1. 请描述下 Activity 的生命周期。

必调用的三个方法：onCreate() --> onStart() --> onResume()，用 AAA 表示

（1）父 Activity 启动子 Activity，子 Actvity 退出，父 Activity 调用顺序如下

AAA --> onFreeze() --> onPause() --> onStop() --> onRestart() --> onStart(),onResume() …

（2）用户点击 Home，Actvity 调用顺序如下

AAA --> onFreeze() --> onPause() --> onStop() -- Maybe --> onDestroy() – Maybe

（3）调用 finish()， Activity 调用顺序如下

AAA --> onPause() --> onStop() --> onDestroy()

（4）在 Activity 上显示 dialog， Activity 调用顺序如下

AAA

（5）在父 Activity 上显示透明的或非全屏的 activity，Activity 调用顺序如下

AAA --> onFreeze() --> onPause()

（6）设备进入睡眠状态，Activity 调用顺序如下

AAA --> onFreeze() --> onPause()

2. 如果后台的 Activity 由于某原因被系统回收了，如何在被系统回收之前保存当前状态？

onSaveInstanceState()

当你的程序中某一个 Activity A 在运行时，主动或被动地运行另一个新的 Activity B，这个时候 A 会执行 onSaveInstanceState()。B 完成以后又会来找 A，这个时候就有两种情况：一是 A 被回收，二是 A 没有被回收，被回收的 A 就要重新调用 onCreate()方法，不同于直接启动的是这回 onCreate()里是带上了参数 savedInstanceState；而没被收回的就直接执行 onResume()，跳过 onCreate()了。

3. 如何将一个 Activity 设置成窗口的样式。

在 AndroidManifest.xml 中定义 Activity 的地方一句话 android:theme="@android:style/Theme.Dialog"或 android:theme="@android:style/Theme.Translucent"就变成半透明的

4. 如何退出 Activity？如何安全退出已调用多个 Activity 的 Application？

对于单一 Activity 的应用来说，退出很简单，直接 finish()即可。

当然，也可以用 killProcess()和 System.exit()这样的方法。

但是，对于多 Activity 的应用来说，在打开多个 Activity 后，如果想在最后打开的 Activity 直接退出，上边的方法都是没有用的，因为上边的方法都是结束一个 Activity 而已。

当然，网上也有人说可以。

就好像有人问，在应用里如何捕获 Home 键，有人就会说用 keyCode 比较 KEYCODE_HOME 即可，而事实上如果不修改 framework，根本不可能做到这一点一样。

所以，最好还是自己亲自试一下。

那么，有没有办法直接退出整个应用呢？

在 2.1 之前，可以使用 ActivityManager 的 restartPackage 方法。

它可以直接结束整个应用。在使用时需要权限 android.permission.RESTART_PACKAGES。

注意不要被它的名字迷惑。

可是，在 2.2，这个方法失效了。

在 2.2 添加了一个新的方法，killBackgroundProcesses()，需要权限 android.permission.KILL_BACKGR

OUND_PROCESSES。
可惜的是，它和 2.2 的 restartPackage 一样，根本起不到应有的效果。

另外还有一个方法，就是系统自带的应用程序管理里，强制结束程序的方法，forceStopPackage()。
它需要权限 android.permission.FORCE_STOP_PACKAGES。
并且需要添加 android:sharedUserId="android.uid.system"属性
同样可惜的是，该方法是非公开的，他只能运行在系统进程，第三方程序无法调用。
因为需要在 Android.mk 中添加 LOCAL_CERTIFICATE := platform。
而 Android.mk 是用于在 Android 源码下编译程序用的。

从以上可以看出，在 2.2，没有办法直接结束一个应用，而只能用自己的办法间接办到。

现提供几个方法，供参考：

1、抛异常强制退出：
该方法通过抛异常，使程序 Force Close。
验证可以，但是，需要解决的问题是，如何使程序结束掉，而不弹出 Force Close 的窗口。

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。

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。

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。

除了第一个，都是想办法把每一个 Activity 都结束掉，间接达到目的。
但是这样做同样不完美。
你会发现，如果自己的应用程序对每一个 Activity 都设置了 nosensor，在两个 Activity 结束的间隙，sensor 可能有效了。
但至少，我们的目的达到了，而且没有影响用户使用。

为了编程方便，最好定义一个 Activity 基类，处理这些共通问题。

5. 请介绍下 Android 中常用的五种布局。

FrameLayout（框架布局），LinearLayout （线性布局），AbsoluteLayout（绝对布局），RelativeLayout（相对布局），TableLayout（表格布局）

6. 请介绍下 Android 的数据存储方式。

一.SharedPreferences 方式

二.文件存储方式

三.SQLite 数据库方式

四.内容提供器（Content provider）方式

五. 网络存储方式

7. 请介绍下 ContentProvider 是如何实现数据共享的。

创建一个属于你自己的 Content provider 或者将你的数据添加到一个已经存在的 Content provider 中，前提是有相同数据类型并且有写入 Content provider 的权限。

8. 如何启用 Service，如何停用 Service。

Android 中的 service 类似于 windows 中的 service，service 一般没有用户操作界面，它运行于系统中不容易被用户发觉，

可以使用它开发如监控之类的程序。

一。步骤

第一步：继承 Service 类

public class SMSService extends Service { }

第二步：在 AndroidManifest.xml 文件中的<application>节点里对服务进行配置:

<service android:name=".DemoService" />

二。Context.startService()和 Context.bindService

服务不能自己运行，需要通过调用 Context.startService()或 Context.bindService()方法启动服务。这两个方法都可

以启动 Service，但是它们的使用场合有所不同。

1.使用 startService()方法启用服务，调用者与服务之间没有关连，即使调用者退出了，服务仍然运行。

使用 bindService()方法启用服务，调用者与服务绑定在了一起，调用者一旦退出，服务也就终止。

2.采用 Context.startService()方法启动服务，在服务未被创建时，系统会先调用服务的 onCreate()方法，

接着调用 onStart()方法。如果调用 startService()方法前服务已经被创建，多次调用 startService()方法并

不会导致多次创建服务，但会导致多次调用 onStart()方法。

采用 startService()方法启动的服务，只能调用 Context.stopService()方法结束服务，服务结束时会调用

onDestroy()方法。

3.采用 Context.bindService()方法启动服务，在服务未被创建时，系统会先调用服务的 onCreate()方法，

接着调用 onBind()方法。这个时候调用者和服务绑定在一起，调用者退出了，系统就会先调用服务的

onUnbind()方法，

。接着调用 onDestroy()方法。如果调用 bindService()方法前服务已经被绑定，多次调用 bindService()方法并不会

导致多次创建服务及绑定(也就是说 onCreate()和 onBind()方法并不会被多次调用)。如果调用者希望与正在绑定的服务

解除绑定，可以调用 unbindService()方法，调用该方法也会导致系统调用服务的 onUnbind()-->onDestroy()方法。

三。Service 的生命周期

1.Service 常用生命周期回调方法如下：

onCreate() 该方法在服务被创建时调用，该方法只会被调用一次，无论调用多少次 startService()或 bindService()方法，

服务也只被创建一次。 onDestroy()该方法在服务被终止时调用。

2. Context.startService()启动 Service 有关的生命周期方法

onStart() 只有采用 Context.startService()方法启动服务时才会回调该方法。该方法在服务开始运行时被调用。

多次调用 startService()方法尽管不会多次创建服务，但 onStart() 方法会被多次调用。

3. Context.bindService()启动 Service 有关的生命周期方法

onBind()只有采用 Context.bindService()方法启动服务时才会回调该方法。该方法在调用者与服务绑定时被调用，

当调用者与服务已经绑定，多次调用 Context.bindService()方法并不会导致该方法被多次调用。

onUnbind()只有采用 Context.bindService()方法启动服务时才会回调该方法。该方法在调用者与服务解除绑定时被调用。

备注：

1. 采用 startService()启动服务

```
Intent intent = new Intent(DemoActivity.this, DemoService.class);
startService(intent);
```

2.Context.bindService()启动

```
Intent intent = new Intent(DemoActivity.this, DemoService.class);
```

```
bindService(intent, conn, Context.BIND_AUTO_CREATE);

//unbindService(conn);//解除绑定
```

9. 注册广播有几种方式，这些方式有何优缺点？请谈谈 Android 引入广播机制的用意。

Android 广播机制（两种注册方法）

在 android 下，要想接受广播信息，那么这个广播接收器就得我们自己来实现了，我们可以继承 BroadcastReceiver，就可以有一个广播接受器了。有个接受器还不够，我们还得重写 BroadcastReceiver 里面的 onReceiver 方法，当来广播的时候我们要干什么，这就要我们自己来实现，不过我们可以搞一个信息防火墙。具体的代码：

```
public class SmsBroadCastReceiver extends BroadcastReceiver
{

    @Override
    public void onReceive(Context context, Intent intent)
    {
        Bundle bundle = intent.getExtras();
        Object[] object = (Object[])bundle.get("pdus");
        SmsMessage sms[]=new SmsMessage[object.length];
        for(int i=0;i<object.length;i++)
        {
            sms[0] = SmsMessage.createFromPdu((byte[])object[i]);
            Toast.makeText(context, "来自"+sms[i].getDisplayOriginatingAddress()+" 的消息是："+sms[i].getDisplayMessageBody(), Toast.LENGTH_SHORT).show();

        }
        //终止广播，在这里我们可以稍微处理，根据用户输入的号码可以实现短信防火墙。
        abortBroadcast();
    }

}
```

当实现了广播接收器，还要设置广播接收器接收广播信息的类型，这里是信息：android.provider.Telephony.SMS_RECEIVED

我们就可以把广播接收器注册到系统里面，可以让系统知道我们有个广播接收器。这里有两种，一种是代码动态注册：

```
//生成广播处理
smsBroadCastReceiver = new SmsBroadCastReceiver();
//实例化过滤器并设置要过滤的广播

IntentFilter intentFilter = new IntentFilter("android.provider.Telephony.SMS_RECEIVED");

//注册广播
BroadCastReceiverActivity.this.registerReceiver(smsBroadCastReceiver, intentFilter);
```

一种是在 AndroidManifest.xml 中配置广播

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="spl.broadCastReceiver"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".BroadCastReceiverActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />

            </intent-filter>
        </activity>

        <!--广播注册-->
        <receiver android:name=".SmsBroadCastReceiver">
            <intent-filter android:priority="20">
```

```
                <action android:name="android.provider.Telephony.SMS_RECEIV
ED"/>
            </intent-filter>
        </receiver>

    </application>

    <uses-sdk android:minSdkVersion="7" />

    <!-- 权限申请 -->
    <uses-permission android:name="android.permission.RECEIVE_SMS"></uses
-permission>

</manifest>
```

两种注册类型的区别是：

1)第一种不是常驻型广播，也就是说广播跟随程序的生命周期。

2)第二种是常驻型，也就是说当应用程序关闭后，如果有信息广播来，程序也会被系统调用自动运行。

10. 请解释下在单线程模型中 Message、Handler、Message Queue、Looper 之间的关系。

Handler 简介：

一个 Handler 允许你发送和处理 Message 和 Runable 对象，这些对象和一个线程的 MessageQueue 相关联。每一个线程实例和一个单独的线程以及该线程的 MessageQueue 相关联。当你创建一个新的 Handler 时，它就和创建它的线程绑定在一起了。这里，线程我们也可以理解为线程的 MessageQueue。从这一点上来看，Handler 把 Message 和 Runable 对象传递给 MessageQueue，而且在这些对象离开 MessageQueue 时，Handler 负责执行他们。

Handler 有两个主要的用途：（1）确定在将来的某个时间点执行一个或者一些 Message 和 Runnable 对象。（2）在其他线程（不是 Handler 绑定线程）中排入一些要执行的动作。

Scheduling Message，即（1），可以通过以下方法完成：

post(Runnable):Runnable 在 handler 绑定的线程上执行，也就是说不创建新线程。

postAtTime(Runnable,long):

postDelayed(Runnable,long):

sendEmptyMessage(int):

sendMessage(Message):

sendMessageAtTime(Message,long):

sendMessageDelayed(Message,long):

post 这个动作让你把 Runnable 对象排入 MessageQueue,MessageQueue 受到这些消息的时候执行他们，当然以一定的排序。sendMessage 这个动作允许你把 Message 对象排成队列，这些 Message 对象包含一些信息，Handler 的 hanlerMessage(Message)会处理这些 Message.当然，handlerMessage(Message)必须由 Handler 的子类来重写。这是编程人员需要作的事。

当 posting 或者 sending 到一个 Hanler 时，你可以有三种行为：当 MessageQueue 准备好就处理，定义一个延迟时间，定义一个精确的时间去处理。后两者允许你实现 timeout,tick,和基于时间的行为。

当你的应用创建一个新的进程时，主线程（也就是 UI 线程）自带一个 MessageQueue，这个 MessageQueue 管理顶层的应用对象（像 activities,broadcast receivers 等）和主线程创建的窗体。你可以创建自己的线程，并通过一个 Handler 和主线程进行通信。这和之前一样，通过 post 和 sendmessage 来完成，差别在于在哪一个线程中执行这么方法。在恰当的时候，给定的 Runnable 和 Message 将在 Handler 的 MessageQueue 中被 Scheduled。

Message 简介：

Message 类就是定义了一个信息，这个信息中包含一个描述符和任意的数据对象，这个信息被用来传递给 Handler.Message 对象提供额外的两个 int 域和一个 Object 域，这可以让你在大多数情况下不用作分配的动作。

尽管 Message 的构造函数是 public 的，但是获取 Message 实例的最好方法是调用 Message.obtain(),或者 Handler.obtainMessage()方法，这些方法会从回收对象池中获取一个。

MessageQueue 简介：

这是一个包含 message 列表的底层类。Looper 负责分发这些 message。Messages 并不是直接加到一个 MessageQueue 中，而是通过 MessageQueue.IdleHandler 关联到 Looper。

你可以通过 Looper.myQueue()从当前线程中获取 MessageQueue。

Looper 简介：

Looper 类被用来执行一个线程中的 message 循环。默认情况，没有一个消息循环关联到线程。在线程中调用 prepare()创建一个 Looper，然后用 loop()来处理 messages，直到循环终止。

大多数和 message loop 的交互是通过 Handler。

下面是一个典型的带有 Looper 的线程实现。

```
  class LooperThread extends Thread {
      public Handler mHandler;

      public void run() {
          Looper.prepare();

          mHandler = new Handler() {
              public void handleMessage(Message msg) {
                  // process incoming messages here
              }
          };
```

```
            Looper.loop();
        }
    }
```

11. AIDL 的全称是什么？如何工作？能处理哪些类型的数据？

AIDL 的英文全称是 Android Interface Define Language

当 A 进程要去调用 B 进程中的 service 时，并实现通信，我们通常都是通过 AIDL 来操作的

A 工程：

首先我们在 net.blogjava.mobile.aidlservice 包中创建一个 RemoteService.aidl 文件，在里面我们自定义一个接口，含有方法 get。ADT 插件会在 gen 目录下自动生成一个 RemoteService.java 文件，该类中含有一个名为 RemoteService.stub 的内部类，该内部类中含有 aidl 文件接口的 get 方法。

说明一：aidl 文件的位置不固定，可以任意

然后定义自己的 MyService 类，在 MyService 类中自定义一个内部类去继承 RemoteService.stub 这个内部类，实现 get 方法。在 onBind 方法中返回这个内部类的对象，系统会自动将这个对象封装成 IBinder 对象，传递给他的调用者。

其次需要在 AndroidManifest.xml 文件中配置 MyService 类，代码如下：

```
<!-- 注册服务 -->
<service android:name=".MyService">
   <intent-filter>
   <!--  指定调用 AIDL 服务的 ID  -->
       <action android:name="net.blogjava.mobile.aidlservice.RemoteService" />
    </intent-filter>
</service>
```

为什么要指定调用 AIDL 服务的 ID,就是要告诉外界 MyService 这个类能够被别的进程访问，只要别的进程知道这个 ID，正是有了这个 ID,B 工程才能找到 A 工程实现通信。

说明：AIDL 并不需要权限

B 工程：

首先我们要将 A 工程中生成的 RemoteService.java 文件拷贝到 B 工程中，在 bindService 方法中绑定 aidl 服务

绑定 AIDL 服务就是将 RemoteService 的 ID 作为 intent 的 action 参数。

说明：如果我们单独将 RemoteService.aidl 文件放在一个包里，那个在我们将 gen 目录下的该包拷贝到 B 工程中。如果我们将 RemoteService.aidl 文件和我们的其他类存放在一起，那么我们在 B 工程中就要建立相应的包，以保证 RmoteService.java 文件的报名正确，我们不能修改 RemoteService.java 文件

bindService(new Inten("net.blogjava.mobile.aidlservice.RemoteService"), serviceConnection, Context.BIND_AUTO_CREATE);

ServiceConnection 的 onServiceConnected(ComponentName name, IBinder service)方法中的 service 参数就是 A 工程中 MyService 类中继承了 RemoteService.stub 类的内部类的对象。