# 1. 技术面试题A

## 1.1. 基本面试题

### 1.1.1. ListView如何优化？如何防止OOM？

1. 使用convertView和ViewHolder，使得控件的绑定只需要运行一次，没用的Item及时释放内存。
2. 如果自定义Item中有图片的话，要对图片进行处理。例如：压缩图片，使用弱引用，Bitmap对象要及时释放。
3. 在Adpater中不要使用static来修饰资源消耗过大的变量（例如Context）。因为，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。
4. 尽量避免在ListView适配器中使用线程，因为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制。

### 1.1.2. 广播注册的两种机制，广播发送的两种机制?

a. 实现广播的方法有两种。一种是在androidmanifest.xml当中去注册广播，另一种方法是在代码中动态注册广播，这两种方法虽然都能实 现广播机制，但是还是有很大不同，所以运用也要分情况。

1）静态注册：如果我们在androidmanifest.xml中去定义的话，那么该广播是在activity 结束之后也不会结束的，原因在于它已经写在了manifest.xml文件当中，也就是注册到了系统当中，所以无论你的activity是否存在，对于该 广播没有影响。

2）动态注册：而在java代码中动态注册广播，在该activity结束后，我们可以注销该广播，也就是它随着activity的消失而消失。

这样解 释，大家都应该清楚了。如果是一些系统应用，比如手机没电后震动啊，后台计算流量啊这样的功能，需要一直存在的，我们可以在 androidmanifest.xml中注册，而一些只有该activity存在时才有意义的广播，比如更改界面等等，就用动态注册比较合 适，activity都没了。该广播还有什么用呢？只会浪费资源而已。

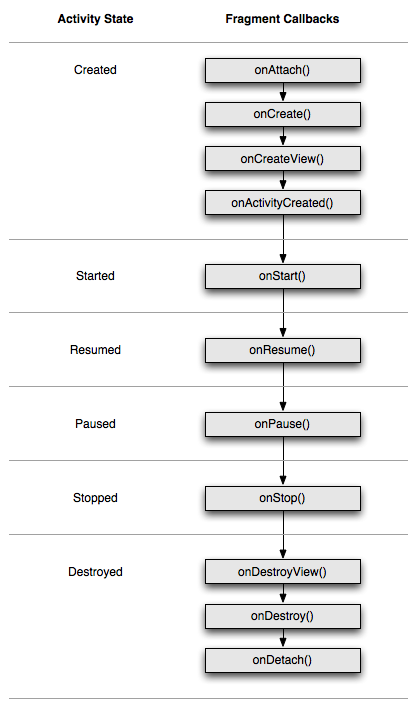
b. 安卓中广播的机制有：

1）普通广播**sendBroadcast()**：按照注册的先后顺序发送至每一个已经注册（订阅）的广播接收器，无法被终止，当然，先接收到广播的receiver可以修改广播数据。。 典型代表：开机启动广播。

2）有序广播**sendOrderedBroadcast()**：按照被接收者的优先级顺序，在被接收者中一次传播。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。每个接收者有权中终止广播，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。如系统收到短信发出的广播就是有序广播。

3）粘性广播**sendStickyBroadcast()**：使用这个api需要权限android.Manifest.permission.BROADCAST\_STICKY,粘性广播的特点是Intent会一直保留到广播事件结束，而这种广播也没有所谓的10秒限制，（10秒限制是指普通的广播如果onReceive方法执行时间太长，超过10秒的时候系统会将这个广播置为可以干掉的candidate，一旦系统资源不够的时候，就会干掉这个广播而让它不执行。）

### 1.1.3. Fragment的生命周期和Activity的生命周期的关系？



fragment的生命周期与管理activity的生命周期极其相似。你所需要去思考的是activity的生命周期如何影响fragment的生命周期。

Activity直接影响它所包含的fragment的生命周期，所以对activity的某个生命周期方法的调用也会产生对fragment相同方法的调用。例如：当activity的onPause()方法被调用时，它所包含的所有的fragment们的onPause()方法都会被调用。

Fragment比activity还要多出几个生命周期回调方法，这些额外的方法是为了与activity的交互而设立，如下：

onAttach()，当fragment被加入到activity时调用（在这个方法中可以获得所在的activity）。

onCreateView()，当activity要得到fragment的layout时，调用此方法，fragment在其中创建自己的layout(界面)。

onActivityCreated()，当activity的onCreated()方法返回后调用此方法。

onDestroyView()，当fragment的layout被销毁时被调用。

onDetach()，当fragment被从activity中删掉时被调用。

一旦activity进入resumed状态（也就是running状态），你就可以自由地添加和删除fragment了。因此，只有当activity在resumed状态时，fragment的生命周期才能独立的运转，其它时候是依赖于activity的生命周期变化的。

### HashMap和HashTable的区别?

1. Hashtable是支持多线程同步访问的，也就是说，是多线程安全的。 HashMap则不是多线程安全的，要想做到多线程安全，需要程序员自己做同步。查看Hashtable的源代码就可以发现，除构造函数外，Hashtable的所有 public 方法声明中都有 synchronized 关键字，而HashMap的源代码中则连 synchronized 的影子都没有。（他们的区别类似于Vector与ArrayList，StringBuffer与StringBuilder）。
2. Hashtable不允许 null 值(key 和 value 都不可以)，HashMap允许 null 值(key和value都可以)。

### 你的APP如何与网络交互？

Android平台有三种网络接口可以使用，他们分别是：java.net.\*(标准Java接口)、org.apache接口和android.net.\*(Android网络接口)。下面分别介绍这些接口的功能和作用。

a. 标准Java接口

java.net.\*提供与联网有关的类，包括流、数据包套接字(socket)、Internet协议、常见Http处理等。比如：创建URL，以及URLConnection/HttpURLConnection对象、设置链接参数、链接到服务器、向服务器写数据、从服务器读取数据等通信。这些在Java网络编程中均有涉及，我们看一个简单的socket编程，实现服务器回发客户端信息。

Apache接口

对于大部分应用程序而言JDK本身提供的网络功能已远远不够，这时就需要Android提供的Apache HttpClient了。它是一个开源项目，功能更加完善，为客户端的Http编程提供高效、最新、功能丰富的工具包支持。

android.net接口

常常使用此包下的类进行Android特有的网络编程，如：访问WiFi,访问Android联网信息，邮件等功能。

### Activity生命周期，在按Back键和Home键时候，生命周期的变化。

HOME键的执行顺序：onPause->onStop

BACK键的顺序： onPause->onStop->onDestroy

Activity生命周期详解：

1. void onCreate(Bundle savedInstanceState)

当Activity被第首次加载时执行。我们新启动一个程序的时候其主窗体的onCreate事件就会被执行。如果Activity被销毁后 （onDestroy后），再重新加载进Task时，其onCreate事件也会被重新执行。注意这里的参数 savedInstanceState（Bundle类型是一个键值对集合，大家可以看成是.Net中的Dictionary）是一个很有用的设计，由于 前面已经说到的手机应用的特殊性，一个Activity很可能被强制交换到后台（交换到后台就是指该窗体不再对用户可见，但实际上又还是存在于某个 Task中的，比如一个新的Activity压入了当前的Task从而“遮盖”住了当前的 Activity，或者用户按了Home键回到桌面，又或者其他重要事件发生导致新的Activity出现在当前Activity之上，比如来电界面）， 而如果此后用户在一段时间内没有重新查看该窗体（Android通过长按Home键可以选择最近运行的6个程序，或者用户直接再次点击程序的运行图标，如 果窗体所在的Task和进程没有被系统销毁，则不用重新加载Process, Task和Task中的Activity，直接重新显示Task顶部的Activity，这就称之为重新查看某个程序的窗体），该窗体连同其所在的 Task和Process则可能已经被系统自动销毁了，此时如果再次查看该窗体，则要重新执行 onCreate事件初始化窗体。而这个时候我们可能希望用户继续上次打开该窗体时的操作状态进行操作，而不是一切从头开始。例如用户在编辑短信时突然来 电，接完电话后用户又去做了一些其他的事情，比如保存来电号码到联系人，而没有立即回到短信编辑界面，导致了短信编辑界面被销毁，当用户重新进入短信程序 时他可能希望继续上次的编辑。这种情况我们就可以覆写Activity的void onSaveInstanceState(Bundle outState)事件，通过向outState中写入一些我们需要在窗体销毁前保存的状态或信息，这样在窗体重新执行onCreate的时候，则会通过 savedInstanceState将之前保存的信息传递进来，此时我们就可以有选择的利用这些信息来初始化窗体，而不是一切从头开始。

2. void onStart()

onCreate事件之后执行。或者当前窗体被交换到后台后，在用户重新查看窗体前已经过去了一段时间，窗体已经执行了onStop事件，但是窗 体和其所在进程并没有被销毁，用户再次重新查看窗体时会执行onRestart事件，之后会跳过onCreate事件，直接执行窗体的onStart事 件。

3. void onResume()

onStart事件之后执行。或者当前窗体被交换到后台后，在用户重新查看窗体时，窗体还没有被销毁，也没有执行过onStop事件（窗体还继续存在于Task中），则会跳过窗体的onCreate和onStart事件，直接执行onResume事件。

4. void onPause()

窗体被交换到后台时执行。

5. void onStop()

onPause事件之后执行。如果一段时间内用户还没有重新查看该窗体，则该窗体的onStop事件将会被执行；或者用户直接按了Back键，将该窗体从当前Task中移除，也会执行该窗体的onStop事件。

6. void onRestart()

onStop事件执行后，如果窗体和其所在的进程没有被系统销毁，此时用户又重新查看该窗体，则会执行窗体的onRestart事件，onRestart事件后会跳过窗体的onCreate事件直接执行onStart事件。

7. void onDestroy()

Activity被销毁的时候执行。在窗体的onStop事件之后，如果没有再次查看该窗体，Activity则会被销毁。

最后用一个实际的例子来说明Activity的各个生命周期。假设有一个程序由2个Activity A和B组成，A是这个程序的启动界面。当用户启动程序时，Process和默认的Task分别被创建，接着A被压入到当前的Task中，依次执行了 onCreate, onStart, onResume事件被呈现给了用户；此时用户选择A中的某个功能开启界面B，界面B被压入当前Task遮盖住了A，A的onPause事件执行，B的 onCreate, onStart, onResume事件执行，呈现了界面B给用户；用户在界面B操作完成后，使用Back键回到界面A，界面B不再可见，界面B的onPause, onStop, onDestroy执行，A的onResume事件被执行，呈现界面A给用户。此时突然来电，界面A的onPause事件被执行，电话接听界面被呈现给用 户，用户接听完电话后，又按了Home键回到桌面，打开另一个程序“联系人”，添加了联系人信息又做了一些其他的操作，此时界面A不再可见，其 onStop事件被执行，但并没有被销毁。此后用户重新从菜单中点击了我们的程序，由于A和其所在的进程和Task并没有被销毁，A的onRestart 和onStart事件被执行，接着A的onResume事件被执行，A又被呈现给了用户。用户这次使用完后，按Back键返回到桌面，A的 onPause, onStop被执行，随后A的onDestroy被执行，由于当前Task中已经没有任何Activity，A所在的Process的重要程度被降到很 低，很快A所在的Process被系统结束

### 如何解析json，有没有看过JsonObject的源码。

常用的json框架：安卓自带的JsonObject,谷歌的Gson，阿里巴巴的FastJson,还有Jackson(SpringMVC中自带的Json解析器)。

### 如何使apk开机自启？

android开机事件会发送一个叫做Android.intent.action.BOOT\_COMPLETED的广播信息,我的程序会在接收到这个监听的时候开启我的应用。但是由于加载sd卡的事件发生在该广播后，故而apk需要安装在sd卡中。

### 图片如何实现异步加载？

采用多线程和线程池

### Intent可以加入什么数据？是否能放入Bitmap？

Android中Intent传递类对象提供了两种方式一种是 通过实现Serializable接口传递对象，一种是通过实现Parcelable接口传递对象。

采用bundle传递bitmap

## 1.2. PL面试题

### 1.2.1. 网络传输中如何处理https？

准备知识：

Keytool工具的使用。

在用Android平台上使用SSL，第一步就是生成证书。

1、证书的生成

1.1生成服务器端的证书py

keytool -genkey -alias test -keystore test.jks

1.2 将keystore中的cert导出来，用来生成客户端的验证证书

[html]

keytool -exportcert -alias test -file test.cert -keystore test.jks

1.3 生成Android平台的证书

因为Android 要求要BC证书，而Java的keytool本身不提供BKS格式，因此要自己手动配置。个人在配置的过程到了文件正在使用中，保存失败的情况，我的做法是将文件备份一下，用unlocker删除后将修改好备份放到原位置就好了。

1.3.1 [下载](http://www.2cto.com/soft) bcprov-ext-jdk15on-146.jar

到官网下载本地jdk对应的jar，复制到C:\Program Files\Java\jdk1.6.0\_43\jre\lib\ext

1.3.2 配置bcprov

在 jdk\_home\jre\lib\security\目录中找到 java.security 在内容增加一行（数字可以自己定义）

[html]

security.provider.11=org.bouncycastle.jce.provider.BouncyCastleProvider

1.3.3 生成android平台的证书

[html]

keytool -importcert -keystore test.bks -file test.cert -storetype BKS -provider org.bouncycastle.jce.provider.BouncyCastleProvider

一、什么是SSL？

  SSL(Secure Sockets Layer 安全套接层),及其继任者传输层安全（Transport Layer Security，TLS）是为网络通信提供安全及数据完整性的一种安全协议。TLS与SSL在传输层对网络连接进行加密。

  SSL/TLS协议位于HTTP协议与传输层协议之间，采用公钥技术，为两个应用间的通讯提供加密、完整性校验以及身份认证。

SSL协议提供了三个功能:

使用公钥证书对双端进行认证(一般仅进行服务器认证,客户端认证为可选)

通信加密(Conﬁdentiality)

数据完整性检查(Data integrity)

二、SSL握手

  SSL会话以称之为SSL握手的一系列消息交换开始,在握手过程中服务器通过公钥技术向客户端认证自身(可选,客户端向服务器认证自身)，客户端和服务器协商加密消息的对称密钥，SSL会话握手后的消息都使用对称密钥加密后传输，对称密钥还用于消息完整性检查。

Client->Server ClientHello

Client向Server发送一个“ClientHello”消息，说明它支持的密码算法列表、压缩方法及最高协议版本，以及稍后将被使用的随机数

Version: 客户端支持TLS协议最高版本号(最高3.3)

RandomNumber:客户端产生的28byte随机数，后续用作加密种子。

CipherSuites 客户端支持的加密算法和密钥大小列表

CompressionMethods 客户可支持的压缩算法

Server->Client ServerHello

服务器向客户端发送“ServerHello”消息，包含服务器选择的密码算法、压缩方法、协议版本以及服务器产生的随机数。

Version: 选择的TLS版本号

RandomeNumber:服务器产生的28字节随机数，后续用作加密种子

CipherSuite: 选择的加密算法和密钥大小。

CompressionMethod: 选择的数据压缩算法

Server->Client Certificate

Server向Client发送自身的证书链,证书链从服务器的公钥证书(public-key

certiﬁcate)开始一直到CA的根证书(certiﬁcate

authority’s root certiﬁcate).

客户端需要对server的证书链进行验证,如果验证不通过，则终止连接。若验证通过，则可从server的公钥证书中获取到server的公钥。验证流程见后续介绍。

Server->Client CertificateRequest

如果Server需要认证Client,则发送此消息请求Client的公钥证书,此消息为可选。消息中包含

Certificate Types: Server可接收的证书类型列表

DistinguishedNames: Server可接收的CA DN列表。

Server->Client ServerHelloDone

Server发送此消息通知Client已完成了初始化协商消息。

Client->Server Certificate

如果Server请求了Client的证书，即存在消息4,则client将自身的证书链信息发送给Server。Server要对Client的证书链进行验证，如果验证不通过，则终止连接，如果验证通过则可从Client的公钥证书中获取到Client的公钥。验证流程见后续介绍.

Client->Server ClientKeyExchange

Client向Server发送premaster secret,并且用Server的公钥加密。premaster secret用于双方计算出对后续消息加密的对称密钥. 使用Server的公钥加密的目的是确认Server具有消息3中所生成公钥证书的私钥，避免发送给伪造的服务器。

Client->Server CertificateVerify

如果Server请求了Client的证书,则还需要发送CertificateVerify消息，Client将到现在为止发送和接收到的握手消息，使用Client的私钥进行签名发送给Server,用于向Server证明其有消息6中声称Client公钥的私钥数据。Server可使用消息6中获得的Client公钥对消息进行验证。

Client->Server ChangeCipherSpec

Client使用客户端随机数,服务器随机数以及premaster secret产生加密消息的对称密钥Master Secret。然后发送此消息告知Server后续消息将使用对称密钥加密

Client->Server Finished

Client向Server发送此消息通知对端协商成功，此消息使用协商的公钥密钥加密。

Server->Client ChangeCipherSpec

Server使用客户端随机数,服务器随机数以及premaster secret产生加密消息的对称密钥Master Secret。然后发送此消息告知Client后续消息将使用对称密钥加密.

Server->Client Finished

Server向Client发送此消息通知对端协商成功，此消息使用协商的公钥密钥加密。

三、SSL通信模式：

1.服务端：

SSL服务端需要通过SSL服务套接字来提供服务接口，而SSL服务套接字需要通过SSL上下文实例来创建。以下是对SSL服务端的启用过程的描述。

（1）通过指定协议（一般是TLS）获取SSL上下文（SSLContext）实例。

（2）通过指定算法（X.509相关）获取密钥管理器工厂（KeyManagerFactory）实例。

（3）通过指定类型和提供者获取密钥库（KeyStore）实例。

（4）密钥库实例使用约定的密码加载（Load）密钥库文件（.keystore）。

（5）密钥管理器工厂实例使用约定的密码和（4）中密钥库进行初始化（Initialize）。

（6）SSL上下文实例通过密钥管理器工厂实例提供的密钥管理器来初始化（Initialize）。

（7）当SSL上下文实力初始化成功后，就可以获取该上下文势力所关联的服务套接字工厂（ServerSocketFactory）实例

（8）服务套接字工厂实例依据指定的服务端口来创建（Create）服务套接字（ServerSocket）。

（9）当SSL服务套接字创建成功，就可以等待客户端的连接，与客户端进行通信。

（10）通信完毕可以关闭服务套接字。

2.客户端

（1）通过指定协议（一般是TLS）获取SSL上下文（SSLContext）实例。

（2）通过指定算法（X.509相关）获取密钥管理器工厂（KeyManagerFactory）实例。

（3）通过指定算法（X.509相关）获取信任管理器工厂（TrustManagerFactory）实例。

（4）通过指定类型和提供者获取密钥库（KeyStore）实例。

（5）通过指定类型和提供者获取信任密钥库（KeyStore）实例。

（6）（4）中密钥库实例使用约定的密码加载（Load）密钥库文件（.keystore）。

（7）（5）中信任密钥库实例使用约定的密码加载（Load）密钥库文件（.keystore）。

（8）密钥管理器工厂实例使用约定的密码和（4）中密钥库进行初始化（Initialize）。

（9）信任密钥管理器工厂实例使用约定的密码和（5）中密钥库进行初始化（Initialize）。

（10）当SSL上下文实力初始化成功后，就可以获取该上下文实例所关联的套接字工厂（SocketFactory）实例

（11）套接字工厂实例依据指定的主机和端口来创建（Create）客户端套接字（Socket）。

（12）当SSL服务套接字创建成功，就可以向服务端发送请求，与服务端进行通信。

（13）通信完毕可以关闭服务套接字。

  服务端代码：

[java]

import java.io.BufferedInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.ServerSocket;

import java.net.Socket;

import java.net.SocketAddress;

import java.security.KeyManagementException;

import java.security.KeyStore;

import java.security.KeyStoreException;

import java.security.NoSuchAlgorithmException;

import java.security.UnrecoverableKeyException;

import java.security.cert.CertificateException;

import javax.net.ServerSocketFactory;

import javax.net.ssl.KeyManager;

import javax.net.ssl.KeyManagerFactory;

import javax.net.ssl.SSLContext;

/\*\*

 \* @author  draem0507@gmail.com

 \* @TODO    java线程开发之四 SSL加密

 \* 开发步骤

 \* 1.生成服务端密钥

 \* 2.导出服务端证书

 \* 3.生成客户端密钥

 \* 4.[程序开发](http://www.2cto.com/kf)测试

 \* 关于证书的生成请参考readme.txt

 \* 参考资料:http://chrui.iteye.com/blog/1018778

 \* @version 1.0

 \* @date 2013-5-7 23:22:45

 \* @update 2013-5-8 10:22:45

 \* @blgos http://www.cnblogs.com/draem0507

 \*/

public class Server {

    private ServerSocket serverSocket;

    private final static char[] password="1qaz2wsx".toCharArray();

    private SSLContext context;

    private InputStream inputStream;

    public Server() {

        inputStream=this.getClass().getResourceAsStream("/test.jks");

        initContext();

        try {

            //直接运行会报 javax.net.ssl.SSLException:

            //ServerSocketFactory factory=     SSLServerSocketFactory.getDefault();

            ServerSocketFactory factory=     context.getServerSocketFactory();

//            serverSocket = new ServerSocket(10000);

            serverSocket=factory.createServerSocket(10000);

            System.out.println("======启动安全SocektServer成功=========");

            while (true) {

                Socket socket = serverSocket.accept();

                new ReceiveSocket(socket).start();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    //ssl 上下文对象的初始化

    private void initContext() {

        try {

            KeyStore store=KeyStore.getInstance("JKS");

            store.load(inputStream, password);

            KeyManagerFactory factory=KeyManagerFactory.getInstance("SunX509");

            factory.init(store,password);

            KeyManager []keyManagers=factory.getKeyManagers();

            context=SSLContext.getInstance("SSL");

            context.init(keyManagers, null    , null);

        } catch (KeyStoreException e) {

            e.printStackTrace();

        } catch (NoSuchAlgorithmException e) {

            e.printStackTrace();

        } catch (CertificateException e) {

            e.printStackTrace();

        } catch (FileNotFoundException e) {

            e.printStackTrace();

        } catch (IOException e) {

            e.printStackTrace();

        } catch (UnrecoverableKeyException e) {

            e.printStackTrace();

        } catch (KeyManagementException e) {

            e.printStackTrace();

        }

    }

    public static void main(String[] args) {

        new Server();

    }

    private class ReceiveSocket extends Thread {

        private Socket socket;

        public ReceiveSocket(Socket socket) {

            this.socket = socket;

        }

        private ObjectInputStream reader;

        private ObjectOutputStream writer;

        @Override

        public void run() {

            try {

                reader=new ObjectInputStream(new BufferedInputStream(socket.getInputStream()));

                //writer=new ObjectOutputStream(socket.getOutputStream());

                // 开启无限循环 监控消息

                    //java.io.EOFException

                Object obj=    reader.readUTF();

                SocketAddress address = socket.getRemoteSocketAddress();

                System.out.println(address.toString() + ">\t" + obj);

//                Object obj=    reader.readObject();

//                    if(obj!=null)

//                    {

//                        User user =(User)obj;

//                        System.out.println("id=="+user.getPassword()+"\tname=="+user.getName());

//                    }

                //    while (true) {}

            } catch (IOException e) {

                e.printStackTrace();

            } finally {

                if (null != reader) {

                    try {

                        reader.close();

                    } catch (IOException e) {

                        e.printStackTrace();

                    }

                }

                if (null != writer) {

                    try {

                        reader.close();

                    } catch (IOException e) {

                        e.printStackTrace();

                    }

                }

                try {

                    socket.close();

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

        }

    }

}

 Android客户端代码：

[java]

protected Void doInBackground(Void... params) {

   Log.i(TAG, "doInBackground");

   try {

    SSLContext context;

    KeyStore ts = KeyStore.getInstance("BKS");

    ts.load(getResources().openRawResource(R.raw.test),

      "1qaz2wsx".toCharArray());

    TrustManagerFactory tmf = TrustManagerFactory

      .getInstance("X509");

    tmf.init(ts);

    TrustManager[] tm = tmf.getTrustManagers();

    context = SSLContext.getInstance("SSL");

    context.init(null, tm, null);

    SocketFactory factory = context.getSocketFactory();

    SSLSocket socket = (SSLSocket) factory.createSocket(

      "192.168.70.249", 10000);

    ObjectOutputStream out = new ObjectOutputStream(

      socket.getOutputStream());

    out.writeUTF(UUID.randomUUID().toString());

    out.flush();

    System.out.println("========客户端发送成功=========");

    ;

    socket.close();

   } catch (Exception ex) {

    ex.printStackTrace();

   }

   return null;

  }

四、HTTPS

 HTTPS可以视为HTTP的安全版本（Secure），其安全基础基于SSL协议（Secure Socket Layer，安全套接字层）。HTTPS在HTTP的基础上添加了一个[加密](http://www.2cto.com/Article/jiami/)和身份验证。其默认端口是443.对于一些对数据安全要求比较高的网络应用，比如网络支付，网上银行，都是采用HTTPS通信机制，其规范：RFC2818

HTTPS URL连接的方式访问HTTPS服务器与HTTP URL访问HTTP服务器的方式基本相同。用到的类：HttpsURLConnection。

代码例子（验证服务器）：

HttpsURLConnection方式

[java]

public static String getHttpsContent(Map<String, String> paramsMap, String urlPath){

        try{

            File bksFile = new File("bks文件路径");

            if(bksFile.exists()){

                InputStream keyStoreInput = new FileInputStream(bksFile);

                String keyStoreType = KeyStore.getDefaultType();

                KeyStore keyStore = KeyStore.getInstance(keyStoreType);

                keyStore.load(keyStoreInput, KEYSTORE\_PASSWORD.toCharArray());

                // Create a TrustManager that trusts the CAs in our KeyStore

                String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();

                TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);

                tmf.init(keyStore);

                KeyManagerFactory kmf = KeyManagerFactory.getInstance("X509");

                kmf.init(keyStore, KEYSTORE\_PASSWORD.toCharArray());

                // Create an SSLContext that uses our TrustManager

                SSLContext sslContext = SSLContext.getInstance("TLS");

                sslContext.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);

                X509HostnameVerifier hostnameVerifier = SSLSocketFactory.ALLOW\_ALL\_HOSTNAME\_VERIFIER;

                HttpsURLConnection.setDefaultHostnameVerifier(hostnameVerifier);

                StringBuilder entityBuilder = new StringBuilder("");

                if(paramsMap!=null && !paramsMap.isEmpty()){

                for(Map.Entry<String, String> entry : paramsMap.entrySet()){

                entityBuilder.append(entry.getKey()).append('=');

                entityBuilder.append(URLEncoder.encode(entry.getValue(), HTTP.UTF\_8));

                entityBuilder.append('&');

                }

                entityBuilder.deleteCharAt(entityBuilder.length() - 1);

                }

                byte[] entity = entityBuilder.toString().getBytes();

                // Tell the URLConnection to use a SocketFactory from our SSLContext

                //URL url = new URL("https://172.16.18.109");

                URL url = new URL(urlPath);

                HttpsURLConnection urlConnection = (HttpsURLConnection) url

                        .openConnection();

                urlConnection.setSSLSocketFactory(sslContext.getSocketFactory());

                urlConnection.setConnectTimeout(5 \* 1000);

                urlConnection.setRequestMethod("POST");

                urlConnection.setDoOutput(true);//允许输出数据

                urlConnection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");

                urlConnection.setRequestProperty("Content-Length", String.valueOf(entity.length));

                OutputStream outStream = urlConnection.getOutputStream();

                outStream.write(entity);

                outStream.flush();

                outStream.close();

                InputStream in = urlConnection.getInputStream();

                StringBuffer sb = new StringBuffer();

                String line = null;

                char ch = '\u0000';

                int temp = 0 ;

                while ((temp = in.read()) != -1) {

                    ch = (char) temp;

                    sb.append((char) temp);

                }

                String result = sb.toString();

                return result;

            }

            return "-1";

        }catch(Exception e){

            e.printStackTrace();

            return "-2";

        }

    }

HttpClient方式：

[java]

public static  synchronized HttpClient getHttpClient(Context context) throws KeyStoreException, NoSuchAlgorithmException, CertificateException, IOException, KeyManagementException, UnrecoverableKeyException

  {

    if(null==httpClient)

      {

              AssetManager sm=   context.getAssets();

              InputStream keyStroreInputStream=sm.open("ca\_zx.bks");

              KeyStore  keyStore=KeyStore.getInstance(KeyStore.getDefaultType());

              keyStore.load(keyStroreInputStream, KEYSTORE\_PASSWORD.toCharArray());

             SSLSocketFactory sf=new MySSLSocketFactory(keyStore);

              sf.setHostnameVerifier(SSLSocketFactory.ALLOW\_ALL\_HOSTNAME\_VERIFIER);

              HttpParams params=new BasicHttpParams();

              HttpProtocolParams.setVersion(params, HttpVersion.HTTP\_1\_1);

              HttpProtocolParams.setContentCharset(params, HTTP.UTF\_8);

              HttpProtocolParams.setUseExpectContinue(params, true);

              ConnManagerParams.setTimeout(params, 10000);

              HttpConnectionParams.setConnectionTimeout(params, 15000);

              HttpConnectionParams.setSoTimeout(params, 20000);

              SchemeRegistry schreg=new SchemeRegistry();

              schreg.register(new Scheme("http", PlainSocketFactory.getSocketFactory(), 80));

              schreg.register(new Scheme("https",   sf , 443));

             ClientConnectionManager conman=new ThreadSafeClientConnManager(params, schreg);

             httpClient=new DefaultHttpClient(conman, params);

      }

      return httpClient;

  }

private static  class MySSLSocketFactory extends SSLSocketFactory

  {

        SSLContext sslContext=SSLContext.getInstance("TLS");

        public MySSLSocketFactory(KeyStore truststore)

                throws NoSuchAlgorithmException, KeyManagementException,

                KeyStoreException, UnrecoverableKeyException {

            super(truststore);

            TrustManager tm=new X509TrustManager() {

                @Override

                public X509Certificate[] getAcceptedIssuers() {

                    return null;

                }

                @Override

                public void checkServerTrusted(X509Certificate[] chain, String authType)

                        throws CertificateException {

                }

                @Override

                public void checkClientTrusted(X509Certificate[] chain, String authType)

                        throws CertificateException {

                }

            };

            KeyManagerFactory kmf = KeyManagerFactory.getInstance("X509");

            kmf.init(truststore, KEYSTORE\_PASSWORD.toCharArray());

            sslContext.init(kmf.getKeyManagers(), new TrustManager[]{ tm}, null);

        }

        @Override

        public Socket createSocket() throws IOException {

            // TODO Auto-generated method stub

            return sslContext.getSocketFactory().createSocket();

        }

        @Override

        public Socket createSocket(Socket socket, String host, int port,

                boolean autoClose) throws IOException, UnknownHostException {

            return sslContext.getSocketFactory().createSocket(socket, host, port, autoClose);

        }

  }

### 1.2.2. 网络交互是否使用过Webkit？

### 1.2.3. 如何动态加载dex？为何谷歌官方不建议使用反射？

**前言**

　　 在目前的软硬件环境下，Native App与Web App在用户体验上有着明显的优势，但在实际项目中有些会因为业务的频繁变更而频繁的升级客户端，造成较差的用户体验，而这也恰恰是Web App的优势。本文对网上Android动态加载jar的资料进行梳理和实践在这里与大家一起分享，试图改善频繁升级这一弊病。

**声明**

　　欢迎转载，但请保留文章原始出处:)

　　　　博客园：http://www.cnblogs.com

　　　　农民伯伯： http://over140.cnblogs.com

　　　　Android中文翻译组：<http://androidbox.sinaapp.com/>

**正文**

　　一、 基本概念和注意点

　　　　1.1　　首先需要了解一点：在Android中可以动态加载，但无法像Java中那样方便动态加载jar

　　　　　　原因：Android的虚拟机(Dalvik VM)是不认识Java打出jar的byte code，需要通过dx工具来优化转换成Dalvik byte code才行。这一点在咱们Android项目打包的apk中可以看出：引入其他Jar的内容都被打包进了classes.dex。

　　　　　　所以这条路不通，请大家注意。

　　　　1.2　　当前哪些API可用于动态加载

　　　　　　1.2.1　　DexClassLoader

　　　　　　　　这个可以加载jar/apk/dex，也可以从SD卡中加载，也是本文的重点。

　　　　　　1.2.3　　PathClassLoader

　　　　　　　　只能加载已经安装到Android系统中的apk文件。

　　二、 准备

　　　　本文主要参考"四、参考文章"中第一篇文章，补充细节和实践过程。

　　　　2.1　　下载开源项目

<http://code.google.com/p/goodev-demo>

　　　　　　将项目导入工程，工程报错的话应该是少了gen文件夹，手动添加即可。注意这个例子是从网上下载优化好的jar(已经优化成dex然后再打包成的jar)到本地文件系统，然后再从本地文件系统加载并调用的。本文则直接改成从SD卡加载。

　　三、实践

　　　　3.1　　编写接口和实现

　　　　　　3.1.1　　接口IDynamic

package com.dynamic;  
  
public interface IDynamic {  
    public String helloWorld();  
}

 　　　　　　3.1.2　　实现类DynamicTest

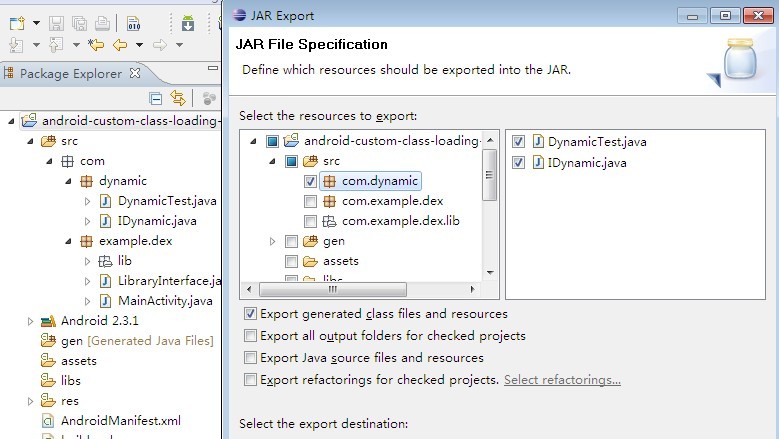
[复制代码](javascript:void(0);)

package com.dynamic;  
  
public class DynamicTest implements IDynamic {  
  
    @Override  
    public String helloWorld() {  
        return "Hello World!";  
    }  
}

[复制代码](javascript:void(0);)

　　　　3.2　　打包并转成dex

　　　　　　3.2.1　　选中工程，常规流程导出即可，如图：



　　　　　　注意：在实践中发现，自己新建一个Java工程然后导出jar是无法使用的，这一点大家可以根据文章一来了解相关原因，也是本文的重点之一。这里打包导出为dynamic.jar

　　　　　　（后期修复：打包请不要把接口文件打进来，参见文章末尾后续维护！）

　　　　　　3.2.2　　将打包好的jar拷贝到SDK安装目录android-sdk-windows\platform-tools下，DOS进入这个目录，执行命名：

dx --dex --output=test.jar dynamic.jar

　　　　3.3　　修改调用例子

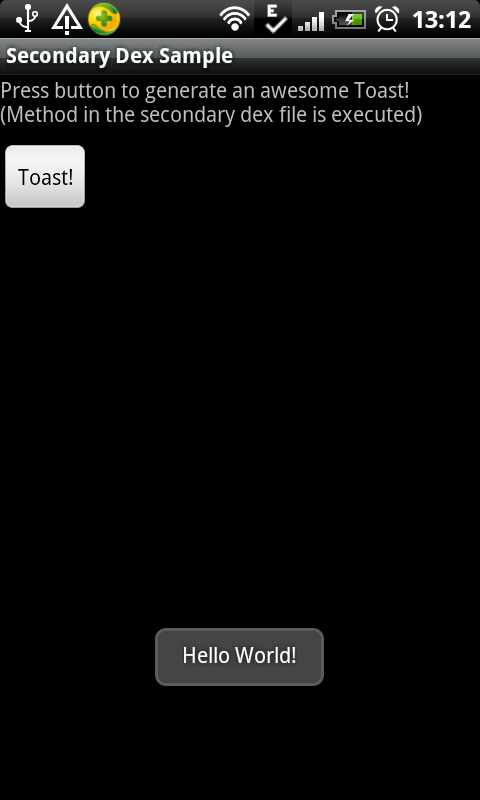
　　　　　　修改MainActivity，如下：

[复制代码](javascript:void(0);)

    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        mToastButton = (Button) findViewById(R.id.toast\_button);  
          
        // Before the secondary dex file can be processed by the DexClassLoader,  
        // it has to be first copied from asset resource to a storage location.  
//        final File dexInternalStoragePath = new File(getDir("dex", Context.MODE\_PRIVATE),SECONDARY\_DEX\_NAME);  
//        if (!dexInternalStoragePath.exists()) {  
//            mProgressDialog = ProgressDialog.show(this,  
//                    getResources().getString(R.string.diag\_title),   
//                    getResources().getString(R.string.diag\_message), true, false);  
//            // Perform the file copying in an AsyncTask.  
//            // 从网络下载需要的dex文件  
//            (new PrepareDexTask()).execute(dexInternalStoragePath);  
//        } else {  
//            mToastButton.setEnabled(true);  
//        }  
          
        mToastButton.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View view) {  
                // Internal storage where the DexClassLoader writes the optimized dex file to.  
                //final File optimizedDexOutputPath = getDir("outdex", Context.MODE\_PRIVATE);  
                final File optimizedDexOutputPath = new File(Environment.getExternalStorageDirectory().toString()  
                    + File.separator + "test.jar");  
                // Initialize the class loader with the secondary dex file.  
//                DexClassLoader cl = new DexClassLoader(dexInternalStoragePath.getAbsolutePath(),  
//                        optimizedDexOutputPath.getAbsolutePath(),  
//                        null,  
//                        getClassLoader());  
                DexClassLoader cl = new DexClassLoader(optimizedDexOutputPath.getAbsolutePath(),  
                    Environment.getExternalStorageDirectory().toString(), null, getClassLoader());  
                Class libProviderClazz = null;  
                  
                try {  
                    // Load the library class from the class loader.  
                    // 载入从网络上下载的类  
//                    libProviderClazz = cl.loadClass("com.example.dex.lib.LibraryProvider");  
                    libProviderClazz = cl.loadClass("com.dynamic.DynamicTest");  
                      
                    // Cast the return object to the library interface so that the  
                    // caller can directly invoke methods in the interface.  
                    // Alternatively, the caller can invoke methods through reflection,  
                    // which is more verbose and slow.  
                    //LibraryInterface lib = (LibraryInterface) libProviderClazz.newInstance();  
                    IDynamic lib = (IDynamic)libProviderClazz.newInstance();  
                      
                    // Display the toast!  
                    //lib.showAwesomeToast(view.getContext(), "hello 世界!");  
                    Toast.makeText(MainActivity.this, lib.helloWorld(), Toast.LENGTH\_SHORT).show();  
                } catch (Exception exception) {  
                    // Handle exception gracefully here.  
                    exception.printStackTrace();  
                }  
            }  
        });  
    }

[复制代码](javascript:void(0);)

　　　　3.4　　执行结果



　　四、参考文章

[[推荐]在Android中动态载入自定义类](http://yunfeng.sinaapp.com/?p=87)

[Android app中加载jar插件](http://www.cnblogs.com/franksunny/archive/2011/09/27/2193085.html)

[关于Android的ClassLoader探索](http://blog.csdn.net/czh0766/article/details/6736826)

[Android App 如何动态加载类](http://blog.csdn.net/quaful/article/details/6096951)

　　五、补充

　　　　大家可以看看DexClassLoader的API文档，里面不提倡从SD卡加载，不安全。此外，我也正在组织翻译组尽快把这个命名空间下的几个类都翻译出来，以供大家参考。

　　　　工程下载：[这里](http://files.cnblogs.com/over140/2011/11/android-custom-class-loading-goodev-demo.zip)，Dex文件下载：[这里](http://files.cnblogs.com/over140/2011/11/dynamic.zip)。大家可以直接把Dex文件拷贝到SD卡，然后运行例子。

　　六、后期维护

　　　　6.1　　2011-12-1　　修复本文错误

　　　　　　感谢网友ppp250和liuzhaocn的反馈，基本按照评论2来修改：

　　　　　　6.1.1　　不需要在本工程里面导出jar，自己新建一个Java工程然后导出来也行。

　　　　　　6.1.2　　导出jar时不能带接口文件，否则会报以下错：

 　　　　　　　　java.lang.IllegalAccessError: Class ref in pre-verified class resolved to unexpected implementation

　　　　　　6.1.3　　将jar优化时应该重新成jar(jar->dex->jar)，如果如下命令：

　　　　　　dx --dex --output=test.jar dynamic.jar

　　　　6.2　　2012-3-29　　本文升级版：

[Android应用开发提高系列（4）——Android动态加载（上）——加载未安装APK中的类](http://www.cnblogs.com/over140/archive/2012/03/29/2423116.html)

　　　　　　请大家参照最新的文章来做动态加载！

**结束**

　　除了翻译组的工作和自己本职的工作以外，很难抽时间出来分享一些开发心得，但正所谓挤挤总是有的，欢迎交流！

**探秘腾讯Android手机游戏平台之不安装游戏APK直接启动法**

2011年09月18日 ⁄ [移动开发](http://blog.zhourunsheng.com/category/%e8%bd%af%e4%bb%b6%e5%bc%80%e5%8f%91/%e7%a7%bb%e5%8a%a8%e5%bc%80%e5%8f%91/) ⁄ [暂无评论](http://blog.zhourunsheng.com/2011/09/%e6%8e%a2%e7%a7%98%e8%85%be%e8%ae%afandroid%e6%89%8b%e6%9c%ba%e6%b8%b8%e6%88%8f%e5%b9%b3%e5%8f%b0%e4%b9%8b%e4%b8%8d%e5%ae%89%e8%a3%85%e6%b8%b8%e6%88%8fapk%e7%9b%b4%e6%8e%a5%e5%90%af%e5%8a%a8%e6%b3%95/#comments) ⁄ 被围观 13,915+

相信这样一个问题，大家都不会陌生，

***“有什么的方法可以使Android的程序APK不用安装，而能够直接启动”。***

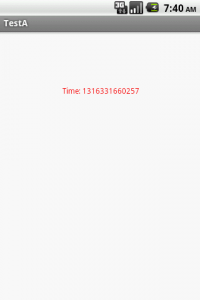
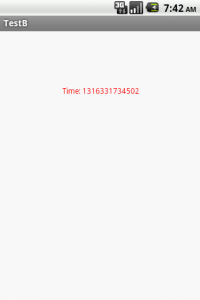
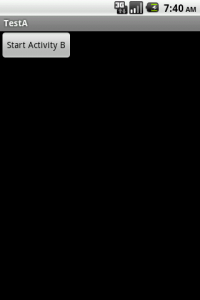
发现最后的结局都是不能实现这个美好的愿望，而腾讯Android手机游戏平台却又能实现这个功能，下载的连连看，五子棋都没有安装过程，但是都能直接运行，这其中到底有什么“玄机”呢，也有热心童鞋问过我这个问题，本文就为大家来揭开这个谜团。

重要说明

***在实践的过程中大家都会发现资源引用的问题，这里重点声明两点：  
1. 资源文件是不能直接inflate的，如果简单的话直接在程序中用代码书写。  
2. 资源文件是不能用R来引用的，因为上下文已经不同了，腾讯的做法是将资源文件打包(\*.pak文件和APK打包在一起)，虽然APK是没有进行安装，但是资源文件是另外解压到指定文件夹下面的，然后将文件夹的地址传给了第三方应用程序，这样第三方应用程序通过File的inputstream流还是可以读取和使用这些资源的。***

实践

我实现了一个小小的Demo，麻雀虽小五脏俱全，为了突出原理，我就尽量简化了程序，通过这个实例来让大家明白后台的工作原理。

1. 下载demo的apk程序[apks](http://carey.u.qiniudn.com/uploads/2011/09/apks.zip)，其中包括了两个apk，分别是A和B
2. 这两个APK可分别安装和运行，A程序界面只显示一个Button，B程序界面会动态显示当前的时间
3. 下面的三幅图片分别为直接启动运行A程序（安装TestA.apk），直接启动运行B程序（安装TestB.apk）和由A程序动态启动B程序（安装TestA.apk，TestB.apk不用安装，而是放在/mnt/sdcard/目录中，即 SD卡上）的截图，细心的同学可以停下来观察一下他们之间的不同  
   
4. 后两幅图片的不同，也即Title的不同，则解释出了我们将要分析的后台实现原理的机制

实现原理

最能讲明白道理的莫过于源码了，下面我们就来分析一下A和B的实现机制，首先来分析TestA.apk的主要代码实现：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | @Override  **public** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);          setContentView(R.layout.main);            Button btn = (Button) findViewById(R.id.btn);          btn.setOnClickListener(**new** OnClickListener() {                @Override  **public** **void** onClick(View v) {                  Bundle paramBundle = **new** Bundle();                  paramBundle.putBoolean("KEY\_START\_FROM\_OTHER\_ACTIVITY", **true**);                  String dexpath = "/mnt/sdcard/TestB.apk";                  String dexoutputpath = "/mnt/sdcard/";                  LoadAPK(paramBundle, dexpath, dexoutputpath);              }          });      } |

**代码解析**：这就是OnCreate函数要做的事情，装载view界面，绑定button事件，大家都熟悉了，还有就是设置程序B的放置路径，因为我程序中代码是从/mnt/sdcard/TestB.apk中动态加载，这也就是为什么要让大家把TestB.apk放在SD卡上面的原因了。关键的函数就是最后一个了LoadAPK，它来实现动态加载B程序。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | **public** **void** LoadAPK(Bundle paramBundle, String dexpath, String dexoutputpath) {          ClassLoader localClassLoader = ClassLoader.getSystemClassLoader();          DexClassLoader localDexClassLoader = **new** DexClassLoader(dexpath,                  dexoutputpath, **null**, localClassLoader);  **try** {              PackageInfo plocalObject = getPackageManager()                      .getPackageArchiveInfo(dexpath, 1);    **if** ((plocalObject.activities != **null**)                      && (plocalObject.activities.length > 0)) {                  String activityname = plocalObject.activities[0].name;                  Log.d(TAG, "activityname = " + activityname);    **Class** localClass = localDexClassLoader.loadClass(activityname);                  Constructor localConstructor = localClass                          .getConstructor(**new** **Class**[] {});                  Object instance = localConstructor.newInstance(**new** Object[] {});                  Log.d(TAG, "instance = " + instance);                    Method localMethodSetActivity = localClass.getDeclaredMethod(                          "setActivity", **new** **Class**[] { Activity.**class** });                  localMethodSetActivity.setAccessible(**true**);                  localMethodSetActivity.invoke(instance, **new** Object[] { **this** });                    Method methodonCreate = localClass.getDeclaredMethod(                          "onCreate", **new** **Class**[] { Bundle.**class** });                  methodonCreate.setAccessible(**true**);                  methodonCreate.invoke(instance, **new** Object[] { paramBundle });              }  **return**;          } **catch** (Exception ex) {              ex.printStackTrace();          }      } |

**代码解析**：这个函数要做的工作如下：加载B程序的APK文件，通过类加载器DexClassLoader来解析APK文件，这样会在SD卡上面生成一个同名的后缀为dex的文件，例如/mnt/sdcard/TestB.apk==>/mnt/sdcard/TestB.dex,接下来就是通过java反射机制，动态实例化B中的Activity对象，并依次调用了其中的两个函数，分别为setActivity和onCreate.看到这里，大家是不是觉得有点奇怪，Activity的启动函数是onCreate，为什么要先调用setActivity，而更奇怪的是setActivity并不是系统的函数，确实，那是我们自定义的，这也就是核心的地方。

好了带着这些疑问，我们再来分析B程序的主代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **public** **class** TestBActivity **extends** Activity {  **private** **static** **final** String TAG = "TestBActivity";  **private** Activity otherActivity;        @Override  **public** **void** onCreate(Bundle savedInstanceState) {  **boolean** b = **false**;  **if** (savedInstanceState != **null**) {              b = savedInstanceState.getBoolean("KEY\_START\_FROM\_OTHER\_ACTIVITY", **false**);  **if** (b) {  **this**.otherActivity.setContentView(**new** TBSurfaceView(  **this**.otherActivity));              }          }  **if** (!b) {  **super**.onCreate(savedInstanceState);  *// setContentView(R.layout.main);*              setContentView(**new** TBSurfaceView(**this**));          }      }    **public** **void** setActivity(Activity paramActivity) {          Log.d(TAG, "setActivity..." + paramActivity);  **this**.otherActivity = paramActivity;      }  } |

**代码解析**：看完程序B的实现机制，大家是不是有种恍然大悟的感觉，这根本就是“偷梁换柱”嘛，是滴，程序B动态借用了程序A的上下文执行环境，这也就是上面后两幅图的差异，最后一幅图运行的是B的程序，但是title表示的却是A的信息，而没有重新初始化自己的，实际上这也是不可能的，所以有些童鞋虽然通过java的反射机制，正确呼叫了被调程序的onCreate函数，但是期望的结果还是没有出现，原因就是这个上下文环境没有正确建立起来，但是若通过startActivity的方式来启动APK的话，android系统会替你建立正确的执行时环境，所以就没问题。至于那个TBSurfaceView，那就是自定义的一个view画面，动态画当前的时间

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | **public** **class** TBSurfaceView **extends** SurfaceView **implements** Callback, Runnable {  **private** SurfaceHolder sfh;  **private** Thread th;  **private** Canvas canvas;  **private** Paint paint;    **public** TBSurfaceView(Context context) {  **super**(context);          th = **new** Thread(**this**);          sfh = **this**.getHolder();          sfh.addCallback(**this**);          paint = **new** Paint();          paint.setAntiAlias(**true**);          paint.setColor(Color.RED);  **this**.setKeepScreenOn(**true**);      }    **public** **void** surfaceCreated(SurfaceHolder holder) {          th.start();      }    **private** **void** draw() {  **try** {              canvas = sfh.lockCanvas();  **if** (canvas != **null**) {                  canvas.drawColor(Color.WHITE);                  canvas.drawText("Time: " + System.currentTimeMillis(), 100,                          100, paint);              }          } **catch** (Exception ex) {              ex.printStackTrace();          } **finally** {  **if** (canvas != **null**) {                  sfh.unlockCanvasAndPost(canvas);              }          }      }    **public** **void** run() {  **while** (**true**) {              draw();  **try** {                  Thread.sleep(100);              } **catch** (InterruptedException e) {                  e.printStackTrace();              }          }      }    **public** **void** surfaceChanged(SurfaceHolder holder, **int** format, **int** width,  **int** height) {      }    **public** **void** surfaceDestroyed(SurfaceHolder holder) {      }  } |

腾讯游戏平台解析

说了这么多，都是背景，O(∩\_∩)O哈哈~

其实腾讯游戏平台就是这么个实现原理，我也是通过它才学习到这种方式的，还得好好感谢感谢呢。

腾讯Android游戏平台的游戏分成两类，第一类是腾讯自主研发的，像斗地主，五子棋，连连看什么的，所以实现机制就如上面的所示，A代表游戏大厅，B代表斗地主类的小游戏。第二类是第三方软件公司开发的，可就不能已这种方式来运作了，毕竟腾讯不能限制别人开发代码的方式啊，所以腾讯就开放了一个sdk包出来，让第三方应用可以和游戏大厅相结合，具体可参见[QQ游戏中心开发者平台](http://dev.g.qq.com/res_sdk.html)，但这同时就损失了一个优点，那就是第三方开发的游戏要通过安装的方式才能运行。

结论

看到这里，相信大家都比较熟悉这个背后的原理了吧，也希望大家能提供更好的反馈信息！

程序源码下载[source](http://carey.u.qiniudn.com/uploads/2011/09/source.zip)

### 1.2.4. 是否做过Framework层的开发？

### 1.2.5. APK升级如何解耦？APK如何拆分？

### 1.2.6. 用过哪些加密算法？

### 1.2.7. 是否了解git，如何使用git？

### 1.2.8. svn版本管理？

### 1.2.9. 如何发布apk？apk发布版本的管理？

### 1.2.10. ndk了解到什么程度？是否直接写过lib？

### 1.2.11. jni的映射原理？如何从java调用到c？

### 1.2.12. 你的团队的架构？

### 1.2.13. 你在原来的团队管理中，你个人对技术和管理投入的比例？

### 1.2.14. 你的团队有多少个骨干？如何安排骨干的工作？如何解决技术难题？

### 1.2.15. 你的团队什么情况下压力大？如何衡量压力？

### 1.2.16. 怎么让新人快速融入团队，达到开发的要求？

## 2. 技术面试题B

## 2.1. 基本面试题

### 2.1.1. 对自己所做的一个项目进行介绍

### 2.1.2. 项目中有哪些难点？

### 2.1.3. UI设计和优化如何去做？

### 2.1.4. 本地数据和服务器数据如何进行同步？

### 2.1.5. ANR？

### 2.1.6. 多线程

### 2.1.7. 设计模式

### 2.1.8. 二维码如何实现？

### 2.1.9. 简要说明下framework层的开发和自己的理解

### 2.1.10. 如何自定义控件，举例说明。

### 2.1.11. 设计文档如何编写？

### 2.1.12你所处的团队如何？（分别有多少人开发、测试等）