim/slide_in_right</i
    <item name="android:activityCloseExitAnimation">@anim/slide_out_right</i
</style>
```

添加 themes.xml 文件：

```
<style name="ThemeActivity">
    <item name="android:windowAnimationStyle">@style/AnimationActivity</item
    <item name="android:windowNoTitle">true</item>
</style>
```

在 AndroidManifest.xml 中给指定的 Activity 指定 theme。

- 覆写 overridePendingTransition 方法

overridePendingTransition(R.anim.fade, R.anim.hold);

3、属性动画，例如一个 button 从 A 移动到 B 点，B 点还是可以响应点击事件，这个原理是什么？（★★）

补间动画只是显示的位置变动，View 的实际位置未改变，表现为 View 移动到其他地方，点击事件仍在原处才能响应。而属性动画控件移动后事件相应就在控件移动后本身进行处理

六、ContentObserver 内容观察者作用及特点（★★★）

一、ContentObserver 目的是观察(捕捉)特定 Uri 引起的数据库的变化，继而做一些相应的处理。

它类似于数据库技术中的触发器(Trigger)，当 ContentObserver 所观察的 Uri 发生变化时，便会触发它。触发器分为表触发器、行触发器，相应地 ContentObserver 也分为“表“ContentObserver、“行”ContentObserver，当然这是与它所监听的 Uri MIME Type 有关的。

1）**注册 ContentObserver 方法**

public final void registerContentObserver(Uri uri,

boolean notifyForDescendents, ContentObserver observer)

功能：为指定的 Uri 注册一个 ContentObserver 派生类实例，当给定的 Uri 发生改变时，回调该实例对象去处理。

参数: uri 表示需要观察的 Uri

notifyForDescendents 为 false 表示精确匹配，即只匹配该 Uri。 为 true 表示可以同时匹配其派生的 Uri。

取消注册 ContentObserver 方法

public final void

unregisterContentObserver(ContentObserver observer)

功能：取消对给定 Uri 的观察

参数： observer ContentObserver 的派生类实例

ContentObserver 类介绍

构造方法 ContentObserver(Handler h)

void onChange(boolean selfChange) 功能 :当观察到的 Uri 发生变化时，回调该方法去处理。所有 ContentObserver 的派生类都需要重载该方法去处理逻辑。

3.观察特定 Uri 的步骤如下：

1、创建我们特定的 ContentObserver 派生类，必须重载父类构造方法，必须重载 onChange()方法去处理回调后的功能实现

2、利用 context.getContentResolover()获 ContentResolover 对象，接着调用 registerContentObserver()方法去注册内容观察者

3、在不需要时，需要手动的调用 unregisterContentObserver()去取消注

册。

例子：监听短信内容变化

在 Activity 中：

```
public class Day0108_contentobserverActivity extends Activity {
    private Handler handler = new Handler(){
        public void handleMessage(android.os.Message msg) {
            switch (msg.what) {
            case 100:
                String body = (String) msg.obj;
                TextView tv = (TextView) findViewById(R.id.tv);
                tv.setText(body);
                break;
            }
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
            ContentResolver cr = getContentResolver();
            ContentObserver smsObserver = new
SmsContentObserver(this,handler);
            //第二个参数,true 表示观察所有有关短信的
            cr.registerContentObserver(Uri.parse("content://sms"),
true, smsObserver);
            //content://sms/inbox //收件箱
            //content://sms/sent //已发送
            //content://sms/draft //草稿箱
            //content://sms/outbox //发件箱
            //content://sms/failed //失败短信
            //content://sms/queued //代发队列
        }
}
//SmsContentObserver 代码如下：
public class SmsContentObserver extends ContentObserver {
    private Handler handler;
    private Context context;
    public SmsContentObserver(Context context,Handler
handler) {
```

```
        super(handler);
        this.handler = handler;
        this.context = context;
    }

    @Override
    public void onChange(boolean selfChange) {
        ContentResolver cr = context.getContentResolver();
        Cursor c = cr.query(Uri.parse("content://sms/inbox"),
null, "0", null, "date desc");
        StringBuilder sb = new StringBuilder();
        while(c.moveToNext()){
            //发件人手机号码
            String sendNumber = c.getString(
            c.getColumnIndex("address"));
            //信息内容
            String body =
c.getString(c.getColumnIndex("body"));
            //readType 表示是否已经读
            int hasRead = c.getInt(c.getColumnIndex("read"));
```

```
                if(hasRead == 0){//表示短信未读
                    System.out.println("短信未读"+sendNumber);
                }
                sb.append(sendNumber+":"+body+"\n");
            }

handler.obtainMessage(100,sb.toString()).sendToTarget();
        }
}
```

2019Android 高级面试题总结（★★★★★）

1. 说下你所知道的设计模式与使用场景

a. 建造者模式：

将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

使用场景比如最常见的 AlertDialog,拿我们开发过程中举例，比如 Camera 开发过程中，可能需要设置一个初始化的相机配置，设置摄像头方向，闪光灯开闭，成像质量等等，这种场景下就可以使用建造者模式

装饰者模式：动态的给一个对象添加一些额外的职责，就增加功能来说，装饰模

式比生成子类更为灵活。装饰者模式可以在不改变原有类结构的情况下曾强类的功能，比如 Java 中的 BufferedInputStream 包装 FileInputStream，举个开发中的例子，比如在我们现有网络框架上需要增加新的功能，那么再包装一层即可，装饰者模式解决了继承存在的一些问题，比如多层继承代码的臃肿，使代码逻辑更清晰

观察者模式：

代理模式：

门面模式：

单例模式：

生产者消费者模式：

2. java 语言的特点与 OOP 思想

这个通过对比来描述，比如面向对象和面向过程的对比，针对这两种思想的对比，还可以举个开发中的例子，比如播放器的实现，面向过程的实现方式就是将播放视频的这个功能分解成多个过程，比如，加载视频地址，获取视频信息，初始化解码器，选择合适的解码器进行解码，读取解码后的帧进行视频格式转换和音频重采样，然后读取帧进行播放，这是一个完整的过程，这个过程中不涉及类的概念，而面向对象最大的特点就是类，封装继承和多态是核心，同样的以播放器为例，一面向对象的方式来实现，将会针对每一个功能封装出一个对象，吧如说

Muxer，获取视频信息，Decoder,解码，格式转换器，视频播放器，音频播放器等，每一个功能对应一个对象，由这个对象来完成对应的功能，并且遵循单一职责原则，一个对象只做它相关的事情

3. 说下 java 中的线程创建方式，线程池的工作原理。

java 中有三种创建线程的方式，或者说四种

1.继承 Thread 类实现多线程

2.实现 Runnable 接口

3.实现 Callable 接口

4.通过线程池

线程池的工作原理：线程池可以减少创建和销毁线程的次数，从而减少系统资源的消耗，当一个任务提交到线程池时

a. 首先判断核心线程池中的线程是否已经满了，如果没满，则创建一个核心线程执行任务，否则进入下一步

b. 判断工作队列是否已满，没有满则加入工作队列，否则执行下一步

c. 判断线程数是否达到了最大值，如果不是，则创建非核心线程执行任务，否则执行饱和策略，默认抛出异常

4. 说下 handler 原理

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

Handler，Message，looper 和 MessageQueue 构成了安卓的消息机制，handler 创建后可以通过 sendMessage 将消息加入消息队列，然后 looper 不断的将消息从 MessageQueue 中取出来，回调到 Hander 的 handleMessage 方法，从而实现线程的通信。

从两种情况来说，第一在 UI 线程创建 Handler,此时我们不需要手动开启 looper，因为在应用启动时 ,在 ActivityThread 的 main 方法中就创建了一个当前主线程的 looper ,并开启了消息队列 ,消息队列是一个无限循环 ,为什么无限循环不会 ANR? 因为可以说，应用的整个生命周期就是运行在这个消息循环中的，安卓是由事件驱动的，Looper.loop 不断的接收处理事件，每一个点击触摸或者 Activity 每一个生命周期都是在 Looper.loop 的控制之下的，looper.loop 一旦结束，应用程序的生命周期也就结束了。我们可以想想什么情况下会发生 ANR，第一，事件没有得到处理，第二，事件正在处理，但是没有及时完成，而对事件进行处理的就是 looper，所以只能说事件的处理如果阻塞会导致 ANR，而不能说 looper 的无限循环会 ANR

另一种情况就是在子线程创建 Handler,此时由于这个线程中没有默认开启的消息队列，所以我们需要手动调用 looper.prepare(),并通过 looper.loop 开启消息

主线程 Looper 从消息队列读取消息，当读完所有消息时，主线程阻塞。子线程往消息队列发送消息，并且往管道文件写数据，主线程即被唤醒，从管道文件读

取数据，主线程被唤醒只是为了读取消息，当消息读取完毕，再次睡眠。因此 loop 的循环并不会对 CPU 性能有过多的消耗。

5. 内存泄漏的场景和解决办法

1.非静态内部类的静态实例

非静态内部类会持有外部类的引用，如果非静态内部类的实例是静态的，就会长期的维持着外部类的引用，组织被系统回收，解决办法是使用静态内部类

2.多线程相关的匿名内部类和非静态内部类

匿名内部类同样会持有外部类的引用，如果在线程中执行耗时操作就有可能发生内存泄漏，导致外部类无法被回收，直到耗时任务结束，解决办法是在页面退出时结束线程中的任务

3.Handler 内存泄漏

Handler 导致的内存泄漏也可以被归纳为非静态内部类导致的，Handler 内部 message 是被存储在 MessageQueue 中的，有些 message 不能马上被处理，存在的时间会很长，导致 handler 无法被回收，如果 handler 是非静态的，就会导致它的外部类无法被回收，解决办法是 1.使用静态 handler，外部类引用使用弱引用处理 2.在退出页面时移除消息队列中的消息

4.Context 导致内存泄漏

根据场景确定使用 Activity 的 Context 还是 Application 的 Context,因为二者生命周期不同，对于不必须使用 Activity 的 Context 的场景（Dialog），一律采用

Application 的 Context,单例模式是最常见的发生此泄漏的场景，比如传入一个 Activity 的 Context 被静态类引用，导致无法回收

5.静态 View 导致泄漏

使用静态 View 可以避免每次启动 Activity 都去读取并渲染 View ,但是静态 View 会持有 Activity 的引用，导致无法回收，解决办法是在 Activity 销毁的时候将静态 View 设置为 null（View 一旦被加载到界面中将会持有一个 Context 对象的引用，在这个例子中，这个 context 对象是我们的 Activity，声明一个静态变量引用这个 View，也就引用了 activity）

6.WebView 导致的内存泄漏

WebView 只要使用一次，内存就不会被释放，所以 WebView 都存在内存泄漏的问题，通常的解决办法是为 WebView 单开一个进程，使用 AIDL 进行通信，根据业务需求在合适的时机释放掉

7.资源对象未关闭导致

如 Cursor，File 等，内部往往都使用了缓冲，会造成内存泄漏，一定要确保关闭它并将引用置为 null

8.集合中的对象未清理

集合用于保存对象，如果集合越来越大，不进行合理的清理，尤其是入股集合是静态的

9.Bitmap 导致内存泄漏

bitmap 是比较占内存的，所以一定要在不使用的时候及时进行清理，避免静态变

量持有大的 bitmap 对象

10.监听器未关闭

很多需要 register 和 unregister 的系统服务要在合适的时候进行 unregister,手动添加的 listener 也需要及时移除

6. 如何避免 OOM?

1.使用更加轻量的数据结构：如使用 ArrayMap/SparseArray 替代 HashMap,HashMap 更耗内存，因为它需要额外的实例对象来记录 Mapping 操作，SparseArray 更加高效，因为它避免了 Key Value 的自动装箱，和装箱后的解箱操作

2.便面枚举的使用，可以用静态常量或者注解@IntDef 替代

3.Bitmap 优化:

a.尺寸压缩：通过 InSampleSize 设置合适的缩放

b.颜色质量：设置合适的 format，ARGB_6666/RBG_545/ARGB_4444/ALPHA_6，存在很大差异

c.inBitmap:使用 inBitmap 属性可以告知 Bitmap 解码器去尝试使用已经存在的内存区域，新解码的 Bitmap 会尝试去使用之前那张 Bitmap 在 Heap 中所占据的 pixel data 内存区域，而不是去问内存重新申请一块区域来存放 Bitmap。利用这种特性，即使是上千张的图片，也只会仅仅只需要占用屏幕所能够显示的图片数量的内存大小，但复用存在一些限制，具体体现在：在 Android 4.4 之前只能重用相同大

小的 Bitmap 的内存，而 Android 4.4 及以后版本则只要后来的 Bitmap 比之前的小即可。使用 inBitmap 参数前，每创建一个 Bitmap 对象都会分配一块内存供其使用，而使用了 inBitmap 参数后，多个 Bitmap 可以复用一块内存，这样可以提高性能

4.StringBuilder 替代 String: 在有些时候,代码中会需要使用到大量的字符串拼接的操作，这种时候有必要考虑使用 StringBuilder 来替代频繁的“+”

5.避免在类似 onDraw 这样的方法中创建对象，因为它会迅速占用大量内存，引起频繁的 GC 甚至内存抖动

6.减少内存泄漏也是一种避免 OOM 的方法

7. 说下 Activity 的启动模式，生命周期，两个 Activity 跳转的生命周期，如果一个 Activity 跳转另一个 Activity 再按下 Home 键在回到 Activity 的生命周期是什么样的

启动模式

Standard 模式:Activity 可以有多个实例，每次启动 Activity，无论任务栈中是否已经有这个 Activity 的实例，系统都会创建一个新的 Activity 实例

SingleTop 模式:当一个 singleTop 模式的 Activity 已经位于任务栈的栈顶,再去启动它时，不会再创建新的实例,如果不位于栈顶，就会创建新的实例

SingleTask 模式:如果 Activity 已经位于栈顶，系统不会创建新的 Activity 实例，和 singleTop 模式一样。但 Activity 已经存在但不位于栈顶时，系统就会把该

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

Activity 移到栈顶，并把它上面的 activity 出栈

SingleInstance 模式:singleInstance 模式也是单例的，但和 singleTask 不同，singleTask 只是任务栈内单例，系统里是可以有多个 singleTask Activity 实例的，而 singleInstance Activity 在整个系统里只有一个实例，启动一singleInstanceActivity 时，系统会创建一个新的任务栈，并且这个任务栈只有他一个 Activity

生命周期

onCreate onStart onResume onPause onStop onDestroy

两个 Activity 跳转的生命周期

1.启动 A

onCreate - onStart - onResume

2.在 A 中启动 B

ActivityA onPause

ActivityB onCreate

ActivityB onStart

ActivityB onResume

ActivityA onStop

3.从 B 中返回 A（按物理硬件返回键）

ActivityB onPause

ActivityA onRestart

ActivityA onStart

ActivityA onResume

ActivityB onStop

ActivityB onDestroy

4.继续返回

ActivityA onPause

ActivityA onStop

ActivityA onDestroy

8. onRestart 的调用场景

（1）按下 home 键之后，然后切换回来，会调用 onRestart()。

（2）从本 Activity 跳转到另一个 Activity 之后，按 back 键返回原来 Activity，会调用 onRestart()；

（3）从本 Activity 切换到其他的应用，然后再从其他应用切换回来，会调用 onRestart()；

说下 Activity 的横竖屏的切换的生命周期，用那个方法来保存数据，两者的区别。触发在什么时候在那个方法里可以获取数据等。

9. 是否了 SurfaceView，它是什么？他的继承方式是什么？他与 View 的区别(从源码角度，如加载，绘制等)。

SurfaceView 中采用了双缓冲机制，保证了 UI 界面的流畅性，同时 SurfaceView 不在主线程中绘制，而是另开辟一个线程去绘制，所以它不妨碍 UI 线程；

SurfaceView 继承于 View，他和 View 主要有以下三点区别：

（1）View 底层没有双缓冲机制，SurfaceView 有；

（2）view 主要适用于主动更新，而 SurfaceView 适用与被动的更新，如频繁的刷新

（3）view 会在主线程中去更新 UI，而 SurfaceView 则在子线程中刷新；

SurfaceView 的内容不在应用窗口上，所以不能使用变换（平移、缩放、旋转等）。也难以放在 ListView 或者 ScrollView 中，不能使用 UI 控件的一些特性比如 View.setAlpha()

View：显示视图，内置画布，提供图形绘制函数、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。

SurfaceView :基于 view 视图进行拓展的视图类，更适合 2D 游戏的开发；是 view 的子类，类似使用双缓机制，在新的线程中更新画面所以刷新界面速度比 view 快，Camera 预览界面使用 SurfaceView。

GLSurfaceView：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。

10.如何实现进程保活

a: Service 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

b: 通过 startForeground 将进程设置为前台进程， 做前台服务，优先级和前台应

用一个级别，除非在系统内存非常缺，否则此进程不会被 kill

c: 双进程 Service： 让 2 个进程互相保护对方，其中一个 Service 被清理后，另外没被清理的进程可以立即重启进程

d: 用 C 编写守护进程(即子进程) : Android 系统中当前进程(Process)fork 出来的子进程，被系统认为是两个不同的进程。当父进程被杀死的时候，子进程仍然可以存活，并不受影响(Android5.0 以上的版本不可行）联系厂商，加入白名单

e.锁屏状态下，开启一个一像素 Activity

11. 说下冷启动与热启动是什么，区别，如何优化，使用场景等。

app 冷启动： 当应用启动时，后台没有该应用的进程，这时系统会重新创建一个新的进程分配给该应用， 这个启动方式就叫做冷启动（后台不存在该应用进程）。冷启动因为系统会重新创建一个新的进程分配给它，所以会先创建和初始化 Application 类，再创建和初始化 MainActivity 类（包括一系列的测量、布局、绘制），最后显示在界面上。

app 热启动： 当应用已经被打开， 但是被按下返回键、Home 键等按键时回到桌面或者是其他程序的时候，再重新打开该 app 时， 这个方式叫做热启动（后台已经存在该应用进程）。热启动因为会从已有的进程中来启动，所以热启动就不会走 Application 这步了，而是直接走 MainActivity（包括一系列的测量、布局、绘制），所以热启动的过程只需要创建和初始化一个 MainActivity 就行了，而不必创建和初始化 Application

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

冷启动的流程

当点击 app 的启动图标时，安卓系统会从 Zygote 进程中 fork 创建出一个新的进程分配给该应用，之后会依次创建和初始化 Application 类、创建 MainActivity 类、加载主题样式 Theme 中的 windowBackground 等属性设置给 MainActivity 以及配置 Activity 层级上的一些属性、再 inflate 布局、当 onCreate/onStart/onResume 方法都走完了后最后才进行 contentView 的 measure/layout/draw 显示在界面上

冷启动的生命周期简要流程：

Application 构造方法 –> attachBaseContext()–>onCreate –>Activity 构造方法 –> onCreate() –> 配置主体中的背景等操作 –>onStart() –> onResume() –> 测量、布局、绘制显示

冷启动的优化主要是视觉上的优化，解决白屏问题，提高用户体验，所以通过上面冷启动的过程。能做的优化如下：

1、减少 onCreate()方法的工作量

2、不要让 Application 参与业务的操作

3、不要在 Application 进行耗时操作

4、不要以静态变量的方式在 Application 保存数据

5、减少布局的复杂度和层级

6、减少主线程耗时

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

12. 为什么冷启动会有白屏黑屏问题？

原因在于加载主题样式 Theme 中的 windowBackground 等属性设置给 MainActivity 发生在 inflate 布局当 onCreate/onStart/onResume 方法之前，而 windowBackground 背景被设置成了白色或者黑色，所以我们进入 app 的第一个界面的时候会造成先白屏或黑屏一下再进入界面。解决思路如下

1.给他设置 windowBackground 背景跟启动页的背景相同，如果你的启动页是张图片那么可以直接给 windowBackground 这个属性设置该图片那么就不会有一闪的效果了

```
<style name=``"Splash_Theme"``
`parent=``"@android:style/Theme.NoTitleBar"``>`
    <item
name=``"android:windowBackground"``>@drawable/splash_bg</item>`
    <item name=``"android:windowNoTitle"``>``true``</item>`</style>`
```

2.采用世面的处理方法，设置背景是透明的，给人一种延迟启动的感觉。,将背景颜色设置为透明色,这样当用户点击桌面 APP 图片的时候，并不会"立即"进入 APP，而且在桌面上停留一会，其实这时候 APP 已经是启动的了，只是我们心机的把 Theme 里的 windowBackground 的颜色设置成透明的，强行把锅甩给了手机应用厂商（手机反应太慢了啦）

```
<style name=``"Splash_Theme"``
`parent=``"@android:style/Theme.NoTitleBar"``>`
    <item name=``"android:windowIsTranslucent"``>``true``</item>`
    <item name=``"android:windowNoTitle"``>``true``</item>`</style>`
```

3.以上两种方法是在视觉上显得更快，但其实只是一种表象，让应用启动的更快，有一种思路，将 Application 中的不必要的初始化动作实现懒加载，比如，在 SpashActivity 显示后再发送消息到 Application，去初始化，这样可以将初始化的动作放在后边，缩短应用启动到用户看到界面的时间

13. Android 中的线程有那些,原理与各自特点

AsyncTask,HandlerThread,IntentService

AsyncTask 原理：内部是 Handler 和两个线程池实现的，Handler 用于将线程切换到主线程，两个线程池一个用于任务的排队，一个用于执行任务，当 AsyncTask 执行 execute 方法时会封装出一个 FutureTask 对象，将这个对象加入队列中，如果此时没有正在执行的任务，就执行它，执行完成之后继续执行队列中下一个任务，执行完成通过 Handler 将事件发送到主线程。AsyncTask 必须在主线程初始化，因为内部的 Handler 是一个静态对象，在 AsyncTask 类加载的时候他就已经被初始化了。在 Android3.0 开始，execute 方法串行执行任务的，一个一个来，3.0 之前是并行执行的。如果要在 3.0 上执行并行任务，可以调用 executeOnExecutor 方法

HandlerThread 原理：继承自 Thread，start 开启线程后，会在其 run 方法中会通过 Looper 创建消息队列并开启消息循环，这个消息队列运行在子线程中，所以可以将 HandlerThread 中的 Looper 实例传递给一个 Handler，从而保证这个

Handler 的 handleMessage 方法运行在子线程中，Android 中使用 HandlerThread 的一个场景就是 IntentService

IntentService 原理：继承自 Service，它的内部封装了 HandlerThread 和 Handler，可以执行耗时任务，同时因为它是一个服务，优先级比普通线程高很多，所以更适合执行一些高优先级的后台任务，HandlerThread 底层通过 Looper 消息队列实现的，所以它是顺序的执行每一个任务。可以通过 Intent 的方式开启 IntentService，IntentService 通过 handler 将每一个 intent 加入 HandlerThread 子线程中的消息队列，通过 looper 按顺序一个个的取出并执行，执行完成后自动结束自己，不需要开发者手动关闭

14. ANR 的原因

1.耗时的网络访问

2.大量的数据读写

3.数据库操作

4.硬件操作（比如 camera)

5.调用 thread 的 join()方法、sleep()方法、wait()方法或者等待线程锁的时候

6.service binder 的数量达到上限

7.system server 中发生 WatchDog ANR

8.service 忙导致超时无响应

9.其他线程持有锁，导致主线程等待超时

10.其它线程终止或崩溃导致主线程一直等待

15. 三级缓存原理

当 Android 端需要获得数据时比如获取网络中的图片，首先从内存中查找（按键查找），内存中没有的再从磁盘文件或 sqlite 中去查找，若磁盘中也没有才通过网络获取

16. LruCache 底层实现原理：

LruCache 中 Lru 算法的实现就是通过 LinkedHashMap 来实现的。LinkedHashMap 继承于 HashMap，它使用了一个双向链表来存储 Map 中的 Entry 顺序关系，对于 get、put、remove 等操作，LinkedHashMap 除了要做 HashMap 做的事情，还做些调整 Entry 顺序链表的工作。

LruCache 中将 LinkedHashMap 的顺序设置为 LRU 顺序来实现 LRU 缓存，每次调用 get(也就是从内存缓存中取图片)，则将该对象移到链表的尾端。

调用 put 插入新的对象也是存储在链表尾端，这样当内存缓存达到设定的最大值时，将链表头部的对象（近期最少用到的）移除。

17. 说下你对 Collection 这个类的理解。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

Collection 是集合框架的顶层接口，是存储对象的容器,Colloction 定义了接口的公用方法如 add remove clear 等等，它的子接口有两个，List 和 Set,List 的特点有元素有序，元素可以重复，元素都有索引（角标），典型的有

Vector:内部是数组数据结构，是同步的（线程安全的）。增删查询都很慢。

ArrayList:内部是数组数据结构，是不同步的（线程不安全的）。替代了 Vector。查询速度快，增删比较慢。

LinkedList:内部是链表数据结构，是不同步的（线程不安全的）。增删元素速度快。

而 Set 的是特点元素无序，元素不可以重复

HashSet：内部数据结构是哈希表，是不同步的。

Set 集合中元素都必须是唯一的，HashSet 作为其子类也需保证元素的唯一性。

判断元素唯一性的方式：

通过存储对象（元素）的 hashCode 和 equals 方法来完成对象唯一性的。

如果对象的 hashCode 值不同，那么不用调用 equals 方法就会将对象直接存储到集合中；

如果对象的 hashCode 值相同，那么需调用 equals 方法判断返回值是否为 true，若为 false, 则视为不同元素，就会直接存储；

若为 true， 则视为相同元素，不会存储。

如果要使用 HashSet 集合存储元素 ,该元素的类必须覆盖 hashCode 方法和 equals

方法。一般情况下，如果定义的类会产生很多对象，通常都需要覆盖 equals，hashCode 方法。建立对象判断是否相同的依据。

TreeSet：保证元素唯一性的同时可以对内部元素进行排序，是不同步的。

判断元素唯一性的方式：

根据比较方法的返回结果是否为 0，如果为 0 视为相同元素，不存；如果非 0 视为不同元素，则存。

TreeSet 对元素的排序有两种方式：

方式一：使元素（对象）对应的类实现 Comparable 接口，覆盖 compareTo 方法。这样元素自身具有比较功能。

方式二：使 TreeSet 集合自身具有比较功能，定义一个比较器 Comparator，将该类对象作为参数传递给 TreeSet 集合的构造函数

18. JVM 老年代和新生代的比例

Java 中的堆是 JVM 所管理的最大的一块内存空间，主要用于存放各种类的实例对象。

在 Java 中，堆被划分成两个不同的区域：新生代（Young）、老年代（Old）。

新生代（Young）又被划分为三个区域：Eden、From Survivor、To Survivor。

这样划分的目的是为了使 JVM 能够更好的管理堆内存中的对象，包括内存的分配以及回收。

堆大小 = 新生代 + 老年代。其中，堆的大小可以通过参数 –Xms、-Xmx 来指

定。

（本人使用的是 JDK1.6，以下涉及的 JVM 默认值均以该版本为准。）

默认的，新生代（Young）与老年代（Old）的比例的值为 1:2（该值可以通过参数 –XX:NewRatio 来指定），即：新生代（Young）= 1/3 的堆空间大小。老年代（Old）= 2/3 的堆空间大小。其中，新生代（Young）被细分为 Eden 和 两个 Survivor 区域，这两个 Survivor 区域分别被命名为 from 和 to，以示区分。

默认的，Edem : from : to = 8 : 1 : 1（可以通过参数 –XX:SurvivorRatio 来设定），即： Eden = 8/10 的新生代空间大小，from = to = 1/10 的新生代空间大小。

JVM 每次只会使用 Eden 和其中的一块 Survivor 区域来为对象服务，所以无论什么时候，总是有一块 Survivor 区域是空闲着的。

因此，新生代实际可用的内存空间为 9/10（即 90%）的新生代空间

19. jvm，jre 以及 jdk 三者之间的关系？JDK(Java Development Kit) 是针对 Java 开发员的产品，是整个 Java 的核心，包括了 Java 运行环境 JRE、Java 工具和 Java 基础类库。

Java Runtime Environment（JRE）是运行 JAVA 程序所必须的环境的集合，包含 JVM 标准实现及 Java 核心类库。

JVM 是 Java Virtual Machine（Java 虚拟机）的缩写，是整个 java 实现跨平台的最核心的部分，能够运行以 Java 语言写作的软件程序。

20. 谈谈你对 JNIEnv 和 JavaVM 理解？

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

1.JavaVm

JavaVM 是虚拟机在 JNI 层的代表，一个进程只有一个 JavaVM，所有的线程共用一个 JavaVM。

2.JNIEnv

JNIEnv 表示 Java 调用 native 语言的环境，是一个封装了几乎全部 JNI 方法的指针。

JNIEnv 只在创建它的线程生效，不能跨线程传递，不同线程的 JNIEnv 彼此独立。

native 环境中创建的线程，如果需要访问 JNI，必须要调用 AttachCurrentThread 关联，并使用 DetachCurrentThread 解除链接。

21. Serializable 与 Parcable 的区别？

1.Serializable （java 自带）

方法：对象继承 Serializable 类即可实现序列化，就是这么简单，也是它最吸引我们的地方

2.Parcelable（Android 专用）：Parcelable 方式的实现原理是将一个完整的对象进行分解，用起来比较麻烦

1）在使用内存的时候，Parcelable 比 Serializable 性能高，所以推荐使用 Parcelable。

2）Serializable 在序列化的时候会产生大量的临时变量，从而引起频繁的 GC。

3）Parcelable 不能使用在要将数据存储在磁盘上的情况，因为 Parcelable 不能很好的保证数据的持续性,在外界有变化的情况下。尽管 Serializable 效率低点，但此时还是建议使用 Serializable 。

4）android 上应该尽量采用 Parcelable，效率至上，效率远高于 Serializable

ANDROID 面试常见 58 题

1、java 中==和 equals 和 hashCode 的区别

基本数据类型的==比较的值相等.

类的==比较的内存的地址，即是否是同一个对象，在不覆盖 equals 的情况下，同比较内存地址，原实现也为 == ，如 String 等重写了 equals 方法.

hashCode 也是 Object 类的一个方法。返回一个离散的 int 型整数。在集合类操作中使用，为了提高查询速度。（HashMap，HashSet 等比较是否为同一个）

如果两个对象 equals，Java 运行时环境会认为他们的 hashcode 一定相等。

如果两个对象不 equals，他们的 hashcode 有可能相等。

如果两个对象 hashcode 相等，他们不一定 equals。

如果两个对象 hashcode 不相等，他们一定不 equals。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

2、int 与 integer 的区别

int 基本类型

integer 对象 int 的封装类

3、String、StringBuffer、StringBuilder 区别

String:字符串常量 不适用于经常要改变值得情况，每次改变相当于生成一个新的对象

StringBuffer:字符串变量 （线程安全）

StringBuilder:字符串变量（线程不安全） 确保单线程下可用，效率略高于 StringBuffer

4、什么是内部类？内部类的作用

内部类可直接访问外部类的属性

Java 中内部类主要分为**成员内部类**、**局部内部类**(嵌套在方法和作用域内)、**匿名内部类**（没构造方法）、**静态内部类**（static 修饰的类，不能使用任何外围类的非 static 成员变量和方法， 不依赖外围类）

5、进程和线程的区别

进程是 cpu 资源分配的最小单位，线程是 cpu 调度的最小单位。

进程之间不能共享资源，而线程共享所在进程的地址空间和其它资源。

一个进程内可拥有多个线程，进程可开启进程，也可开启线程。

一个线程只能属于一个进程，线程可直接使用同进程的资源,线程依赖于进程而存在。

6、final，finally，finalize 的区别

final:修饰类、成员变量和成员方法，类不可被继承，成员变量不可变，成员方法不可重写

finally:与 try...catch...共同使用，确保无论是否出现异常都能被调用到

finalize:类的方法,垃圾回收之前会调用此方法,子类可以重写 finalize()方法实现对资源的回收

7、Serializable 和 Parcelable 的区别

Serializable Java 序列化接口 在硬盘上读写 读写过程中有大量临时变量的生成，内部执行大量的 i/o 操作，效率很低。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

Parcelable Android 序列化接口 效率高 使用麻烦 在内存中读写（AS 有相关插件 一键生成所需方法），对象不能保存到磁盘中

8、静态属性和静态方法是否可以被继承？是否可以被重写？以及原因？

可继承 不可重写 而是被隐藏

如果子类里面定义了静态方法和属性，那么这时候父类的静态方法或属性称之为"隐藏"。如果你想要调用父类的静态方法和属性，直接通过父类名.方法或变量名完成。

9、成员内部类、静态内部类、局部内部类和匿名内部类的理解，以及项目中的应用

ava 中内部类主要分为**成员内部类**、**局部内部类**(嵌套在方法和作用域内)、**匿名内部类**（没构造方法）、**静态内部类**（static 修饰的类，不能使用任何外围类的非 static 成员变量和方法， 不依赖外围类）

使用内部类最吸引人的原因是：每个内部类都能独立地继承一个（接口的）实现，所以无论外围类是否已经继承了某个（接口的）实现，对于内部类都没有影响。

因为 Java 不支持多继承，支持实现多个接口。但有时候会存在一些使用接口很难解决的问题，这个时候我们可以利用内部类提供的、可以继承多个具体的或者

抽象的类的能力来解决这些程序设计问题。可以这样说，接口只是解决了部分问题，而内部类使得多重继承的解决方案变得更加完整。

10、string 转换成 integer 的方式及原理

String integer Intrger.parseInt(string);

Integer string Integer.toString();

11、哪些情况下的对象会被垃圾回收机制处理掉？

1.所有实例都没有活动线程访问。

2.没有被其他任何实例访问的循环引用实例。

3.Java 中有不同的引用类型。判断实例是否符合垃圾收集的条件都依赖于它的引用类型。

要判断怎样的对象是没用的对象。这里有 2 种方法：

1.采用标记计数的方法：

给内存中的对象给打上标记，对象被引用一次，计数就加 1，引用被释放了，计数就减一，当这个计数为 0 的时候，这个对象就可以被回收了。当然，这也就引发了一个问题：循环引用的对象是无法被识别出来并且被回收的。所以就有了第二种方法：

2.采用根搜索算法：

从一个根出发，搜索所有的可达对象，这样剩下的那些对象就是需要被回收的

12、静态代理和动态代理的区别，什么场景使用？

静态代理类：

由程序员创建或由特定工具自动生成源代码，再对其编译。在程序运行前，代理类的.class 文件就已经存在了。动态代理类：在程序运行时，运用反射机制动态创建而成。

14、Java 中实现多态的机制是什么？

答：方法的重写 Overriding 和重载 Overloading 是 Java 多态性的不同表现

重写 Overriding 是父类与子类之间多态性的一种表现

重载 Overloading 是一个类中多态性的一种表现.

16、说说你对 Java 反射的理解

JAVA 反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性。从对象出发，通过反射（Class 类）可以取得取得类的完整信息（类名 Class 类型，所在

包、具有的所有方法 Method[]类型、某个方法的完整信息（包括修饰符、返回值类型、异常、参数类型）、所有属性 Field[]、某个属性的完整信息、构造器 Constructors），调用类的属性或方法自己的总结： 在运行过程中获得类、对象、方法的所有信息。

17、说说你对 Java 注解的理解

元注解

元注解的作用就是负责注解其他注解。java5.0 的时候，定义了 4 个标准的 meta-annotation 类型，它们用来提供对其他注解的类型作说明。

1.@Target

2.@Retention

3.@Documented

4.@Inherited

18、Java 中 String 的了解

在源码中 string 是用 final 进行修饰，它是不可更改，不可继承的常量。

19、String 为什么要设计成不可变的？

1、字符串池的需求

字符串池是方法区（Method Area）中的一块特殊的存储区域。当一个字符串已经被创建并且该字符串在 池 中，该字符串的引用会立即返回给变量，而不是重新创建一个字符串再将引用返回给变量。如果字符串不是不可变的，那么改变一个引用（如: string2）的字符串将会导致另一个引用（如: string1）出现脏数据。

2、允许字符串缓存哈希码

在 java 中常常会用到字符串的哈希码，例如： HashMap 。String 的不变性保证哈希码始终一，因此，他可以不用担心变化的出现。 这种方法意味着不必每次使用时都重新计算一次哈希码——这样，效率会高很多。

3、安全

String 广泛的用于 java 类中的参数，如：网络连接（Network connetion），打开文件（opening files ）等等。如果 String 不是不可变的，网络连接、文件将会被改变——这将会导致一系列的安全威胁。操作的方法本以为连接上了一台机器，但实际上却不是。由于反射中的参数都是字符串，同样，也会引起一系列的安全问题。

20、Object 类的 equal 和 hashCode 方法重写，为什么？

首先 equals 与 hashcode 间的关系是这样的：

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

1、如果两个对象相同（即用 equals 比较返回 true），那么它们的 hashCode 值一定要相同；

2、如果两个对象的 hashCode 相同，它们并不一定相同(即用 equals 比较返回 false)

由于为了提高程序的效率才实现了 hashcode 方法，先进行 hashcode 的比较，如果不同，那没就不必在进行 equals 的比较了，这样就大大减少了 equals 比较的次数，这对比需要比较的数量很大的效率提高是很明显的

21、List,Set,Map 的区别

Set 是最简单的一种集合。集合中的对象不按特定的方式排序，并且没有重复对象。 Set 接口主要实现了两个实现类：HashSet： HashSet 类按照哈希算法来存取集合中的对象，存取速度比较快

TreeSet ：TreeSet 类实现了 SortedSet 接口，能够对集合中的对象进行排序。

List 的特征是其元素以线性方式存储，集合中可以存放重复对象。

ArrayList() : 代表长度可以改变得数组。可以对元素进行随机的访问，向 ArrayList()中插入与删除元素的速度慢。

LinkedList(): 在实现中采用链表数据结构。插入和删除速度快，访问速度慢。

Map 是一种把键对象和值对象映射的集合，它的每一个元素都包含一对键对象和值对象。 Map 没有继承于 Collection 接口 从 Map 集合中检索元素时，只要给出键对象，就会返回对应的值对象。

HashMap：Map 基于散列表的实现。插入和查询“键值对”的开销是固定的。可以通过构造器设置容量 capacity 和负载因子 load factor，以调整容器的性能。

LinkedHashMap： 类似于 HashMap，但是迭代遍历它时，取得“键值对”的顺序是其插入次序，或者是最近最少使用(LRU)的次序。只比 HashMap 慢一点。而在迭代访问时发而更快，因为它使用链表维护内部次序。

TreeMap ： 基于红黑树数据结构的实现。查看“键”或“键值对”时，它们会被排序(次序由 Comparabel 或 Comparator 决定)。TreeMap 的特点在 于，你得到的结果是经过排序的。TreeMap 是唯一的带有 subMap()方法的 Map，它可以返回一个子树。

WeakHashMao ：弱键(weak key)Map，Map 中使用的对象也被允许释放: 这是为解决特殊问题设计的。如果没有 map 之外的引用指向某个“键”，则此“键”可以被垃圾收集器回收。

26、ArrayMap 和 HashMap 的对比

1、存储方式不同

HashMap 内部有一个 HashMapEntry<K, V>[]对象，每一个键值对都存储在这个对象里，当使用 put 方法添加键值对时，就会 new 一个 HashMapEntry 对象，

2、添加数据时扩容时的处理不一样，进行了 new 操作，重新创建对象，开销很大。ArrayMap 用的是 copy 数据，所以效率相对要高。

3、ArrayMap 提供了数组收缩的功能，在 clear 或 remove 后，会重新收缩数组，是否空间

4、ArrayMap 采用二分法查找；

29、HashMap 和 HashTable 的区别

1 HashMap 不是线程安全的，效率高一点、方法不是 Synchronize 的要提供外同步，有 containsvalue 和 containsKey 方法。

hashtable 是 ,线程安全 ,不允许有 null 的键和值 ,效率稍低 ,方法是是 Synchronize 的。有 contains 方法方法。Hashtable 继承于 Dictionary 类

30、HashMap 与 HashSet 的区别

hashMap:HashMap 实现了 Map 接口,HashMap 储存键值对,使用 put()方法将元素放入 map 中,HashMap 中使用键对象来计算 hashcode 值,HashMap 比较快，因为是使用唯一的键来获取对象。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

HashSet 实现了 Set 接口，HashSet 仅仅存储对象，使用 add()方法将元素放入 set 中，HashSet 使用成员对象来计算 hashcode 值，对于两个对象来说 hashcode 可能相同，所以 equals()方法用来判断对象的相等性，如果两个对象不同的话，那么返回 false。HashSet 较 HashMap 来说比较慢。

31、HashSet 与 HashMap 怎么判断集合元素重复？

HashSet 不能添加重复的元素，当调用 add（Object）方法时候，

首先会调用 Object 的 hashCode 方法判 hashCode 是否已经存在，如不存在则直接插入元素；如果已存在则调用 Object 对象的 equals 方法判断是否返回 true，如果为 true 则说明元素已经存在，如为 false 则插入元素。

33、ArrayList 和 LinkedList 的区别，以及应用场景

ArrayList 是基于数组实现的，ArrayList 线程不安全。

LinkedList 是基于双链表实现的：

使用场景：

（1）如果应用程序对各个索引位置的元素进行大量的存取或删除操作，ArrayList 对象要远优于 LinkedList 对象；

（2）如果应用程序主要是对列表进行循环，并且循环时候进行插入或者删除操作，LinkedList 对象要远优于 ArrayList 对象；

34、数组和链表的区别

数组：是将元素在内存中连续存储的；它的优点：因为数据是连续存储的，内存地址连续，所以在查找数据的时候效率比较高；它的缺点：在存储之前，我们需要申请一块连续的内存空间，并且在编译的时候就必须确定好它的空间的大小。在运行的时候空间的大小是无法随着你的需要进行增加和减少而改变的，当数据两比较大的时候，有可能会出现越界的情况，数据比较小的时候，又有可能会浪费掉内存空间。在改变数据个数时，增加、插入、删除数据效率比较低。

链表：是动态申请内存空间，不需要像数组需要提前申请好内存的大小，链表只需在用的时候申请就可以，根据需要来动态申请或者删除内存空间，对于数据增加和删除以及插入比数组灵活。还有就是链表中数据在内存中可以在任意的位置，通过应用来关联数据（就是通过存在元素的指针来联系）

35、开启线程的三种方式？

ava 有三种创建线程的方式，分别是继承 Thread 类、实现 Runable 接口和使用线程池

36、线程和进程的区别？

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。别把它和栈内存搞混，每个线程都拥有单独的栈内存用来存储本地数据。

38、run()和 start()方法区别

这个问题经常被问到，但还是能从此区分出面试者对 Java 线程模型的理解程度。start()方法被用来启动新创建的线程，而且 start()内部调用了 run()方法，这和直接调用 run()方法的效果不一样。当你调用 run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。

39、如何控制某个方法允许并发访问线程的个数？

semaphore.acquire() 请求一个信号量，这时候的信号量个数-1（一旦没有可使用的信号量，也即信号量个数变为负数时，再次请求的时候就会阻塞，直到其他线程释放了信号量）

semaphore.release() 释放一个信号量，此时信号量个数+1

40、在 Java 中 wait 和 seelp 方法的不同；

Java 程序中 wait 和 sleep 都会造成某种形式的暂停，它们可以满足不同的需要。wait()方法用于线程间通信，如果等待条件为真且其它线程被唤醒时它会释放锁，

而 sleep()方法仅仅释放 CPU 资源或者让当前线程停止执行一段时间，但不会释放锁。

41、谈谈 wait/notify 关键字的理解

等待对象的同步锁,需要获得该对象的同步锁才可以调用这个方法,否则编译可以通过，但运行时会收到一个异常：IllegalMonitorStateException。

调用任意对象的 wait() 方法导致该线程阻塞，该线程不可继续执行，并且该对象上的锁被释放。

唤醒在等待该对象同步锁的线程(只唤醒一个,如果有多个在等待),注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由 JVM 确定唤醒哪个线程，而且不是按优先级。

调用任意对象的 notify()方法则导致因调用该对象的 wait()方法而阻塞的线程中随机选择的一个解除阻塞（但要等到获得锁后才真正可执行）。

42、什么导致线程阻塞？线程如何关闭？

阻塞式方法是指程序会一直等待该方法完成期间不做其他事情，ServerSocket 的 accept()方法就是一直等待客户端连接。这里的阻塞是指调用结果返回之前，当前线程会被挂起，直到得到结果之后才会返回。此外，还有异步和非阻塞式方法在任务完成前就返回。

一种是调用它里面的 stop()方法

另一种就是你自己设置一个停止线程的标记 （推荐这种）

43、如何保证线程安全？

1.synchronized；

2.Object 方法中的 wait,notify；

3.ThreadLocal 机制 来实现的。

44、如何实现线程同步？

1、synchronized 关键字修改的方法。2、synchronized 关键字修饰的语句块 3、使用特殊域变量（volatile）实现线程同步

45、线程间操作 List

List list = Collections.synchronizedList(new ArrayList());

46、谈谈对 Synchronized 关键字，类锁，方法锁，重入锁的理解

java 的对象锁和类锁：java 的对象锁和类锁在锁的概念上基本上和内置锁是一致的，但是，两个锁实际是有很大的区别的，对象锁是用于对象实例方法，或者一个对象实例上的，类锁是用于类的静态方法或者一个类的 class 对象上的。我们

知道，类的对象实例可以有很多个，但是每个类只有一个 class 对象，所以不同对象实例的对象锁是互不干扰的，但是每个类只有一个类锁。但是有一点必须注意的是，其实类锁只是一个概念上的东西，并不是真实存在的，它只是用来帮助我们理解锁定实例方法和静态方法的区别的

49、synchronized 和 volatile 关键字的区别

1.volatile 本质是在告诉 jvm 当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取；synchronized 则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。

2.volatile 仅能使用在变量级别；synchronized 则可以使用在变量、方法、和类级别的

3.volatile 仅能实现变量的修改可见性，不能保证原子性；而 synchronized 则可以保证变量的修改可见性和原子性

4.volatile 不会造成线程的阻塞；synchronized 可能会造成线程的阻塞。

5.volatile 标记的变量不会被编译器优化；synchronized 标记的变量可以被编译器优化

51、ReentrantLock 、synchronized 和 volatile 比较

ava 在过去很长一段时间只能通过 synchronized 关键字来实现互斥，它有一些缺点。比如你不能扩展锁之外的方法或者块边界，尝试获取锁时不能中途取消等。Java 5 通过 Lock 接口提供了更复杂的控制来解决这些问题。 ReentrantLock 类实现了 Lock，它拥有与 synchronized 相同的并发性和内存语义且它还具有可扩展性。

53、死锁的四个必要条件？

死锁产生的原因

1. 系统资源的竞争

系统资源的竞争导致系统资源不足，以及资源分配不当，导致死锁。

2. 进程运行推进顺序不合适

互斥条件：一个资源每次只能被一个进程使用，即在一段时间内某 资源仅为一个进程所占有。此时若有其他进程请求该资源，则请求进程只能等待。

请求与保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源 已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。

《Android 架构师面试题精编解析大全》

不可剥夺条件:进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能 由获得该资源的进程自己来释放（只能是主动释放)。

循环等待条件: 若干进程间形成首尾相接循环等待资源的关系

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。

死锁的避免与预防：

死锁避免的基本思想:

系统对进程发出每一个系统能够满足的资源申请进行动态检查,并根据检查结果决定是否分配资源,如果分配后系统可能发生死锁,则不予分配,否则予以分配。这是一种保证系统不进入死锁状态的动态策略。

理解了死锁的原因，尤其是产生死锁的四个必要条件，就可以最大可能地避免、预防和解除死锁。所以，在系统设计、进程调度等方面注意如何让这四个必要条件不成立，如何确定资源的合理分配算法，避免进程永久占据系统资源。此外，也要防止进程在处于等待状态的情况下占用资源。因此，对资源的分配要给予合理的规划。

死锁避免和死锁预防的区别：

死锁预防是设法至少破坏产生死锁的四个必要条件之一,严格的防止死锁的出现,而死锁避免则不那么严格的限制产生死锁的必要条件的存在,因为即使死锁的必要条件存在,也不一定发生死锁。死锁避免是在系统运行过程中注意避免死锁的最终发生。

56、什么是线程池，如何使用?

创建线程要花费昂贵的资源和时间 ,如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从 JDK1.5 开始，Java API 提供了 Executor 框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。

57、Java 中堆和栈有什么不同？

为什么把这个问题归类在多线程和并发面试题里？因为栈是一块和线程紧密相关的内存区域。每个线程都有自己的栈内存，用于存储本地变量，方法参数和栈调用，一个线程中存储的变量对其它线程是不可见的。而堆是所有线程共享的一片公用内存区域。对象都在堆里创建，为了提升效率线程会从堆中弄一个缓存到自己的栈，如果多个线程使用该变量就可能引发问题，这时 volatile 变量就可以发挥作用了，它要求线程从主存中读取变量的值。

58、有三个线程 T1，T2，T3，怎么确保它们按顺序执行？

在多线程中有多种方法让线程按特定顺序执行，你可以用线程类的 join()方法在一个线程中启动另一个线程，另外一个线程完成该线程继续执行。为了确保三个线程的顺序你应该先启动最后一个(T3 调用 T2，T2 调用 T1)，这样 T1 就会先完成而 T3 最后完成。

线程间通信

我们知道线程是 CPU 调度的最小单位。在 Android 中主线程是不能够做耗时操作的，子线程是不能够更新 UI 的。而线程间通信的方式有很多，比如广播，Eventbus，接口回掉，在 Android 中主要是使用 handler。handler 通过调用 sendmessage 方法，将保存消息的 Message 发送到 Messagequeue 中，而 looper 对象不断的调用 loop 方法，从 messageueue 中取出 message，交给 handler 处理，从而完成线程间通信。

第三章：开源框架实战

一、 自我介绍（★★★★★）

从姓名，工作多长时间，及项目开发的周期来说。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

（1）姓名

（2）工作时间

（3）开发周期

例如， 你好，我是 XXX，到现在为止我工作了一年半，期间有做过三款 app。最近的一个项目开发了两个月。（这时候可以把你准备好的手机里的 app 打开给面试官看）

二、 开发中都使用过哪些框架、平台（★★★★★）

1. EventBus（事件处理）

2. xUtils（网络、图片、ORM）

xUtils 分为四大模块：

① DbUtils 模块：Android 中的 orm 框架（对象关系映射，它的作用是在关系型数据库和业务实体对象之间作为一个映射），一行代码就可以进行增删改查。（Logo 新闻内容缓存到数据库 当没有网络的时候）

② ViewUtils 模块：android 中的 ioc 框架(生命周期由框架控制)，完全注解的方式就可以进行对 UI 绑定和事件的绑定。

③ HttpUtils 模块：（请求服务器 客户端 传过去标示 head=”md5” ）

a. 支持同步，异步方式的请求。

b. 支持大文件上传，上传大文件不会 oom(内存溢出)。

c. 支持 GET,POST,DELETE 请求。

④ BitmapUtil 模块：

可以先说下三级缓存的原理：

1. 从缓存中加载。
2. 从本地文件中加载（数据库，SD）
3. 从网络加载。

a.加载 bitmap 的时候无需考虑 bitmap 加载过程中出现的 oom(内存溢出)和 android 容器快速滑动的时候出现的图片错位等现象。（16M）

b. 支持加载网络图片和本地图片。

c. 内存管理使用的 lru 算法（移除里面是有频率最少的对象），更好的管理 bitmap 的内存。

d.可配置线程加载的数量，缓存的大小，缓存的路径，加载显示的动画等。

清除缓存是怎么做的？

（1）清除内存的缓存。

（2）数据库，SD。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

注：需要添加一下权限<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

3. JPush（推送平台）

推送的好处：

a. 及时主动性。（这是推送服务最基本的特点，即当有新的信息需要提交时，依据传送信息的类型和重要性不同，推送软件会主动提醒用户接收新信息。从而提高了用户获取信息的及时性。）

b. 针对目的性。（推送服务提供的信息是根据用户的特定需求定的，这充分体现了用户的个性化需求。这种个性化的服务还是动态的，用户只需在定制之初描述信息需求，推送软件就会自动跟踪用户的使用倾向，实时地完成特定信息的推送。）

c. 便捷高效性。（用户只需输入一次信息请求，就可获得连续的信息服务。推送服务还采用信息代理机制，可以自动跟踪用户的信息需求。这样的推送服务既节省了用户主动拉取的时间，又减少了冗余信息的传递提高了信息的匹配度，从而大大方便了用户，提高了效率。）

我们在项目中主要使用的是极光推送，在极光的官网里（https://www.jpush.cn/）下载 android 的 demo，将 demo 中的 aapid 换成自己申请的，测试推送，然后集成到自己的项目中去。

4. 友盟（统计平台）

5. 有米（优米）（广告平台）

6. 百度地图

1) 下载百度地图移动版 API(Android)开发包

要在 Android 应用中使用百度地图 API，就需要在工程中引用百度地图 API 开发包，这个开发包包含两个文件：baidumapapi.jar 和 libBMapApiEngine.so。

下载地址：

http://dev.baidu.com/wiki/static/imap/files/BaiduMapApi_Lib_Android_1.0.zip

2) 申请 API Key

和使用 Google map api 一样，在使用百度地图 API 之前也需要获取相应的 API Key。百度地图 API Key 与你的百度账户相关联，因此您必须先有百度帐户，才能获得 API Key；并且，该 Key 与您引用 API 的程序名称有关。

百度 API Key 的申请要比 Google 的简单多了，其实只要你有百度帐号，应该不超过 30 秒就能完成 API Key 的申请。申请地址：http://dev.baidu.com/wiki/static/imap/key/

3) 创建一个 Android 工程

这里需要强调一点：百度地图移动版 api 支持 Android 1.5 及以上系统，因此我们创建的工程应基于 Android SDK 1.5 及以上。

工程创建完成后，将 baidumapapi.jar 和 libBMapApiEngine.so 分别拷贝到工程的根目录及 libs/armeabi 目录下，并在工程属性->Java Build Path->Libraries 中选择“Add JARs”，选定 baidumapapi.jar，这样就可以在应用中使用百度地图 API 了。

7. bmob（服务器平台、短信验证、邮箱验证、第三方支付）

8.阿里云 OSS（云存储）

9.ShareSDK（分享平台、第三方登录）

SDK 简介： ShareSDK 是为 iOS 的 APP 提供社会化功能的一个组件，开发者只需 10 分钟即可集成到自己的 APP 中，它不仅支持如 QQ、微信、新浪微博、腾讯微博、开心网、人人网、豆瓣、网易微博、搜狐微博、facebook、twitter、google+等国内外主流社交平台，还有强大的统计分析管理后台，可以实时了解用户、信息流、回流率、传播效应等数据，有效的指导日常运营与推广，同时为 APP 引入更多的社会化流量。

主要功能：

a. 支持分享到主流的各大平台上。（国内主要的分享平台：QQ ，微信 ，新浪微博 ，腾讯微博 国外的：facebook twitter google+ ）

b. 支持获取授权用户资料及其他用户资料，可以通过 sdk 制作使用新浪微博登录，QQ 登录等。

c. 支持关注官方微博，支持@好友，插入话题，图片。

d. 支持一键分享，用户可以一次性将内容分享至全部的社交平台。

使用：

（1）获取 SharedSDK。（SharedSDK 官网：http://wiki.mob.com/）

（2）将 SharedSDK 集成（导入）到项目的 libs 目录下。

（3）配置 AndroidManifest.xml 权限

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

（4）添加代码，启动 SDK。SMSSDK.initSDK(**this**, "<您的 appkey>", "<您的 appsecret>");

10. Gson（解析 json 数据框架）

根据服务器返回的 Gson 数据来设计类的模型，让 Gson 解析字符串为对应的对象模型。简单来讲就是　根据 json 的数据结构定义出相应的 javabean --->"new"出 Gson 的实例

gson---->gson.fromJson(jsonString,JavaBean.class) 即可.

面试概要：

可以先说下 Gson 的作用，然后在向后拓展下。（Gson 呢，是 google 提供的一个快速解析 json 数据的开源框架,原来我们解析数据的时候都是 jsonObject jsonArray 一层层解析，我发现这样层层解析很浪费时间，于是我在业余时间研究了 Gson，Gson 满足了我们快速开发的特性，只要从服务器拿到 json 数据用 Gson 解析，Gson 就会返回一个数据对象，我们就可以直接对数据进行操作了。原来解析可能需要十几分钟的事，现在两三分钟就搞定了）

补充：为什么数据要以 json 形式传输？

① 易读性

② 高效率

11.imageLoader （图片处理框架）

目前主流的图片缓存处理框架，只要配置相关参数即可。

12.zxing （二维码扫描）

三、 都使用过哪些自定义控件（★★★★）

1．pull2RefreshListView

2．LazyViewPager

3．SlidingMenu

4．SmoothProgressBar

5．自定义组合控件

6．ToggleButton

7．自定义吐司（Toast）

四、 自定义控件：绘制圆环的实现过程（★★）

使用自定义控件绘制一个圆环,需要重现的方法是 OnDraw()实现对 view 的绘制，从而输出符合自己需求的 view 控件

观察圆环的组成部分:

外层圆+中间百分比文字+不断变化进度的弧形圈

--->分析:每一个组成部分需要的属性,构成几个关键的自定义属性

1:外层圆的颜色

2:弧形进度圈的颜色

3:中间百分比文字的颜色

4:中间百分比文字的大小

5:圆环的宽度(作为进度弧形圈的宽度)

6:*首页当中也有一个圆环进度 ,为了兼容使用首页的圆环进度,增加一个自定义属性，绘制进度弧形圈的风格(实心[Fill]，空心[Stroken])

分析完毕-->绘制步骤：

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

OKHTTP 的实际开发中的用法

1:构造方法当中初始化画笔对象，获取自定义的属性值.

2:重写 Ondraw 方法

---2.1:绘制最外层的圆

-关键方法 canvas.drawCircle(center, center, radius, paint); //画出圆环

*:计算半径、中心点坐标、画笔设置

paint.setColor(roundColor); //设置圆环的颜色

paint.setStyle(Paint.Style.STROKE); //设置空心

paint.setStrokeWidth(roundWidth); //设置圆环的宽度---这个宽度也是提供给进度弧形圈绘制的时候覆盖的宽度

paint.setAntiAlias(true); //消除锯齿

中心点坐标

int center = getWidth() / 2; //获取圆心的 x 坐标

半径:

int radius = (int) (center - roundWidth/2) ---画图说明最容易理解

---2.2:绘制中间的百分比文字

--关键方法：canvas.drawText(percent + "%", center - textWidth / 2, center + textSize / 2, paint); //画出进度百分比

测量画笔上的文本宽度

Float textWidth =

paint.measureText(percent + "%");

画笔设置

paint.setStrokeWidth(0);

paint.setColor(textColor);

paint.setTextSize(textSize);

paint.setTypeface(Typeface.DEFAULT_BOLD); //设置字体

绘制的文字的位置,由参数 2,3 的 X,Y 坐标值决定--圆环的中心点位置显示

X:表示从哪开始绘制,如果你直接中心点开始绘制-->画图说明最容易理解

-->正确的 X=center - textWidth / 2；Y = center + textSize / 2 --（因为 android 坐标系与数学坐标系 Y 轴值是相反的，也可以画图说明,这里的 textsize 就可以代表高度,paint.measureText 测量方法执行之后，默认的文字

高度就是根据文字大小计算的，相当于 wrap_content，所以 textSize 就是本身文字所占的高度值）

*:绘制的进度要转换为百分比形式：int percent = (int) (((float) progress / (float) max) * 100);

---2.3:绘制进度弧形圈

---关键方法：canvas.drawArc(oval, 0, 360 * progress / max, false, paint); //根据进度画圆弧

参数解释：

oval：绘制的弧形的范围轮廓

0:从多少角度开始绘制

360 * progress / max:绘制弧形扫过的角度对应的区域

false:不包含圆心，如果是 true，表示包含圆心

paint:绘制使用的画笔

画笔设置

paint.setStrokeWidth(roundWidth); //设置进度弧形圈的宽度，必须保持和外层圆的 StrokeWidth 一致，确保弧形圈绘制的时候覆盖的范围就是外层圆的宽度

paint.setColor(roundProgressColor); //设置进度的颜色

弧形范围计算

RectF oval = new RectF(center - radius, center - radius, center + radius, center + radius); --->画图说明最容易理解

left:center - radius

top:center-radius

right:center+radius

bottom:center+radius

*:注意，因为 progress 是相对于 100 当中占比多少，而弧形总共是按照角度分成 360 分的，所以绘制弧形圈指定参数扫过的区域角度需要计算转换一下

=360 * progress / max（max=100）

*:对应自定义属性中的最后一个属性是实心还是空心，绘制的时候需要判断一下,兼容首页的圆环进度处理

```
    switch (style) {
                case STROKE:{
                    paint.setStyle(Paint.Style.STROKE);
                    canvas.drawArc(oval,  0,  360  *
progress / max, false, paint);  //根据进度画圆弧
                    break;
                }
```

```
                case FILL:{

paint.setStyle(Paint.Style.FILL_AND_STROKE);
                    if(progress !=0)
                        canvas.drawArc(oval, 0, 360 *
progress / max, true, paint);  //根据进度画圆弧
                    break;
                }
            }
```

最后提供一个设置进度，根据进度重新绘制圆环的方法

```
public void setProgress(int progress) {
            if(progress < 0){
                throw                                   new
IllegalArgumentException("progress not less than
0");
            }
            if(progress > max){
                progress = max;
            }
```

```
            if(progress <= max){
                this.progress = progress;
                postInvalidate();
            }
}
```

五、 自定义控件：摩天轮的实现过程（★★）

① 摩天轮控件是可以通过触摸旋转的 ,但旋转的过程中保持子 View 的方向。

② 摩天轮控件中触摸旋转，惯性转动的基础是自控件的摆放，使用了三角函数来确定子 view 的中心点位置，注意在分析阶段我们把摩天轮的中心点作为参考点(0,0)，在写代码的时候，记得偏移到控件左上方。

③ 在触摸旋转和惯性旋转时，我们需要改变所有的孩子的位置，其实也就是孩子中心点与圆心连线的角度，但因为角度间隔相等，所以只需要改变第一个孩子的角度，然后其他孩子与第一个孩子保持角度间隔即可，然后调用 requestLayout 重新摆放孩子即可。

④ 触摸旋转和惯性转动，都使用了 GestureDector 帮助我们判断触摸事件的类型，以及提供给我们所需要的参数。

⑤ 在触摸旋转中，我们通过计算当前点的角度和上一次点的角度，再算出两点的差，就可以知道需要旋转多少角度了。

⑥ 比较难的地方就是把惯性滑动的像素速度转化为角速度，对于没有学过微积分的同学可能想不到。然后我们使用了值动画不断地改变角度来模拟转动，又使用了减速插值器来模拟减速。

⑦ 因为摩天轮的触摸事件处理都是写在了 onTouchEven，t 方法中，如果子控件消费了触摸事件，会导致摩天轮的 onToucheEvent 方法没有被调用，所以我们需要在 onInterceptTouchEvent 方法中返回 ture，然后再检测单击事件发送给被点击的孩子。

六、 自定义控件：可拖拽排序的 GridLayout 的实现过程（★★）

⑧ 为什么没有使用 GridView，而是使用了 GridLayout；

因为 GridView 必然需要一个 Adapter，而 Adapter 需要数据集，如果在拖动中想要处理好三者的关系，在编码实现上具有一定的难度。

GridLayout 是安卓 4.0 出现的一个新的布局，不需要适配器即可让每个子 View 占据一定的格子，需要注意的是 GridLayout 的孩子可以横向、纵向地合并单元格。

⑨ 布局动画是什么？

可详见 http://www.cnblogs.com/mengdd/p/3305973.html

布局动画可帮助我们非常简单地处理 ViewGroup 的子控件的出现、消失、移动，是安卓 3.0 后出现的一个类。

⑩ Drag Drop 框架是怎么回事？

DragDrop 框架是安卓 3.0 后出现的，用户可以通过此框架以控件的形式移动“数据”，其重要的方法有两个，都是 view 上的：startDrag 和 setOnDragListener。

startDrag 方法要传递 4 个参数，但最重要的就是第二个 ShadowBuilder，用于构建跟随手指移动的影子，一般可在创建 ShadowBuilder 对象时，传入一个 view 作为影子的原形即可；

当产生影子后，安卓系统会发送 DragEvent 给当前应用中所有可见的 View，如果想要接收 DragEvent，可通过 View 的 setOnDragListener 方法，设置一个拖拽监听；

DragEvent 有 6 种类型，分别是开始，结束、进入、退出、移动、放下。需要注意的是，如果影子的中心点在设置了监听的控件之外移动或放下，是不会收到相应事件的。

在不同的事件类型中，还可以获得不同的数据，如图：

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

getAction() value	getClipDescription() value	getLocalState() value	getX() value	getY() value	getClipData() value	getResult() value
ACTION_DRAG_STARTED	X	X	X	X		
ACTION_DRAG_ENTERED	X	X	X	X		
ACTION_DRAG_LOCATION	X	X	X	X		
ACTION_DRAG_EXITED	X	X				
ACTION_DROP	X	X	X	X	X	
ACTION_DRAG_ENDED	X	X				X

其中，x 和 y 是对于对设置监听的控件的左上角的位置，而 Clip 数据是 startDrag 的第一个参数的数据，LocalState 是 startDrag 的第三个数据，Result 是 Drop 事件的返回值。

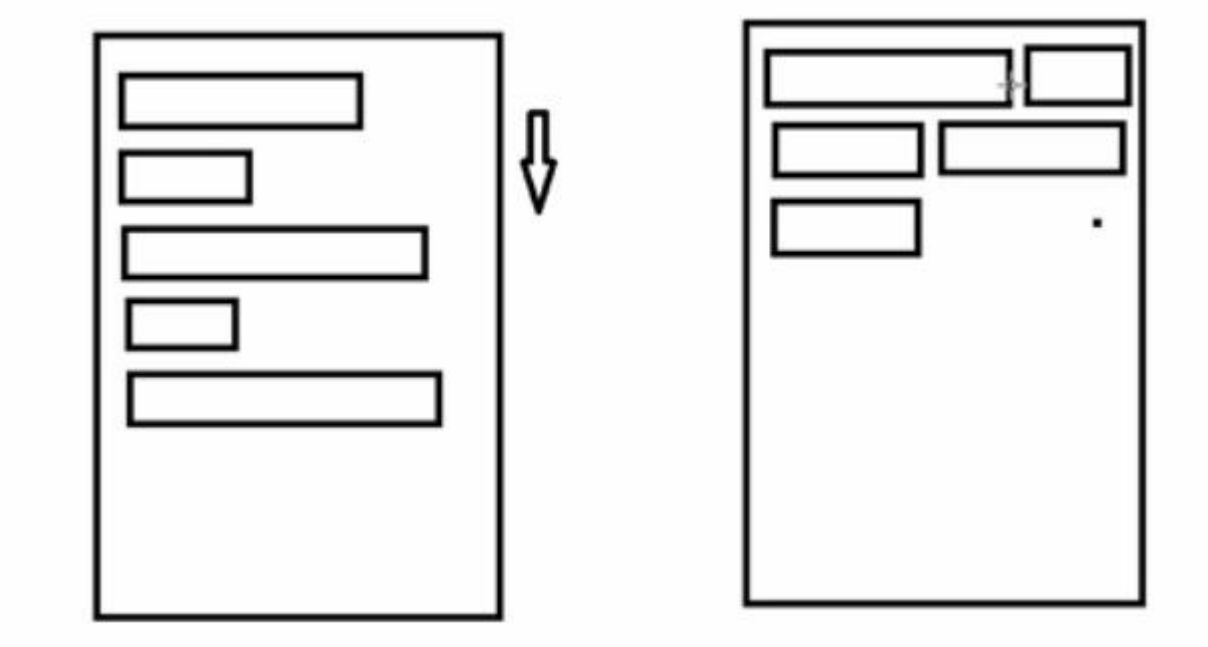
七、 流式布局的实现过程（★★★★）

流式布局的特点以及应用场景

特点：当上面一行的空间不够容纳新的 TextView 时候，

才开辟下一行的空间

原理图：



场景：主要用于关键词搜索或者热门标签等场景

2.自定义 ViewGroup,重点重写下面两个方法

1) onMeasure:测量子 view 的宽高，设置自己的宽和高

2) onLayout:设置子 view 的位置

onMeasure:根据子 view 的布局文件中属性 ,来为子 view 设置测量模式和测量值

测量=测量模式+测量值；

测量模式有 3 种：

EXACTLY：表示设置了精确的值，一般当 childView 设置其宽、高为精确值、match_parent 时，ViewGroup 会将其设置为 EXACTLY；

AT_MOST：表示子布局被限制在一个最大值内，一般当 childView 设置其宽、高为 wrap_content 时，ViewGroup 会将其设置为 AT_MOST；

UNSPECIFIED :表示子布局想要多大就多大 ,一般出现在 AadapterView 的 item 的 heightMode 中、ScrollView 的 childView 的 heightMode 中；此种模式比较少见

4) LayoutParams

ViewGroup LayoutParams :每个 ViewGroup 对应一个 LayoutParams; 即 ViewGroup -> LayoutParams.

getLayoutParams 不知道转为哪个对应的 LayoutParams ,其实很简单 ,就是如下：

子View.getLayoutParams 得到的LayoutParams对应的就是 子View 所在的父控件的 LayoutParams;

例如，LinearLayout 里面的子 view.getLayoutParams ->LinearLayout.LayoutParams

所以 咱们的 FlowLayout 也需要一个 LayoutParams ,由于上面的效果图是子 View 的 margin，

所以应该使用 MarginLayoutParams。即 FlowLayout->MarginLayoutParams

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

八、 项目的流程（★★★★★）

这个在项目中不一定问，但是大家要知道项目开发的流程，作为有经验的程序员程序的流程是一定知道的。

按时间轴来排列：

立项：确定项目、负责人、开发的周期、成本、人力、物力

需求：文档、原型图

开发：编码

测试：测试人员

上线：产品部门

维护：修复新的 bug

升级:改版、添加新的功能

九、 项目中常见面试题（2019.5.5 更新）

一、网络篇（★）

1．什么是 socket?,TCP 和 UDP 每个数据包的大小？

(2016.5.5)

Socket 通常也称作“套接字”，用于描述 IP 地址和端口，是一个通信连的句柄，应用程序通常通过“套接字”向网络发送请求或者应答网络请求，它就是网络通

信过程中端点的抽象表示。它主要包括 TCP，UDP 两个协议。

UDP 包的大小是 1492 - IP 头(20) - UDP 头(8) = 1464 字节

TCP 包的大小是 1492 - IP 头(20) - TCP 头(20) = 1452 字节

2．用 volley 如何下载数据较大的文件 Volley 的二次封装是怎样做的(2019.5.5)（★★）

（1） Volley 本身不支持大数据文件的下载，如果必须使用的 volley 进行下载，可进行断点下载；将每次的下载文件尽可能的切割成更小的文件进行下载。

（2） 二次封装

① 直接封装一个数据接口，方便我们在 Activity 中直接得到网络请求的结果数据。

② 封装一个 JsonObject 请求结果的泛型封装类，方便对不同的实体类的解析。

③ 封装 Volley 请求方法，包括 GET，POST 等主流请求方式，及 json 文件的解析方法。最后，在 Activity 中的调用。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

3．大文件上传到服务器是怎么做到的(2016.5.5)（★★）

利用文件分割将大文件分割为小文件可以解决问题；android 端和服务端存在 RandomAccessFile 随机访问文件类，方便对文件进行分割。

4．如何实现文件断点上传（★★★★★）

在 Android 中上传文件可以采用 HTTP 方式 ,也可以采用 Socket 方式，但是 HTTP 方式不能上传大文件，这里介绍一种通过 Socket 方式来进行断点续传的方式，服务端会记录下文件的上传进度，当某一次上传过程意外终止后，下一次可以继续上传，这里用到的其实还是 J2SE 里的知识。

这个上传程序的原理是：客户端第一次上传时向服务端发送

“Content-Length=35;filename=WinRAR_3.90_SC.exe;sourceid= “这种格式的字符串，服务端收到后会查找该文件是否有上传记录，如果有就返回已经上传的位置，否则返回新生成的 sourceid 以及 position 为 0，类似 sourceid=2324838389;position=0“这样的字符串，客户端收到返回后的字符串后再从指定的位置开始上传文件。

5．XUtils 和 Volley 的区别在哪里？(2019.5.5)（★★★）

Volley: Volley 是 Andrdoid 平台的网络通讯库，能使网络通讯更快，更简单，更健壮；默认 Android2.3 及以上基于 HttpURLConnection，2.3 以下

使用基于 HttpClient；适用于数据量小，网络通信频繁的操作。

Xutils：源于 Afinal 框架，对 Afinal 进行了大量重构，使得 xUtils 支持大文件上传，更全面的 http 请求协议支持，拥有更加灵活的 ORM，更多的事件注解支持且不受混淆影响，同时 xutils 集成了较多的工具类。

6．Tcp/Ip 及 UDP 的差别在哪里？ 平常中我们是如何来进行使用的？(2019.5.5)（★★）

TCP 协议和 UDP 协议特性区别总结：

（1）TCP 是面向连接(Connection oriented)的协议，UDP 是无连接(Connection less)协议；

（2）TCP 可靠，UDP 不可靠；

（3）TCP 有序，UDP 无序；

（4） TCP 无界，UDP 有界；

（5）TCP 有流量控制（拥塞控制），UDP 没有；

（6）TCP 传输慢，UDP 传输快；

（7）TCP 是重量级的，UDP 是轻量级的

（8） TCP 的头部比 UDP 大；

使用：

TCP 一般用于文件传输（FTP HTTP 对数据准确性要求高，速度可以相对慢），发送或接收邮件（POP IMAP SMTP 对数据准确性要求高，非紧急应用），远程登录（TELNET SSH 对数据准确性有一定要求，有连接的概念）等等，TCP 可以用于网络数据库，分布式高精度计算系统的数据传输。

UDP 一般用于即时通信（QQ 聊天对数据准确性和丢包要求比较低，但速度必须快），在线视频（RTSP 速度一定要快，保证视频连续，但是偶尔花了一个图像帧，人们还是能接受的），网络语音电话（VoIP 语音数据包一般比较小，需要高速发送，偶尔断音或串音也没有问题）等等；UDP 还用于服务系统内部之间的数据传输，因为数据可能比较多，内部系统局域网内的丢包错包率又很低，即便丢包，顶多是操作无效，这种情况下，UDP 经常被使用。

7．你了解 Android 网络的重试机制，以及 Android 的重试机制，你怎么看？(2019.5.5)（★）

因为经常需要跟网络编程打交道，并不是你的每次请求，服务端都会给你想要的结果。重试机制虽然并不能解决这种情况，但是却可以大大减少这种情况的发生。

8．给我说说 Http？(2019.5.6)（★）

网络由下往上分为物理层、数据链路层、网络层、传输层、会话层、表示层和应

用层。我知道 IP 协议对应于网络层，TCP 协议对应于传输层，而 HTTP 协议对应于应用层，那么 http 的是负责主要解决如何包装数据，我们用的是 http1.1 版本而不是 1.0 版本，因为 1.0 他是非持续链接，android App 一般是获取数据和展示数据，数据交互一般使用 PUt 和 Post 请求，然后说一下二者区别。

9．sqlite 数据库去重复怎么做?(2019.5.23) (★★)

在列名前加 **distinct** 即可去重复，示例如下：

```
Select distinct * from XXX；
```

二、框架搭建

10．讲下怎么搭建框架(2019.5.5)

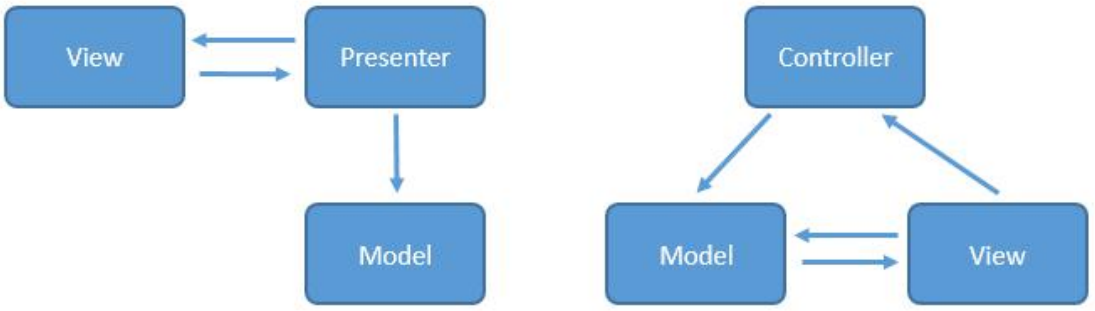
这个面试题一般是独立开发都会问的，这里可以参考我的“怎样构建 Android App（小白篇）”

11．简述一下 MVC 和 MVP 这两种设计模式，说说各自的特点 (2019.5.5) (★★★★★)

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

MVP和MVC对比图



MVP模式　　MVC模式

在 MVC 里，View 是可以直接访问 Model 的！从而，View 里会包含 Model 信息，不可避免的还要包括一些业务逻辑。 在 MVC 模型里，更关注的 Model 的不变，而同时有多个对 Model 的不同显示，及 View。所以，在 MVC 模型里，Model 不依赖于 View，但是 View 是依赖于 Model 的。不仅如此，因为有一些业务逻辑在 View 里实现了，导致要更改 View 也是比较困难的，至少那些业务逻辑是无法重用的。

MVP 的优点：

1、模型与视图完全分离，我们可以修改视图而不影响模型；

2、可以更高效地使用模型，因为所有的交互都发生在一个地方——Presenter 内部；

3、我们可以将一个 Presenter 用于多个视图，而不需要改变 Presenter 的逻辑。这个特性非常的有用，因为视图的变化总是比模型的变化频繁；

4、如果我们把逻辑放在 Presenter 中，那么我们就可以脱离用户接口来测试这

些逻辑（单元测试）

MVP 主要解决就是把逻辑层抽出来成 P 层，要是遇到需求逻辑上的更改就可以只需要修改 P 层了或者遇到逻辑上的大概我们可以直接从写一个 P 也可以，很多开发人员把所有的东西都写在了 Activity/Fragment 里面这样一来遇到频繁改需求或者逻辑越来越复杂的时候 ,Activity /Fragment 里面就会出现过多的混杂逻辑导致出错，所以 MVP 模式对于 APP 来对控制逻辑和 UI 的解耦来说是一个不错的选择！

12．说说 mvc 模式的原理，它在 android 中的运用。（★★★★）

MVC 英文即 Model-View-Controller，即把一个应用的输入、处理、输出流程按照 Model、View、Controller 的方式进行分离，这样一个应用被分成三个层——模型层、视图层、控制层。

Android 中界面部分也采用了当前比较流行的 MVC 框架，在 Android 中 M 就是应用程序中二进制的数据，V 就是用户的界面。Android 的界面直接采用 XML 文件保存的，界面开发变的很方便。在 Android 中 C 也是很简单的，一个 Activity 可以有多个界面，只需要将视图的 ID 传递到 setContentView()，就指定了以哪个视图模型显示数据。

在 Android SDK 中的数据绑定，也都是采用了与 MVC 框架类似的方法来

显示数据。在控制层上将数据按照视图模型的要求（也就是 Android SDK 中的 Adapter）封装就可以直接在视图模型上显示了，从而实现了数据绑定。比如显示 Cursor 中所有数据的 ListActivity，其视图层就是一个 ListView，将数据封装为 ListAdapter，并传递给 ListView，数据就在 ListView 中显示。

另一种方式解答：

a.模型（model）对象：是应用程序的主体部分，所有的业务逻辑都应该写在该层。

b. 视图（view）对象：是应用程序中负责生成用户界面的部分。也是在整个 mvc 架构中用户唯一可以看到的一层，接收用户的输入，显示处理结果。

c.控制器（control）对象：是根据用户的输入，控制用户界面数据显示及更新 model 对象状态的部分

View：自定义 View 或 ViewGroup，负责将用户的请求通知 Controller，并根据 model 更新界面；

Controller：Activity 或者 Fragment，接收用户请求并更新 model；

Model：数据模型，负责数据处理相关的逻辑，封装应用程序状态，响应状态查询，通知 View 改变，对应 Android 中的 datebase、SharePreference 等。

13．你一般在开发项目中都使用什么设计模式？如何来重构，优化你的代码？（★★★）

较为常用的就是单例设计模式和工厂设计模式以及观察者设计模式,一般需

要保证对象在内存中的唯一性时就是用单例模式,例如对数据库操作的 SqliteOpenHelper 的对象。厂模式主要是为创建对象提供过渡接口，以便将创建对象的具体过程屏蔽隔离起来，达到提高灵活性的目的。观察者模式定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新

14． 单例设计模式的缺点是什么 (上海)(2019.5.5)（★★）

主要优点：

1、 提供了对唯一实例的受控访问。

2、 由于在系统内存中只存在一个对象， 因此可以节约系统资源， 对于一些需要频繁创建和销毁的对象单例模式无疑可以提高系统的性能。

3、 允许可变数目的实例。

主要缺点：

1、 由于单利模式中没有抽象层， 因此单例类的扩展有很大的困难。

2、 单例类的职责过重， 在一定程度上违背了“ 单一职责原则” 。

3、 滥用单例将带来一些负面问题， 如为了节省资源将数据库连接池对象设计为的单例类， 可能会导致共享连接池对象的程序过多而出现连接池溢出； 如果实例化的对象长时间不被利用， 系统会认为是垃圾而被回收， 这将导致对象状态的丢失。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

15．说说观察者模式，里面主要有什么方法，各自的用途（杭州，2016.5.6）

notifyChange():观察数据库内容是否发生改变，如果改变，通知观察者

registerContentObserver():注册观察者

三、图片

16．图片的三级缓存(2019.5.5)（★★★★★）

1、图片三级缓存就是内存-文件-网络 三层 cache 机制，其中内存缓存包括强引用缓存和软引用缓存（SoftReference）。通过 url 向网络获取图片的时候，先从内存中找，如果内存中没有，再从缓存文件中查找，如果缓存文件中也没有，再从网络上通过 http 请求获取图片。在键值对（key-value）中，图片缓存的 key 是图片 url 的 hash 值，value 就是 bitmap。所以，只要一个 url 被下载过，其图片就被缓存起来了。

2、从代码上来说，采用一个 ImageManager 来负责图片的管理和缓存，函数接口为 public void loadBitmap(String url, Handler handler) ；其中 url 为要下载的图片地址，handler 为图片下载成功后的回调，在 handler 中处理 message，而 message 中包含了图片的信息以及 bitmap 对象。

17．LruCache 的怎么回收 bitmap(2019.5.5)（★）

LruCache 每次添加 Bitmap 图片缓存的时候(put 操作)，都会调用 sizeOf 方法，返回 Bitmap 的内存大小给 LruCache，然后循环增加这个 size。

当这个 Size 内存大小超过初始化设定的 cacheMemory 大小时，则遍历 map 集合，把最近最少使用的元素 remove 掉

18．图片优化底层怎么实现(2019.5.5)（★★）

图片优化其实就是内存进行优化 ,比如 bitmap 中有方法 recycle()进行回收。

从 BitmapFactory 的源代码可以看到，生成 Bitmap 对象最终都是通过 JNI 调用方式实现的。所以，加载 Bitmap 到内存里以后，是包含两部分内存区域的。简单的说，一部分是 Java 部分的，一部分是 C 部分的。这个 Bitmap 对象是由 Java 部分分配的，不用的时候系统就会自动回收了，但是那个对应的 C 可用的内存区域，虚拟机是不能直接回收的，这个只能调用底层的功能释放。所以需要调用 recycle()方法来释放 C 部分的内存。从 Bitmap 类的源代码也可以看到，recycle()方法里也的确是调用了 JNI 方法了的。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

19．图片压缩的详细过程(2019.5.5)（★★★★）

使用 BitmapFactory.Options 设置 inSampleSize 就可以缩小图片。属性值 inSampleSize 表示缩略图大小为原始图片大小的几分之一。即如果这个值为 2，则取出的缩略图的宽和高都是原始图片的 1/2 ,图片的大小就为原始大小的 1/4。

如果知道图片的像素过大，就可以对其进行缩小。那么如何才知道图片过大呢？

使用 BitmapFactory.Options 设置 inJustDecodeBounds 为 true 后 ,再使用 decodeFile()等方法 ,并不会真正的分配空间 ,即解码出来的 Bitmap 为 null，但是可计算出原始图片的宽度和高度，即 options.outWidth 和 options.outHeight。通过这两个值，就可以知道图片是否过大了。

20．当服务器上给我们传回来的图片过小的时候 ,我们应该怎么处理？(2019.5.5)（★★★）

可以通过 android 中的 bitmap 对象获取图像文件信息，进行图像剪切、旋转、缩放等操作，并可以指定格式保存图像文件。首先通过 Bitmap 获取图片的大小，根据你需要计算出缩放比例即可。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

21．Volley 框架中，关于图片的处理是如何处理的？(2019.5.5)（★★★）

Volley 框架图片处理是用 Volley 里面的的 ImageLoader 来实现加载图片的功能，从源码角度来分析加载流程；

使用 ImageLoader 来加载图片步骤：

(1)创建一个 RequestQueue 对象；

(2)创建一个 ImageLoader 对象；

获取一个 ImageListener 对象(通过 ImageLoader 的 getImageListener 方法来获取)；

(4)调用 ImageLoader 的 get 方法来加载图片；

22．图片的 URL 传递，我们用的是 javabean 还是对象传递，还是 map 集合传递(2019.5.5)（★★）

图片的 URL 传递，我们一般用对象进行传递，比如用 bitmap 对象进行传递。用 intent.putExtra("bitmap", bitmap);传递过去。因为 intent 只能传递实现序列化的类，即继承 java 的 Serializable 接口或者 android 的 Parcelable 接口。故 javabean 没有实现该接口所以不能传递。

如果是从网上拉取的图片，可以定义一个下载的工具类，并定义一个 static 的静态 Map 集合，每拉取成功一张图片就把该图片存入 Map 中作为缓存。key

是该图片的拉取地址。然后通过 intent 把地址传递过去。在新 activity 中通过该工具类取得静态 map，并通过传递过来的地址中 map 中取出该图片

23．加载大图片的时候，如何防止 OOM 异常(2019.5.5)（★★★★）

在加载图片时候先检查一下图片的大小：用 BitmapFactory.Options 参数，参数中的 inJustDecodeBounds 属性设置为 true 就可以让解析图片方法禁止为 bitmap 分配内存，返回值也不再是一个 Bitmap 对象，而是 null。虽然 Bitmap 是 null 了，但是 BitmapFactory.Options 的 outWidth、outHeight 和 outMimeType 属性都会被赋值。从而可以得到图片的宽、高、大小。得到图片的大小后 ,我们就可以决定是否把整张图片加载到内存中还是加载一个压缩版的图片到内存中，从而就可以解决 OOM 异常。

24．谈谈你对 Bitmap 的理解，什么时候应该手动调用 bitmap.recycle()（★★★）

Bitmap 是 android 中经常使用的一个类 ,它代表了一个图片资源。Bitmap 消耗内存很严重，如果不注意优化代码，经常会出现 OOM 问题，优化方式通常有这么几种： 1. 使用缓存； 2. 压缩图片； 3. 及时回收；

至于什么时候需要手动调用 recycle，这就看具体场景了，原则是当我们不再使

用 Bitmap 时，需要回收之。另外，我们需要注意，2.3 之前 Bitmap 对象与像素数据是分开存放的 ,Bitmap 对象存在 java Heap 中而像素数据存放在 Native Memory 中，这时很有必要调用 recycle 回收内存。但是 2.3 之后，Bitmap 对象和像素数据都是存在 Heap 中，GC 可以回收其内存

四、控件篇

25．请问 Gridview 能添加头布局吗(杭州 2019.5.6)？

GridView 本身没有添加头布局的方法 api

可以使用 ScrollView 与 GridView 结合，让 GridView 充满 ScrollView，不让 GridView 滑动而只让 ScrollView 滑动；具体做法是重载 GridView 的 onMeasure()方法。示例如下：

```
public class MyGridView extends GridView {
        public MyGridView(Context context, AttributeSet attrs) {
            super(context, attrs);
        }

        public MyGridView(Context context) {
    super(context);
}

public MyGridView(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);
}

@Override
public void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
```

```
            int expandSpec = MeasureSpec.makeMeasureSpec(
                    Integer.MAX_VALUE >> 2, MeasureSpec.AT_MOST);
            super.onMeasure(widthMeasureSpec, expandSpec);
        }
    }
```

26．新闻详细页 WebView 是怎么使用的？（★）

WebView 是 View 的子类，可以让你在 activity 中显示网页。

（1）添加网络访问权限。

（2）调用 webview 的 loadUrl 加载网络地址。

27．如何清除 webview 的缓存（★★★）

webview 的缓存包括网页数据缓存（存储打开过的页面及资源）、H5 缓存（即 AppCache）,webview 会将我们浏览过的网页 url 已经网页文件(css、图片、js 等)保存到数据库表中，如下；

/data/data/package_name/database/webview.db

/data/data/package_name/database/webviewCache.db

所以，我们只需要根据数据库里的信息进行缓存的处理即可。

28．webview 播放视频，5.0 以上没有全屏播放按钮(2019.5.5)（★★）

实现全屏的时候把 webview 里的视频放到一个 View 里面 然后把 webview

隐藏掉；即可实现全屏播放。

29．webview 调用 js(2019.5.5)（★★★★）

（1）android 中利用 webview 调用网页上的 js 代码。

首先将 webview 控件的支持 js 的属性设置为 true，，然后通过 loadUrl 就可以直接进行调用，如下所示：

```
mWebView.getSettings().setJavaScriptEnabled(true);
mWebView.loadUrl("javascript:test()");
```

（2）网页上调用 android 中 java 代码的方法

在网页中调用 java 代码，需要在 webview 控件中添加 javascriptInterface。如下所示：

```
mWebView.addJavascriptInterface(new Object() {
        public void clickOnAndroid() {
           mHandler.post(new Runnable() {
                        public void run() {
                              Toast.makeText(Test.this, “测试调用 java”，
                                                    Toast.LENGTH_LONG).show();
              }
           });
        }
    }, "demo");
```

在网页中，只需要像调用 js 方法一样，进行调用就可以

```
<div id='b'>
<aonclick="window.demo.clickOnAndroid()">b.c</a>
</div>
```

（3）Java 代码调用 js 并传参

首先需要带参数的 js 函数，如 function test(str)，然后只需在调用 js 时传入参数即可，如下所示：

mWebView.loadUrl("javascript:test('aa')");

（4）Js 中调用 java 函数并传参

首先一样需要带参数的函数形式，但需注意此处的参数需要 final 类型，即得到以后不可修改，如果需要修改其中的值，可以先设置中间变量，然后进行修改。

如下所示：

```
mWebView.addJavascriptInterface(new Object() {
        public void clickOnAndroid(final int i) {
           mHandler.post(new Runnable() {
              public void run() {
                      int j = i;
                      j++;
           Toast.makeText(Test.this, "测试调用 java" + String.valueOf(j),
                                                    Toast.LENGTH_LONG).show();
              }
           });
        }
    }, "demo");
```

然后在 html 页面中，利用如下代码

```
<div id='b'>
<aonclick="window.demo.clickOnAndroid(2)">b.c</a>
</div>,
```

即可实现调用

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

30．Webview 中是如何控制显示加载完成的进度条的 (2019.5.5)（★★★）

在 WebView 的 setWebChromClient() 中，重写 WebChromClient 的 openDialog()和 closeDialog()方法；实现监听进度条的显示与关闭。

31．你使用过 HTM5 开发过 Android 应用程序吗？谈谈你对移动应用开发领域 webApp 开发与 NativeApp 开发之争的看法？(2019.5.5)（★）

目前移动应用开发有三种方案，分别是 Native 原生 App，Hybrid 混合 App，以及 HTML5 Web App。

Native（原生）：

丰富的用户体验，平台的指向性，久经考验的移动应用开发途；对于游戏应用这类对性能、图形处理要求较高，但不太在乎文件尺寸的软件来说，原生应用才是最理想的选择——不过大家其实也可以利用 PhoneGap 实现游戏开发。

HTML 5：

更快的开发周期，跨平台运行，实时更新。对于任何追求极致轻量化的网站（或者 Web 应用）都应该通过 HTML 5 进行创建，并使用 Bootstrap 或者 Foundation 等技术作为响应层。响应式 Web 设计为设备提供一套极度精简化

的访问门户，技术人员还能够根据需求每天对其加以调整。

32．View， surfaceView， GLsurfaceView 他们的区别(上海鸿巍企业管理咨询有限公司， 医疗类)？(2019.5.5)（★★）

View：显示视图，内置画布，提供图形绘制函数、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。必须在 UI 的主线程中更新画面，用于被动更新画面。

SurfaceView：基于 view 视图进行拓展的视图类，更适合 2D 游戏的开发；是 view 的子类，类似使用双缓机制，在新的线程中更新画面所以刷新界面速度比 view 快。UI 线程和子线程中都可以。在一个新启动的线程中重新绘制画面，主动更新画面。

GLSurfaceView：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，比如你更新画面的时间过长，那么你的主 UI 线程会被你正在画的函数阻塞。那么将无法响应按键，触屏等消息。

当使用 surfaceView 由于是在新的线程中更新画面所以不会阻塞你的 UI 主线程。但这也带来了另外一个问题，就是事件同步，涉及到线程同步。

33．说说 RecycleView 和 ListView 的区别(2019.5.5)（★★★）

RecyclerView 是 ListView 的增强版。整体上看 RecyclerView 架构，更加灵活，

通过设置它提供的不同 LayoutManager，ItemDecoration，ItemAnimator 优质的效果。想要控制其显示的方式，可以通过布局管理器 LayoutManager；想要控制 Item 间的间隔（可绘制），可以通过 ItemDecoration；想要控制 Item 增删的动画，可以通过 ItemAnimator；如果想要要控制点击、长按事件，需要自己写。

RecyclerView.LayoutManager 是一个抽象类，系统为其提供了 3 个实现类：

LinearLayoutManager 现行管理器，支持横向、纵向。

GridLayoutManager 网格布局管理器

StaggeredGridLayoutManager 瀑布就式布局管理器

五、系统篇

34．framework 层的开发是怎样的（★★）

Framework 层开发，

关于 framework，更多是做的应用层之下的系统层面的东西。比如电源管理、消息队列、包管理等等，还包括对硬件的支持及系统提供给上层的硬件功能调用接口。framework 的学习必然离不开不断编译 rom 和刷机。这就要求有耐心有时间有兴趣。而且由于 framework 层多数模块都是以 JNI 方式被调用的，因此你需要有比较扎实的 C 语言基础，之少能看懂程序结构。除此之外，对你想要详细研读的模块在应用层的应用需要有必要的理解。我刚开始看源码的时候是从

电源管理模块开始看的，就是因为当时对 android 系统自带的电源提醒方式以及电量通知不太满意，想重新定义更多层级的提醒。刚开始也是一头雾水，但还是硬着头皮一点点啃。这个过程中，为了防止忘记之前看过什么，所以又不断对看过的源码做注释并做阅读笔记。大概两三个月，虽然了解的也比较浅显，但是我的目的达到了。源码之路漫漫，看个三五年都不一定敢说能够整体吃透。虽如此，但只要有钻研的方向和基本的能力，相信工作中遇到的 framework 层的改动应当还是能够负担得住的。

需要的：Android 版本完整的源码，linux 系统的主机，smb 服务，xshell 软件，sourcinsght_软件，软件（eclipse，AS）

35．Android 系统编程与 Java 编程有何异同之处，谈谈你的看法（★★）

	Java 编程	Android 编程
1.执行入口点	main	manifest.xml 配置文件
2.UI	frame	Activity
3.是否给予配置文件	否	是
4.是否基于组件	否	是 activity，service， broadcastReceiver，contentprovider

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

5.布局文件　　new　　findViewById

总的来说，区别在于 android 程序是基于组件给予配置的。

1：Android 使用的开发包是 J2EE 包的一个子集。

2：在使用 Android 开发使用线程技术比较多，但使用 J2EE 开发常不用手动去创建线程

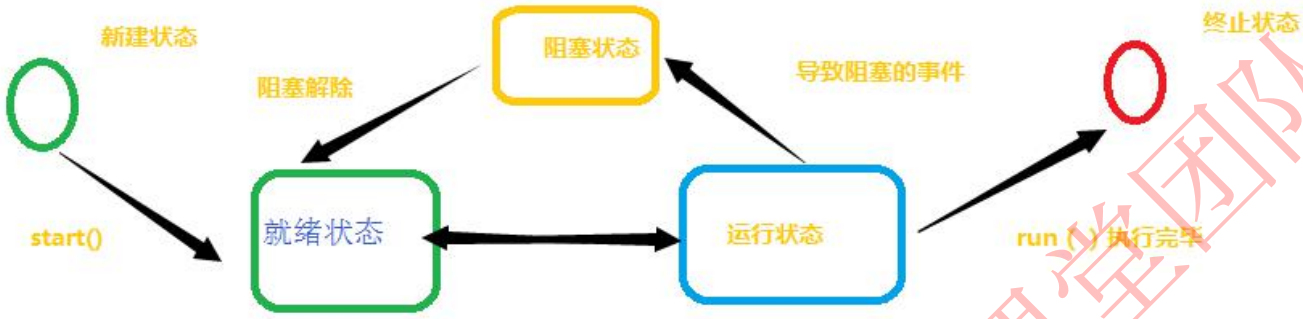
3：在 Android 经常使用到线程，而在线程中的处理是不能直接调用 UI 界面，所以就涉及到线程与 UI 的比较多，所以对 Android 的 Handler 用得比较多。

4：在使用 Android 开发时，有一些底层的网络通信就需要使用 Socket 通信，在 J2EE 开发中，几乎不会用到 Socket 技术

36．谈一谈 java 线程模型（★★★）

线程是一个程序内部的顺序控制流,

线程生命周期：新建,就绪,运行,阻塞,终止



线程基本状态转换

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

创建线程的两种方式：实现 Runnable 接口，继承 Thread 类

比较：

实现 Runnable 接口

可以将 CPU，代码和数据分开，形成清晰的模型；

线程体 run()方法所在的类还可以从其他类继承一些有用的属性和方法

有利于保持程序风格的一致性

继承 Thread 类

Thread 子类无法再从其他类继承

编写简单，run()方法的当前对象就是线程对象，可直接操纵

线程优先级：

默认为 5，最小为 1，最大为 10，不提供优先级低的在优先级高的执行结束后执行

线程串行化

join()

线程休眠

sleep()

临界资源问题

并发线程有机的交替，确保共享的数据在关键的时段被专用

多个线程间共享的数据称为临界资源

在 Java 语言中，引入了对象互斥锁的概念，来保证共享数据操作的完整性

每个对象都对应于一个可称为“互斥锁”的标记，这个标记用来保证在任何一个时刻，只能有一个线程访问该对象

关键字 synchronized 来与对象的互斥锁联系。当某个对象用 synchronized 修饰时，表明该对象在任一时刻只能有一个线程访问

并发运行的多个线程彼此间等待，都无法运行的状态称为线程死锁

线程同步通信

为避免死锁，在线程进入阻塞状态时应尽量释放其锁定的资源，以为其他的线程提供运行的机会

wait()

notify()/notifyAll()

线程间数据传输

使用管道流

类的同步性与线程安全

37．谈谈 Android 的 GC

Java 语言建立了垃圾收集机制，用以跟踪正在使用的对象和发现并回收不再使用(引用)的对象。该机制可以有效防范动态内存分配中可能发生的两个危险：因内存垃圾过多而引发的内存耗尽，以及不恰当的内存释放所造成的内存非法引用。

垃圾收集算法的核心思想是：对虚拟机可用内存空间，即堆空间中的对象进行识别，如果对象正在被引用，那么称其为存活对象，反之，如果对象不再被引用，则为垃圾对象，可以回收其占据的空间，用于再分配。垃圾收集算法的选择和垃圾收集系统参数的合理调节直接影响着系统性能，因此需要开发人员做比较深入的了解。

38．java 多线程同步锁

1、线程同步的目的是为了保护多个线程反问一个资源时对资源的破坏。

2、线程同步方法是通过锁来实现，每个对象都有切仅有一个锁，这个锁与一个特定的对象关联，线程一旦获取了对象锁，其他访问该对象的线程就无法再访问该对象的其他同步方法。

3、对于静态同步方法，锁是针对这个类的，锁对象是该类的 Class 对象。静态和非静态方法的锁互不干预。一个线程获得锁，当在一个同步方法中访问另外对象上的同步方法时，会获取这两个对象锁。

4、对于同步，要时刻清醒在哪个对象上同步，这是关键。

5、编写线程安全的类，需要时刻注意对多个线程竞争访问资源的逻辑和安全做出正确的判断，对“原子”操作做出分析，并保证原子操作期间别的线程无法访问竞争资源。

6、当多个线程等待一个对象锁时，没有获取到锁的线程将发生阻塞。

7、死锁是线程间相互等待锁锁造成的，在实际中发生的概率非常的小。真让你

写个死锁程序，不一定好使，呵呵。但是，一旦程序发生死锁，程序将死掉。

39．线程和进程间的通讯都有哪些方式（★★）

1 、一个 Android 程序开始运行时，会单独启动一个 Process 。

默认情况下，所有这个程序中的 Activity 或者 Service 都会跑在这个 Process 。

默认情况下，一个 Android 程序也只有一个 Process ，但一个 Process 下却可以有许多个 Thread。

2 、一个 Android 程序开始运行时，就有一个主线程 Main Thread 被创建。该线程主要负责 UI 界面的显示、更新和控件交互，所以又叫 UI Thread 。

一个 Android 程序创建之初，一个 Process 呈现的是单线程模型 — 即 Main Thread ，所有的任务都在一个线程中运行。所以， Main Thread 所调用的每一个函数，其耗时应该越短越好。而对于比较费时的工作，应该设法交给子线程去做，以避免阻塞主线程（主线程被阻塞，会导致程序假死 现象）。

3 、 Android 单线程模型： Android UI 操作并不是线程安全的并且这些操作必须在 UI 线程中执行。如果在子线程中直接修改 UI ，会导致异常。

那么我们线程之间怎么通讯呢？

Handler，AsyncTask，广播，文件，sp

线程之间：意图，内容提供者，File，网络，广播，aidl

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

40．Android 系统中 GC 什么情况下会出现内存泄露呢？ 视频编解码/内存泄露

导致内存泄漏主要的原因是，先前申请了内存空间而忘记了释放。如果程序中存在对无用对象的引用，那么这些对象就会驻留内存，消耗内存，因为无法让垃圾回收器 GC 验证这些对象是否不再需要。如果存在对象的引用，这个对象就被定义为"有效的活动"，同时不会被释放。要确定对象所占内存将被回收，我们就要务必确认该对象不再会被使用。典型的做法就是把对象数据成员设为 null 或者从集合中移除该对象。但当局部变量不需要时，不需明显的设为 null，因为一个方法执行完毕时，这些引用会自动被清理。

Java 带垃圾回收的机制,为什么还会内存泄露呢?

```
Vector v = new Vector(10);
 for (int i = 1; i < 100; i++){
        Object o = new Object();
 v.add(o);
 o = null;
}//此时，所有的 Object 对象都没有被释放，因为变量 v 引用这些对象。
```

Java 内存泄露的根本原因就是 保存了不可能再被访问的变量类型的引用

检测内存工具 heap

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

41．说说 LruCache 底层原理（★★★★★）

LruCache 使用一个 LinkedHashMap 简单的实现内存的缓存，没有软引用，都是强引用。如果添加的数据大于设置的最大值，就删除最先缓存的数据来调整内存。

maxSize 是通过构造方法初始化的值，他表示这个缓存能缓存的最大值是多少。

size 在添加和移除缓存都被更新值，他通过 safeSizeOf 这个方法更新值。safeSizeOf 默认返回 1，但一般我们会根据 maxSize 重写这个方法，比如认为 maxSize 代表是 KB 的话，那么就以 KB 为单位返回该项所占的内存大小。

除异常外首先会判断 size 是否超过 maxSize，如果超过了就取出最先插入的缓存，如果不为空就删掉，并把 size 减去该项所占的大小。这个操作将一直循环下去，直到 size 比 maxSize 小或者缓存为空。

42．jni 的调用过程?（★★★）

1. 安装和下载 Cygwin，下载 Android NDK。
2. ndk 项目中 JNI 接口的设计。
3. 使用 C/C++实现本地方法。
4. JNI 生成动态链接库.so 文件。

5. 将动态链接库复制到 java 工程，在 java 工程中调用，运行 java 工程即可。

43．怎样保证 App 不被杀死?（★★）

强烈建议不要这么做，不仅仅从用户角度考虑，作为 Android 开发者也有责任去维护 Android 的生态环境。当然从可行性讲，谷歌也不会让容易的实现。同时这样的 app 一般属于流氓应用

通常为了保证自己 app 避免被杀死，我们一般使用以下方法：

1.Service 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.通过 startForeground 将进程设置为前台进程，做前台服务，优先级和前台应用一个级别 ，除非在系统内存非常缺，否则此进程不会被 kill

3..双进程 Service：让 2 个进程互相保护，其中一个 Service 被清理后，另外没被清理的进程可以立即重启进程

4.QQ 黑科技:在应用退到后台后，另起一个只有 1 像素的页面停留在桌面上，让自己保持前台状态，保护自己不被后台清理工具杀死

5.在已经 root 的设备下，修改相应的权限文件，将 App 伪装成系统级的应用（Android4.0 系列的一个漏洞，已经确认可行）

6.Android 系统中当前进程(Process)fork 出来的子进程，被系统认为是两个不同的进程。当父进程被杀死的时候，子进程仍然可以存活，并不受影响。鉴于目前提到的在 Android-Service 层做双守护都会失败，我们可以 fork 出 c 进程，

多进程守护。死循环在那检查是否还存在，具体的思路如下（Android5.0 以下可行）

1.用 C 编写守护进程(即子进程)，守护进程做的事情就是循环检查目标进程是否存在，不存在则启动它。

2. 在 NDK 环境中将 1 中编写的 C 代码编译打包成可执行文件(BUILD_EXECUTABLE)。

3.主进程启动时将守护进程放入私有目录下，赋予可执行权限，启动它即可。

7 联系厂商，加入白名单

六、优化篇（★★★★★）

44．性能优化有哪些

性能优化，是 App 开发的常态，也是一个好的 App 的基本需要，好的体验的一种方式，也是我们走向高级程序员的基本能力，我把性能优化归类为 5 类：

1.Reckon（计算）

首先需要知道你的 app 所消耗内存的情况，知己知彼才能百战不殆

2.Reduce（减少）

消耗更少的资源

3.Reuse（重用）

当第一次使用完以后，尽量给其他的使用

Reuse 重用，减少内存消耗的重要手段之一。

核心思路就是将已经存在的内存资源重新使用而避免去创建新的，最典型的使用就是缓存（Cache）和池（Pool）。

5.Recycle（回收）

返回资源给生产流

4.Review（检查）

回顾检查你的程序，看看设计或代码有什么不合理的地方。

具体的可以看文档《Android 内存优化之 5R 法则》

45．我们的 app 发热很厉害，这块的话 你有什么解决方案(2016/5/11)？

不是安卓机的通病，其他系统的手机，如果电流没有优化好，器件布局不合理，一样会存在发热严重的问题。

对于那些在待机状态下也出现发热严重的，建议试试把移动数据关掉；手机上如果装的 app 多，发出接入移动网络的请求也多，会出现难以进入睡眠模式的情况，这个时候尽管屏幕是灭屏状态，但其实待机电流可能高达好几百毫安；手机不仅发热大，而且还特别耗电

在的安卓手机，硬件配置性能过剩，动辄 8 核，运行内存也不断变大，多任务运行也更加流畅，加之应用越做越大，cpu 的运算压力太大，很容易发热。很多软件常驻后台，不断自启，使得手机发热严重！系统优化的不够也是原因之一！

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

46．android 优化工具有哪些

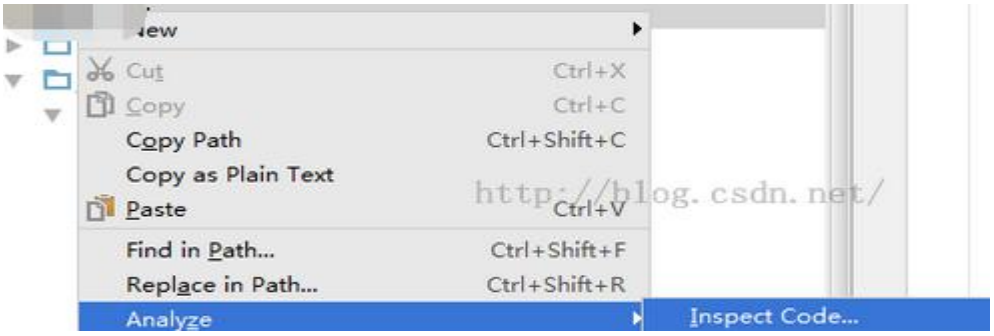
优化是一个 App 走向完善，成熟的必经之路，也是我们 HM 程序员走向更高层次应该思考的。

常常开发我们借助了很多工具，正所谓“工欲善其事，必先利其器”，Android 优化也是一样的，那么我们 Android 开发中，优化要用到哪些工具呢？

代码优化：

lint for android studio 工具

代码 在性能测试之前，首先要对工程源码进行排错和调优。Android Lint 可以通过扫描和检查对 Android 工程可能存在的问题进行审查，其是一种静态测试工具，通过发现代码中可能存在的问题来在应用发布前保证程序质量优化。早期的 lint 工具可以与 Eclipse 集成，如今 android-studio 已经内置这一工具。通过在工程标题上右键->Analyze->Inspect Code 可以打开该工具。



Android Lint 可以检查出的错误包括：

1. Missing translations (and unused translations) 没有翻译的文本

2. Layout performance problems (all the issues the old layoutopt tool used to find, and more) 布局性能
3. Unused resources 未使用的冗余资源
4. Inconsistent array sizes (when arrays are defined in multiple configurations)在多个配置中的数组大小不一致文件
5. Accessibility and internationalization problems (hardcoded strings, missing contentDescription, etc)国际化问题
6. Icon problems (like missing densities, duplicate icons, wrong sizes, etc) 图标问题
7. Usability problems (like not specifying an input type on a text field) 可用性问题，如正则
8. Manifest errors 清单文件出错

应用性能检测工具

每当处理或者排查性能问题的时候，我都遵循这些原则：

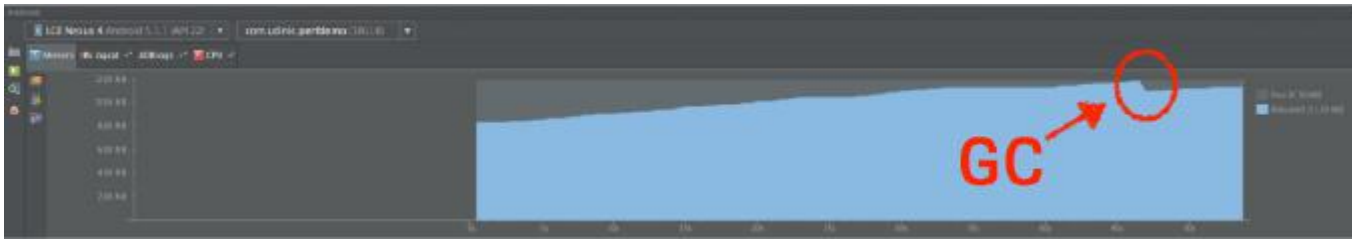
- **持续测量**： 用你的眼睛做优化从来就不是一个好主意。同一个动画看了几遍之后，你会开始想像它运行地越来越快。数字从来都不说谎。使用我们即将讨论的工具，在你做改动的前后多次测量应用程序的性能。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

- **使用慢速设备：**如果你真的想让所有的薄弱环节都暴露出来，慢速设备会给你更多帮助。有的性能问题也许不会出现在更新更强大的设备上，但不是所有的用户都会使用最新和最好的设备。
- **权衡利弊 ：**性能优化完全是权衡的问题。你优化了一个东西 —— 往往是以损害另一个东西为代价的。很多情况下，损害的另一个东西可能是查找和修复问题的时间，也可能是位图的质量，或者是应该在一个特定数据结构中存储的大量数据。你要随时做好取舍的准备。

Android Studio 最近改进了很多，有越来越多的工具可以帮助我们找出和分析性能问题。Android 窗口中的内存页告诉我们，随着时间的推移有多少数据在栈上分配。它看上去像这样：



我们在图中看到一个小的下降，这里发生了一次 GC 事件 ，移除了堆上不需要的对象和释放了空间。

图中的左边有两个工具可用：堆转储和分配跟踪器

47．说说 ANR,怎样避免

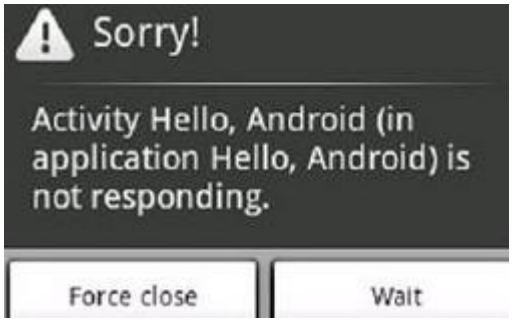
ANR(Application Not Responding) Android 系统中应用无响应

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

是 Android 系统中比较常见的问题 ,当出现 ANR 时一般情况会弹出一个带有以下文字的对话框提示：

Activity XXX(in XXXXX) is not responding. 比如：



从大体上分,两种情况导致 ANR：

第一类：dispatchTimeout 输入事件分发超时，一般是由于主线程在 5 秒之内没有响应输入事件。

第二类 :BroadcastReceiver 没有在系统设定的时间(一般是 10s)内完成并返回。

从细节上归纳(ANR 可能是由主线程导致也可能是由非主线程导致)：

由于主线程导致的情况：

1.耗时网络访问

2.当有大量数据读写操作时再请求数据读写

3.数据库操作(比如其他大数据量应用访问数据库导致数据库负载过重时)

4.硬件操作(比如 Camera)

5.调用 thread_join() / Sleep() / Wait() 或者等待 locker 的时候

6.Service binder 数量达到上限

7.在 system_server 中发生 WatchDog ANR

8.Service 忙导致超时无响应

由于非主线程导致的情况：

1.非主线程持有 lock，导致主线程等待 lock 超时

2.非主线程终止或者崩溃导致主线程一直等待

既然我们知道了导致 ANR 是怎样引起的，那么我们该怎样避免 ANR 呢？

1、运行在主线程里的任何方法都尽可能少做事情。特别是，Activity 应该在它的关键生命周期方法（如 onCreate()和 onResume()）里尽可能少的去做创建操作。（可以采用重新开启子线程的方式，然后使用 Handler+Message 的方式做一些操作，比如更新主线程中的 ui 等）

2、应用程序应该避免在 BroadcastReceiver 里做耗时的操作或计算。但不再是在子线程里做这些任务（因为 BroadcastReceiver 的生命周期短），替代的是，如果响应 Intent 广播需要执行一个耗时的动作的话，应用程序应该启动一个 Service。

3、避免在 Intent Receiver 里启动一个 Activity，因为它会创建一个新的画面，并从当前用户正在运行的程序上抢夺焦点。如果你的应用程序在响应 Intent 广

播时需要向用户展示什么，你应该使用 Notification Manager 来实现。

检测工具

总结：ANR 异常也是在程序中自己经常遇到的问题，主要的解决办法自己最常用的就是不要在主线程中做耗时的操作，而应放在子线程中来实现，比如采用 Handler+mesage 的方式，或者是有时候需要做一些和网络相互交互的耗时操作就采用 Asyntask 异步任务的方式（它的底层其实 Handler+mesage 有所区别的是它是线程池）等，在主线程中更新 UI。

48．怎样对 android 进行优化？

1) 对 listview 的优化。
2) 对图片的优化。
3) 对内存的优化。

- 尽量不要使用过多的静态类 static
- 数据库使用完成后要记得关闭 cursor
- 广播使用完之后要注销

49．请介绍下 AsyncTask 的内部实现，适用的场景是

AsyncTask 内部也是 Handler 机制来完成的，只不过 Android 提供了执行

框架来提供线程池来执行相应地任务，因为线程池的大小问题，所以 AsyncTask 只应该用来执行耗时时间较短的任务，比如 HTTP 请求，大规模的下载和数据库的更改不适用于 AsyncTask，因为会导致线程池堵塞，没有线程来执行其他的任务，导致的情形是会发生 AsyncTask 根本执行不了的问题。

50．谈谈二叉树，说说他的遍历方式？（上海 2016.5.6）

1、先序遍历：先序遍历是先输出根节点，再输出左子树，最后输出右子树。

2、中序遍历：中序遍历是先输出左子树，再输出根节点，最后输出右子树。

3、后序遍历：后序遍历是先输出左子树，再输出右子树，最后输出根节点。

七、视频播放篇（★★★）

51．视频播放 ndk 开发，你能说说吗？

首先我们先得到本地视频，当然如果需要添加其他的例如视频的长度和宽度可以使用 Video 类中的方法，获取到视频后，那么接下来就是解码视频，解码视频可以使用的是 ffmpeg，通过使用 NDK 编译手机可以使用的 FFmpeg 库。

首先可以到官网 http://ffmpeg.org/download.html 下载最新的源码,然后在 ffmpeg 目录下面创建一个脚本 config.sh，ndk 端使用 socket 来接受网络摄像

机的视频流然后使用 ffmpeg 解码，再使用 ndk 里面的 ativewindow 来播放。然后在 java 代码这一端，使用 surfaceView 来播放，

注意这里的 NDK 要换成本地的路径。

52．你对视频类的软件是如何看待的， 如果我现在需要开发一个视频类的 app 你觉得最大的问题在哪里？

现在的视频类软件很火，比如斗鱼直播、熊猫直播、网页版的、android 手机版的等等这些视频软件访问率是越来越大，从而盈利就很大，这样导致现在越来越多的公司都开始开发视频类软件。如果现在开发一个视频类的 app 其实也不难，首先网上有大量的资源可以利用，视频解码器也很多，如：ffmpeg，vlc 等等，github 上也有很多这些资料，直接有下载源代码运行 demo 就行。如果非要说问题的话应该就是视频解码上需要点时间。

因为网上的东西可能在自己用时都会出现一些小错误。需要去测试。

53．直播的应用大概是怎么实现的？

直播应用大概实现步骤：录制音视频->编码->流媒体传输->服务器->流媒体传输到其他 app->解码->播放

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

android 本身对音视频流媒体传输协议,以及音视频编解码支持有限.所以对于直播类应用,要自己解码实现。常用解码：vitamio、webRTC、ffmpeg、vlc、ijkplayer。同时流媒体协议有(RTP RTCP RTSP RTMP HLS)

视频直播一般有两个条件：1、直播地址 2、播放器；

对于直播地址我们可以利用很多软件获取连接，播放器，现在开源的也有很多，最常见的就是 ffmpeg，但是如果直接用 ffmpeg 开发工作量比较大，我们可以使用第三方的播放器库，例如 vlc,vitamio 等等，例如使用 vitamio 库。

首先建立一个项目，命名为 Live,项目建立好了以后我们需要配置 vitamio 需要的环境，然后添加依赖库，接着添加一个主界面，进行配置，接着直接写播放界面类，并进行配置。简单的直播就可以实现了。

方案二：现在 h5 越来越火了，用 h5 也能实现视频直播，h5 不用加载插件直接播放视频，还能跨平台播放。

pc 端还是用 flash 播放，但移动端通过 html5 方式来做。

而 html5 的 video 标签只支持 mp4、webm、ogg 三种格式.目前所有主流浏览器最新版本都支持 html5（除了 Opera）

八、APP 业务以及安全篇（★★★★★）

54．打开 APP 后实现更新的具体操作

在 App 启动的时候，可以在 MainActivity 里面调用后台更新接口。如果服务器有更新，就弹出更新对话框，显示更新日志和更新包的相关信息，比如大小等。如果是强制升级，禁用取消按钮。如果用户点击升级，调用系统的下载程序或者自己写下载程序，下载新的安装包。下载完成之后，调用安装指令，进行安装。如果需求是每次 App 从后台切换到前台和 App 启动的话，就要结合 Activity 的生命周期，比如可以在 onstart 里面调用更新接口。

55．登陆注册,登陆状态保存

登录之后，一般服务器会返回一串字符串 Token,这里面包含有使用这个 token 的用户信息，把这个 token 保存在本地，建议保存在 data/data/pacakgename/下面。在需要安全校验的接口，后台会要求客户端把 token 作为头信息传入到服务器，服务器收到该接口请求后，先检查 toekn 的合法性，如果 token 合法，就进行相应的操作，返回数据等。如果校验不通过，返回特定的返回码（httpStatus）。客户端在接口到这个错误码的时候，清空本地 token。跳转至登录页，要求用户重新登录。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

56．微信朋友圈怎么实现

这题不是真正的让你去实现朋友圈功能，而是拿着朋友圈的功能来让你分析里面可能涉及到的技术点。

主界面：

（1） 选择网络请求框架获取数据

（2） ListView 的优化（或 RecycleView）

（3） 图片的缓存处理，主流的图片缓存框架是 ImageLoader 框架

发朋友圈：

多图片选择：引用 github 上的 android-multiple-images-selector 进行实现

录制小视频：调用系统摄像头图纸视频，通过 VideoView 进行展示。

定位：使用百度地图等第三方 SDK

57．Zxing 是做什么的？

ZXing 是一个开源 Java 类库用于解析多种格式的 1D/2D 条形码。目标是能够对 QR 编码、Data Matrix、UPC 的 1D 条形码进行解码。 其提供了多种平台下的客户端包括：J2ME、J2SE 和 Android。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

58．怎样防范 APP 被反编译

（1）加壳保护：就是在程序的外面再包裹上另外一段代码，保护里面的代码不被非法修改或反编译，在程序运行的时候优先取得程序的控制权做一些我们自己想做的工作。

（2）dex 文件格式 ：apk 生成后所有的 java 生成的 class 文件都被 dx 命令整合成了一个 classes.dex 文件，当 apk 运行时 dalvik 虚拟机加载 classes.dex 文件并且用 dexopt 命令进行进一步的优化成 odex 文件。在这个过程中修改 dalvik 指令来达到我们的目的。

59．如何去记录用户是否看过这条新闻？

1) 在新闻详情页用 sharePrefence 存入当前新闻的 id。
2) 在新闻列表的 getView 方法中判断当前 id 在数据中是否存在，存在的话将字体设置为灰色。

60．在项目中，你是如何缓存数据的？

缓存数据可以减少服务器的压力，提高客户端的响应速度，在一定的程度上支持离线功能，通常我们这么做缓存处理的：

（1） 对需要缓存的数据，进行缓存保存到本地；然后把文件的相关信息如 url，路经，下载时间存放进路径，我会把存储路径 url 作为唯一的标识，便于

进行缓存使用时进行查询；还需要根据时间，判断缓存是否过期，从而实现缓存效果。

（2） 直接的本地文件缓存，使用 File.lastModified()方法得到文件的最后修改时间，与当前时间判断是否过期，从而实现缓存效果。

61． 如何实现第三方登录(以微信为例子)

第三方的集成，有多重，一般在第三方的官方有相关文档；大致的流程和微信类似。

(1) 下载微信官方最新的 sdk，集成到 app 中，放入项目的 libs 目录下

(2) 在项目 application 的 onCreate()方法中注册，这样，在整个 app 的中都可以使用

(3) 在 app 的包名目录下新建一个 wxapi 目录，然后在此目录下新建 WXEntryActivity.java 文件 ,如下 ,app 的包名为 carjob.com.cn ,新建的 wxapi 目录如下。（注意：一定是包名目录下新建，不要在其他目录新建，否则 WXEntryActivity.java 里的 public void onResp(BaseResp resp)方法不会被调用）

WXEntryActivity 继承 Activity ，实现 IWXAPIEventHandler ，并重写 protected void onNewIntent(Intent intent)

public void onReq(BaseReq arg0)、public void onResp(BaseResp resp)

(3) 发送微信登录的请求，调用 sendReq（request）方法；SDK 回调上一步中的 onReq（ ）方法，然后根据返回值做相关处理。

62．谈谈你在工作中是怎样解决一个 bug

思路大体和上面差不多，

1) 异常附近多打印 log 信息；

2) 分析 log 日志，锁定 bug 的原因，实在不行的话进行断点调试；

3) 调试不出结果，上 Stack Overflow 贴上异常信息，查找上面的资源

4) 再多看看代码，或者从源代码中查找相关信息

5) 实在不行，找师傅来解决！

63．andorid 应用第二次登录实现自动登录

我来说一个实际案例的流程吧。前置条件是所有用户相关接口都走 https，非用户相关列表类数据走 http。

1. 第一次登陆 getUserInfo 里带有一个长效 token，该长效 token 用来判断用户是否登陆和换取短 token

2. 把长效 token 保存到 SharedPreferences

3. 接口请求用长效 token 换取短 token，短 token 服务端可以根据你的接口最后一次请求作为标示，超时时间为一天。

4. 所有接口都用短效 token 5. 如果返回短效 token 失效，执行 3 再直接当前接口 6. 如果长效 token 失效（用户换设备或超过两周），提示用户登录。

Android 应用中验证码登陆都有哪些实现方案

验证码应该只有两种获取方式： 从服务器端获取图片， 通过短信服务，将验证码发送给客户端这两种。

64．Android 应用中验证码登陆都有哪些实现方案

验证码应该只有两种获取方式： 从服务器端获取图片， 通过短信服务，将验证码发送给客户端这两种。

65．手机 APP 安全登录的几种方式

登录过程的用户认证，常见的手段有密码**加密传输、动态密码、验证码**等。

加密传输：根据密码生成一个散列值，把散列值发送给服务器。服务器计算库中用户密码的散列值，然后和客户端传来的散列值比较，一致的话，登录成功。

动态密码：市面上适合 App 使用的最常见的方式是利用手机短信进

行动态密码认证

验证码：服务器一旦发现登录有异常，如 IP 变化、短时间内登录次数过多等，会向 App 下发一个图片，用户把图片中要求输入的数据和用户名、用户密码一起提交给服务器

（2）**减少用户输入次数的自动登录。**

App 登录成功后，服务器会告诉 App 一个 session，后续交流都使用 session。但通常为了安全起见 session 都是要设置有效期的，从 1 星期到 20 天都见过。那么，为了不让用户在 session 失效后重新登录，减少用户的手动输入用户名和用户密码的次数，引入了“自动登录”概念。

66．请问怎样监听 Home 建 ,请问 Hone 建能实现 Back 功能吗？(5/11)

在低版本 android 中，可以通过重写 onKeyDown()方法，达到监听 Home 键，间接实现 Back 键功能。但是在 android 4.0 之后 Home 键的这个重写方式失效。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

67． 你怎么看待混合式 App 开发？

在目前，混合开发效率相对原生开发效率较高，不可忽视混合开发是未来的一个开发方向，淘宝 APP 的实现就是一个很好的例子，但是普通公司（技术积累不够）混合开发出来的产品，目前体验还是没有原生的好；流畅度，安全性无法和原生开发相比。

九、支付(2016.5.5)（★★★）

68．集成支付宝支付你是怎样处理公钥和私钥的,在代码中如何获取到的,微信支付的支付限额是多少？

参考答案：

1.支付宝支付采用了 RSA 加密签名的安全通信机制，开发者可以通过支付宝的公钥验证消息的来源，同时使用自己的私钥进行信息加密。RSA 算法及数字签名机制是服务窗平台与开发者网关安全通信的基础

2.微信支付用户使用零钱支付额度是多少？

1）添加过银行卡的用户（绑定后解绑了也算），实物限额单笔单日：50000 元；虚拟限额：单笔 3000 元，单日 6000 元，没有月限额。

2）未添加过银行卡的用户，零钱支付额度为：单笔单日 1000 元，单月 1000

元。超过额度需要添加银行卡完成支付。

69．是否会集成支付，你是怎么做的？他们的区别在哪里？

参考答案：

答：我之前写过 2 篇帖子，可以看看

支付宝支付

微信支付

十、地图(2016.5.5)：

70．今天面试官问百度地图经纬度和你的 gps 之间怎么转换的？（★★★）

直接使用百度提供的百度地图坐标转换 API，是一套以 HTTP 形式提供的坐标转换接口，可以用于将常用的非百度坐标（目前支持 GPS 设备获取的坐标、google 地图坐标、soso 地图坐标、amap 地图坐标、mapbar 地图坐标）转换成百度地图中使用的坐标，并可将转化后的坐标在百度地图 JavaScript API、车联网 API、静态图 API、web 服务 API 等产品中使用。

使用方法：

第一步，申请秘钥（ak），作为访问服务的依据；

第二步，按照请求参数说明拼写发送 http 请求的 url，注意需使用第一步申请的

ak；

第三步，接收返回的数据（json 或者 xml 格式）。

71．定位项目中，如何选取定位方案，如何平衡耗电与实时位置的精度？（★★★）

方案 1：始定位，Application 持有一个全局的公共位置对象，然后隔一定时间自动刷新位置，每次刷新成功都把新的位置信息赋值到全局的位置对象，然后每个需要使用位置请求的地方都使用全局的位置信息进行请求。 该方案好处：请求的时候无需再反复定位，每次请求都使用全局的位置对象，节省时间。 该方案弊端：耗电，每隔一定时间自动刷新位置，对电量的消耗比较大。

方案 2：按需定位，每次请求前都进行定位。这样做的好处是比较省电，而且节省资源，但是请求时间会变得相对较长

十一、推送(2016.5.5)

72．极光推送的缺陷(高版本的会出现闪)（★★★）

问推送的话，可以扩展下你对推送的了解。

例子：其实做推送的有很多，如云巴，百度，蝴蝶，极光，个推等。我基本上是用极光和个推。本人在用期间基本没出现什么问题。但我同事在用时好像在 android 高版本上出现过闪屏现象。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

其实从技术上讲，百度、腾讯的技术实力并不会比个推、极光差，但是，这些大公司的大牛基本上核心不在推送。所以技术方面，我认为单从推送这个业务上来讲，以个推、极光相对要好些。技术上每家各有长短，但是独立推送方更愿意在技术上下功夫；

总的来说极光推送蛮不错的，至少有时出了问题后联系客服能得到很好的解决，所以很多 app 应用愿意去一直使用。因此现在用极光都占大多数。

73．极光推送的效率问题？(2016.5.5)（★★★）

极光推送效率：高稳定性，大容量，高并发：目前极光推送平台支持 十亿级用户，高达 20 万/秒的下行速度

十二、工作学习篇(2016.5.5)（★★★★）

74．喜欢逛哪些国外网站

（1）StackOverFlow（工作上大部分的 bug 都在上面能得到解答）

http://stackoverflow.com/

（2）GitHub(github 里面有很多的优秀开源库，而且还有中文社区)

https://github.com/

https://github.com/Trinea/android-open-project（中文社区）

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

（3）Android developer 谷歌官方开发文档

https://developer.android.com/guide/index.html

（4）搜索工具 google 与 DuckDuckgo（鸭子搜索），搜索出来的资料相对精辟

https://www.google.com.hk/?gws_rd=ssl

https://duckduckgo.com/

（5）codota 搜索最好的 Android 代码

http://www.codota.com/

拓展：

国内网站： CSDN（国内知名的技术论坛，很多大牛都在上面开了博客）

http://blog.csdn.net/?ref=toolbar_logo

国内外优秀博客：

https://github.com/android-cn/android-dev-cn 国内

https://github.com/android-cn/android-dev-com 国外

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

75．github 和 git 的区别

github 是一个用 git 做版本控制的项目托管平台。提供 git 托管服务，你可以把你自己的 git 仓库托管在它网站上，这样在任何一台电脑上都可以同步回来。

git 是一个版本控制工具；一般公司还是用 SVN。

76．最后一个问题　你最近在看什么安卓的书，有什么阅后感？

郭霖的《第一行代码》：对基础的总结较好，但是不够细，书中只是略带开发技巧；毕竟是一本入门书。

中级进阶书籍（大家可介绍这两本书）：

徐宜生的《Android 群英传》

任玉刚《Android 开发艺术探索》

这两本书在 Android 的知识点上都进行了原理上的解释，看完之后会有种醍醐灌顶的感觉，对自己所掌握的知识点有了一个全面的深入了解。两本书都涉及了源码原理部分，但是《Android 开发艺术探索》比《Android 群英传》更加细致，有了更多的源码讲解，显得晦涩难懂。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

77．你们以前公司主要做的是哪一个方面的业务

这题考察大家的是否是真的在该公司做过，是否对自己公司的业务进行过了解，方便更好的参与工作中；一个连自己做什么都不知道的员工怎么能做出很好的产品。首先大家要对自己简历上的公司事前进行一个简单的了解，准备好自己的说辞，面试中不断的优化说辞，以备面试中被问到这个问题。

示例：我之前在一家电商公司工作，这两年电商 app 很火，公司当时做的业务是做一个类似于返利网的业务模式；用户通过公司的网上商城进入其他商城如淘宝，国美等平台买东西，公司就可以获得其他电商平台的返利。同时，通过本公司平台前往其他平台购买商品的用户，可获取本公司相应的积分；该积分累计到一定程度，可参与公司定期举办的“免费商品房”摇号权利。

78．你在工作过程中,觉得收获最大的是什么,提升最多的是什么

考察部分：工作态度，是否能够解决业务工作

事例：感触最深的是第一份工作，刚开始进公司只是了解技术开发；对公司的整体环境还不大熟悉，当时自己就努力熟悉公司的大环境；第一份工作让自己学到了更多，并影响了后来的一些工作方式，真正对工作游刃有余。IT 技术更新快，做技术之后，在技术部门的强度工作中养成了不断学习的习惯。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

79．谈谈你的职业规划

这个题目的背后是了解求职者的求职念头和对自己中长期职业发展的思索。而此时需要传递给 HR 的信息是：

1、你是有备而来的，由于对自己是有职业规划的，所以求职投简历都是经由一番思索的，而不是海投盲投的结果。

2、表明自己的上风，体现出自己是这份工作的不二人选。

3、你有着良好的职业规划，清晰地知道未来三到五年的计划和打算，表明你是符合企业要求的不乱型员工。

4、其实在面试的时候回答面试问题职业规划并没有那么难，关键是求职者对自身职场发展的方向，及职业目标有一个清楚的认识，给自己定好位，这样面试中才能做到游刃有余。

职业规划范文：

第一年：快速融入工作环境，同公司文化融为一体，承担起在公司的职责。熟悉公司的业务，不断发现自己需要学习改进的地方。提升自己，进而为自己提出切合实际的目标。

第二、三年：丰富自己的专业知识，根据自己的工作表现适时调整自己的规划，进行能力与经验的积累，提升自己的价值。

第四、五年：调整自己的规划，进入一个新的层次，向更高点看齐。程序员中有高级程序员，或者做部门负责人。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

十、 即时通讯是是怎么做的?（★★★★★）

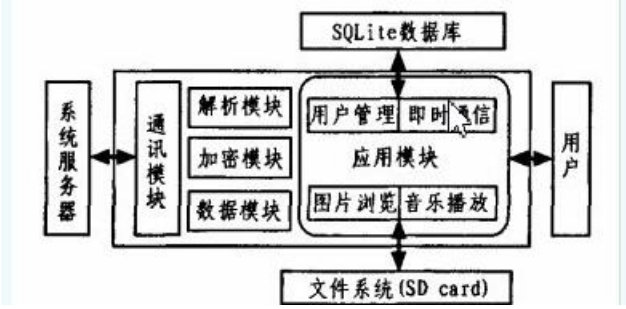
答: 1.我在之前的项目中使用了 asmark 开源框架实现的即时通讯功能.该框架基于开源的 XMPP 即时通信协议，采用 C / S 体系结构，通过 GPRS 无线网络用 TCP 协议连接到服务器，以架设开源的 Openfn'e 服务器作为即时通讯平台。

2. 客户端基于 Android 平台进行开发。负责初始化通信过程，进行即时通信时，由客户端负责向服务器发起创建连接请求。系统通过 GPRS 无线网络与 Internet 网络建立连接，通过服务器实现与 Android 客户端的即时通信脚。

3.服务器端则采用 Openfire 作为服务器。允许多个客户端同时登录并且并发的连接到一个服务器上。服务器对每个客户端的连接进行认证，对认证通过的客户端创建会话，客户

端与服务器端之间的通信就在该会话的上下文中进行。

客户端的补充内容:



《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

a. 客户端分为五大模块开发:

通讯模块负责与服务器建立通讯旧。通过创建 3 个线程来进行处理。分别负责消息的发送、接收和系统信息的发送.

解析模块主要用来解析 XML 数据流根据解析元素不同类型封装成不同的数据对象.

数据模块定义整个客户端中大部分的数据类型和对象；

应用模块包括即时通信、图片浏览和音乐播放。是客户端和用户交流的接口.

加密模块对发送和接收的消息进行加解密。以确保通讯数据的安全。

b. xmpp 协议补充内容:

1.XMPP（可扩展消息处理现场协议）是基于可扩展标记语言（XML）的协议，它用于即时消息（IM）以及在线现场探测。它在促进服务器之间的准即时操作。这个协议可能最终允许因特网用户向因特网上的其他任何人发送即时消息，即使其操作系统和浏览器不同。

2.XMPP 协议的优势

与 HTTP 相比，XMPP 具有如下的优势：

（1）能够“推送”数据，而不是“拉”。

（2）防火墙友好(???)

（3）牢固的身份验证和安全机制

（4）为许多不同的问题提供大量即开即用的工具

3.XMPP 协议的不足

每种协议都有各自的优缺点，在许多场合中 XMPP 并不是完成任务的最佳工具或受到某种限制。

（1）有状态协议

（2）社区和部署不及 HTTP 广泛

（3）对于简单的请求，其开销比 HTTP 大

（4）仍然需要专门的实现

4.XMPP 协议的数据格式

在 XMPP 中 ,各项工作都是通过在一个 XMPP 流上发送和接收 XMPP 节来完成的。核心 XMPP 工具集由三种基本节组成，这三种节分别为<presence>、出席<message>、<iq>。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

XMPP 流由两份 XML 文档组成，通信的每个方向均有一份文档。这份文档有一个根元素<stream:stream>，这个根元素的子元素由可路由的节以及与流相关的顶级子元素构成。

下面给出一段简短的 XMPP 会话：roster 花名册

拉取联系人

```
<stream:stream>

    <iq type='get'>

        <query xmlns='jabber:iq:roster'/>

    </iq>

    <presence type=’ available’ >

    <message   from='laowang@itcast.cn' />

    </stream:stream>
```

简单解释下：

（1）laowang 登陆后，将自己的第一节（一个<iq>元素）发送出去，这个<iq>元素请求 laowang 的联系人列表。

（2）接下来，他使用<presence>节通知服务器他在线并且可以访问。

（3）当 laowang 注意到 cui 也在线时，他发送了一条<message>节给 cui。

（4）最后，laowang 发送了一个<presence>节告诉服务器自己不可访问并关闭<stream:stream>元素，然后结束会话。

所有三种节都支持一组通用的属性：

（1）from

（2）to

（3）type

（4）id

5.1 presence 节

<presence>节控制并报告实体 Jid 的可访问性，

Mode: 在线 chat、离线 away 、离开 xa、请勿打扰 dnd

Type:available, unavailable,subscribe,subscribed,unsubscribe,

Unsubscribed,error

此外，这个节还用来建立和终止向其他实体发布出席订阅。

普通 presence 节

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

普通<presence>节不含 type 属性 ,或者 type 属性值为 unavailable 或 error。type 属性没有 available 值，因为可以通过缺少 type 属性来指出这种情况。

用户通过发送不带 to 属性，直接发往服务器的<presence>节来操纵自己的出席状态。

```
<presence/>

<presence type='unavailable'/>

<presence>

    <show>away</show>

    <status>at the ball</status>

<presence/>

<presence>

    <status>touring the countryside</status>

    <priority>10</priority>

</presence>
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
<presence>

    <priority>10</priority>

</presence>
```

简单解释：

（1）<show>元素用来传达用户的可访问性，只能出现在<presence>节中。该元素的值可以为：away、chat、dnd 和 xa，分别表示离开、有意聊天、不希望被打扰和长期离开。

（2）<status>元素时一个人类可读的字符串，在接收者的聊天客户端中，这个字符串一般会紧挨着联系人名字显示。

（3）<priority>元素用来指明连接资源的优先级，介于-128~127，默认值为 0。

扩展 presence 节

开发人员希望能够扩展<presence>节以包含更详细的信息，比如用户当前听的歌或个人的情绪。

因为<presence>节会广播给所有联系人，并且在 XMPP 网络流量中占据了很大的份额，所以不鼓励这种做法。这类扩展应该交给额外信息传送协议来处理。

出席订阅

用户的服务器会自动地将出席信息广播给那些订阅该用户出席信息的联系人。类似的，用户从所有他已经出席订阅的联系人那里接收到出席更新信息。

与一些社交网络和 IM 系统不同，在 XMPP 中，出席订阅是有方向的。我订阅了你的出席信息，但是你并不一定订阅了我的出席信息。

可以通过设置 type 的值来识别出席订阅节：subscribe(请求加好友)、unsubscribe（删除好友）、subscribed（答应加好友的请求）、unsubscribed（拒绝加好友的请求）。

```
public enum Type {

        /**

         * The user is available to receive messages (default).

         */

         available,

         /**

          * The user is unavailable to receive messages.

          */
```

```
        unavailable,

        /**

         * Request subscription to recipient's presence.

         */

        subscribe,

        /**

         * Grant subscription to sender's presence.

         */

        subscribed,

        /**

         * Request removal of subscription to sender's
presence.

         */

        unsubscribe,
```

```
        /**
         * Grant removal of subscription to sender's
presence.
         */
        unsubscribed,
        /**
         * The presence packet contains an error message.
         */
        error
}
```

定向出席

定向出席是一种直接发给另一个用户或其他实体的普通<presence>节，这种节用来向那些没有进行出席订阅（通常因为只是临时需要出席信息）的实体传达出席状态信息。

message 节

<message>节用来从一个实体向另一个实体发送消息。这些可以是简单的聊天信息，也可以是任何结构化信息。例如，绘制指令、游戏状态和新游戏变动状况。

<message>属于发送后不管的类型，没有内在的可靠性，就像电子邮件一样。在有些情况下（比如向不存在的服务器发送信息），发送者可能会收到一个错误提示节，从中了解出现的问题。可以通过在应用程序协议中增加确认机制来实现可靠传送。

消息类型

<message>节有几种不同的类型，这些类型由 type 属性指出，可取的值有：chat、error、normal、groupchat 和 headline。该属性是可选的，如果没有指定 type 值，默认为 normal。

normal　a normal text message used in email like interface

groupchat Chat message sent to a groupchat server for group chats.

chat　Chat message sent to a groupchat server for group chats

消息内容

尽管<message>节可以包含任意扩展元素，但<body>和<thread>元素是为向消息中添加内容提供的正常机制。这两种子元素均是可选的。

Sets the thread id of the message, which is a unique identifier for a sequence of "chat" messages. thread the thread id of the message.

IQ 节

<iq>节表示的是Info/Query ,它为XMPP通信提供请求和响应机制。它与HTTP协议的基本工作原理非常相似 ,允许获取和设置查询 ,与 HTTP 的 GET 和 POST 动作类似。

<iq>节有四种 ,通过 type 属性区分 ,其中两种请求 get 和 set ,两种响应 result 和 error。每一个 IQ-get 或 IQ-set 节均必须接收响应的 IQ-result 和 IQ-error 节。此外，每一对<iq>必须匹配 id 属性。

Get： 获取当前域值

Set： 设置或替换 get 查询的值

Result： 说明成功的响应了先前的查询

Error： 查询和响应中出现的错误

The base IQ (Info/Query) packet. IQ packets are used to get

and set information on the server, including authentication, roster operations, and creating accounts.

Each IQ packet has a specific type that indicates what type of action is being taken: "get", "set", "result", or "error".

IQ packets can contain a single child element that exists in a specific XML namespace. The combination of the element name and namespace determines what type of IQ packet it is.

query xmlns="jabber:iq:auth"

十一、 设计模式六大原则（★★★）

1.单一职责原则:不要存在多于一个导致类变更的原因。

通俗的说:即一个类只负责一项职责。

2.里氏替换原则:所有引用基类的地方必须能透明地使用其子类的对象。

通俗的说:当使用继承时。类 B 继承类 A 时，除添加新的方法完成新增功能 P2 外，尽量不要重写父类 A 的方法，也尽量不要重载父类 A 的方法。如果子类对这些非抽象方法任意修改，就会对整个继承体系造成破坏。子类可以扩展父类

的功能，但不能改变父类原有的功能。

3.依赖倒置原则:高层模块不应该依赖低层模块，二者都应该依赖其抽象；抽象不应该依赖细节；细节应该依赖抽象。

通俗的说:在 java 中，抽象指的是接口或者抽象类，细节就是具体的实现类，使用接口或者抽象类的目的，是制定好规范和契约，而不去涉及任何具体的操作，把展现细节的任务交给他们的实现类去完成。依赖倒置原则的核心思想是**面向接口编程.**

4.接口隔离原则:客户端不应该依赖它不需要的接口；一个类对另一个类的依赖应该建立在最小的接口上。

通俗的说:建立单一接口，不要建立庞大臃肿的接口，尽量细化接口，接口中的方法尽量少。也就是说，我们要为各个类建立专用的接口，而不要试图去建立一个很庞大的接口供所有依赖它的类去调用。

5.迪米特法则:一个对象应该对其他对象保持最少的了解

通俗的说:尽量降低类与类之间的耦合。

6.开闭原则:一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。

通俗的说:**用抽象构建框架，用实现扩展细节。**因为抽象灵活性好，适应性广，只要抽象的合理，可以基本保持软件架构的稳定。而软件中易变的细节，我们用

从抽象派生的实现类来进行扩展，当软件需要发生变化时，我们只需要根据需求重新派生一个实现类来扩展就可以了。

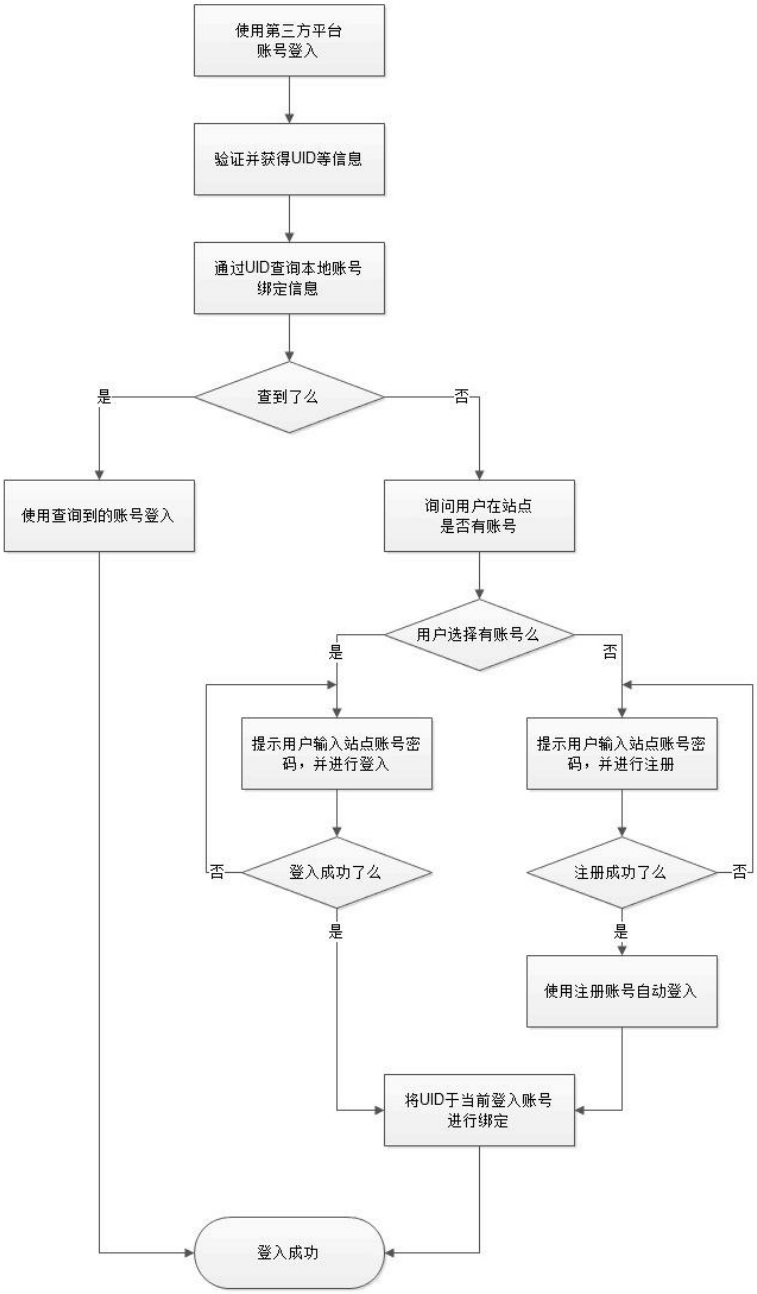
十二、 第三方登陆（★★★★）

参考资料：

http://blog.csdn.net/infsafe/article/details/10209401

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队



1、你们需要支持用户注册

2、你们需要在应用登录的时候提供第三方平台的图标

3、用户点击第三方平台图标以后，你们尝试判断用户是否已经授权

4、如果用户授权，获取他的唯一识别符，比方说 WeiboDb 里面的 weiboId 这个字段

5、如果用户没有授权，引导用户授权，授权成功后也可以获取 weibo Id

6、然后用这个唯一识别符登录你们的系统，如果用户已经注册，则应该让用户登录到你们的系统，流程结束

7、如果你们的系统发现用户没有注册，引导用户进入你们应用的注册页面，并通过 share sdk 的 showuser 方法获取用户资料，自动帮助用户完成注册资料的填写，然后等待用户确认

8、如果用户确认了注册信息，你们的应用就根据他的信息完成这注册操作，如果操作成功，则应该让用户登录到你们的系统，流程结束

十三、 第三方支付（★★★★★）

申请流程

注册支付宝账号??进行实名认证??提交审核资料??审核通过

备注：申请通过后会获得：合作者身份 ID（PID），该 ID 在项目配置中需要用到

开发流程：

第一步：

下载 API 开发文档后，即可获取官方 Demo，该 Demo 中需要将审核通过后获取的 PID 替换，并且输入支付宝收款账户即可。这里非常简单，就不过多叙述。

第二步：

官方 Api 开发文档中，存在一个 openssl 的文件夹，该文件夹主要是用于生成支付宝所需要用到的公钥以及私钥。打开该文件夹可以看到详细的生成方式，根据提示生成公钥及私钥，请注意，密钥需要经过 pkcs8 二次加密。

第三步：

将生成的公钥和私钥配置到 Demo 中。

第四步（可省略）：

为了方便后期维护 ,建议将支付宝相关的方法及配置项抽取出来做为单独的一个类，后期需要使用直接调用即可。

十四、 常见框架分析（★★★★★）

1. Otto 与 EventBus

这两个框架的实现原理差不多，在开发中常用 Otto.

我们假设这样一种业务场景，现在在做一款及时聊天应用，我们在聊天页面进行收发信息，同时也要实时更新前一页面的聊天记录，这时我们该如何去实现？可以使用的是广播接收器 BroadCastReceiver ,在接收和发送消息的时候就不停去发送广播，然后在需要实时更新的地方进行接收更新。实现的思想比较简单，

也不存在代码上的耦合问题，但是有个弊端。弊端就是需要去在很多地方实现 BroadCastRecevier，代码虽不算冗余，但比较多，看起来很是不爽。使用 Otto 能解决代码体积的问题。Otto 是一款目前比较流行事件总线框架，旨在保持应用各页面和模块之间通信高效的前提下，对应用进行解耦。Otto 是基于观察者设计模式，简单来说，如果你想订阅某个消息，使用@Subcribe 注解即可进行接收，同时使用

Bus.post(Object obj)进行消息的发布，这样的设计达到了完全的解耦。

Otto 使用过程:

一、Bus 实例化

Bus 这个类是整个框架的灵魂，它负责消息的发布和接收，整个流程都是经过这个 Bus 去实现的。Bus 的实例化推荐使用单例，就是说整个应用内只实例化一个 Bus 对象，所有的消息的处理都是经过这单一的实例去实现。因为要实现消息的接受者接收到发布的消息，一定要经过同一个 Bus 对象的处理。Bus 的构造器可以接收 ThreadEnforcer 类型的参数，ThreadEnforcer 其实是一个接口，它自身有两个实现，分别表示 Bus 运行在 Main Thread 中还是异步线程中。

```
public final class BusProvider {
  private static final Bus BUS = new Bus(ThreadEnforcer.MAIN);

  public static Bus getInstance() {
    return BUS;
  }

  private BusProvider() {
    // No instances.
  }
}
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

二、注册和解绑 Bus

根据具体的业务需求进行 Bus 的注册和解绑，对于 android 中的组件，一般是基于生命周期方法中去实现；同时如果是任意你自定义的类中都可以进行。下面展示的是在 Activity 和 Fragment 里面实现。

```
@Override public void onResume() {
  super.onResume();
  BusProvider.getInstance().register(this);
}

@Override public void onPause() {
  super.onPause();
  BusProvider.getInstance().unregister(this);
}
```

三、消息的发布

发布消息是整个框架中最重要的部分，它允许你告诉所有的订阅者一个事件已经触发。任何一个类的实例对象都可以通过总线 Bus 去发布，同时也只能被订阅这种对象的接受者所接收。下面展示的是通过 Bus 去发布一个消息，消息的内容是 LocationChangeEvent，所以 LocationChangeEvent 的接受者都能接收到此发布的消息。注意的是，发布消息只能一个 Object 对象。

```
// Post new location event
lastLatitude = DEFAULT_LAT;
lastLongitude = DEFAULT_LON;
LocationChangedEvent event = new LocationChangedEvent(
        lastLatitude, lastLongitude);
BusProvider.getInstance().post(event);
```

四、消息的订阅

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

消息的订阅和发布之前都要在当前的类中进行 Bus 的注册。订阅是对消息发布的补充，当消息发布的事件调用之后，对应的消息订阅者就能立即接收到此消息。实现订阅功能是通过自定义方法实现的，方法的名称可以随意，同时还得需要满足三个条件。

```
@Subscribe public void onLocationChanged(LocationChangedEvent event) {
  locationEvents.add(0, event.toString());
  if (adapter != null) {
    adapter.notifyDataSetChanged();
  }
}
```

1、方法前使用@Subscribe 注解

2、访问修饰符为 public

3、单一参数，根据你想订阅的消息进行设置

注：使用之前，记得进行注册；使用完毕，记得释放。

```
@Subscribe public void onLocationChanged(LocationChangedEvent event) {
  locationEvents.add(0, event.toString());
  if (adapter != null) {
    adapter.notifyDataSetChanged();
  }
}
```

五、消息的 produce

当订阅者注册完毕，针对特定的消息，通常也需要获取当前已知的值。这个时候，就需要用到 produce。同样的使用 produce 的方法名称可以随意，同时有三点需要注意。

1、方法前使用@produce 注解

2、访问修饰符为 public

3、无参，返回值是基于订阅者参数类型

```
@Produce public LocationChangedEvent produceLocationEvent() {
  // Provide an initial value for location based on the last known position.
  return new LocationChangedEvent(lastLatitude, lastLongitude);
}
```

当然 Otto 的缺点也是有的，要实现上述订阅/发布模型的功能，付出的代价就是对各个注册 Bus 的类进行反射。如果大量的使用的情况下，对应用的性能多少有点副影响。

实现原理:

这个框架所实现的功能本质是在一个对象中调用任意的另一个或多个对象中的方法，而不需要将这个或者这些对象传入调用者.

如果把它们都传入调用者去调用它们的方法，那么调用者就依赖这些对象，程序的耦合度就高了，所以说这个框架是用来解耦的.

无论 EventBus 还是 Otto 都有一个注册(register)的方法，方法参数是需要订阅事件的对象.

register 方法会拿到参数的 class 文件,

Otto 通过反射获取类中有@Subscribe 注解的方法,

将该 Method 对象和参数放入 Otto 内部的一个集合中,

在发布事件时调用 post 方法，post 方法会根据参数类型，在这个集合中找到 register 时放入的相对应 Method 对象,

调用这个 Method 所需要的对象是 register 时传入的参数，所需的参数是 post

时传入的参数，这些都已经有了，直接将它 invoke

EventBus 与 Otto 不同仅在于 Otto 是通过注解来确定哪些方法是需要接收事件的方法，而 EventBus 是通过固定的方法名来确定的，所以在项目上线，代码混淆时，使用了 EventBus 的类都不能混淆，因此在项目中使用 Otto 会更多一点.

1.ImageLoader

一个强大的图片加载框架，很好的处理了图片加载的多线程，缓存，内存溢出等问题.

1. 多线程异步加载和显示图片（图片来源于网络、sd 卡、assets 文件夹，drawable 文件夹（不能加载 9patch） 可以加载视频缩略图）

Uri 参数:

http:// site.com/image.png

file:/// mnt/sdcard/image.png

file:/// mnt/sdcard/video.mp4

content:// media/external/images/media/13

content://media/external/video/media/13

assets:// image.png

drawable://R.drawable.img

可以这么使用

assets: Scheme.ASSETS.wrap(path)

drawable: Scheme.DRAWABLE.wrap(path)

file: Scheme.FILE.wrap(path)

content: Scheme.CONTENT.wrap(path)

http: Scheme.HTTP.wrap(path)

https: Scheme.HTTPS.wrap(path)

2、支持监视加载的过程，可以暂停加载图片，在经常使用的 ListView、GridView 中，可以设置滑动时暂停加载，停止滑动时加载图片

3、高度可定制化（可以根据自己的需求进行各种配置，如：线程池，图片下载器，内存缓存策略等）

4、支持图片的内存缓存，SD 卡（文件）缓存

5、在网络速度较慢时，还可以对图片进行加载并设置下载监听

6、减少加载大图片 OOM 的情况

7、避免同一 Uri 加载过程中重复开启任务加载

框架中三个核心类:

ImageLoader

ImageLoaderConfiguration

DisplayImageOptions

ImageLoaderConfiguration 是给 ImageLoader 做配置的，DisplayImageOptions 是显示图片的一些配置. 这两个类都使用了构建器模式.

下面是使用的例子，代码中的注释写的很详细:

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

MyApplication.java

最简单的使用范例:

```
public class MyApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
            File cacheDir = StorageUtils.getCacheDirectory(this);
            ImageLoaderConfiguration config = new ImageLoaderConfiguration
                    .Builder(this)
                    .denyCacheImageMultipleSizesInMemory()
                    .diskCache(new UnlimitedDiskCache(cacheDir))
                    .diskCacheExtraOptions(480, 800, null)
                    .build();
                    ImageLoader.getInstance().init(config);
    }
}
```

在 AndroidManifest 文件的 application 节点下配置:

android:name="cn.itcast.demo.app.MyApplication"

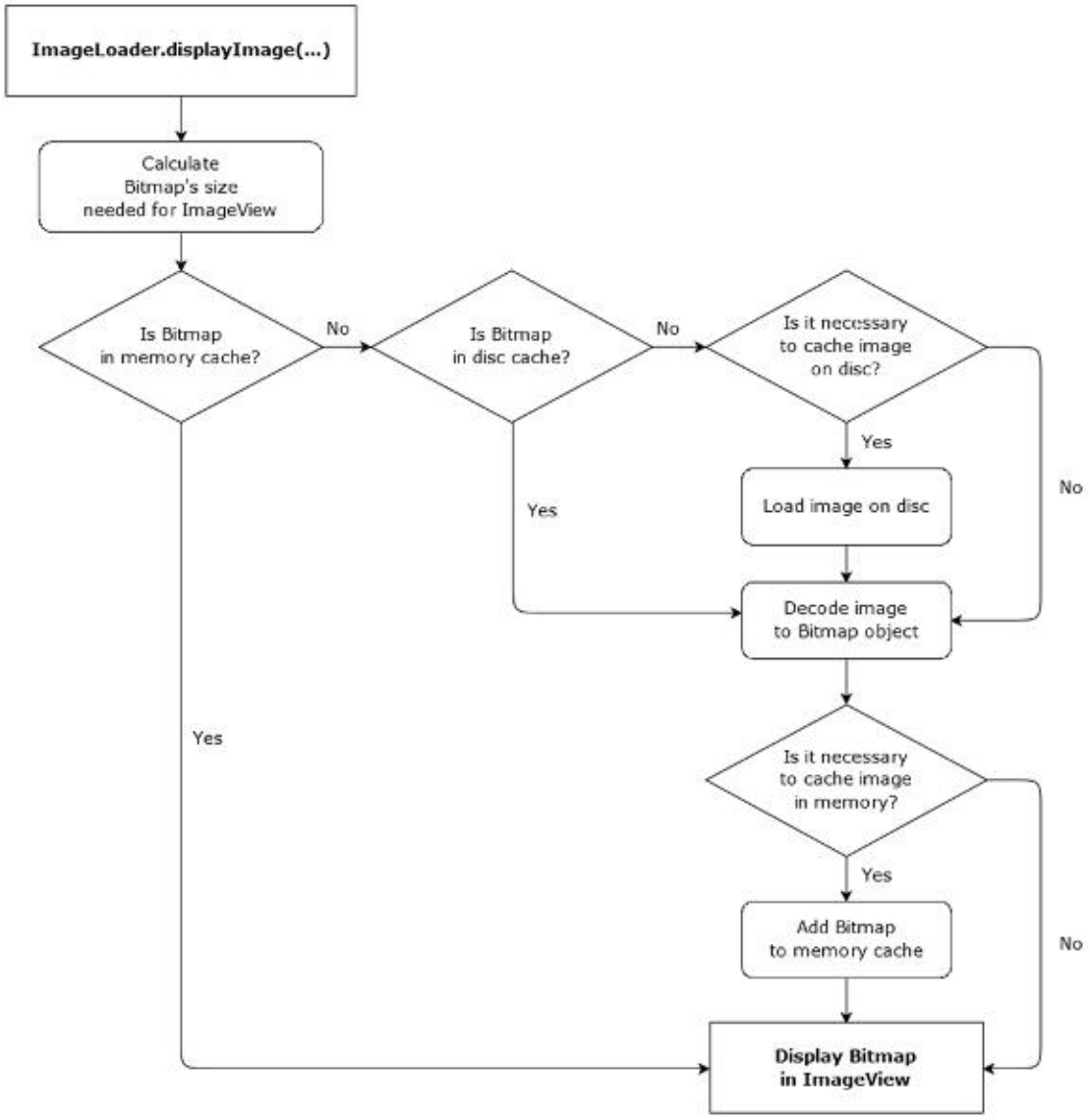
使用:

ImageLoader.getInstance().displayImage(url);

ImageLoader 显示图片一般使用 displayImage 方法，它的执行流程图:

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队



可以看到就是讲过的三级缓存.

2.BufferKnife 和 AndroidAnnotations

这是两个轻量级的 IOC 框架，类似 J2ee 的 Spring. 安卓类似的框架还有很多，这两个用的比较多，也较具有代表性，这两个框架的实现原理不是很重要，但是它的思想必须要了解.

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

IOC 是面向对象编程中非常重要的一个原则, IOC 的意思是控制反转: Inversion of Control，控制反转分为依赖注入和依赖查找，这两个框架是依赖注入.

控制反转指的是创建对象的功能转移了,可以把它看做是工厂模式的升华，也是为了解耦

ButterKnife 是通过注解+反射来实现的，作用在于解耦和减少代码书写量．缺点在于使用反射实现，会影响程序运行速度.

注解：其实注解没有什么其它的用途，它只是一个标志，当你需要给某个类，莫个方法,或某个属性打一个特定的标记，就可以使用注解.

ButterKnife 的使用比较简单，举个栗子，如果你想自动 findViewById，那么需要两步

1. 给属性加上@InjectView(R.id.xx) 括号中的参数是这个控件的 id

2. 在 Activity 的 setContentView 之后调用 ButterKnife.inject(this);

实现过程:

inject 方法会拿到传入的 Activity 的 class 对象,

获取类中所有的 Field,

判断该属性是否被@InjectView 注解,

如果被注解，获取注解的值

调用 findViewById()传入注解的值，获取 view 对象

将 view 对象设置给这个 field

AndroidAnnotations 框架是另一种实现模式，它在编译时会给注解的类生成一

个子类，在这个子类做了初始化和绑定事件等操作，在使用时应该使用这个子类，比如注解 Activity 时，在 AndroidManifest 文件配置时是配置生成的这个子类而不是原类，子类的类名是原类类名+_,比如原类名是 MainActivity，生成的子类类名就是 MainActivity_.

相比其它 IOC 框架它的优点在于没有使用反射运行更快，缺点在于会生成大量类文件，增加 apk 体积.

ButterKnife 使用示例:

```
public class MainActivity extends Activity {

    //相当于 findViewById(R.id.tv)
    @InjectView(R.id.tv)
    TextView tv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.inject(this);//将对象注入,就能对它用 ButterKnife 的注解
    }
    //给 id=R.id.tv 的控件设置点击事件
    @OnClick(R.id.tv)
    public void show(View v){
```

```
        Toast.makeText(this,"butterKnif!", Toast.LENGTH_LONG).show();
    }
}
```

3.Picasso, ImageLoader, Fresco, Glide 优劣

首先看 Fresco，它的优点是其他几个框架没有的，或者说是其他几个框架的短板.

Fresco:

优点:

1. 图片存储在安卓系统的匿名共享内存，而不是虚拟机的堆内存中，图片的中间缓冲数据也存放在本地堆内存，所以，应用程序有更多的内存使用，不会因为图片加载而导致 oom，同时也减少垃圾回收器频繁调用回收 Bitmap 导致的界面卡顿，性能更高.
2. 渐进式加载 JPEG 图片，支持图片从模糊到清晰加载
3. 图片可以以任意的中心点显示在 ImageView，而不仅仅是图片的中心.
4. JPEG 图片改变大小也是在 native 进行的，不是在虚拟机的堆内存，同样减少 OOM
5. 很好的支持 GIF 图片的显示

缺点:

1. 框架较大，影响 Apk 体积

2. 使用较繁琐

ImageLoader, Picasso, Glide: 这三者实现机制都差不多

ImageLoader:

比较老的框架，稳定，加载速度适中，缺点在于不支持 GIF 图片加载，使用稍微繁琐，并且缓存机制没有和 http 的缓存很好的结合，完全是自己的一套缓存机制.

Picasso:

使用方便，一行代码完成加载图片并显示，框架体积小，

缺点在于不支持 GIF，并且它可能是想让服务器去处理图片的缩放，它缓存的图片是未缩放的，并且默认使用 ARGB_8888 格式缓存图片，缓存体积大.

Glide:

可以说是 Picasso 的升级版，有 Picasso 的优点，并且支持 GIF 图片加载显示，图片缓存也会自动缩放，默认使用 RGB_565 格式缓存图片，是 Picasso 缓存体积的一半.

4.Volley 和 Ok-http,android-async-http,retrofit

volley 是一个简单的异步 http 库，仅此而已。比较适合小的而频繁的 Http 请求

缺点是不支持同步，这点会限制开发模式；

不能 post 大数据，所以不适合用来上传文件。

android-async-http,与 volley 一样是异步网络库，使用了 nio 的方式实现，不过 nio 更适合大量连接的情况，对于移动平台有点杀鸡用牛刀的味道。volley 是封装的 httpUrlConnection，android-async-http 封装的 httpClient，而 android 平台不推荐用 HttpClient 了，所以这个库已经不适合 android 平台了。

okhttp 是高性能的 http 库，支持同步、异步，而且实现了 spdy、http2、websocket 协议，api 很简洁易用，和 volley 一样实现了 http 协议的缓存。

picasso 就是利用 okhttp 的缓存机制实现其文件缓存，实现的很优雅，很正确，反例就是 UIL(universal image loader)，自己做的文件缓存，而且不遵守 http 缓存机制。

retrofit 与 picasso 一样都是在 okhttp 基础之上做的封装，项目中可以直接用了。

picasso、uil 都不支持 inbitmap，项目中有用到 picasso 的富图片应用需要注意这点。

okhttp 和 async http 是一个基础的通信库，都很强大，但需要自己封装使用才更方便。另外 okhttp 已经被谷歌官方用在 android 源码中了。 retrofit 和 volley 是属于比较高级点的封装库了，其中 retrofit 是默认使用 okhttp，volley 也支持 okhttp 作为其底层通信的部件。retrofit 的特点是使用清晰简单的接口，非常方便，而 volley 在使用的时候也还简单，不过要使用高级一点的功能需要自己自定义很多东西.

推荐使用 retrofit+okhttp. 最新的 retorfit 已经支持 nio 了，而且 retorfit 内置了 RxJava.

5. RxJava

RxJava 是一个异步操作库，一个在 Java VM 上使用可观测的序列来组成异步的、基于事件的程序的库.它是仿照.net 的 Reactive Extensions（Rx）类库设计的，可以帮助我们更容易的进行相应式编程. RxJava 的优势在于简洁.

这个简洁不是指代码书写量少，而是代码逻辑与结构简洁. 异步操作很关键的一点是程序的简洁性，因为在调度过程比较复杂的情况下，异步代码经常会既难写也难被读懂。 Android 创造的 AsyncTask 和 Handler ，其实都是为了让异步代码更加简洁。

RxJava 的简洁的与众不同之处在于，随着程序逻辑变得越来越复杂，它依然能够保持简洁，能够很大程度提升代码的阅读性，易于后期升级维护. 他通过一种扩展的观察者模式来实现的, RxJava 有四个基本概念：Observable (可观察者，即被观察者)、 Observer (观察者)、 subscribe (订阅事件)。Observable 和 Observer 通过 subscribe() 方法实现订阅关系，从而 Observable 可以在需要的时候发出事件来通知 Observer.

它的缺点在于有一定的学习成本.

6. Volley

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

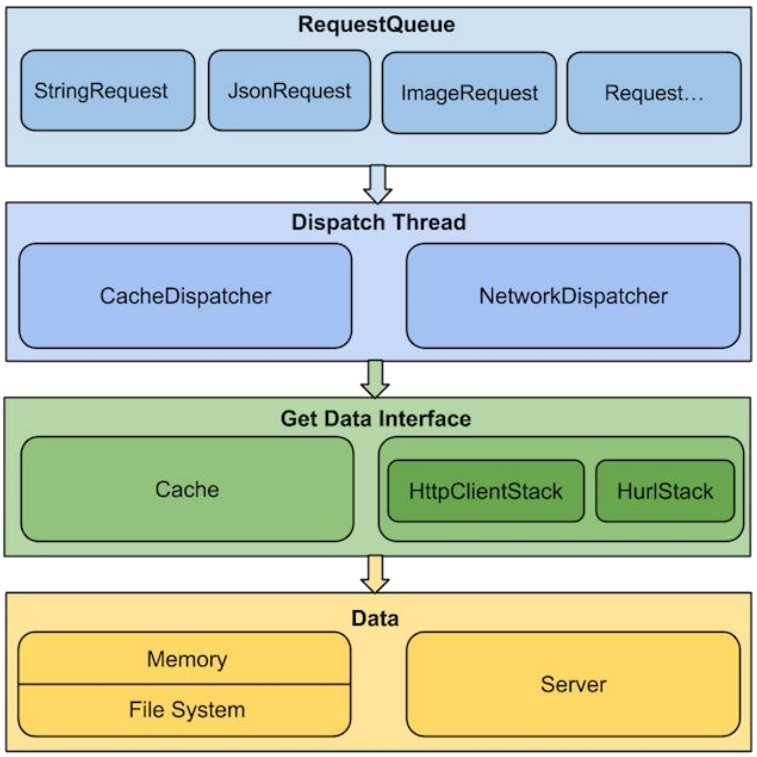
Volley 的主要特点

(1). 扩展性强。Volley 中大多是基于接口的设计，可配置性强。

(2). 一定程度符合 Http 规范，包括返回 ResponseCode(2xx、3xx、4xx、5xx) 的处理，请求头的处理，缓存机制的支持等。并支持重试及优先级定义。

(3). 默认 Android2.3 及以上基于 HttpURLConnection，2.3 以下基于 HttpClient 实现

(4). 提供简便的图片加载工具。



上面是 Volley 的总体设计图，主要是通过两种 Diapatch Thread 不断从 RequestQueue 中取出请求，根据是否已缓存调用 Cache 或 Network 这两类

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

数据获取接口之一，从内存缓存或是服务器取得请求的数据，然后交由 ResponseDelivery 去做结果分发及回调处理. 也是三级缓存.

Volley 的优缺点:

优点: Volley 的优点在于请求队列的管理，适合小而频繁的请求，如果 app 比较小，网络请求要求不高的情况下可以使用 volley，通常情况下是要结合其他框架一起来使用，比如 volley+okhttp.

缺点: 下载大文件性能不佳，不支持大文件上传

Volley 简单使用示范:

注意要添加联网权限.

```
public class MainActivity extends Activity {


    @InjectView(R.id.tv)
    TextView tv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.inject(this);
    }
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
    @OnClick(R.id.tv)
    public void show(View v){
        String url =
"http://news.xinhuanet.com/photo/2016-01/29/128684788_14540614045291n.jpg
";
        RequestQueue requestQueue = Volley.newRequestQueue(this);
        MyResponseListener responseListener = new MyResponseListener();

        ImageRequest imageRequest = new ImageRequest(
                url, //图片的 url
                new MyResponseListener(), //响应回调接口
                720,   //图片宽
                1280, //图片高
                ImageView.ScaleType.FIT_XY,
                Bitmap.Config.RGB_565,
                new MyErrorListener()); //请求错误的回调接口
        requestQueue.add(imageRequest); //添加到请求队列
    }

    private class MyErrorListener implements Response.ErrorListener{
```

```
        @Override
        public void onErrorResponse(VolleyError error) {
            //Do Something...
        }
    }

    private class MyResponseListener implements Response.Listener<Bitmap>{
        @Override
        public void onResponse(Bitmap response) {
            Drawable drawable = (Drawable) new
BitmapDrawable(getResources(), response);
            tv.setBackground(drawable);
        }
    }
}
```

Ok-http 使用简单范例

```
public class MainActivity extends Activity {

    @InjectView(R.id.tv)
    TextView tv;
```

```
    private OkHttpClient okHttpClient;
    private Call call;
    private ResponseCallBack responseCallBack;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.inject(this);
        okHttpClient = new OkHttpClient(); //创建 Client
        String url =
"http://news.xinhuanet.com/photo/2016-01/29/128684788_14540614045291n.jpg
";
        Request request = new Request.Builder() //创建请求
                .url(url)
                .build();
        call = okHttpClient.newCall(request); //创建调用对象
        responseCallBack = new ResponseCallBack();   //创建回调接口
        call.enqueue(responseCallBack); //  调用
    }
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
    private class ResponseCallBack implements Callback{
        @Override
        public void onFailure(Call call, IOException e) { }

        @Override
        public void onResponse(Call call, Response response) throws IOException {
            InputStream in = response.body().byteStream();
            Bitmap bitmap = BitmapFactory.decodeStream(in);
            final Drawable drawable = (Drawable) new
BitmapDrawable(getResources(), bitmap);
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    tv.setBackground(drawable);
                }
            });
        }
    }
}
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

第四章：Java 面试题

一、Java 基础（2019.9.20）

1、Java 中引用类型都有哪些

（1）在虚拟机内存不足的情况下，也不会回收强引用对象。如果我们把（强引用）对象置为 null，会大大加大 垃圾回收执行频率。几乎只要我们给出建议（GC），jvm 就会回收。

强引用，例如下面代码：

```
Object o=new Object();
Object o1=o;
```

（2）对于软引用，如果不显式的置为 null 的话，和强引用差不多，垃圾回收不会执行。只会等到内存不足的时候才会调用。

（3）对于弱引用，就算你不显式的把他置为 null，垃圾回收也会立即执行。

（4 ）虚引用，相当于 null。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

2、什么是重载，什么是重写，有什么区别？

重载(Overloading)：

（1）Overloading 是一个类中多态性的一种表现，让类以统一的方式处理不同类型数据的一种手段。多个同名函数同时存在，具有不同的参数个数/类型。

（2）重载的时候，方法名要一样，但是参数类型和个数不一样，返回值类型可以相同，也可以不相同。无法以返回型别作为重载函数的区分标准。

重写（Overriding):

（1）父类与子类之间的多态性，对父类的函数进行重新定义。即在子类中定义某方法与其父类有相同的名称和参数。

（2）若子类中的方法与父类中的某一方法具有相同的方法名、返回类型和参数表，则新方法将覆盖原有的方法。如需父类中原有的方法，可使用 super 关键字，该关键字引用了当前类的父类。

3、String、StringBuffer 和 StringBuilder 的区别

JAVA 早期平台提供了两个类：String 和 StringBuffer，它们可以储存和操作字符串，即包含多个字符的字符数据。这个 String 类提供了数值不可改变的字符串。而这个 StringBuffer 类提供的字符串可以进行修改。当你知道字符数据要改变的时候你就可以使用 StringBuffer。。String 类是不可变类，任何对 String 的改变都会引发新的 String 对象的生成；而 StringBuffer 则是可变类，任何对

它所指代的字符串的改变都不会产生新的对象。

StringBuilder 是后面引入的，它与 StringBuffer 类的区别在于，新引入的 StringBuilder 类不是线程安全的，但其在单线程中的性能比 StringBuffer 高。（有兴趣的可以去读下《Think in Java》描述 HashTable 和 HashMap 区别的那部分章节比较熟悉的话，就是支持线程同步保证线程安全而导致性能下降的问题）

典型地，你可以使用 StringBuffers 来动态构造字符数据。另外，String 实现了 equals 方法，new String(“abc”).equals(newString(“abc”)的结果为 true，而 StringBuffer 没有实现 equals 方法，所以，new StringBuffer(“abc”).equals(newStringBuffer(“abc”)的结果为 false。

4、关键字 final 和 static 是怎么使用的

final 有着“终态的”“这是无法改变的”含义，阻止了多态和继承。

具体使用有：

final 类不能被继承，没有子类，final 类中的方法默认是 final 的。

final 方法不能被子类的方法覆盖，但可以被继承。

final 成员变量表示常量，只能被赋值一次，赋值后值不再改变。

final 不能用于修饰构造方法。

注意：父类的 private 成员方法是不能被子类方法覆盖的，因此 private 类型的

方法默认是 final 类型的。

static 表示“全局”或者“静态”的意思，用来修饰成员变量和成员方法，也可以形成静态 static 代码块，但是 Java 语言中没有全局变量的概念。

被 static 修饰的成员变量和成员方法独立于该类的任何对象,static 对象可以在它的任何对象创建之前访问，无需引用任何对象。

static 前面也可用 public 或 private 来修饰 ,其中 private 是访问权限限定 ,static 表示不要实例化就可以使用。

主要用于静态变量，静态方法，static 代码块

静态变量：对于静态变量在内存中只有一个拷贝（节省内存），JVM 只为静态分配一次内存，在加载类的过程中完成静态变量的内存分配，可用类名直接访问（方便），当然也可以通过对象来访问（但是这是不推荐的）。

静态方法： 静态方法可以直接通过类名调用，任何的实例也都可以调用，因此静态方法中不能用 this 和 super 关键字，不能直接访问所属类的实例变量和实例方法(就是不带 static 的成员变量和成员成员方法)，只能访问所属类的静态成员变量和成员方法。因为实例成员与特定的对象关联！

static 代码块：atic 代码块也叫静态代码块，是在类中独立于类成员的 static 语句块，可以有多个，位置可以随便放，它不在任何的方法体内，JVM 加载类时会执行这些静态的代码块，如果 static 代码块有多个，JVM 将按照它们在类中出现的先后顺序依次执行它们，每个代码块只会被执行一次

static 和 final 一起使用：

static final 用来修饰成员变量和成员方法，可简单理解为“全局常量”！

对于变量，表示一旦给值就不可修改，并且通过类名可以访问。

对于方法，表示不可覆盖，并且可以通过类名直接访问。

特别要注意一个问题：

对于被 static 和 final 修饰过的实例常量，实例本身不能再改变了，但对于一些容器类型（比如，ArrayList、HashMap）的实例变量，不可以改变容器变量本身，但可以修改容器中存放的对象，这一点在编程中用到很多。

5、TCP/IP 协议簇分哪几层？TCP、IP、XMPP、HTTP、分别属于哪一层？（2016.01.24）

通讯协议采用了 4 层的层级结构，每一层都呼叫下一层所提供的网络来完成自己的需求。这 4 层分别为：

应用层：应用程序间沟通的层，如简单电子邮件传输（SMTP）、文件传输协议（FTP）、网络远程访问协议（Telnet）、超文本传输协议(HTTP)、可扩展通讯和表示协议（XMPP）等。

传输层：在此层中，它提供了节点间的数据传送服务，如传输控制协议（TCP）、用户数据报协议（UDP）等，TCP 和 UDP 给数据包加入传输数据并把它传输到下一层中，这一层负责传送数据，并且确定数据已被送达并接收。

互连网络层：负责提供基本的数据封包传送功能，让每一块数据包都能

够到达目的主机（但不检查是否被正确接收），如网际协议（IP）。

网络接口层：对实际的网络媒体的管理，定义如何使用实际网络（如 Ethernet、SerialLine 等）来传送数据。

6、谈一谈 java 线程模型

① 一对一模型

每个用户线程都对应一个内核线程，由于每个用户线程都有各自的内核线程，所以他们互不影响，即使其中一个线程阻塞，也允许另一个线程继续执行，这无疑是此模型的优点，但也存在一个严重的缺陷，由于一对一的关系，有多少个用户线程就代表有多少个内核线程，而内核线程的开销较大，一般操作系统都会有内核线程数量的限制，用户线程的数量也被限制。

② 多对一模型

由一条内核线程实现若干个用户线程的并发功能，线程的管理在用户空间中进行，一般不需要切换到内核态，效率较高，而且比起一对一模型，支持的线程数量更大。但此模型有个致命的弱点是如果一个线程执行了阻塞调用，所有线程都将阻塞，并且任意时刻只能有一个线程访问内核。另外，对线程的所有操作都将由用户应用自己处理。所以一般除了在不支持多线程的操作系统被迫使用此模型外，在多线程操作系统中基本不使用。

③ 多对多模型

多对多模型的提出是为了解决前面两种模型的缺点，多个用户线程与多个内

核线程映射形成多路复用。前面提到的一对一模型存在受内核线程数量限制的问题，多对一模型虽然解决了数量限制问题，但它存在一个线程阻塞导致所有线程阻塞的风险，而且由于一个内核线程只能调度一个线程导致并发性不强。看看多对多模型如何解决这些问题，由于多对一是多对多的子集，所以多对多具备多对一的优点，线程数不受限制。除此之外，多个内核线程可处理多个用户线程，当某个线程阻塞时，将可以调度另外一个线程执行，这从另一方面看也是增强了并发性。

7、java 多线程同步锁

（1）无论 synchronized 关键字加在方法上还是对象上，他取得的锁都是对象，而不是把一段代码或函数当作锁┅而且同步方法很可能还会被其他线程的对象访问。

（2）每个对象只有一个锁（lock）和之相关联。

（3）实现同步是要很大的系统开销作为代价的，甚至可能造成死锁，所以尽量避免无谓的同步控制。

二、Java 中的设计模式（★★★★）

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

1、你所知道的设计模式有哪些

Java 中一般认为有 23 种设计模式，我们不需要所有的都会，但是其中常用的几种设计模式应该去掌握。下面列出了所有的设计模式。需要掌握的设计模式我单独列出来了，当然能掌握的越多越好。

总体来说设计模式分为三大类：

创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

2、单例设计模式

最好理解的一种设计模式，分为懒汉式和饿汉式。

- 饿汉式：

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
public class Singleton {
    // 直接创建对象
    public static Singleton instance = new Singleton();

    // 私有化构造函数
    private Singleton() {
    }

    // 返回对象实例
    public static Singleton getInstance() {
        return instance;
    }
}
```

懒汉式：

```
public class Singleton {
    // 声明变量
    private static volatile Singleton singleton2 = null;

    // 私有构造函数
    private Singleton2() {
    }

    // 提供对外方法
    public static Singleton2 getInstance() {
        if (singleton2 == null) {
            synchronized (Singleton2.class) {
                if (singleton == null) {
                    singleton = new Singleton();
                }
            }
        }
        return singleton;
    }
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

3、工厂设计模式

工厂模式分为工厂方法模式和抽象工厂模式。

- 工厂方法模式

工厂方法模式分为三种：普通工厂模式，就是建立一个工厂类，对实现了同一接口的一些类进行实例的创建。

多个工厂方法模式，是对普通工厂方法模式的改进，在普通工厂方法模式中，如果传递的字符串出错，则不能正确创建对象，而多个工厂方法模式是提供多个工厂方法，分别创建对象。

静态工厂方法模式，将上面的多个工厂方法模式里的方法置为静态的，不需要创建实例，直接调用即可。

- 普通工厂模式

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
public interface Sender {
    public void Send();
}
public class MailSender implements Sender {

    @Override
    public void Send() {
        System.out.println("this is mail sender!");
    }
}
public class SmsSender implements Sender {

    @Override
    public void Send() {
        System.out.println("this is sms sender!");
    }
}
public class SendFactory {
    public Sender produce(String type) {
        if ("mail".equals(type)) {
            return new MailSender();
        } else if ("sms".equals(type)) {
            return new SmsSender();
        } else {
            System.out.println("请输入正确的类型!");
            return null;
        }
    }
}
```

多个工厂方法模式

该模式是对普通工厂方法模式的改进，在普通工厂方法模式中，如果传递的字符串出错，则不能正确创建对象，而多个工厂方法模式是提供多个工厂方法，分别创建对象。

```
public class SendFactory {
    public Sender produceMail(){
        return new MailSender();
    }

    public Sender produceSms(){
        return new SmsSender();
    }
}

public class FactoryTest {
    public static void main(String[] args) {
       SendFactory factory = new SendFactory();
       Sender sender = factory.produceMail();
       sender.send();
    }
}
```

- 静态工厂方法模式，将上面的多个工厂方法模式里的方法置为静态的，不需要创建实例，直接调用即可。

```
public class SendFactory {
    public static Sender produceMail(){
        return new MailSender();
    }

    public static Sender produceSms(){
        return new SmsSender();
    }
}


public class FactoryTest {
    public static void main(String[] args) {
        Sender sender = SendFactory.produceMail();
        sender.send();
    }
}
```

抽象工厂模式

工厂方法模式有一个问题就是，类的创建依赖工厂类，也就是说，如果想要拓展程序，必须对工厂类进行修改，这违背了闭包原则，所以，从设计角度考虑，有一定的问题，如何解决？就用到抽象工厂模式，创建多个工厂类，这样一旦需要增加新的功能，直接增加新的工厂类就可以了，不需要修改之前的代码。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
public interface Provider {
    public Sender produce();
}
---------------------------------------------------------------------
public interface Sender {
    public void send();
}
---------------------------------------------------------------------
public class MailSender implements Sender {

    @Override
    public void send() {
        System.out.println("this is mail sender!");
    }
}
---------------------------------------------------------------------
public class SmsSender implements Sender {

    @Override
    public void send() {
        System.out.println("this is sms sender!");
    }
}
---------------------------------------------------------------------
public class SendSmsFactory implements Provider {

    @Override
    public Sender produce() {
        return new SmsSender();
    }
}
```

```
public class SendMailFactory implements Provider {

    @Override
    public Sender produce() {
        return new MailSender();
    }
}
------------------------------------------------------------------------
public class Test {
    public static void main(String[] args) {
        Provider provider = new SendMailFactory();
        Sender sender = provider.produce();
        sender.send();
    }
}
```

4、建造者模式（Builder）

工厂类模式提供的是创建单个类的模式，而建造者模式则是将各种产品集中起来进行管理，用来创建复合对象，所谓复合对象就是指某个类具有不同的属性，其实建造者模式就是前面抽象工厂模式和最后的 Test 结合起来得到的。

```
public class Builder {
    private List<Sender> list = new ArrayList<Sender>();

    public void produceMailSender(int count) {
        for (int i = 0; i < count; i++) {
            list.add(new MailSender());
        }
    }

    public void produceSmsSender(int count) {
        for (int i = 0; i < count; i++) {
            list.add(new SmsSender());
        }
    }
}
```

```
public class TestBuilder {
    public static void main(String[] args) {
        Builder builder = new Builder();
        builder.produceMailSender(10);
    }
}
```

5、适配器设计模式

适配器模式将某个类的接口转换成客户端期望的另一个接口表示，目的是消除由于接口不匹配所造成的类的兼容性问题。主要分为三类：类的适配器模式、对象的适配器模式、接口的适配器模式。

- 类的适配器模式

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
public class Source {
    public void method1() {
        System.out.println("this is original method!");
    }
}
----------------------------------------------------------------
public interface Targetable {
    /* 与原类中的方法相同 */
    public void method1();
    /* 新类的方法 */
    public void method2();
}
public class Adapter extends Source implements Targetable {
    @Override
    public void method2() {
        System.out.println("this is the targetable method!");
    }
}
public class AdapterTest {
    public static void main(String[] args) {
        Targetable target = new Adapter();
        target.method1();
        target.method2();
    }
}
```

对象的适配器模式

基本思路和类的适配器模式相同，只是将 Adapter 类作修改，这次不继承 Source 类，而是持有 Source 类的实例，以达到解决兼容性的问题。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
public class Wrapper implements Targetable {
    private Source source;

    public Wrapper(Source source) {
        super();
        this.source = source;
    }

    @Override
    public void method2() {
        System.out.println("this is the targetable method!");
    }

    @Override
    public void method1() {
        source.method1();
    }
}
----------------------------------------------------------------
public class AdapterTest {

        public static void main(String[] args) {
            Source source = new Source();
            Targetable target = new Wrapper(source);
            target.method1();
            target.method2();
        }
    }
```

接口的适配器模式

接口的适配器是这样的：有时我们写的一个接口中有多个抽象方法，当我们写该接口的实现类时，必须实现该接口的所有方法，这明显有时比较浪费，因为并不是所有的方法都是我们需要的，有时只需要某一些，此处为了解决这个问题，我们引入了接口的适配器模式，借助于一个抽象类，该抽象类实现了该接口，实

现了所有的方法，而我们不和原始的接口打交道，只和该抽象类取得联系，所以我们写一个类，继承该抽象类，重写我们需要的方法就行。

6、装饰模式（Decorator）

顾名思义，装饰模式就是给一个对象增加一些新的功能，而且是动态的，要求装饰对象和被装饰对象实现同一个接口，装饰对象持有被装饰对象的实例。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
public interface Sourceable {
    public void method();
}
-------------------------------------------------------
public class Source implements Sourceable {
    @Override
    public void method() {
        System.out.println("the original method!");
    }
}
-------------------------------------------------------
public class Decorator implements Sourceable {
    private Sourceable source;
    public Decorator(Sourceable source) {
        super();
        this.source = source;
    }

    @Override
    public void method() {
        System.out.println("before decorator!");
        source.method();
        System.out.println("after decorator!");
    }
}
-------------------------------------------------------
public class DecoratorTest {
    public static void main(String[] args) {
        Sourceable source = new Source();
        Sourceable obj = new Decorator(source);
        obj.method();
    }
}
```

7、策略模式（strategy）

策略模式定义了一系列算法，并将每个算法封装起来，使他们可以相互替换，

且算法的变化不会影响到使用算法的客户。需要设计一个接口，为一系列实现类提供统一的方法，多个实现类实现该接口，设计一个抽象类（可有可无，属于辅助类），提供辅助函数。策略模式的决定权在用户，系统本身提供不同算法的实现，新增或者删除算法，对各种算法做封装。因此，策略模式多用在算法决策系统中，外部用户只需要决定用哪个算法即可。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
public interface ICalculator {
    public int calculate(String exp);
}
-----------------------------------------------------------
public class Minus extends AbstractCalculator implements ICalculator {

    @Override
    public int calculate(String exp) {
        int arrayInt[] = split(exp, "-");
        return arrayInt[0] - arrayInt[1];
    }
}
-----------------------------------------------------------
public class Plus extends AbstractCalculator implements ICalculator {

    @Override
    public int calculate(String exp) {
        int arrayInt[] = split(exp, "\\+");
        return arrayInt[0] + arrayInt[1];
    }
}
-----------------------------------------------------------
public class AbstractCalculator {
    public int[] split(String exp, String opt) {
        String array[] = exp.split(opt);
        int arrayInt[] = new int[2];
        arrayInt[0] = Integer.parseInt(array[0]);
        arrayInt[1] = Integer.parseInt(array[1]);
        return arrayInt;
    }
}
```

```
public class StrategyTest {
    public static void main(String[] args) {
        String exp = "2+8";
        ICalculator cal = new Plus();
        int result = cal.calculate(exp);
        System.out.println(result);
    }
}
```

8、观察者模式（Observer）

观察者模式很好理解，类似于邮件订阅和 RSS 订阅，当我们浏览一些博客或 wiki 时，经常会看到 RSS 图标，就这的意思是，当你订阅了该文章，如果后续有更新，会及时通知你。其实，简单来讲就一句话：当一个对象变化时，其它依赖该对象的对象都会收到通知，并且随着变化！对象之间是一种一对多的关系。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
public interface Observer {
    public void update();
}

public class Observer1 implements Observer {
    @Override
    public void update() {
        System.out.println("observer1 has received!");
    }
}

public class Observer2 implements Observer {
    @Override
    public void update() {
        System.out.println("observer2 has received!");
    }
}

public interface Subject {
     /*增加观察者*/
    public void add(Observer observer);

    /*删除观察者*/
    public void del(Observer observer);
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

```
    /*通知所有的观察者*/
    public void notifyObservers();

    /*自身的操作*/
    public void operation();
}

public abstract class AbstractSubject implements Subject {

    private Vector<Observer> vector = new Vector<Observer>();

    @Override
    public void add(Observer observer) {
        vector.add(observer);
    }

    @Override
    public void del(Observer observer) {
        vector.remove(observer);
    }

    @Override
    public void notifyObservers() {
        Enumeration<Observer> enumo = vector.elements();
        while (enumo.hasMoreElements()) {
            enumo.nextElement().update();
        }
    }
}

public class MySubject extends AbstractSubject {

    @Override
    public void operation() {
        System.out.println("update self!");
         notifyObservers();
    }
}

public class ObserverTest {
    public static void main(String[] args) {
        Subject sub = new MySubject();
        sub.add(new Observer1());
```

```
        sub.add(new Observer2());
        sub.operation();
    }
}
```

第五章：Flutter 相关面试题全解析

Dart 部分（10 月更新）

1. Dart 语言的特性？

- Productive（生产力高，Dart 的语法清晰明了，工具简单但功能强大）
- Fast（执行速度快，Dart 提供提前优化编译，以在移动设备和 Web 上获得可预测的高性能和快速启动。）
- Portable（易于移植，Dart 可编译成 ARM 和 X86 代码，这样 Dart 移动应用程序可以在 iOS、Android 和其他地方运行）
- Approachable（容易上手，充分吸收了高级语言特性，如果你已经知道 C++，C 语言，或者 Java，你可以在短短几天内用 Dart 来开发）
- Reactive（响应式编程）

2.Dart 的一些重要概念？

- 在 Dart 中，一切都是对象，所有的对象都是继承自 Object
- Dart 是强类型语言，但可以用 var 或 dynamic 来声明一个变量，Dart 会自动推断其数据类型，dynamic 类似 c#
- 没有赋初值的变量都会有默认值 null
- Dart 支持顶层方法，如 main 方法，可以在方法内部创建方法

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

- Dart 支持顶层变量，也支持类变量或对象变量
- Dart 没有 public protected private 等关键字，如果某个变量以下划线（_）开头，代表这个变量在库中是私有的

3.Dart 当中的 「..」表示什么意思？

- Dart 当中的 「..」意思是 「级联操作符」，为了方便配置而使用。
- 「..」和「.」不同的是 调用「..」后返回的相当于是 this，而「.」返回的则是该方法返回的值 。
- **2. Dart 的作用域**
- Dart 没有 「public」「private」等关键字，默认就是公开的，私有变量使用 下划线 _开头。
- **3. Dart 是不是单线程模型？是如何运行的？**
- Dart 是单线程模型，如何运行的看这张图：
-
- 引用《Flutter 中文网》里的话：
- Dart 在单线程中是以消息循环机制来运行的，其中包含两个任务队列，一个是“微任务队列” microtask queue，另一个叫做“事件队列” event queue。
- 入口函数 main() 执行完后，消息循环机制便启动了。首先会按照先进先出的顺序逐个执行微任务队列中的任务，当所有微任务队列执行完后便开始执行事件队列中的任务，事件任务执行完毕后再去执行微任务，如此循环往复，生生不息。

4. Dart 多任务如何并行的？

- 刚才也说了，既然 Dart 不存在多线程，那如何进行多任务并行？

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

- Dart 当中提供了一个 **类似于新线程，但是不共享内存的独立运行的 worker - isolate。**
- 那他们是如何交互的？
- 这里引用 flutter 入门之 dart 中的并发编程、异步和事件驱动详解 中的一部分答案：
-
- 在 dart 中，一个 Isolate 对象其实就是一个 isolate 执行环境的引用，一般来说我们都是通过当前的 isolate 去控制其他的 isolate 完成彼此之间的交互，而当我们想要创建一个新的 Isolate 可以使用 Isolate.spawn 方法获取返回的一个新的 isolate 对象，两个 isolate 之间使用 SendPort 相互发送消息，而 isolate 中也存在了一个与之对应的 ReceivePort 接受消息用来处理，但是我们需要注意的是，ReceivePort 和 SendPort 在每个 isolate 都有一对，只有同一个 isolate 中的 ReceivePort 才能接受到当前类的 SendPort 发送的消息并且处理。

5.dart 是值传递还是引用传递？

dart 是值传递。我们每次调用函数，传递过去的都是对象的内存地址，而不是这个对象的复制。
先来看段代码

```
main(){

  Test a = new Test(5);

  print("a 的初始值为: ${a.value}");

  setValue(a);

  print("修改后 a 的值为: ${a.value}");

}
```

```
class Test{

  int value = 1;

  Test(int newValue){

    this.value = newValue;

  }

}



setValue(Test s){

  print("修改 value 为 100");
```

Flutter 部分（11 月更新）

1. Flutter 是什么？

Flutter 是谷歌的移动 UI 框架，可以快速在 iOS 和 Android 上构建高质量的原生用户界面。 Flutter 可以与现有的代码一起工作。在全世界，Flutter 正在被越来越多的开发者和组织使用，并且 Flutter 是完全免费、开源的。

2. Flutter 特性有哪些？

快速开发（毫秒级热重载）

- 绚丽 UI（内建漂亮的质感设计 Material Design 和 Cupertino Widget 和丰富平滑的动画效果和平台感知）
- 响应式(Reactive，用强大而灵活的 API 解决 2D、动画、手势、效果等难题)
- 原生访问功能

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

堪比原生性能

3.基础知识

Flutter 和 React Native 不同主要在于 **Flutter UI 是直接通过 skia 渲染的 ，而 React Native 是将 js 中的控件转化为原生控件，通过原生去渲染的** ，相关更多可查看：《移动端跨平台开发的深度解析》。

Flutter 中存在 `Widget` 、 `Element` 、`RenderObject` 、`Layer` 四棵树，其中 `Widget` **与** `Element` **是一对多的关系** ，

`Element` 中持有 `Widget` 和 `RenderObject` ， 而 `Element` **与** `RenderObject` **是一一对应的关系（除去** `Element` **不存在** `RenderObject` **的情况，如** `ComponentElement` **是不具备** `RenderObject`**）** ，

当 `RenderObject` 的 `isRepaintBoundary` 为 `true` 时，那么个区域形成一个 `Layer`，所以**不是每个** `RenderObject` **都具有** `Layer` **的，因为这受** `isRepaintBoundary` **的影响。**

更多相关可查阅 《Flutter 完整开发实战详解(九、 深入绘制原理)》

《Android 架构师面试题精编解析大全》

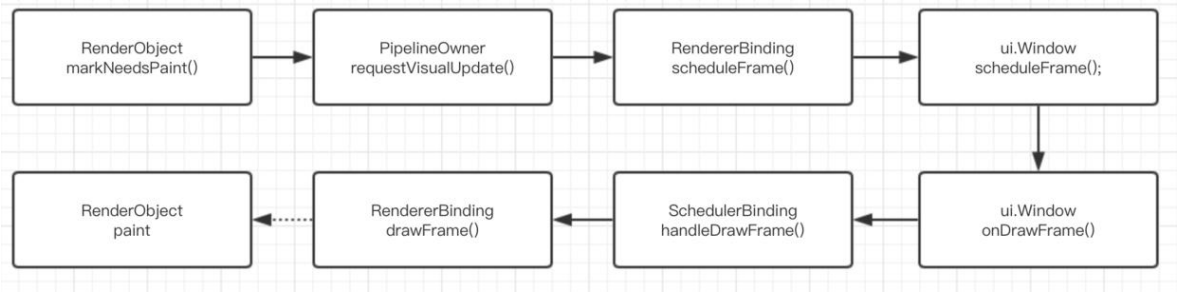
------腾讯享学课堂团队

Flutter 中 Widget 不可变，每次保持在一帧，如果**发生改变是通过** State **实现跨帧状态保存**，而**真实完成布局和绘制数组的是** RenderObject ，Element 充当两者的桥梁， State **就是保存在** Element **中。**

Flutter 中的 BuildContext **只是接口，而** Element **实现了它。**

Flutter 中 setState **其实是调用了** markNeedsBuild ，该方法内部**标记此** Element **为** Dirty ，然后在下一帧 WidgetsBinding.drawFrame 才会被绘制，这可以看出 setState **并不是立即生效的。**

Flutter 中 RenderObject 在 attch/layout 之后会通过 markNeedsPaint(); 使得页面重绘，流程大概如下：

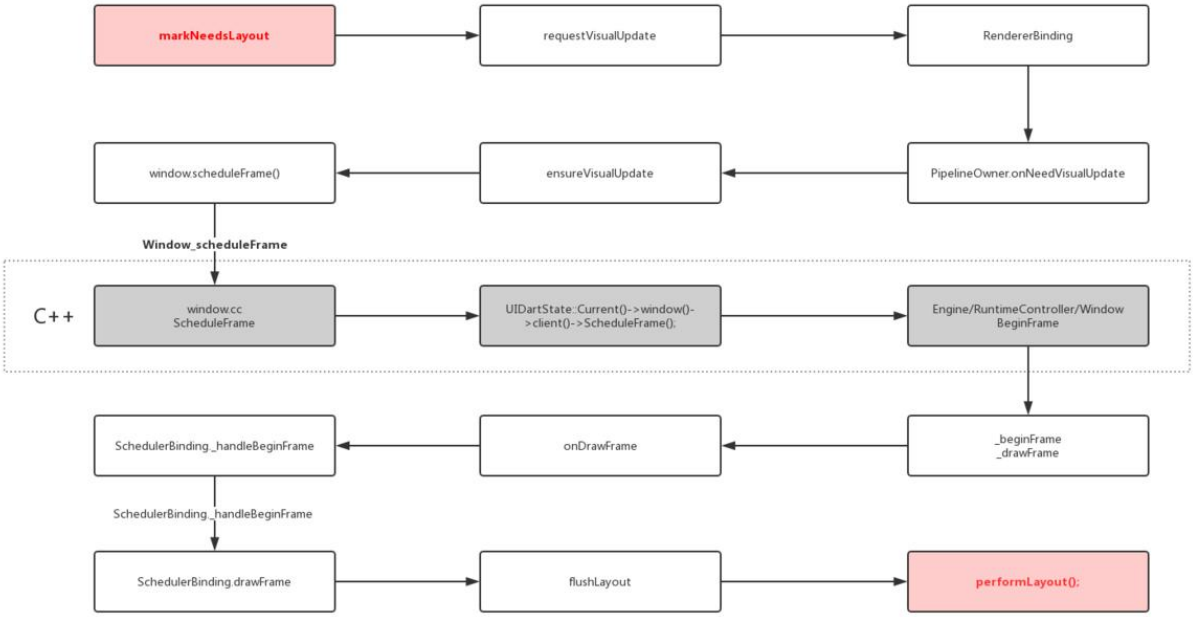


通过 isRepaintBoundary 往上确定了更新区域，通过 requestVisualUpdate 方法触发更新往下绘制。

正常情况 RenderObject 的布局相关方法调用顺序是 ： layout ->performResize ->performLayout ->markNeedsPaint，但是用户一般不会直接调用 layout，而是**通过** markNeedsLayout ，具体流程如下：

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队



Flutter 中一般 **json** 数据从 `String` 转为 `Object` 的过程中都需要先经过 `Map` 类型。

Flutter 中 `InheritedWidget` 一般用于**状态共享**，如 `Theme` 、`Localizations` 、 `MediaQuery` 等，都是通过它实现共享状态，这样我们可以通过 `context` 去获取共享的状态，比如 `ThemeData theme = Theme.of(context);`

在 `Element` 的 `inheritFromWidgetOfExactType` 方法实现里，有一个 `Map<Type, InheritedElement> _inheritedWidgets` 的对象。

_inheritedWidgets 一般情况下是空的，只有当父控件是 InheritedWidget 或者本身是 InheritedWidgets 时才会有被初始化，而当父控件是 InheritedWidget 时，这个 Map 会被一级一级往下传递与合并 。

所以当我们通过 context 调用 inheritFromWidgetOfExactType 时，就可以往上查找到父控件的 Widget 。

Flutter 中默认主要通过 runtimeType 和 key 判断更新：

```
static bool canUpdate(Widget oldWidget, Widget newWidget) {
return oldWidget.runtimeType == newWidget.runtimeType
&& oldWidget.key == newWidget.key;
  }
}
```

4.Flutter 中的生命周期

initState() 表示当前 State 将和一个 BuildContext 产生关联，但是此时 BuildContext 没有完全装载完成，如果你需要在该方法中获取 BuildContext ，可以 new Future.delayed(const Duration(seconds: 0, (){//context}); 一下。

didChangeDependencies() 在 initState() 之后调用 ,当 State 对象的依赖关系发生变化时，该方法被调用，初始化时也会调用。

`deactivate()` 当 `State` 被暂时从视图树中移除时，会调用这个方法，同时页面切换时，也会调用。

`dispose()` Widget 销毁了，在调用这个方法之前，总会先调用 deactivate()。

`didUpdateWidge` 当 `widget` 状态发生变化时，会调用。

通过 `StreamBuilder` 和 `FutureBuilder` 我们可以快速使用 `Stream` 和 `Future` 快速构建我们的异步控件: 《Flutter 完整开发实战详解(十一、全面深入理解 Stream)》

Flutter 中 `runApp` 启动入口其实是一个 `WidgetsFlutterBinding` ，它主要是通过 `BindingBase` 的子类 `GestureBinding` 、`ServicesBinding` 、`SchedulerBinding` 、`PaintingBinding` 、`SemanticsBinding` 、`RendererBinding` 、`WidgetsBinding` 等，通过 `mixins` 的组合而成的。

Flutter 中的 Dart 的线程是以**事件循环和消息队列**的形式存在，包含两个任务队列，**一个是 microtask 内部队列，一个是 event 外部队列，而 microtask 的优先级又高于 event 。**

> 因为 microtask 的优先级又高于 event， 同时会阻塞 event 队列，所以如果 microtask 太多就可能会对触摸、绘制等外部事件造成阻塞卡顿哦。

Flutter 中存在**四大线程，分别为** UI Runner、GPU Runner、IO Runner，Platform Runner **（原生主线程）**，同时在 Flutter 中可以通过 isolate 或者 compute 执行真正的跨线程异步操作。

5.PlatformView

Flutter 中通过 PlatformView 可以嵌套原生 View 到 Flutter UI 中，这里面其实是使用了 Presentation＋VirtualDisplay＋Surface 等实现的，大致原理就是：

使用了类似副屏显示的技术，VirtualDisplay 类代表一个虚拟显示器，调用 DisplayManager 的 createVirtualDisplay() 方法，将虚拟显示器的内容渲染在一个 Surface 控件上，然后将 Surface 的 id 通知给 Dart，让 engine 绘制时，在内存中找到对应的 Surface 画面内存数据，然后绘制出来。em... **实时控件截图渲染显示技术。**

> **Flutter 的 Debug 下是 JIT 模式，release 下是 AOT 模式。**

> Flutter 中可以通过 mixins AutomaticKeepAliveClientMixin ，然后重写 wantKeepAlive 保持住页面，记得在被保持住的页面 build 中调用 super.build 。（因为 mixins 特性）。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

Flutter 手势事件主要是通过竞技判断的：

主要有 hitTest 把所有需要处理的控件对应的 RenderObject ， 从 child 到 parent 全部组合成列表，从最里面一直添加到最外层。

然后从队列头的 child 开始 for 循环执行 handleEvent 方法，执行 handleEvent 的过程不会被拦截打断。

一般情况下 Down 事件不会决出胜利者，大部分时候是在 MOVE 或者 UP 的时候才会决出胜利者。

竞技场关闭时只有一个的就直接胜出响应 ,没有胜利者就拿排在队列第一个强制胜利响应。

同时还有 didExceedDeadline 处理按住时的 Down 事件额外处理 ,同时手势处理一般在 GestureRecognizer 的子类进行。

更多详细请查看：《Flutter 完整开发实战详解(十三、全面深入触摸和滑动原理)》

Flutter 中 ListView 滑动其实都是通过改变 ViewPort 中的 child 布局来实现显示的。

常用状态管理的：目前有 scope_model 、flutter_redux 、fish_redux 、bloc + Stream 等几种模式，具体可见：《Flutter 完整开发实战详解(十二、全面深入理解状态管理设计)》

6.Platform Channel

Flutter 中可以通过 Platform Channel 让 Dart 代码和原生代码通信的：

- BasicMessageChannel ：用于传递字符串和半结构化的信息。
- MethodChannel ：用于传递方法调用（method invocation）。
- EventChanne l: 用于数据流（event streams）的通信。

同时 Platform Channel 并非是线程安全的，更多详细可查阅闲鱼技术的《深入理解 Flutter Platform Channel》

其中基础数据类型映射如下：

7.Android 启动页

Android 中 Flutter 默认启动时会在 FlutterActivityDelegate.java 中读取 AndroidManifset.xml 内 meta-data 标签，其中

`io.flutter.app.android.SplashScreenUntilFirstFrame` 标志位如果为 ture ，就会启动 Splash 画面效果（类似 IOS 的启动页面）。

启动时原生代码会读取 `android.R.attr.windowBackground` 得到指定的 `Drawable` ， 用于显示启动闪屏效果，之后并且通过 `flutterView.addFirstFrameListener`，在 `onFirstFrame` 中移除闪屏。

8. Flutter 和 Dart 的关系是什么？

Flutter 是一个使用 Dart 语言开发的跨平台移动 UI 框架，通过自建绘制引擎，能高性能、高保真地进行移动开发。Dart 囊括了多数编程语言的优点，它更符合 Flutter 构建界面的方式。

9. Widget 和 element 和 RenderObject 之间的关系？

- Widget 是用户界面的一部分,并且是不可变的。
- Element 是在树中特定位置 Widget 的实例。
- RenderObject 是渲染树中的一个对象，它的层次结构是渲染库的核心。

Widget 会被 inflate（填充）到 Element，并由 Element 管理底层渲染树。Widget 并不会直接管理状态及渲染,而是通过 State 这个对象来管理状态。Flutter 创建 Element 的可见树，相对于 Widget 来说，是可变的，通常界面开发中，我们不用直接操作 Element,而是由框架层实现内部逻辑。就如一个 UI 视图树中，可能包含有多个 TextWidget(Widget

被使用多次)，但是放在内部视图树的视角，这些 TextWidget 都是填充到一个个独立的 Element 中。Element 会持有 renderObject 和 widget 的实例。记住，Widget 只是一个配置，RenderObject 负责管理布局、绘制等操作。

在第一次创建 Widget 的时候，会对应创建一个 Element，然后将该元素插入树中。如果之后 Widget 发生了变化，则将其与旧的 Widget 进行比较，并且相应地更新 Element。重要的是，Element 不会被重建，只是更新而已。

10. mixin extends implement 之间的关系?

继承(关键字 extends)、混入 mixins (关键字 with)、接口实现(关键字 implements)。这三者可以同时存在，前后顺序是 extends -> mixins -> implements。

Flutter 中的继承是单继承，子类重写超类的方法要用@Override，子类调用超类的方法要用 super。

在 Flutter 中 ,Mixins 是一种在多个类层次结构中复用类代码的方法。mixins 的对象是类，mixins 绝不是继承，也不是接口，而是一种全新的特性，可以 mixins 多个类，mixins 的使用需要满足一定条件。

11. 使用 mixins 的条件是什么？

因为 mixins 使用的条件，随着 Dart 版本一直在变，这里讲的是 Dart2.1 中使用 mixins 的条件：

mixins 类只能继承自 object

mixins 类不能有构造函数

一个类可以 mixins 多个 mixins 类

可以 mixins 多个类，不破坏 Flutter 的单继承

12. mixin 怎么指定异常类型？

on 关键字可用于指定异常类型。 on 只能用于被 mixins 标记的类，例如 mixins X on A，意思是要 mixins X 的话，得先接口实现或者继承 A。这里 A 可以是类，也可以是接口，但是在 mixins 的时候用法有区别.

on 一个类：

```
class A {
void a(){
    print("a");
  }
}



mixin X on A{
void x(){
    print("x");
  }
}

class mixinsX extends A with X{
```

```
}
```

on 的是一个接口： 得首先实现这个接口，然后再用 mix

```
class A {
void a(){
   print("a");
  }
}

mixin X on A{
void x(){
   print("x");
  }
}
class implA implements A{
  @override
void a() {}
}
class mixinsX2 extends implA with X{
}
```

13. Flutter main future mirotask 的执行顺序?

普通代码都是同步执行的，结束后会开始检查 microtask 中是否有任务，若有则执行，执行完继续检查 microtask ,直到 microtask 列队为空。最后会去执行 event 队列(future)。

14. Future 和 Isolate 有什么区别？

future 是异步编程，调用本身立即返回，并在稍后的某个时候执行完成时再获得返回结果。在普通代码中可以使用 await 等待一个异步调用结束。

isolate 是并发编程，Dartm 有并发时的共享状态，所有 Dart 代码都在 isolate 中运行，包括最初的 main()。每个 isolate 都有它自己的堆内存，意味着其中所有内存数据，包括全局数据，都仅对该 isolate 可见，它们之间的通信只能通过传递消息的机制完成，消息则通过端口(port)收发。isolate 只是一个概念，具体取决于如何实现，比如在 Dart VM 中一个 isolate 可能会是一个线程，在 Web 中可能会是一个 Web Worker。

15. Stream 与 Future 是什么关系？

Stream 和 Future 是 Dart 异步处理的核心 API。Future 表示稍后获得的一个数据，所有异步的操作的返回值都用 Future 来表示。但是 Future 只能表示一次异步获得的数据。而 Stream 表示多次异步获得的数据。比如界面上的按钮可能会被用户点击多次，所以按钮上的点击事件（onClick）就是一个 Stream 。简单地说，Future 将返回一个值，而 Stream 将返回多次值。Dart 中统一使用 Stream 处理异步事件流。Stream 和一般的集合类似，都是一组数据，只不过一个是异步推送，一个是同步拉取。

16. Stream 两种订阅模式？

Stream 有两种订阅模式：**单订阅(single)** 和 **多订阅（broadcast）**。单订阅就是只能有一个订阅者，而广播是可以有多个订阅者。这就有点类似于消息服务(Message Service)

的处理模式。单订阅类似于点对点，在订阅者出现之前会持有数据，在订阅者出现之后就才转交给它。而广播类似于发布订阅模式，可以同时有多个订阅者，当有数据时就会传递给所有的订阅者，而不管当前是否已有订阅者存在。

Stream 默认处于单订阅模式，所以同一个 stream 上的 listen 和其它大多数方法只能调用一次，调用第二次就会报错。但 Stream 可以通过 transform() 方法（返回另一个 Stream）进行连续调用。通过 Stream.asBroadcastStream() 可以将一个单订阅模式的 Stream 转换成一个多订阅模式的 Stream，isBroadcast 属性可以判断当前 Stream 所处的模式。

17. await for 如何使用?

await for 是不断获取 stream 流中的数据，然后执行循环体中的操作。它一般用在直到 stream 什么时候完成，并且必须等待传递完成之后才能使用，不然就会一直阻塞。

```
Stream<String> stream = new Stream<String>.fromIterable(['不开心', '面试', '没', '过']);
main() async{
  print('上午被开水烫了脚');
  await for(String s in stream){
    print(s);
  }
  print('晚上还没吃饭');
}
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

18. Flutter 中的 Widget、State、Context 的核心概念？是为了解决什么问题？

Widget：在 Flutter 中，几乎所有东西都是 Widget。将一个 Widget 想象为一个可视化的组件（或与应用可视化方面交互的组件），当你需要构建与布局直接或间接相关的任何内容时，你正在使用 Widget。

Widget 树：Widget 以树结构进行组织。包含其他 Widget 的 widget 被称为父 Widget(或 widget 容器)。包含在父 widget 中的 widget 被称为子 Widget。

Context：仅仅是已创建的所有 Widget 树结构中的某个 Widget 的位置引用。简而言之，将 context 作为 widget 树的一部分，其中 context 所对应的 widget 被添加到此树中。一个 context 只从属于一个 widget，它和 widget 一样是链接在一起的，并且会形成一个 context 树。

State：定义了 StatefulWidget 实例的行为，它包含了用于"**交互/干预**"Widget 信息的行为和布局。应用于 State 的任何更改都会强制重建 Widget。

这些状态的引入，主要是为了解决多个部件之间的交互和部件自身状态的维护。

19. Widget 的两种类型是什么？

StatelessWidget：一旦创建就不关心任何变化，在下次构建之前都不会改变。它们除了依赖于自身的配置信息（在父节点构建时提供）外不再依赖于任何其他信息。比如典型的

Text、Row、Column、Container 等，都是 StatelessWidget。它的生命周期相当简单：初始化、通过 build()渲染。

StatefulWidget：在生命周期内，该类 Widget 所持有的数据可能会发生变化，这样的数据被称为 **State**，这些拥有动态内部数据的 Widget 被称为 StatefulWidget。比如复选框、Button 等。State 会与 Context 相关联，并且此关联是永久性的，State 对象将永远不会改变其 Context，即使可以在树结构周围移动，也仍将与该 context 相关联。当 **state 与 context 关联时**，state 被视为**已挂载**。StatefulWidget 由两部分组成，在初始化时必须要在 createState()时初始化一个与之相关的 State 对象。

20. State 对象的初始化流程？

initState()：一旦 State 对象被创建，initState 方法是第一个（构造函数之后）被调用的方法。可通过重写来执行额外的初始化，如初始化动画、控制器等。重写该方法时，应该首先调用 super.initState()。在 initState 中，无法真正使用 context，因为框架还没有完全将其与 state 关联。initState 在该 State 对象的生命周期内将不会再次调用。

didChangeDependencies()：这是第二个被调用的方法。在这一阶段，context 已经可用。如果你的 Widget 链接到了一个 InheritedWidget 并且/或者你需要初始化一些 listeners（基于 context），通常会重写该方法。

build(BuildContext context)：此方法在 didChangeDependencies()、didUpdateWidget()之后被调用。每次 State 对象更新（或当 InheritedWidget 有新的

通知时）都会调用该方法！我们一般都在 build 中来编写真正的功能代码。为了强制重建，可以在需要的时候调用 setState((){...})方法。

dispose()：此方法在 Widget 被废弃时调用。可重写该方法来执行一些清理操作（如解除 listeners），并在此之后立即调用 super.dispose()。

21. Widget 唯一标识 Key 有那几种？

在 flutter 中，每个 widget 都是被唯一标识的。这个唯一标识在 build 或 rendering 阶段由框架定义。该标识对应于可选的 Key 参数，如果省略，Flutter 将会自动生成一个。

在 flutter 中，主要有 4 种类型的 Key：GlobalKey（确保生成的 Key 在整个应用中唯一，是很昂贵的，允许 element 在树周围移动或变更父节点而不会丢失状态）、LocalKey、UniqueKey、ObjectKey。

22. 什么是 Navigator? MaterialApp 做了什么？

Navigator 是在 Flutter 中负责管理维护页面堆栈的导航器。MaterialApp 在需要的时候，会自动为我们创建 Navigator。Navigator.of(context)，会使用 context 来向上遍历 Element 树，找到 MaterialApp 提供的_NavigatorState 再调用其 push/pop 方法完成导航操作。

23.flutter 与 React Native 有什么不同？

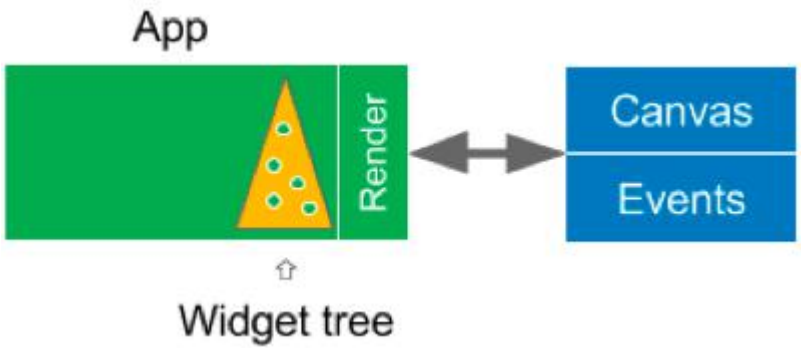
React Native 利用 JavaScript 桥将其小部件转换为 OEM 小部件。而且由于它不断地进行这种转换（比较和更新周期），因此会产生瓶颈并导致性能下降。

虽然仍然使用反应式视图的优势，但 Flutter 并没有使用这种桥将其自己的小部件转换为 OEM 小部件。除了快速和流畅的 UI 性能和可预测性之外，作为此项的另一个优势，您在 Android KitKat 设备上看到的内容与您在 Android Pie 上获得的内容相同。这种兼容性是显而易见的，因为 Flutter 不使用 OEM 小部件，并且不受不同 Android 版本之间的 UI / UX 更改的影响。

24.为什么说 flutter 是原生的

Flutter 应用程序与平台的交互

Flutter 使用名为 Skia 的图形引擎在应用程序端执行所有 UI 呈现。这意味着它不依赖于平台提供的 OEM 小部件。它只需要平台的画布来绘制自己的渲染。这确保了可预测性和开发人员对小部件和布局的完全控制。



Flutter 内部小部件树

除此之外，Flutter 将其结构保持为小部件树。顺便说一下，Flutter 中的几乎所有东西都是一个小部件，它使您能够在小部件内部的小部件结构中构建您的应用程序。此内部树结构允许 Skia 仅呈现需要更新的小部件，并从缓存中检索未更改的甚至移动的小部件。

25.讲一下 flutter 的几个特点/优缺点

Dart 是用于开发 Flutter 应用程序的面向对象，垃圾收集的编程语言。它也是由谷歌创建的，但它是开源的，因此它在 Google 内外都有社区。

《Android 架构师面试题精编解析大全》

除了 Google 的起源之外，Dart 还被选为 Flutter 的语言，原因如下：它是极少数可以同时编译 AOT（提前）和 JIT（即时）的语言之一。

在应用程序开发过程中使用 JIT 编译，因为它可以通过动态编译代码来实现热重新加载（我将在下一个问题中详细讨论）和快速开发周期。
完成开发并准备发布后，将使用 AOT 编译。然后将代码 AOT 编译为本机代码，从而实现应用程序的快速启动和高性能执行。
就个人而言，我对 Dart 的经验是，如果您是熟悉 Java 或类似语言的开发人员，只需要几天的时间就可以习惯它。因此，如果您是 Android 开发人员，那么这种语言的学习曲线应该非常低。

凭借其干净但灵活的语法，Dart 可以被识别为仅包含任何高级编程语言中最需要的功能的语言。

26.什么是 ScopedModel / BLoC 模式？

ScopedModel 和 BLoC（业务逻辑组件）是常见的 Flutter 应用程序架构模式，可帮助将业务逻辑与 UI 代码分离，并使用更少的状态窗口小部件

27.什么是 stateWidget 和 statelessWidget？

内边距 margin 和外边距边距 padding

```
body: Center(

 child: Container(

   child:  new Text("hello zzl ",

   style: TextStyle(

     fontSize: 40.0,

   ),

     textAlign: TextAlign.center,

   ),

   alignment: Alignment.topCenter,
```

```
    width: 500.0,
    height: 400.0,
    color: Colors.blue,
    padding: const EdgeInsets.all(50.0),//外边距
   // margin: const EdgeInsets.all(100.0),//内边距
  ),
  ),
```

填充控件 Padding

Padding 的布局分为两种情况：

当 child 为空的时候，会产生一个宽为 left+right，高为 top+bottom 的区域；
当 child 不为空的时候，Padding 会将布局约束传递给 child，根据设置的 padding 属性，缩小 child 的布局尺寸。然后 Padding 将自己调整到 child 设置了 padding 属性的尺寸，在 child 周围创建空白区域。

```
import 'package:flutter/material.dart';
class PaddingDemo extends StatelessWidget{
  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title:new Text("padding 填充控件"),
      ),
      body: new Padding(
```

```
        padding: const EdgeInsets.all(8.0),
       child: new Image.asset("images/hua3.png"),
     ),
   );
  }
}
void main(){
  runApp(new MaterialApp(
    title: "padding 填充控件",
    home: new PaddingDemo(),
  ));
```

28.如何在 Flutter 中定义边距和填充？

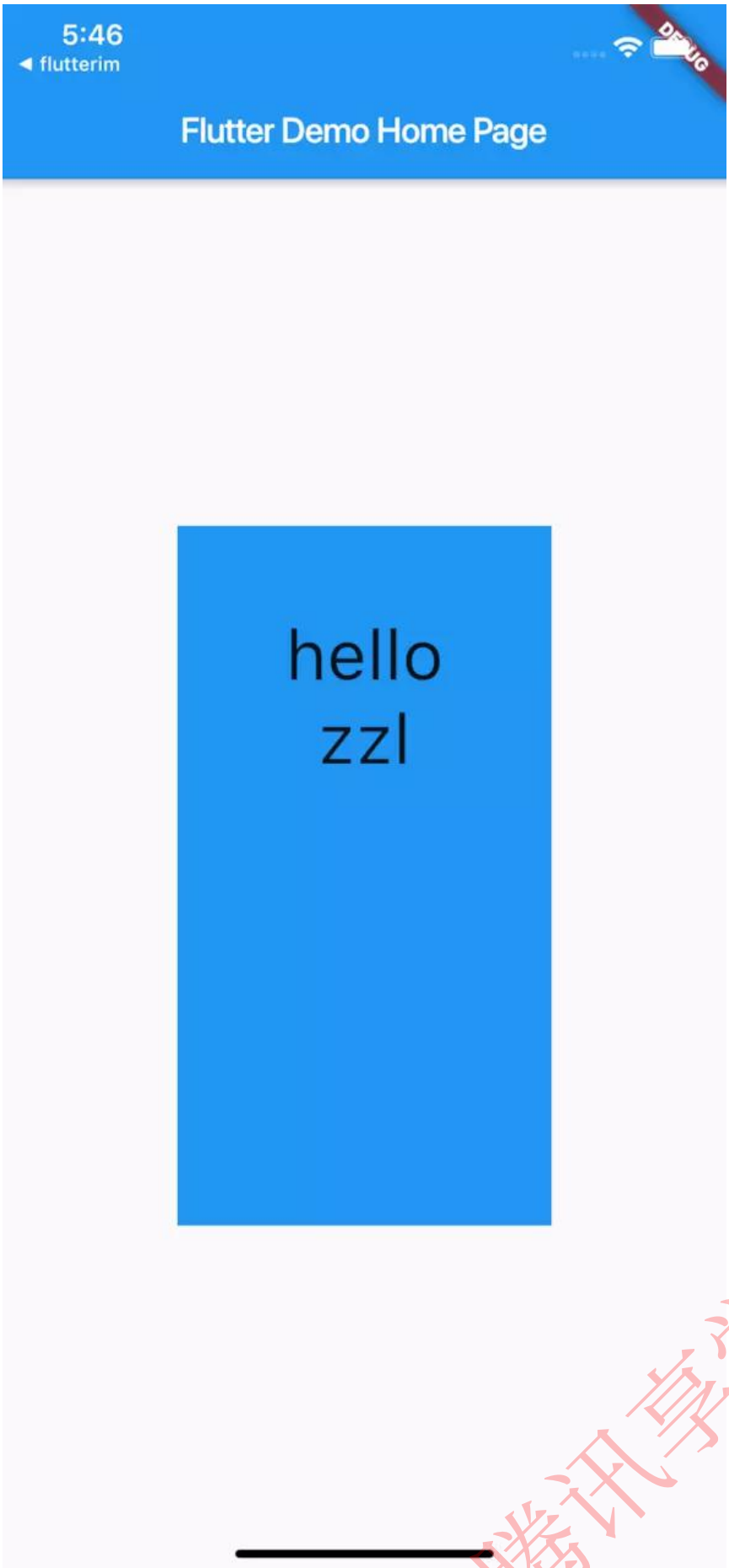
内边距 margin 和外边距边距 padding

```
body: Center(
  child: Container(
    child:  new Text("hello zzl ",
    style: TextStyle(
      fontSize: 40.0,
    ),
      textAlign: TextAlign.center,
```

```
  ),
  alignment: Alignment.topCenter,
  width: 500.0,
  height: 400.0,
  color: Colors.blue,
  padding: const EdgeInsets.all(50.0),//外边距
 // margin: const EdgeInsets.all(100.0),//内边距
),
),
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队



《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

填充控件 Padding

Padding 的布局分为两种情况：

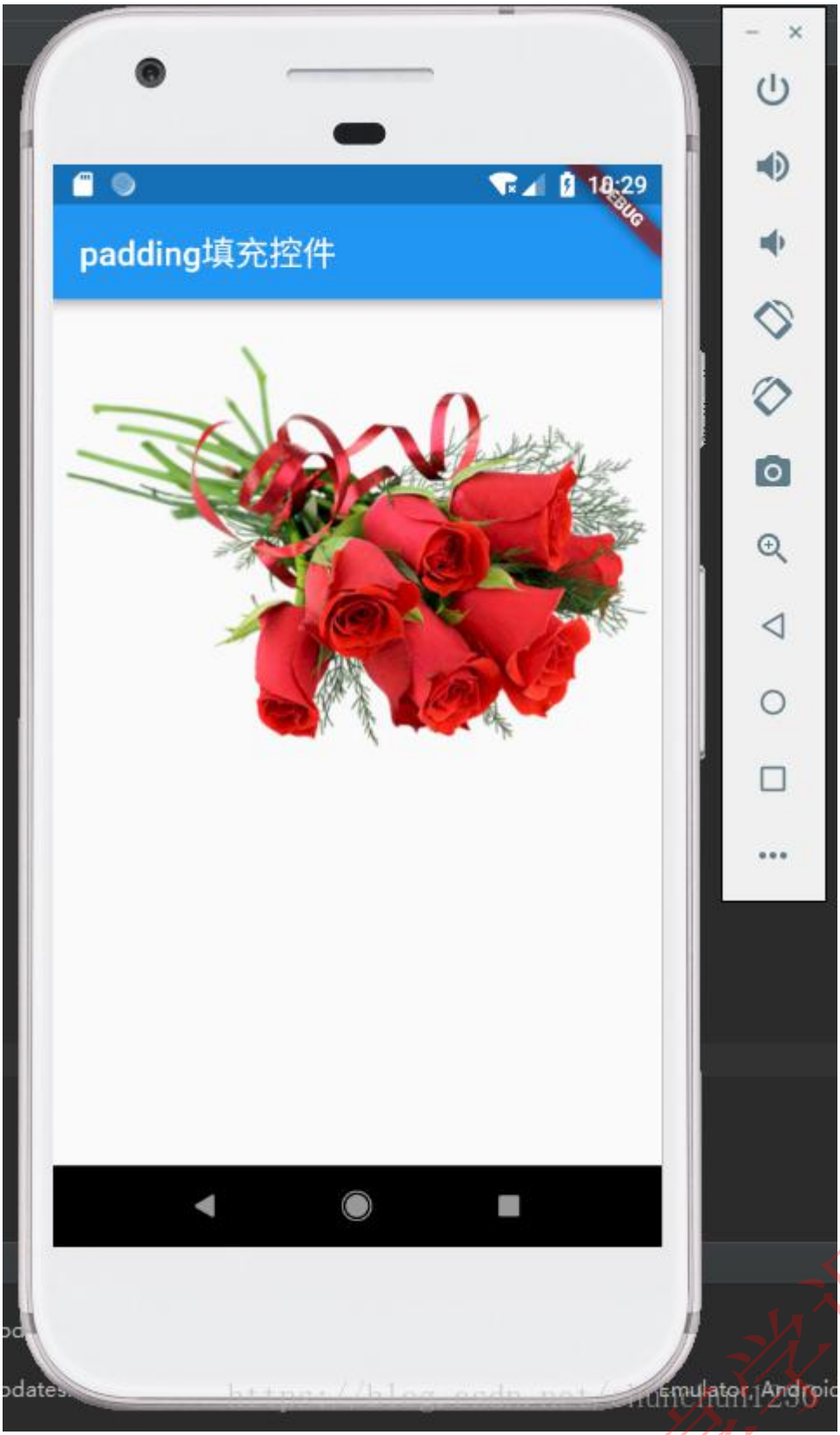
当 child 为空的时候，会产生一个宽为 left+right，高为 top+bottom 的区域；
当 child 不为空的时候，Padding 会将布局约束传递给 child，根据设置的 padding 属性，缩小 child 的布局尺寸。然后 Padding 将自己调整到 child 设置了 padding 属性的尺寸，在 child 周围创建空白区域。

```
import 'package:flutter/material.dart';

class PaddingDemo extends StatelessWidget{

  @override

  Widget build(BuildContext context) {

    return new Scaffold(

      appBar: new AppBar(

        title:new Text("padding 填充控件"),

      ),

      body: new Padding(

          padding: const EdgeInsets.all(8.0),

        child: new Image.asset("images/hua3.png"),

      ),

    );

  }

}

void main(){

  runApp(new MaterialApp(
```

```
  title: "padding 填充控件",

  home: new PaddingDemo(),

));
```

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队



《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

29.谈一下 flutter state 的生命周期

State 的生命周期

从上面的例子中可以看到， 会要求提供一个含有视图树的 。

既然 能够控制一个视图的状态，那它肯定会有一系列的生命周期。

上图就是 State 的生命周期图。

1. `StatefulWidget.createState()`

Framework 调用会通过调用 `StatefulWidget.createState()` 来创建一个 State。

1. `initState()`

新创建的 State 会和一个 产生关联，此时认为 State 已经被安装好了，`initState()` 函数将会被调用。

通常，我们可以重写这个函数，进行初始化操作。

1. `didChangeDependencies()`

在 `initState()` 调用结束后，这个函数会被调用。

事实上，当 State 对象的依赖关系发生变化时，这个函数总会被 Framework 调用。

1. `build()`

经过以上步骤，系统认为一个 State 已经准备好了，就会调用 `build()` 来构建视图。

我们需要在这个函数中，返回一个 Widget。

1. `deactivate()`

当 State 被暂时从视图树中移除时，会调用这个函数。

页面切换时，也会调用它，因为此时 State 在视图树中的位置发生了变化，需要先暂时移除后添加。

⚠️注意，重写的时候必须要调用 `super.deactivate()`。

1. `dispose()`

当 State 被永久的从视图树中移除，Framework 会调用该函数。

在销毁前触发，我们可以在这里进行最终的资源释放。

在调用这个函数之前，总会先调用 `deactivate()`。

⚠️注意，重写的时候必须要调用 `super.dispose()`。

1. `didUpdateWidget(covariant T oldWidget)`

当 widget 的配置发生变化时，会调用这个函数。

比如， 的时候就会调用这个函数。

这个函数调用后，会调用 `build()`。

1. `setState()`

当我需要更新 State 的视图时，需要手动调用这个函数，它会触发 `build()` 。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

31.Flutter 和 RN 的对比。

32.说一下 Hot Reload，Hot Restart，热更新三者的区别和原理。

33.Flutter 是如何做到一套 Dart 代码可以编译运行在 Android 和 iOS 平台的？所以说具体的原理。

34.Flutter 不具备反射，如果要使用反射，你应该如何使用？说一下大概的思路。

35.Flutter 在不使用 WebView 和 JS 方案的情况下。如何做到热更新？说一下大概思路。

36.如何让 Flutter 编译出来的 APP 的包大小尽可能的变小？

37.我们这个项目时一个综合系统的老项目，里面有 Android，iOS，还有 Web 代码，是一个混合开发的项目，现在需要迁移到 Flutter，加入你加入团队做这个项目的迁移工作，你觉得这个项目如何工程化、容器化以及架构演变应该从哪些维度思考？

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

38.APP 启动速度以及页面加载速度一直是我们比较关心的一个问题，特别是混合开发项目，谈谈你对 Flutter 渲染优化有哪些见解？

39.谈谈 Flutter 的内存回收管理机制，以及你平时是怎么处理内存的？内存泄漏和内存溢出你是怎么解决的？

40.再问一个简单一点的，你是如何把控混合项目开发时的生命周期（比如类似安卓的 onCreate、onResume 这种）和路由管理的？

41.Flutter for web 和 Flutter1.9 推出的 Flutter Web 有何本质上的区别？

42.谈谈你认为的 Flutter Web 应该如何改进？哪些内容可以改造之后可以用于平时的 Web 开发。谈谈你的改造方案。

43.谈谈如何打造低延迟的视频直播？为什么这样用？

44. StatefulWidget 的生命周期

- `initState()`：Widget 初始化当前 State，在当前方法中是不能获取到 Context 的，如想获取，可以试试 `Future.delayed()`
- `didChangeDependencies()`：在 `initState()` 后调用，State 对象依赖关系发生变化的时候也会调用。
- `deactivate()`：当 State 被暂时从视图树中移除时会调用这个方法，页面切换时也会调用该方法，和 Android 里的 `onPause` 差不多。

- `dispose()`：Widget 销毁时调用。
- `didUpdateWidget`：Widget 状态发生变化的时候调用。

借用 CoorChice 文章 里的一张图：

45. Flutter 如何与 Android iOS 通信？

Flutter 通过 `PlatformChannel` 与原生进行交互，其中 `PlatformChannel` 分为三种：

1. BasicMessageChannel：用于传递字符串和半结构化的信息。
2. MethodChannel：用于传递方法调用。Flutter 主动调用 Native 的方法，并获取相应的返回值。
3. EventChannel：用于数据流（event streams）的通信。

具体可以查看 闲鱼技术：深入理解 Flutter Platform Channel。

46. 什么是 Widgets、RenderObjects 和 Elements？

- `Widget` 仅用于存储渲染所需要的信息。
- `RenderObject` 负责管理布局、绘制等操作。
- `Element` 才是这颗巨大的控件树上的实体。

具体可以查看 [译] Flutter，什么是 Widgets、RenderObjects 和 Elements？

47. 说一下什么是状态管理，为什么需要它？

首先状态其实是一个概念上的东西，区分全局状态和局部状态。

局部状态比如说一个控件中输入的信息，全局状态比如是登陆后从后台请求回来的 userId。

当全局状态越来越多，多个页面共享一个状态时，我们就需要管理它。

常用的状态管理有：

- ScopedModel
- BLoC
- Redux / FishRedux
- Provider

48. 说一下 BLoC 模式？

具体可以查看： Vadaski - Flutter | 状态管理探索篇——BLoC(三)

这里引用一部分：

> BLoC 是一种利用 reactive programming 方式构建应用的方法，这是一个由流构成的完全异步的世界。

49. 如何统一管理错误页面？

我们都知道，如果在 Flutter 当中出错的话，那就是一片红。

可以使用 `ErrorWidget.builder` 来自定义一个 Widget 就 ok 了。

《Android 架构师面试题精编解析大全》

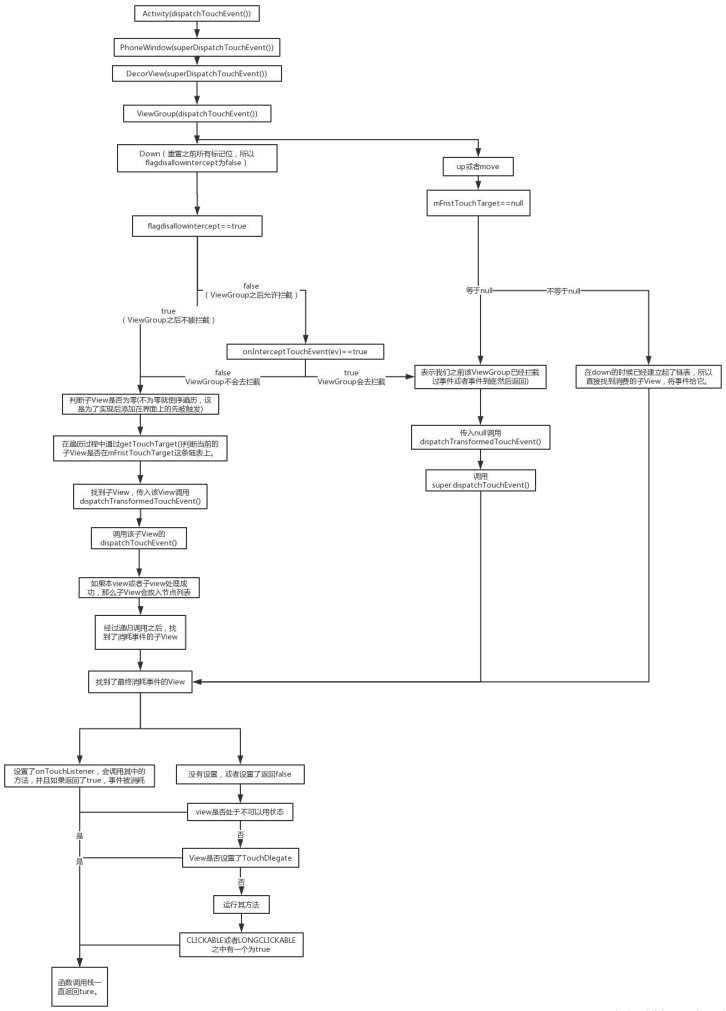
------腾讯享学课堂团队

具体可以看一下 小德 - 教你自定义 Flutter 错误页面

第六章：Android 高频面试题集锦

2019（腾讯、阿里、百度、美团、字节跳动、京东等）Android 高频面试题集锦（12 月更新完善）

1.android 事件分发机制，请详细说下整个流程



《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

事件分发（面试）.png

2.android view 绘制机制和加载过程，请详细说下整个流程

- 1.ViewRootImpl 会调用 performTraversals(),其内部会调用 performMeasure()、 performLayout、performDraw()。
- 2.performMeasure() 会调用最外层的 ViewGroup 的 measure()-->onMeasure(),ViewGroup 的 onMeasure()是抽象方法，但其提供了 measureChildren()，这之中会遍历子 View 然后循环调用 measureChild()这之中会用 getChildMeasureSpec()+父 View 的 MeasureSpec+子 View 的 LayoutParam 一起获取本 View 的 MeasureSpec ，然后调用子 View 的 measure() 到 View 的 onMeasure()-->setMeasureDimension(getDefaultSize(),getDefaultSize()),getDefaultSize() 默认返回 measureSpec 的测量数值，所以继承 View 进行自定义的 wrap_content 需要重写。
- 3.performLayout()会调用最外层的 ViewGroup 的 layout(l,t,r,b),本 View 在其中使用 setFrame()设置本 View 的四个顶点位置。在 onLayout(抽象方法)中确定子 View 的位置，如 LinearLayout 会遍历子 View，循环调用 setChildFrame()-->子 View.layout()。
- 4.performDraw()会调用最外层 ViewGroup 的 draw():其中会先后调用 background.draw()(绘制背景)、onDraw()(绘制自己)、dispatchDraw()(绘制子 View)、onDrawScrollBars()(绘制装饰)。
- 5.MeasureSpec 由 2 位 SpecMode(UNSPECIFIED、 EXACTLY(对应精确值和 match_parent)、AT_MOST(对应 warp_content))和 30 位 SpecSize 组成一个 int,DecorView 的 MeasureSpec 由窗口大小和其 LayoutParams 决定 ,其他 View 由父 View 的 MeasureSpec 和本 View 的 LayoutParams 决定。ViewGroup 中有 getChildMeasureSpec()来获取子 View 的 MeasureSpec。
- 6.三种方式获取 measure()后的宽高：
    - 1.Activity#onWindowFocusChange()中调用获取
    - 2.view.post(Runnable)将获取的代码投递到消息队列的尾部。
    - 3.ViewTreeObservable.

3.android 四大组件的加载过程，请详细介绍下

- 1.android 四大组件的加载过程:这是我总结的一篇博客

4.Activity 的启动模式

- 1.standard:默认标准模式，每启动一个都会创建一个实例，
- 2.singleTop：栈顶复用，如果在栈顶就调用 onNewIntent 复用，从 onResume()开始
- 3.singleTask：栈内复用，本栈内只要用该类型 Activity 就会将其顶部的 activity 出栈
- 4.singleInstance：单例模式，除了 3 中特性，系统会单独给该 Activity 创建一个栈，

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

5.A、B、C、D 分别是四种 Activity 的启动模式，那么 A->B->C->D->A->B->C->D 分别启动，最后的 activity 栈是怎么样的

- 1.这个题目需要深入了解 activity 的启动模式
- 2.最后的答案是：两个栈，前台栈是只有 D，后台栈从底至上是 A、B、C

6.Activity 缓存方法

- 1.配置改变导致 Activity 被杀死，横屏变竖屏：在 onStop 之前会调用 onSaveInstanceState()保存数据在重建 Activity 之后，会在 onStart()之后调用 onRestoreInstanceState(),并把保存下来的 Bundle 传给 onCreate()和它会默认重建 Activity 当前的视图，我们可以在 onCreate()中，回复自己的数据。
- 2.内存不足杀掉 Activity，优先级分别是：前台可见，可见非前台，后台。

7.Service 的生命周期，两种启动方法，有什么区别

- 1.context.startService() ->onCreate()- >onStart()->Service running-->(如 果 调 用 context.stopService())->onDestroy() ->Service shut down
    - 1.如果 Service 还没有运行，则调用 onCreate()然后调用 onStart()；
    - 2.如果 Service 已经运行，则只调用 onStart()，所以一个 Service 的 onStart 方法可能会重复调用多次。
    - 3.调用 stopService 的时候直接 onDestroy，
    - 4.如果是调用者自己直接退出而没有调用 stopService 的话，Service 会一直在后台运行。该 Service 的调用者再启动起来后可以通过 stopService 关闭 Service。
- 2.context.bindService()->onCreate()->onBind()->Service running-->onUnbind() -> onDestroy() ->Service stop
    - 1.onBind 将返回给客户端一个 IBind 接口实例，IBind 允许客户端回调服务的方法，比如得到 Service 运行的状态或其他操作。
    - 2.这个时候会把调用者和 Service 绑定在一起，Context 退出了,Service 就会调用 onUnbind->onDestroy 相应退出。
    - 3.所以调用 bindService 的生命周期为：onCreate --> onBind(只一次，不可多次绑定) --> onUnbind --> onDestory。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

8.怎么保证 service 不被杀死

- 1.提升 service 优先级
- 2.提升 service 进程优先级
- 3.onDestroy 方法里重启 service

9.静态的 Broadcast 和动态的有什么区别

- 1.动态的比静态的安全
- 2.静态在 app 启动的时候就初始化了 动态使用代码初始化
- 3.静态需要配置 动态不需要
- 4.生存期，静态广播的生存期可以比动态广播的长很多
- 5.优先级动态广播的优先级比静态广播高

10.Intent 可以传递哪些数据类型

- 1.Serializable
- 2.charsequence: 主要用来传递 String，char 等
- 3.parcelable
- 4.Bundle

11.Json 有什么优劣势、解析的原理

- 1.JSON 的速度要远远快于 XML
- 2.JSON 相对于 XML 来讲，数据的体积小
- 3.JSON 对数据的描述性比 XML 较差
- 4.解析的基本原理是：词法分析

12.一个语言的编译过程

- 1.词法分析：将一串文本按规则分割成最小的结构，关键字、标识符、运算符、界符和常量等。一般实现方法是自动机和正则表达式
- 2.语法分析：将一系列单词组合成语法树。一般实现方法有自顶向下和自底向上
- 3.语义分析：对结构上正确的源程序进行上下文有关性质的审查
- 4.目标代码生成
- 5.代码优化：优化生成的目标代码，

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

13.动画有哪几类，各有什么特点

- 1.动画的基本原理：其实就是利用插值器和估值器，来计算出各个时刻 View 的属性，然后通过改变 View 的属性来，实现 View 的动画效果。
- 2.View 动画:只是影像变化，view 的实际位置还在原来的地方。
- 3.帧动画是在 xml 中定义好一系列图片之后，使用 AnimationDrawable 来播放的动画。
- 4.View 的属性动画：
    - 1.插值器：作用是根据时间的流逝的百分比来计算属性改变的百分比
    - 2.估值器：在 1 的基础上由这个东西来计算出属性到底变化了多少数值的类

14.Handler、Looper 消息队列模型，各部分的作用

- 1.MessageQueue：读取会自动删除消息，单链表维护，在插入和删除上有优势。在其 next()中会无限循环，不断判断是否有消息，有就返回这条消息并移除。
- 2.Looper :Looper 创建的时候会创建一个 MessageQueue ,调用 loop()方法的时候消息循环开始 ,loop()也是一个死循环 ,会不断调用 messageQueue 的 next() ,当有消息就处理 ,否则阻塞在 messageQueue 的 next()中。当 Looper 的 quit()被调用的时候会调用 messageQueue 的 quit(),此时 next()会返回 null，然后 loop()方法也跟着退出。
- 3.Handler ：在主线程构造一个 Handler ，然后在其他线程调用 sendMessage(),此时主线程的 MessageQueue 中会插入一条 message，然后被 Looper 使用。
- 4.系统的主线程在 ActivityThread 的 main()为入口开启主线程，其中定义了内部类 Activity.H 定义了一系列消息类型，包含四大组件的启动停止。
- 5.MessageQueue 和 Looper 是一对一关系，Handler 和 Looper 是多对一

15.怎样退出终止 App

- 1.自己设置一个 Activity 的栈，然后一个个 finish()

16.Android IPC:Binder 原理

- 1.在 Activity 和 Service 进行通讯的时候，用到了 Binder。
    - 1.当属于同个进程我们可以继承 Binder 然后在 Activity 中对 Service 进行操作
    - 2.当不属于同个进程，那么要用到 AIDL 让系统给我们创建一个 Binder，然后在 Activity 中对远端的 Service 进行操作。

- 2.系统给我们生成的 Binder：
    - 1.Stub 类中有:接口方法的 id ,有该 Binder 的标识 ,有 asInterface(IBinder)(让我们在 Activity 中获取实现了 Binder 的接口，接口的实现在 Service 里，同进程时候返回 Stub 否则返回 Proxy)，有 onTransact()这个方法是在不同进程的时候让 Proxy 在 Activity 进行远端调用实现 Activity 操作 Service
    - 2.Proxy 类是代理，在 Activity 端，其中有:IBinder mRemote(这就是远端的 Binder)，两个接口的实现方法不过是代理最终还是要在远端的 onTransact()中进行实际操作。

- 3.哪一端的 Binder 是副本，该端就可以被另一端进行操作，因为 Binder 本体在定义的时候可以操作本端的东西。所以可以在 Activity 端传入本端的 Binder ,让 Service 端对其进行操作称为 Listener，可以用 RemoteCallbackList 这个容器来装 Listener ,防止 Listener 因为经历过序列化而产生的问题。
- 4.当 Activity 端向远端进行调用的时候，当前线程会挂起，当方法处理完毕才会唤醒。
- 5.如果一个 AIDL 就用一个 Service 太奢侈，所以可以使用 Binder 池的方式，建立一个 AIDL 其中的方法是返回 IBinder，然后根据方法中传入的参数返回具体的 AIDL。
- 6.IPC 的方式有：Bundle（在 Intent 启动的时候传入，不过是一次性的），文件共享(对于 SharedPreference 是特例，因为其在内存中会有缓存)，使用 Messenger(其底层用的也是 AIDL，同理要操作哪端，就在哪端定义 Messenger)，AIDL，ContentProvider(在本进程中继承实现一个 ContentProvider，在增删改查方法中调用本进程的 SQLite，在其他进程中查询)，Socket

17.描述一次跨进程通讯

- 1.client、proxy、serviceManager、BinderDriver、impl、service
- 2.client 发起一个请求 service 信息的 Binder 请求到 BinderDriver 中，serviceManager 发现 BinderDiriver 中有自己的请求 然后将 clinet 请求的 service 的数据返回给 client 这样完成了一次 Binder 通讯
- 3.clinet 获取的 service 信息就是该 service 的 proxy，此时调用 proxy 的方法，proxy 将请求发送到 BinderDriver 中，此时 service 的 Binder 线程池循环发现有自己的请求，然后用 impl 就处理这个请求最后返回，这样完成了第二次 Binder 通讯
  4.中间 client 可挂起，也可以不挂起，有一个关键字 oneway 可以解决这个

18.android 重要术语解释

- 1.ActivityManagerServices，简称 AMS，服务端对象，负责系统中所有 Activity 的生命周期
- 2.ActivityThread，App 的真正入口。当开启 App 之后，会调用 main()开始运行，开启消息循环队列，这就是传说中的 UI 线程或者叫主线程。与 ActivityManagerServices 配合，一起完成 Activity 的管理工作

- 3.ApplicationThread，用来实现 ActivityManagerService 与 ActivityThread 之间的交互。在 ActivityManagerService 需要管理相关 Application 中的 Activity 的生命周期时，通过 ApplicationThread 的代理对象与 ActivityThread 通讯。
- 4.ApplicationThreadProxy，是 ApplicationThread 在服务器端的代理，负责和客户端的 ApplicationThread 通讯。AMS 就是通过该代理与 ActivityThread 进行通信的。
- 5.Instrumentation，每一个应用程序只有一个 Instrumentation 对象，每个 Activity 内都有一个对该对象的引用。Instrumentation 可以理解为应用进程的管家，ActivityThread 要创建或暂停某个 Activity 时，都需要通过 Instrumentation 来进行具体的操作。
- 6.ActivityStack，Activity 在 AMS 的栈管理，用来记录已经启动的 Activity 的先后关系，状态信息等。通过 ActivityStack 决定是否需要启动新的进程。
- 7.ActivityRecord，ActivityStack 的管理对象，每个 Activity 在 AMS 对应一个 ActivityRecord，来记录 Activity 的状态以及其他的管理信息。其实就是服务器端的 Activity 对象的映像。
- 8.TaskRecord，AMS 抽象出来的一个“任务”的概念，是记录 ActivityRecord 的栈，一个“Task”包含若干个 ActivityRecord。AMS 用 TaskRecord 确保 Activity 启动和退出的顺序。如果你清楚 Activity 的 4 种 launchMode，那么对这个概念应该不陌生。

19.理解 Window 和 WindowManager

- 1.Window 用于显示 View 和接收各种事件，Window 有三种类型：应用 Window(每个 Activity 对应一个 Window)、子 Window(不能单独存在，附属于特定 Window)、系统 window(Toast 和状态栏)
- 2.Window 分层级，应用 Window 在 1-99、子 Window 在 1000-1999、系统 Window 在 2000-2999.WindowManager 提供了增删改 View 三个功能。
- 3.Window 是个抽象概念：每一个 Window 对应着一个 View 和 ViewRootImpl，Window 通过 ViewRootImpl 来和 View 建立联系，View 是 Window 存在的实体，只能通过 WindowManager 来访问 Window。
- 4.WindowManager 的实现是 WindowManagerImpl 其再委托给 WindowManagerGlobal 来对 Window 进行操作，其中有四个 List 分别储存对应的 View、ViewRootImpl、WindowManger.LayoutParams 和正在被删除的 View
- 5.Window 的实体是存在于远端的 WindowMangerService 中，所以增删改 Window 在本端是修改上面的几个 List 然后通过 ViewRootImpl 重绘 View，通过 WindowSession(每个应用一个)在远端修改 Window。
- 6.Activity 创建 Window：Activity 会在 attach()中创建 Window 并设置其回调(onAttachedToWindow()、dispatchTouchEvent()),Activity 的 Window 是由 Policy 类创建 PhoneWindow 实现的。然后通过 Activity#setContentView()调用 PhoneWindow 的 setContentView。

20.Bitmap 的处理

- 1.当使用 ImageView 的时候，可能图片的像素大于 ImageView，此时就可以通过 BitmapFactory.Option 来对图片进行压缩，inSampleSize 表示缩小 2^(inSampleSize-1)倍。
- 2.BitMap 的缓存：
    - 1.使用 LruCache 进行内存缓存。
    - 2.使用 DiskLruCache 进行硬盘缓存。
    - 3.实现一个 ImageLoader 的流程：同步异步加载、图片压缩、内存硬盘缓存、网络拉取
        - 1.同步加载只创建一个线程然后按照顺序进行图片加载
        - 2.异步加载使用线程池，让存在的加载任务都处于不同线程
        - 3.为了不开启过多的异步任务，只在列表静止的时候开启图片加载

21.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。

22.ClassLoader 的基础知识

- 1.双亲委托：一个 ClassLoader 类负责加载这个类所涉及的所有类，在加载的时候会判断该类是否已经被加载过，然后会递归去他父 ClassLoader 中找。
- 2.可以动态加载 Jar 通过 URLClassLoader
- 3.ClassLoader 隔离问题 JVM 识别一个类是由：ClassLoader id+PackageName+ClassName。
- 4.加载不同 Jar 包中的公共类：
    - 1.让父 ClassLoader 加载公共的 Jar，子 ClassLoader 加载包含公共 Jar 的 Jar，此时子 ClassLoader 在加载公共 Jar 的时候会先去父 ClassLoader 中找。(只适用 Java)
    - 2.重写加载包含公共 Jar 的 Jar 的 ClassLoader，在 loadClass 中找到已经加载过公共 Jar 的 ClassLoader，也就是把父 ClassLoader 替换掉。(只适用 Java)
    - 3.在生成包含公共 Jar 的 Jar 时候把公共 Jar 去掉。

23.插件化框架描述：dynamicLoadApk 为例子

- 1.可以通过 DexClassLoader 来对 apk 中的 dex 包进行加载访问
- 2.如何加载资源是个很大的问题，因为宿主程序中并没有 apk 中的资源，所以调用 R 资源会报错，所以这里使用了 Activity 中的实现 ContextImpl 的 getAssets()和 getResources()再加上反射来实现。

- 3.由于系统启动 Activity 有很多初始化动作要做，而我们手动反射很难完成，所以可以采用接口机制，将 Activity 的大部分生命周期提取成接口，然后通过代理 Activity 去调用插件 Activity 的生命周期。同时如果像增加一个新生命周期方法的时候，只需要在接口中和代理中声明一下就行。
- 4.缺点：
    - 1.慎用 this，因为在 apk 中使用 this 并不代表宿主中的 activity，当然如果 this 只是表示自己的接口还是可以的。除此之外可以使用 that 代替 this。
    - 2.不支持 Service 和静态注册的 Broadcast
    - 3.不支持 LaunchMode 和 Apk 中 Activity 的隐式调用。

24.热修复：Andfix 为例子

- 1.大致原理：apkpatch 将两个 apk 做一次对比，然后找出不同的部分。可以看到生成的 apatch 了文件，后缀改成 zip 再解压开，里面有一个 dex 文件。通过 jadx 查看一下源码，里面就是被修复的代码所在的类文件,这些更改过的类都加上了一个_CF 的后缀，并且变动的方法都被加上了一个叫@MethodReplace 的 annotation，通过 clazz 和 method 指定了需要替换的方法。然后客户端 sdk 得到补丁文件后就会根据 annotation 来寻找需要替换的方法。最后由 JNI 层完成方法的替换。
- 2.无法添加新类和新的字段、补丁文件很容易被反编译、加固平台可能会使热补丁功能失效

25.线程同步的问题，常用的线程同步

- 1.sycn：保证了原子性、可见性、有序性
- 2.锁：保证了原子性、可见性、有序性
    - 1.自旋锁:可以使线程在没有取得锁的时候，不被挂起，而转去执行一个空循环。
        - 1.优点:线程被挂起的几率减少，线程执行的连贯性加强。用于对于锁竞争不是很激烈，锁占用时间很短的并发线程。
        - 2.缺点:过多浪费 CPU 时间，有一个线程连续两次试图获得自旋锁引起死锁
    - 2.阻塞锁:没得到锁的线程等待或者挂起，Sycn、Lock
    - 3.可重入锁:一个线程可多次获取该锁，Sycn、Lock
    - 4.悲观锁:每次去拿数据的时候都认为别人会修改，所以会阻塞全部其他线程 Sycn、Lock
    - 5.乐观锁:每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。cas
    - 6.显示锁和内置锁:显示锁用 Lock 来定义、内置锁用 synchronized。
    - 7.读-写锁:为了提高性能，Java 提供了读
- 3.volatile

- 1.只能保证可见性，不能保证原子性
- 2.自增操作有三步，此时多线程写会出现问题

- 4.cas
  - 1.操作:内存值 V、旧的预期值 A、要修改的值 B，当且仅当预期值 A 和内存值 V 相同时，将内存值修改为 B 并返回 true，否则什么都不做并返回 false。
  - 2.解释:本地副本为 A，共享内存为 V，线程 A 要把 V 修改成 B。某个时刻线程 A 要把 V 修改成 B，如果 A 和 V 不同那么就表示有其他线程在修改 V，此时就表示修改失败，否则表示没有其他线程修改，那么把 V 改成 B。
  - 3.局限:如果 V 被修改成 V1 然后又被改成 V，此时 cas 识别不出变化，还是认为没有其他线程在修改 V，此时就会有问题
  - 4.局限解决:将 V 带上版本。

- 5.线程不安全到底是怎么回事：
  - 1.一个线程写，多个线程读的时候，会造成写了一半就去读
  - 2.多线程写，会造成脏数据

26.Asynctask 和线程池，GC 相关（怎么判断哪些内存该 GC，GC 算法）

- 1.Asynctask：异步任务类，单线程线程池+Handler
- 2.线程池：
  - 1.ThreadPoolExecutor：通过 Executors 可以构造单线程池、固定数目线程池、不固定数目线程池。
  - 2.ScheduledThreadPoolExecutor：可以延时调用线程或者延时重复调度线程。

- 3.GC 相关：重要
  - 1.搜索算法：
    - 1.引用计数
    - 2.图搜索，可达性分析
  - 2.回收算法：
    - 1.标记清除复制：用于青年代

        - 2.标记整理：用于老年代
    - 3.堆分区：
        - 1.青年区 eden 80%、survivor1 10%、survivor2 10%
        - 2.老年区
    - 4.虚拟机栈分区：
        - 1.局部变量表
        - 2.操作数栈
        - 3.动态链接
        - 4.方法返回地址
    - 5.GC Roots:
        - 1.虚拟机栈(栈桢中的本地变量表)中的引用的对象
        - 2.方法区中的类静态属性引用的对象
        - 3.方法区中的常量引用的对象
        - 4.本地方法栈中 JNI 的引用的对象

27.网络

- 1.ARP 协议:在 IP 以太网中，当一个上层协议要发包时，有了该节点的 IP 地址，ARP 就能提供该节点的 MAC 地址。
- 2.HTTP HTTPS 的区别:
    - 1.HTTPS 使用 TLS(SSL)进行加密
    - 2.HTTPS 缺省工作在 TCP 协议 443 端口
    - 3.它的工作流程一般如以下方式:
        - 1.完成 TCP 三次同步握手
        - 2.客户端验证服务器数字证书，通过，进入步骤 3
        - 3.DH 算法协商对称加密算法的密钥、hash 算法的密钥
        - 4.SSL 安全加密隧道协商完成
        - 5.网页以加密的方式传输，用协商的对称加密算法和密钥加密，保证数据机密性；用协商的 hash 算法进行数据完整性保护，保证数据不被篡改
    - 3.http 请求包结构，http 返回码的分类，400 和 500 的区别

- 1.包结构：
  - 1.请求：请求行、头部、数据
  - 2.返回：状态行、头部、数据
- 2.http 返回码分类：1 到 5 分别是，消息、成功、重定向、客户端错误、服务端错误

○ 4.Tcp

- 1.可靠连接，三次握手，四次挥手
  - 1.三次握手：防止了服务器端的一直等待而浪费资源，例如只是两次握手，如果 s 确认之后 c 就掉线了，那么 s 就会浪费资源
    - 1.syn-c = x，表示这消息是 x 序号
    - 2.ack-s = x + 1，表示 syn-c 这个消息接收成功。syn-s = y，表示这消息是 y 序号。
    - 3.ack-c = y + 1，表示 syn-s 这条消息接收成功
- 2.四次挥手：TCP 是全双工模式
  - 1.fin-c = x，表示现在需要关闭 c 到 s 了。ack-c = y,表示上一条 s 的消息已经接收完毕
  - 2.ack-s = x + 1，表示需要关闭的 fin-c 消息已经接收到了，同意关闭
  - 3.fin-s = y + 1，表示 s 已经准备好关闭了，就等 c 的最后一条命令
  - 4.ack-c = y + 1，表示 c 已经关闭，让 s 也关闭
- 3.滑动窗口，停止等待、后退 N、选择重传
- 4.拥塞控制，慢启动、拥塞避免、加速递减、快重传快恢复

TCP 协议与 UDP 协议的区别

TCP（Transmission Control Protocol，传输控制协议）是面向连接的协议，也就是说，在收发数据前，必须和对方建立可靠的连接。一个 TCP 连接必须要经过三次“对话”才能建立起来，其中的过程非常复杂，

只简单的描述下这三次对话的简单过程：主机 A 向主机 B 发出连接请求数据包：“我想给你发数据，可以吗？”，这是第一次对话；主机 B 向主机 A 发送同意连接和要求同步（同步就是两台主机一个在发送，一个在接收，协调工作）的数据包：“可以，你什么时候发？”，这是第二次对话；主机 A 再发出一个数据包确认主机 B 的要求同步：“我现在就发，你接着吧！”，这是第三次对话。三次“对话”的目的是使数据包的发送和接收同步，经过三次“对话”之后，主机 A 才向主机 B 正式发送数据。详细点说就是：

TCP 三次握手过程

1 主机 A 通过向主机 B 发送一个含有同步序列号的标志位的数据段给主机 B ,向主机 B 请求建立连接,通过这个数据段,

主机 A 告诉主机 B 两件事:我想要和你通信;你可以用哪个序列号作为起始数据段来回应我.

2 主机 B 收到主机 A 的请求后,用一个带有确认应答(ACK)和同步序列号(SYN)标志位的数据段响应主机 A,也告诉主机 A 两件事:

我已经收到你的请求了,你可以传输数据了;你要用哪仵序列号作为起始数据段来回应我

3 主机 A 收到这个数据段后,再发送一个确认应答,确认已收到主机 B 的数据段:"我已收到回复,我现在要开始传输实际数据了

这样 3 次握手就完成了,主机 A 和主机 B 就可以传输数据了.

3 次握手的特点

没有应用层的数据

SYN 这个标志位只有在 TCP 建产连接时才会被置 1

握手完成后 SYN 标志位被置 0

TCP 建立连接要进行 3 次握手,而断开连接要进行 4 次

1 当主机 A 完成数据传输后,将控制位 FIN 置 1,提出停止 TCP 连接的请求

2 主机 B 收到 FIN 后对其作出响应,确认这一方向上的 TCP 连接将关闭,将 ACK 置 1

3 由 B 端再提出反方向的关闭请求,将 FIN 置 1

4 主机 A 对主机 B 的请求进行确认,将 ACK 置 1,双方向的关闭结束.

由 TCP 的三次握手和四次断开可以看出,TCP 使用面向连接的通信方式,大大提高了数据通信的可靠性,使发送数据端

和接收端在数据正式传输前就有了交互,为数据正式传输打下了可靠的基础

名词解释

ACK　TCP 报头的控制位之一,对数据进行确认.确认由目的端发出,用它来告诉发送端这个序列号之前的数据段

都收到了.比如,确认号为 X,则表示前 X-1 个数据段都收到了,只有当 ACK=1 时,确认号才有效,当 ACK=0 时,确认号无效,这时会要求重传数据,保证数据的完整性.

SYN　同步序列号,TCP 建立连接时将这个位置 1

FIN　发送端完成发送任务位,当 TCP 完成数据传输需要断开时,提出断开连接的一方将这位置 1

UDP（User Data Protocol，用户数据报协议）

（1） UDP 是一个非连接的协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP 传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP 把每个消息段放在队列中，应用程序每次从队列中读一个消息段。

（2） 由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。

（3） UDP 信息包的标题很短，只有 8 个字节，相对于 TCP 的 20 个字节信息包的额外开销很小。

（4）吞吐量不受拥挤控制算法的调节 ,只受应用软件生成数据的速率、传输带宽、源端和终端主机性能的限制。

（5）UDP 使用**尽最大努力交付**，即不保证可靠交付，因此主机不需要维持复杂的链接状态表（这里面有许多参数）。

（6）UDP 是**面向报文**的。发送方的 UDP 对应用程序交下来的报文，在添加首部后就向下交付给 IP 层。既不拆分，也不合并，而是保留这些报文的边界，因此，应用程序需要选择合适的报文大小。

小结 TCP 与 UDP 的区别：

1.基于连接与无连接；

2.对系统资源的要求（TCP 较多，UDP 少）；

3.UDP 程序结构较简单；

4.流模式与数据报模式 ；

5.TCP 保证数据正确性，UDP 可能丢包，TCP 保证数据顺序，UDP 不保证。

以下内容来自百度百科：

第一次握手：建立连接时，客户端发送 syn 包（syn=j）到服务器，并进入 SYN_SENT 状态，等待服务器确认；SYN：同步序列编号（***Synchronize Sequence Numbers***）。

第二次握手：服务器收到 syn 包，必须确认客户的 SYN（ack=j+1），同时自己也发送一个 SYN 包（syn=k），即 SYN+ACK 包，此时服务器进入 SYN_RECV 状态；

第三次握手：客户端收到服务器的 SYN+ACK 包，向服务器发送确认包 ACK(ack=k+1），此包发送完毕，客户端和服务器进入 ESTABLISHED（TCP 连接成功）状态，完成三次握手。

完成三次握手，客户端与服务器开始传送数据，在上述过程中，还有一些重要的概念：

未连接队列

在三次握手协议中，服务器维护一个未连接队列，该队列为每个客户端的 SYN 包（ syn=j ）开设一个条目，该条目表明服务器已收到 SYN 包，并向客户发出确认，正在等待客户的确认包。这些条目所标识的连接在服务器处于 SYN_RECV 状态，当服务器收到客户的确认包时，删除该条目，服务器进入 ESTABLISHED 状态。

关闭 TCP 连接：改进的三次握手

对于一个已经建立的连接，TCP 使用改进的三次握手来释放连接（使用一个带有 FIN 附加标记的报文段）。TCP 关闭连接的步骤如下：

第一步，当主机 A 的应用程序通知 TCP 数据已经发送完毕时，TCP 向主机 B 发送一个带有 FIN 附加标记的报文段（ FIN 表示英文 finish ）。

第二步，主机 B 收到这个 FIN 报文段之后，并不立即用 FIN 报文段回复主机 A，而是先向主机 A 发送一个确认序号 ACK，同

时通知自己相应的应用程序：对方要求关闭连接（先发送 ACK 的目的是为了防止在这段时间内，对方重传 FIN 报文段）。

第三步，主机 B 的应用程序告诉 TCP：我要彻底的关闭连接，TCP 向主机 A 送一个 FIN 报文段。

第四步 ,主机 A 收到这个 FIN 报文段后 ,向主机 B 发送一个 ACK 表示连接彻底释放。

28.数据库性能优化：索引和事务，需要找本专门的书大概了解一下

29.13.APK 打包流程和其内容

- 1.流程
    - 1.aapt 生成 R 文件
        - 2.aidl 生成 java 文件
        - 3.将全部 java 文件编译成 class 文件
        - 4.将全部 class 文件和第三方包合并成 dex 文件
        - 5.将资源、so 文件、dex 文件整合成 apk
        - 6.apk 签名
        - 7.apk 字节对齐
- 2.内容：so、dex、asset、资源文件

30.网络劫持的类型原理：可以百度一下了解一下具体概念

- 1.DNS 劫持、欺骗、污染
- 2.http 劫持：重定向、注入 js，http 注入、报文扩展

31.java 类加载过程：

- 1.加载时机：创建实例、访问静态变量或方法、反射、加载子类之前
- 2.验证：验证文件格式、元数据、字节码、符号引用的正确性
- 3.加载：根据全类名获取文件字节流、将字节流转化为静态储存结构放入方法区、生成 class 对象
- 4.准备：在堆上为静态变量划分内存
- 5.解析：将常量池中的符号引用转换为直接引用
- 6.初始化：初始化静态变量
- 7.书籍推荐：**深入理解 java 虚拟机**，博客推荐：Java/Android 阿里面试 JVM 部分理解

32.retrofit 的了解

- 1.动态代理创建一个接口的代理类
- 2.通过反射解析每个接口的注解、入参构造 http 请求
- 3.获取到返回的 http 请求，使用 Adapter 解析成需要的返回值。

33.bundle 的数据结构，如何存储

- 1.键值对储存
- 2.传递的数据可以是 boolean、byte、int、long、float、double、string 等基本类型或它们对应的数组，也可以是对象或对象数组。
- 3.当 Bundle 传递的是对象或对象数组时，必须实现 Serializable 或 Parcelable 接口

34.listview 内点击 buttom 并移动的事件流完整拦截过程：

- 1.点下按钮的时候：
    - 1.产生了一个 down 事件，activity-->phoneWindow-->ViewGroup-->ListView-->botton,中间如果有重写了拦截方法，则事件被该 view 拦截可能消耗。
    - 2.没拦截，事件到达了 button，这个过程中建立了一条事件传递的 view 链表
    - 3.到 button 的 dispatch 方法-->onTouch-->view 是否可用-->Touch 代理
- 2.移动点击按钮的时候:

    - 1.产生 move 事件，listView 中会对 move 事件做拦截
    - 2.此时 listView 会将该滑动事件消费掉
    - 3.后续的滑动事件都会被 listView 消费掉

- 3.手指抬起来时候：前面建立了一个 view 链表，listView 的父 view 在获取事件的时候，会直接取链表中的 listView 让其进行事件消耗。

35.service 的意义：不需要界面，在后台执行的程序

36.android 的 IPC 通信方式，线程（进程间）通信机制有哪些

- 1.ipc 通信方式：binder、contentprovider、socket
- 2.操作系统进程通讯方式：共享内存、socket、管道

37.操作系统进程和线程的区别

- 1.简而言之,一个程序至少有一个进程,一个进程至少有一个线程.
- 2.线程的划分尺度小于进程，使得多线程程序的并发性高。
- 3.另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。
- 4.多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配

38.HashMap 的实现过程：Capacity 就是 buckets 的数目，Load factor 就是 buckets 填满程度的最大比例。如果对迭代性能要求很高的话不要把 capacity 设置过大，也不要把 load factor 设置过小。

- 1.简单来说 HashMap 就是一个会自动扩容的**数组链表**
- 2.put 过程

    - 1.对 key 的 hashCode()做 hash，然后再计算 index;

- 2.如果没碰撞直接放到 bucket 里；
- 3.如果碰撞了，以链表的形式存在 buckets 后；
- 4.如果碰撞导致链表过长(大于等于 TREEIFY_THRESHOLD)，就把链表转换成红黑树；
- 5.如果节点已经存在就替换 old value(保证 key 的唯一性)
- 6.如果 bucket 满了(超过 load factor*current capacity)，就要 resize。

- 3.resize：当 put 时，如果发现目前的 bucket 占用程度已经超过了 Load Factor 所希望的比例，那么就会发生 resize。在 resize 的过程，简单的说就是把 bucket 扩充为 2 倍，之后重新计算 index，把节点再放到新的 bucket 中
- 4.get 过程

  - 1.根据 key 的 hash 算出数组下表
  - 2.使用 equals 遍历链表进行比较

39.mvc、mvp、mvvm：

- 1.mvc:数据、View、Activity，View 将操作反馈给 Activity，Activitiy 去获取数据，数据通过观察者模式刷新给 View。循环依赖

  - 1.Activity 重，很难单元测试
  - 2.View 和 Model 耦合严重

- 2.mvp:数据、View、Presenter，View 将操作给 Presenter，Presenter 去获取数据，数据获取好了返回给 Presenter，Presenter 去刷新 View。PV，PM 双向依赖

  - 1.接口爆炸
  - 2.Presenter 很重

- 3.mvvm:数据、View、ViewModel，View 将操作给 ViewModel，ViewModel 去获取数据，数据和界面绑定了，数据更新界面更新。

  - 1.viewModel 的业务逻辑可以单独拿来测试
  - 2.一个 view 对应一个 viewModel 业务逻辑可以分离，不会出现全能类
  - 3.数据和界面绑定了，不用写垃圾代码，但是复用起来不舒服

40.java 的线程如何实现

- 1.Thread 继承
- 2.Runnale

- 3.Future
- 4.线程池

41.ArrayList 如何删除重复的元素或者指定的元素；

- 1.删除重复：Set
- 2.删除指定：迭代器

42.如何设计在 UDP 上层保证 UDP 的可靠性传输；

- 1.简单来讲，要使用 UDP 来构建可靠的面向连接的数据传输，就要实现类似于 TCP 协议的超时重传，有序接受，应答确认，滑动窗口流量控制等机制,等于说要在传输层的上一层（或者直接在应用层）实现 TCP 协议的可靠数据传输机制。
- 2.比如使用 UDP 数据包+序列号，UDP 数据包+时间戳等方法，在服务器端进行应答确认机制，这样就会保证不可靠的 UDP 协议进行可靠的数据传输。
- 3.基于 udp 的可靠传输协议有：RUDP、RTP、UDT

43.Java 中内部类为什么可以访问外部类

- 1.因为内部类创建的时候，需要外部类的对象，在内部类对象创建的时候会把外部类的引用传递进去

44.设计移动端的联系人存储与查询的功能，要求快速搜索联系人，可以用到哪些数据结构？数据库索引，平衡二叉树(B 树、红黑树)

45.红黑树特点

- 1.root 节点和叶子节点是黑色
- 2.红色节点后必须为黑色节点
- 3.从 root 到叶子每条路径的黑节点数量相同

46.linux 异步和同步 i/o:

- 1.同步：对于 client，client 一直等待，但是 client 不挂起：主线程调用
- 2.异步：对于 client，client 发起请求，service 好了再回调 client：其他线程调用，调用完成之后进行回调
- 3.阻塞：对于 service，在准备 io 的时候会将 service 端挂起，直至准备完成然后唤醒 service：bio
- 3.非阻塞：对于 service，在准备 io 的时候不会将 service 端挂起，而是 service 一直去轮询判断 io 是否准备完成，准备完成了就进行操作：nio、linux 的 select、poll、epoll
- 4.多路复用 io：非阻塞 io 的一种优化，java nio，用一个线程去轮询多个 io 端口是否可用，如果一个可用就通知对应的 io 请求，这使用一个线程轮询可以大大增强性能。
    - 1.我可以采用 多线程+ 阻塞 IO 达到类似的效果，但是由于在多线程 + 阻塞 IO 中，每个 socket 对应一个线程，这样会造成很大的资源占用。
    - 2.而在多路复用 IO 中，轮询每个 socket 状态是内核在进行的，这个效率要比用户线程要高的多。
- 5.异步 io：aio，用户线程完全不感知 io 的进行，所有操作都交给内核，io 完成之后内核通知用户线程。
    - 1.这种 io 才是异步的，2、3、4 都是同步 io，因为内核进行数据拷贝的过程都会让用户线程阻塞。
    - 2.异步 IO 是需要操作系统的底层支持，也就是内核支持，Java 7 中，提供了 Asynchronous IO

47.ConcurrentHashMap 内部实现，HashTable 的实现被废弃的原因:

- 1.HashTable 容器在竞争激烈的并发环境下表现出效率低下的原因，是因为所有访问 HashTable 的线程都必须竞争同一把锁，那假如容器里有多把锁，每一把锁用于锁容器其中一部分数据，那么当多线程访问容器里不同数据段的数据时，线程间就不会存在锁竞争，从而可以有效的提高并发访问效率，这就是 ConcurrentHashMap 所使用的锁分段技术，首先将数据分成一段一段的存储，然后给每一段数据配一把锁，当一个线程占用锁访问其中一个段数据的时候，其他段的数据也能被其他线程访问。
- 2.ConcurrentHashMap 是由 Segment 数组结构和 HashEntry 数组结构组成。Segment 是一种可重入锁 ReentrantLock，在 ConcurrentHashMap 里扮演锁的角色，HashEntry 则用于存储键值对数据。一个 ConcurrentHashMap 里包含一个 Segment 数组，Segment 的结构和 HashMap 类似，是一种数组和链表结构， 一个 Segment 里包含一个 HashEntry 数组，每个 HashEntry 是一个链表结构的元素，每个 Segment 守护者一个 HashEntry 数组里的元素,当对 HashEntry 数组的数据进行修改时，必须首先获得它对应的 Segment 锁。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

48.HandlerThread 是什么

- 1.MessageQueue + Looper + Handler

49.IntentService 是什么

- 1.含有 HandlerThread 的 Service，可以多次 startService()来多次在子线程中进行 onHandlerIntent()的调用。

50.class 和 dex

- 1.dvm 执行的是 dex 格式文件，jvm 执行的是 class 文件，android 程序编译完之后生产 class 文件。然后 dex 工具会把 class 文件处理成 dex 文件，然后把资源文件和.dex 文件等打包成 apk 文件。
- 2.dvm 是基于寄存器的虚拟机，而 jvm 执行是基于虚拟栈的虚拟机。寄存器存取速度比栈快的多，dvm 可以根据硬件实现最大的优化，比较适合移动设备。
- 3.class 文件存在很多的冗余信息，dex 工具会去除冗余信息，并把所有的 class 文件整合到 dex 文件中。减少了 I/O 操作，提高了类的查找速度

51.内存泄漏

- 1.其他线程持有一个 Listener，Listener 操作 activity。那么在线程么有完毕的时候，activity 关闭了，原本是要被回收的但是，不能被回收。
- 2.例如 Handler 导致的内存泄漏，Handler 就相当于 Listener。
- 3.在 activity 关闭的时候注意停止线程，或者将 Listener 的注册取消
- 3.使用弱引用，这样即使 Listener 持有了 activity，在 GC 的时候还是会被回收
- 4.工具:LeakCanary

52.过度绘制、卡顿优化:

- 1.过度绘制：
    - 1.移除 Window 默认的 Background：getWidow.setBackgroundDrawable(null);
    - 2.移除 XML 布局文件中非必需的 Background
    - 3.减少布局嵌套(扁平化的一个体现，减少 View 数的深度，也就减少了 View 树的遍历时间，渲染的时候，前后期的工作，总是按 View 树结点来)

  - 4.在引入布局文件里面，最外层可以用 merge 替代 LinearLayout,RelativeLayout，这样把子 UI 元素直接衔接在 include 位置
  - 5.工具：HierarchyViewer 查看视图层级

- 2.卡顿优化：16ms 数据更新

53.apk 瘦身:

- 1.classes.dex：通过代码混淆，删掉不必要的 jar 包和代码实现该文件的优化
- 2.资源文件：通过 Lint 工具扫描代码中没有使用到的静态资源
- 3.图片资源：使用 tinypng 和 webP，下面详细介绍图片资源优化的方案,矢量图
- 4.SO 文件将不用的去掉，目前主流 app 一般只放一个 arm 的 so 包

54.ANR 的形成，各个组件上出现 ARN 的时间限制是多少

- 1.只要是主线程耗时的操作就会 ARN 如 io
- 2.broadcast 超时时间为 10 秒 按键无响应的超时时间为 5 秒 前台 service 无响应的超时时间为 20 秒，后台 service 为 200 秒

55.Serializable 和 Parcelable 的区别

- 1.P 消耗内存小
- 2.网络传输用 S 程序内使用 P
- 3.S 将数据持久化方便
- 4.S 使用了反射 容易触发垃圾回收 比较慢

56.Sharedpreferences 源码简述

- 1.储存于硬盘上的 xml 键值对，数据多了会有性能问题
- 2.ContextImpl 记录着 SharedPreferences 的重要数据，文件路径和实例的键值对
- 3.在 xml 文件全部内加载到内存中之前，读取操作是阻塞的，在 xml 文件全部内加载到内存中之后，是直接读取内存中的数据
- 4.apply 因为是异步的没有返回值, commit 是同步的有返回值能知道修改是否提交成功
- 5.多并发的提交 commit 时，需等待正在处理的 commit 数据更新到磁盘文件后才会继续往下执行，从而降低效率; 而 apply 只是原子更新到内存，后调用 apply 函数会直接覆盖前面内存数据，从一定程度上提高很多效率。 3.edit()每次都是创建新的 EditorImpl 对象.
- 6.博客推荐：全面剖析 SharedPreferences

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

57.操作系统如何管理内存的：

- 1.使用寄存器进行将进程地址和物理内存进行映射
- 2.虚拟内存进行内存映射到硬盘上增大内存
- 3.虚拟内存是进行内存分页管理
- 4.页表实现分页，就是 页+地址偏移。
- 5.如果程序的内存在硬盘上，那么就需要用页置换算法来将其调入内存中：先进先出、最近未使用最少等等
- 6.博客推荐：现代操作系统部分章节笔记

58.浏览器输入地址到返回结果发生了什么

- 1.DNS 解析
- 2.TCP 连接
- 3.发送 HTTP 请求
- 4.服务器处理请求并返回 HTTP 报文
- 5.浏览器解析渲染页面
- 6.连接结束

59.java 泛型类型擦除发生在什么时候，通配符有什么需要注意的。

- 1.发生在编译的时候
- 2.PECS，extends 善于提供精确的对象 A 是 B 的子集，Super 善于插入精确的对象 A 是 B 的超集
- 3.博客推荐：Effective Java 笔记（不含反序列化、并发、注解和枚举）、android 阿里面试 java 基础锦集

60.activity 的生命周期

- 1.a 启动 b，后退键再到 a 的生命周期流程：a.create-->a.start-->a.resume-->a.pause-->b.create-->b.start-->b.resume-->b 界面绘制-->a.stop-->b.pause-->b.stop-->b.destroy-->a.restart-->a.start-->a.resume
- 2.意外销毁会调用 saveInstance，重新恢复的时候回调用 restoreInstance。储存数据的时候使用了委托机制，从 activity-->window-->viewGroup-->view 会递归调用 save 来保持本 view 的数据，restore 则是递归恢复本 view 数据。我们可以在里面做一些自己需要的数据操作。

《Android 架构师面试题精编解析大全》

------腾讯享学课堂团队

61.面试常考的算法

- 1.快排、堆排序为首的各种排序算法
- 2.链表的各种操作：判断成环、判断相交、合并链表、倒数 K 个节点、寻找成环节点
- 3.二叉树、红黑树、B 树定义以及时间复杂度计算方式
- 4.动态规划、贪心算法、简单的图论
- 5.推荐书籍：**算法导论**，将图论之前的例子写一遍

62.Launcher 进程启动另外一个进程的过程：启动一个 app

63.开源框架源码

- 1.Fresco
    - 1.mvc 框架：
        - 1.Controller 控制数据显示在 Hierarchy 中的 Drawable 的显隐
        - 2.ImagePipeline 在 Controller 中负责进行数据获取 ,返回的数据是 CloseableImage
        - 3.Drawee 把除了初始化之外的操作全部交给 Holder 去做 ,Holder 持有 Controller 和 Hierarchy
    - 2.Drawable 层次以及绘制：
        - 1.如果要绘制一次 Drawable 就调用 invalidateSelf()来触发 onDraw()
        - 2.Drawable 分为：容器类(保存一些 Drawable)、自我绘制类(进度条)、图形变换类(scale、rotate、矩阵变换)、动画类(内部不断刷新，进行 webp 和 gif 的帧绘制)
        - 3.ImagePipeline 返回的 CloseableImage 是由一个个 DrawableFactory 解析成 Drawable 的
        - 4.webp 和 gif 动画是由 jni 代码解析的 ,然后其他静态图片是根据不同的 android 平台使用 BitmapFactory 来解析的
    - 3.职责链模式：producer 不做操作标 n，表示只是提供一个 consumer。获取图片--》解码图片缓存 Producer--》后台线程 Producer--》client 图片处理 producer(n)--》解码 producer(n)--》旋转或剪裁 producer(n)--》编码图片内存缓存 producer--》读硬盘缓存 producer--》写硬盘缓存 producer(n)--》网络 producer 提供 CloseableImage《--解码图片缓存 consumer《--client 图片处理 consumer《--解码 consumer《--旋转或剪裁 consumer《--编码图片内存缓存 consumer《--写硬盘缓存 consumer《--图片数据
    - 4.内存缓存：

- 1.一个 CountingLruMap 保存已经没有被引用的缓存条目，一个 CountingLruMap 保存所有的条目包括没有引用的条目。每当缓存策略改变和一定时间缓存配置的更新的时候，就会将 待销毁条目 Map 中的条目一个个移除，直到缓存大小符合配置。
- 2.这里的引用计数是用 Fresco 组件实现的引用计数器。
- 3.缓存有一个代理类，用来追踪缓存的存取。
- 4.CountingLruMap 是使用 LinkedHashMap 来储存数据的。

○ 5.硬盘缓存：

- 1.DefaultDiskStorage 使用 Lru 策略。
- 2.为了不让所有的文件集中在一个文件中，创建很多命名不同的文件夹，然后使用 hash 算法把缓存文件分散
- 3.DiskStorageCache 封装了 DefaultDiskStorage，不仅进行缓存存取追踪，并且其在内存里面维持着一个 <key,value> 的键值对，因为文件修改频繁，所有只是定时刷新，因此如果在内存中找不到，还要去硬盘中找一次。
- 4.删除硬盘的缓存只出现在硬盘数据大小超限的时候，此时同时也会删除缓存中的 key，所以不会出现内存中有 key，但是硬盘上没有的情况。
- 5.在插入硬盘数据的时候，采用的是插入器的形式。返回一个 Inserter，在 Inserter.writeData()中传入一个 CallBack(里面封装了客户端插入数据的逻辑和文件引用)，让内部实现调用 CallBack 的逻辑来插入文件数据，前面写的文件后缀是.temp,只有调用 commit()之后才会修改后缀，让文件对客户端可见。
- 6.使用了 java 提供的 FileTreeVisitor 来遍历文件

○ 6.对象池：

- 1.使用数组来存储一个桶，桶内部是一个 Queue。数组下标是数据申请内存的 byte 大小，桶内部的 Queue 存的是内存块的。所以数组使用的是稀疏数组
- 2.申请内存的方式有两种 1.java 堆上开辟的内存 2.ashme 的本地内存中开辟的内存

○ 7.设计模式：Builder、职责链、观察者、代理、组合、享元、适配器、装饰者、策略、生产者消费者、提供者

○ 8.自定义计数引用：类似 c++智能指针

- 1.使用一个静态 IdentityHashMap <储存需要被计数引用的对象,其被引用的次数>
- 2.用 SharedReference 分装需要被计数引用的对象 ,提供一个销毁资源的销毁器，提供一个静态工厂方法来复制自己，复制一个引用计数加一。提供一个方法销毁自己，表示自己需要变成无人引用的对象了，此时引用计数减一。

        - 3.引用计数归零，销毁器将销毁资源，如 bitmap 的 recycle 或者是 jni 内存调用 jni 方法归还内存。

    - 9.博客推荐：Android Fresco 源码文档翻译、从零开始撸一个 Fresco 之硬盘缓存、从零开始撸一个 Fresco 之 gif 和 Webp 动画、从零开始撸一个 Fresco 之内存缓存、从零开始撸一个 Fresco 之总结

- 2.oKhttp：
    - 1.同步和异步：
        - 1.异步使用了 Dispatcher 来将存储在 Deque 中的请求分派给线程池中各个线程执行。
        - 2.当任务执行完成后，无论是否有异常，finally 代码段总会被执行，也就是会调用 Dispatcher 的 finished 函数，它将正在运行的任务 Call 从队列 runningAsyncCalls 中移除后，主动的把缓存队列向前走了一步。
    - 2.连接池：
        - 1.一个 Connection 封装了一个 socket，ConnectionPool 中储存 s 着所有的 Connection，StreamAllocation 是引用计数的一个单位
        - 2.当一个请求获取一个 Connection 的时候要传入一个 StreamAllocation，Connection 中存着一个弱引用的 StreamAllocation 列表，每当上层应用引用一次 Connection，StreamAllocation 就会加一个。反之如果上层应用不使用了，就会删除一个。
        - 3.ConnectionPool 中会有一个后台任务定时清理 StreamAllocation 列表为空的 Connection。5 分钟时间，维持 5 个 socket
    - 3.选择路线与建立连接
        - 1.选择路线有两种方式：
            - 1.无代理，那么在本地使用 DNS 查找到 ip，注意结果是数组，即一个域名有多个 IP，这就是自动重连的来源
            - 2.有代理 HTTP：设置 socket 的 ip 为代理地址的 ip，设置 socket 的端口为代理地址的端口
            - 3.代理好处：HTTP 代理会帮你在远程服务器进行 DNS 查询，可以减少 DNS 劫持。
        - 2.建立连接

        - 1.连接池中已经存在连接，就从中取出(get)RealConnection，如果没有命中就进入下一步
        - 2.根据选择的路线(Route)，调用 Platform.get().connectSocket 选择当前平台 Runtime 下最好的 socket 库进行握手
        - 3.将建立成功的 RealConnection 放入(put)连接池缓存
        - 4.如果存在 TLS，就根据 SSL 版本与证书进行安全握手
        - 5.构造 HttpStream 并维护刚刚的 socket 连接，管道建立完成

    - 4.职责链模式：缓存、重试、建立连接等功能存在于拦截器中网络请求相关，主要是网络请求优化。网络请求的时候遇到的问题
    - 5.博客推荐：Android 数据层架构的实现_上篇、Android 数据层架构的实现_下篇

- 3.okio

    - 1.简介；

        - 1.sink：自己--》别人
        - 2.source：别人--》自己
        - 3.BufferSink：有缓存区域的 sink
        - 4.BufferSource：有缓存区域的 source
        - 5.Buffer：实现了 3、4 的缓存区域，内部有 Segment 的双向链表，在在转移数据的时候，只需要将指针转移指向就行

    - 2.比 java io 的好处：

        - 1.减少内存申请和数据拷贝
        - 2.类少，功能齐全，开发效率高

    - 3.内部实现：

        - 1.Buffer 的 Segment 双向链表，减少数据拷贝
        - 2.Segment 的内部 byte 数组的共享，减少数据拷贝
        - 3.SegmentPool 的共享和回收 Segment
        - 4.sink 和 source 中被实际操作的其实是 Buffer，Buffer 可以充当 sink 和 source
        - 5.最终 okio 只是对 java io 的封装，所有操作都是基于 java io 的