1. 存放原生资源文件，系统在编译时不会编译assets下的资源文件
2. 通过AssetsManager工具类去加载
3. 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。
4. **分别是**：
   1. LinearLayout(线性布局)
      1. 线性布局，每一个LinearLayout里面又可分为垂直布局（android:orientation="vertical"）和水平布局（android:orientation="horizontal" ）。当垂直布局时，每一行就只有一个元素，多个元素依次垂直往下；水平布局时，只有一行，每一个元素依次向右排列TableLayout(表格布局)
   2. RelativeLayout(相对布局)
      1. 相对布局可以理解为某一个元素为参照物，来定位的布局方式
   3. AbsoluteLayout(绝对布局)
      1. 绝对布局用X,Y坐标来指定元素的位置，这种布局方式也比较简单，但是在屏幕旋转时，往往会出问题，而且多个元素的时候，计算比较麻烦
   4. FrameLayout(框架布局)
      1. 所有东西依次都放在左上角，会重叠，这个布局比较简单，也只能放一点比较简单的东西
   5. TableLayout（表格布局）
      1. 每一个TableLayout里面有表格行TableRow，TableRow里面可以具体定义每一个元素
5. **分别有四种：**

在android里，有4种activity的启动模式，分别为：

“standard” (默认)

“singleTop”

“singleTask”

“singleInstance”

它们主要有如下不同：

1. 如何决定所属task

“standard”和”singleTop”的activity的目标task，和收到的Intent的发送者在同一个task内，除非intent包括参数FLAG\_ACTIVITY\_NEW\_TASK。

如果提供了FLAG\_ACTIVITY\_NEW\_TASK参数，会启动到别的task里。

“singleTask”和”singleInstance”总是把activity作为一个task的根元素，他们不会被启动到一个其他task里。

2. 是否允许多个实例

“standard”和”singleTop”可以被实例化多次，并且存在于不同的task中，且一个task可以包括一个activity的多个实例；

“singleTask”和”singleInstance”则限制只生成一个实例，并且是task的根元素。 singleTop要求如果创建intent的时候栈顶已经有要创建 的Activity的实例，则将intent发送给该实例，而不发送给新的实例。

3. 是否允许其它activity存在于本task内

“singleInstance”独占一个task，其它activity不能存在那个task里；如果它启动了一个新的activity，不管新的activity的launch mode 如何，新的activity都将会到别的task里运行（如同加了FLAG\_ACTIVITY\_NEW\_TASK参数）。

而另外三种模式，则可以和其它activity共存。

4. 是否每次都生成新实例

“standard”对于没一个启动Intent都会生成一个activity的新实例；

“singleTop”的activity如果在task的栈顶的话，则不生成新的该activity的实例，直接使用栈顶的实例，否则，生成该activity的实例。

比如现在task栈元素为A-B-C-D（D在栈顶），这时候给D发一个启动intent，如果D是 “standard”的，则生成D的一个新实例，栈变为A－B－C－D－D。

如果D是singleTop的话，则不会生产D的新实例，栈状态仍为A-B-C-D

如果这时候给B发Intent的话，不管B的launchmode是”standard” 还是 “singleTop” ，都会生成B的新实例，栈状态变为A-B-C-D-B。

“singleInstance”是其所在栈的唯一activity，它会每次都被重用。

“singleTask”如果在栈顶，则接受intent，否则，该intent会被丢弃，但是该task仍会回到前台。

当已经存在的activity实例处理新的intent时候，会调用onNewIntent()方法 如果收到intent生成一个activity实例，那么用户可以通过back键回到上一个状态；如果是已经存在的一个activity来处理这个intent的话，用户不能通过按back键返回到这之前的状态。

6.

不设置Activity的android:configChanges时，切屏会重新调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次；设置Activity的android:configChanges="orientation"时，切屏还是会重新调用各个生命周期，切横、竖屏时只会执行一次；设置Activity的android:configChanges="orientation|keyboardHidden"时，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法

7.

简单的说，Handler获取当前线程中的looper对象，looper用来从存放Message的MessageQueue中取出Message，再有Handler进行Message的分发和处理.

8.

onCreate(),onStart(),onResume(),onPause(),onStop(),onRestart(),onDestroy()

9.

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。

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。

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

10.

1. Serializable，将对象序列化成二进制数据传递

2. charsequence: 主要用来传递String，char等

3. parcelable: 这个android提供的一种新的类型，用来封装数据的容器，和Serializable相似，但是序列化的方式不同

4.Bundle： Bundle是将数据传递到另一个上下文中或保存，或者回复自己状态的数据存储方式，数据不是持久化状态。

11.

Collections是个java.util下的类，它包含有各种有关集合操作的静态方法。 Collection是个java.util下的接口，它是各种集合结构的父接口。

12.

int是基本数据类型，而Integer是java提供的类（包装类）。因为java是运用堆栈结构的，int基本类型放在栈里，而Integer类是放在堆里的，这是他俩的本质区别。从生命周期上想。int的生命周期短相对于Integer。

13.

显式跳转

隐式跳转

14.

Application not response

将耗时操作放在分线程，如下载，联网等

15.

Intent是意图，负责对应用中一次操作的动作、动作涉及数据、附加数据进行描述，Android则根据此Intent的描述，负责找到对应的组件，将 Intent传递给调用的组件，并完成组件的调用；intentfilter则是意图过滤器，用于描述intent的各种属性，筛选出符合intent属性的组件

16.

Sharepreference，文件存储，SQLite存储，contentprovider存储，网络存储

17.

Activity是android四大组件之一，负责界面的显示和用户的交互

Intent是意图，负责组件之间信息的传递和跳转

Intentfilter，是意图过滤器，负责过滤符合条件的组件数据

Service，服务，四大组件之一，在后台运行

Broadcast，广播，四大组件之一，主要负责信息的传递

Broadcastreceiver，广播接收器，四大组件之一，负责接收广播

18.

活动进程

可视进程

服务进程

后台进程

空进程

19.

andriod提供了 Handler 和Looper 来满足线程间的通信。Handler先进先出原则。Looper类用来管理特定线程内对象之间的消息交换(MessageExchange)。

　　1)Looper: 一个线程可以产生一个Looper对象，由它来管理此线程里的MessageQueue(消息队列)。

　　2)Handler: 你可以构造Handler对象来与Looper沟通，以便push新消息到Message Queue里;或者接收Looper从MessageQueue取出)所送来的消息。

　　3) Message Queue(消息队列):用来存放线程放入的消息。

4)线程：UI thread 通常就是mainthread，而Android启动程序时会替它建立一个MessageQueue。

20.

Acitivity的进程，当处理Intent的时候，会产生一个对应的Service

Android的进程处理器现在会尽可能的不kill掉你

非常容易使用

21.

1.Activity.finish();

也可以用killProcess()和System.exit()这样的方法

2.用ActivityManager的restartPackage方法

22.

Android提供了ContentProvider，一个程序可以通过实现一个ContentProvider的抽象接口将自己的数据完全暴露出去，而且ContentProviders是以类似数据库中表的方式将数据暴露，也就是说ContentProvider就像一个“数据库”。那么外界获取其提供的数据，也就应该与从数据库中获取数据的操作基本一样，只不过是采用URI来表示外界需要访问的“数据库”。外部访问通过ContentResolver去访问并操作这些被暴露的数据。

23.

TweenAnimation

FrameAnimation

propertyAnimation

24.

最常用的集合类是 List 和 Map。 List 的具体实现包括 ArrayList 和 Vector，它们是可变大小的列表，比较适合构建、存储和操作任何类型对象的元素列表。 List 适用于按数值索引访问元素的情形。

Map 提供了一个更通用的元素存储方法。 Map 集合类用于存储元素对（称作"键"和"值"），其中每个键映射到一个值。

25.

单例就是一个类只有一个实例

方法：

饿汉式和懒汉式

私有化构造器，饿汉式是直接创建好类的实例，在需要的时候返回它，懒汉式是在需要使用类的实例的时候才去创建，需要注意线程安全问题

26.

内存缓存

本地缓存

网络缓存

28.

Convertview

Viewholder

缓存数据复用

29.

assets目录下存放的资源代表应用无法直接访问的原生资源，应用程序需要通过AssetsManager以二进制流的形式来读取资源。SDK编译时不会再R类中为原生资源创建索引项。

res/raw目录下也是存放原生资源（音频、视频），推荐放入assets目录下。

res/raw和assets的相同点：

1.两者目录下的文件在打包后会原封不动的保存在apk包中，不会被编译成二进制。

res/raw和assets的不同点：

1.res/raw中的文件会被映射到R.java文件中，访问的时候直接使用资源ID即R.id.filename；assets文件夹下的文件不会被映射到R.java中，访问的时候需要AssetManager类。

2.res/raw不可以有目录结构，而assets则可以有目录结构（在其目录下可以再建文件夹）

30.

# 主要实现三个方法：

# onMeasure 测量控价的大小

# onLayout 控制显示的位置

# onDraw 将试图绘制出来，画图

31.

onAttach(),onCreate(),onCreateView(),onActivityCreated(),onStart(),onResume(),onPause(),onStop(),onDestroyView(),onDestroy(),onDetach()

32.

a) dvm执行的是.dex 文件，而jvm 执行的是.class。Android 工程编译后的所有.class字节码会被dex工具抽

取到一个.dex文件中。

b) 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。

c) .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

33.

Activity 、 Service 、 ContentProvider 如 果 要 使 用 则 必 须 在 AndroidManifest.xml 中 进 行 注 册 ， 而

BroadcastReceiver则有两种注册方式，静态注册和动态注册。其中静态注册就是指在AndroidManifest.xml中进行注册，而动态注册时通过代码注册。

34.

a) foreground activity

b) visible activity

c) background activity

d) empty process

35.

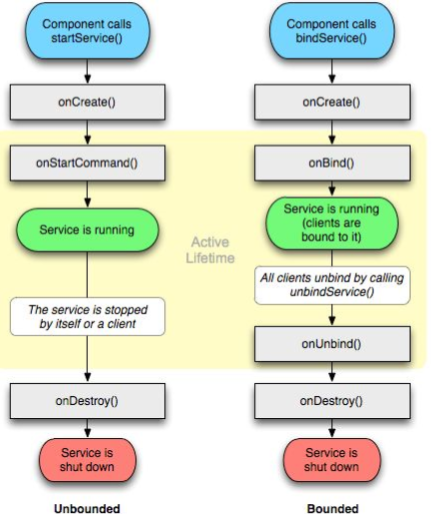
一般情况下比如说有两个activity,分别叫A,B,当在A里面激活B组件的时候, A会调用 onPause()方法,然后B调

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。

36.



37.

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。

38.

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。

39.

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）。

40.

Android Interface Define Language

安卓接口定义语言，用于进程间通信

41.

Broadcast Receiver用于接收并处理广播通知(broadcast announcements)。多数的广播是系统发起的，如地域变换、电量不足、来电短信等。程序也可以播放一个广播。程序可以有任意数量的broadcast receivers来响应它觉得重要的通知。Broadcast receiver可以通过多种方式通知用户: 启动activity、使用NotificationManager、开启背景灯、振动设备、播放声音等，最典型的是在状态栏显示一个图标，这样用户就可以点它打开看通知内容。通常我们的某个应用或系统本身在某些事件(电池电量不足、来电短信)来临时会广播一个Intent出去，我们利用注册一个broadcastreceiver来监听这些Intent并获取Intent中的数据。

42.

Create table person(\_id int primarykey autoincrement,name char,age int)

43.

Padding 内边距

Margin 外边距

44.

overridePendingTransition

45.

final修饰符（关键字）如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承。因此一个类不能既被声明为 abstract的，又被声明为final的。将变量或方法声明为final，可以保证它们在使用中不被改变。被声明为final的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为final的方法也同样只能使用，不能重载 finally再异常处理时提供 finally 块来执行任何清除操作。如果抛出一个异常，那么相匹配的 catch 子句就会执行，然后控制就会进入 finally 块（如果有的话）。

finalize方法名。Java 技术允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的。它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize() 方法以整理系统资源或者执行其他清理工作。finalize() 方法是在垃圾收集器删除对象之前对这个对象调用的。

46.

GC是垃圾收集器。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一： System.gc() Runtime.getRuntime().gc()

47.

没有，是java的保留字

48

sleep()方法是使线程停止一段时间的方法。在sleep 时间间隔期满后，线程不一定立即恢复执行。这是因为在那个时刻，其它线程可能正在运行而且没有被调度为放弃执行，除非(a)“醒来”的线程具有更高的优先级 (b)正在运行的线程因为其它原因而阻塞。 wait()是线程交互时，如果线程对一个同步对象x 发出一个wait()调用，该线程会暂停执行，被调对象进入等待状态，直到被唤醒或等待时间到。

49.

封装、继承、多态

50.

在开发 Android 移动客户端的时候往往要使用多线程来进行操作， 我们通常会将耗时的操作放在单独的线程执行， 避免其占用主线程而给用户带来不好的用户体验。 但是在子线程中无法 去操作主线程（UI 线程） ，在子线程中操作 UI 线程会出现错误。因此 android 提供了一个类 Handler 来在子线程中来更新 UI 线程，用发消息的机制更新 UI 界面，呈现给用户。 这样就解决了子线程更新 UI 的问题。但是费时的任务操作总会启动一些匿名的子线程，太多的子线程给系统带来巨大的负担，随之带来一些性能问题。因此 android 提供了一个工具类AsyncTask，顾名思义异步执行任务。这个 AsyncTask 生来就是处理一些后台的比较耗时的任务，给用户带来良好用户体验的，从编程的语法上显得优雅了许多，不再需要子线程和Handler就可以完成异步操作并且刷新用户界面。