Android知识点

1. socket 和 http tcp/ip协议区别
2. 对socket通信 和 http协议、tcp/ip的理解和使用
3. 说下android下的多线程
4. Json 和xml的解析数据的使用
5. Android os 开发 框架原理 性能优化、内存优化
6. Android UI 优化
7. Android常用的数据库开发sqlite
8. Android 保存数据的方式
9. Android动画类型
10. Android自定义view的流程
11. Android网络编程即socket和http tcp/ip 、json 和 xml的使用
12. 2d绘图 openGL
13. 多版本控制、多尺寸处理
14. webView如何使用javascript和js
15. 无线网络和多媒体开发
16. 怎样控制好app使用流量多的问题
17. 怎样使app和系统通讯录同步
18. 对3G、wifi技术的理解
19. Xmpp的使用经验
20. Android 界面开发，熟悉屏幕分辨率兼容和操作系统版本兼容原理
21. Android进程间的通信
22. 简述android的开发流程
23. 了解的开源项目
24. NDK 、JNI、和aidl的使用
25. 了解那些消息推送、用过那些第三方消息推送
26. 谈下 Androidx下的文件读取和权限设置
27. Intent 和 intentfilter的区别
28. Handler的原理
29. [Android中的Handler, Looper, MessageQueue和Thread](http://www.cnblogs.com/xirihanlin/archive/2011/04/11/2012746.html)
30. Android后台运行线程

9.Android动画类型

ValueAnimator、AnimatorSet、ObjectAnimator、ViewPropertyAnimator、

AnimatorInflater、LayoutTransition

1. **socket 和 http tcp/ip协议区别**

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。通过初步的了解，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，三者从本质上来说没有可比性，**socket则是对TCP/IP协议的封装和应用**（程序员层面上）。也可以说，**TPC/IP协议是传输层协议，主要解决数据如何在网络中传输，而HTTP是应用层协议，主要解决如何包装数据。关于TCP/IP和HTTP协议的关系，**网络有一段比较容易理解的介绍：

  “我们在传输数据时，可以只使用（传输层）TCP/IP协议，但是那样的话，如果没有应用层，便无法识别数据内容，如果想要使传输的数据有意义，则必须使用到应用层协议，应用层协议有很多，比如HTTP、FTP、TELNET等，也可以自己定义应用层协议。**WEB使用HTTP协议作应用层协议，以封装HTTP文本信息，然后使用TCP/IP做传输层协议将它发到网络上。**

而我们平时说的最多的socket是什么呢，**实际上socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口（API），通过Socket，我们才能使用TCP/IP协议**。实际上，Socket跟TCP/IP协议没有必然的联系。Socket编程接口在设计的时候，就希望也能适应其他的网络协议。所以说，**Socket的出现只是使得程序员更方便地使用TCP/IP协议栈而已，是对TCP/IP协议的抽象，从而形成了我们知道的一些最基本的函数接口，比如create、listen、connect、accept、send、read和write等等”**

**http协议：应用层协议，并且http协议是基于tcp连接的，主要解决的是如何包装协议的**

**tcp协议：运输层协议，通常也叫做tcp/ip协议，主要解决数据如何在网络中传输**

**udp协议：运输层协议，用户数据报协议，不可靠的协议，只负责把应用层的协议的数据传送到ip层的数据报，而不管数据是否到达。**

**ip协议：网络层。**

**socket连接：长连接**

**http连接：短连接**

**实际上，传输层的TCP是基于网络层的IP协议的，而应用层的HTTP协议又是基于传输层的TCP协议的，而Socket本身不算是协议，就像上面所说，它只是提供了一个针对TCP或者UDP编程的接口**。

下面是一些经常在笔试或者面试中碰到的重要的概念，特在此做摘抄和总结。

**一。什么是TCP连接的三次握手**

第一次握手：客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认；  
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状态，完成三次握手。  
  握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连接之前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP连接的请求，断开过程需要经过“四次握手”（过程就不细写了，就是服务器和客户端交互，最终确定断开）

**二。利用Socket建立网络连接的步骤**

建立Socket连接至少需要一对套接字，其中一个运行于客户端，称为ClientSocket ，另一个运行于服务器端，称为ServerSocket 。

套接字之间的连接过程分为三个步骤：服务器监听，客户端请求，连接确认。

1。服务器监听：服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态，等待客户端的连接请求。

2。客户端请求：指客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。

3。连接确认：当服务器端套接字监听到或者说接收到客户端套接字的连接请求时，就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端确认了此描述，双方就正式建立连接。而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。

**三。HTTP链接的特点**

HTTP协议即超文本传送协议(Hypertext Transfer Protocol )，是Web联网的基础，也是手机联网常用的协议之一，HTTP协议是建立在TCP协议之上的一种应用。

HTTP连接最显著的特点是客户端发送的每次请求都需要服务器回送响应，在请求结束后，会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。

**四。TCP和UDP的区别**（考得最多。。快被考烂了我觉得- -//）

1。TCP是面向链接的，虽然说网络的不安全不稳定特性决定了多少次握手都不能保证连接的可靠性，但TCP的三次握手在最低限度上（实际上也很大程度上保证了）保证了连接的可靠性；而UDP不是面向连接的，UDP传送数据前并不与对方建立连接，对接收到的数据也不发送确认信号，发送端不知道数据是否会正确接收，当然也不用重发，所以说UDP是无连接的、不可靠的一种数据传输协议。

2。也正由于1所说的特点，使得UDP的开销更小数据传输速率更高，因为不必进行收发数据的确认，所以UDP的实时性更好。

知道了TCP和UDP的区别，就不难理解为何采用TCP传输协议的MSN比采用UDP的QQ传输文件慢了，但并不能说QQ的通信是不安全的，因为程序员可以手动对UDP的数据收发进行验证，比如发送方对每个数据包进行编号然后由接收方进行验证啊什么的，即使是这样，UDP因为在底层协议的封装上没有采用类似TCP的“三次握手”而实现了TCP所无法达到的传输效率。

**Socket 和 HTTP TCP/IP UDP之间的联系**

网络七层由下往上分别为物理层、数据链路层、网络层、传输层、会话层、表示层和应用层，一般编程人员接触最多的就是应用层和运输层，再往下的就是所谓的媒体层了，不是我们研究的对象。

下面是应用层、运输层，网络层、链路层通信协议概图。我们经常接触到的一般是：

**http协议：应用层协议，并且http协议是基于tcp连接的，主要解决的是如何包装协议的**

**tcp协议：运输层协议，通常也叫做tcp/ip协议，主要解决数据如何在网络中传输**

**udp协议：运输层协议，用户数据报协议，不可靠的协议，只负责把应用层的协议的数据传送到ip层的数据报，而不管数据是否到达。**

**ip协议：网络层。**

**socket连接：长连接**

**http连接：短连接**



下面详细介绍网络通信中的几个概念与原理：

**（1）Socket(套接字)**:

套接字（socket）是通信的基石，是支持TCP/IP协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：连接使用的协议，本地主机的IP地址，本地进程的协议端口，远地主机的IP地址，远地进程的协议端口。

应用层通过传输层进行数据通信时，TCP会遇到同时为多个应用程序进程提供并发服务的问题。多个TCP连接或多个应用程序进程可能需要通过同一个 TCP协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与TCP／IP协议交互提供了套接字(Socket)接口。应用层可以和传输层通过Socket接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

ps：说白了，socket就是一组调用接口（API），封装了做tcp/ip开发的网络接口，通过Socket，我们才能使用TCP/IP协议。

**（2）Tcp连接**：

第一次握手：客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认；   
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状态，完成三次握手。      
握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭 连接之前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP连接的请求，断开过程需要经过“四次握手”。

**（3）Socket连接**:

由于通常情况下Socket连接就是TCP连接，因此Socket连接一旦建立，通信双方即可开始相互发送数据内容，直到双方连接断开。但在实际网络应用中，客户端到服务器之间的通信往往需要穿越多个中间节点，例如路由器、网关、防火墙等，大部分防火墙默认会关闭长时间处于非活跃状态的连接而导致 Socket 连接断连，因此需要通过轮询告诉网络，该连接处于活跃状态。

建立Socket连接至少需要一对套接字，其中一个运行于客户端，称为ClientSocket ，另一个运行于服务器端，称为ServerSocket 。

套接字之间的连接过程分为三个步骤：服务器监听，客户端请求，连接确认。

1。服务器监听：服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态，等待客户端的连接请求。

2。客户端请求：指客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。

3。连接确认：当服 务器端套接字监听到或者说接收到客户端套接字的连接请求时，就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端 确认了此描述，双方就正式建立连接。而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。

**（4）Http连接**：

HTTP协议是建立在TCP协议之上的一种应用，**HTTP连接使用的是“请求—响应”的方式，不仅在请求时需要先建立连接，而且需要客户端向服务器发出请求后，服务器端才能回复数据。在请求结束后，会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。由于HTTP在每次请求结束后都会主动释放连接，因此HTTP连接是一种“短连接”**，要保持客户端程序的在线状态，需要不断地向服务器发起连接请求。通常的做法是即时不需要获得任何数据，客户端也保持每隔一段固定的时间向服务器发送一次“保持连接”的请求，服务器在收到该请求后对客户端进行回复，表明知道客户端“在线”。若服务器长时间无法收到客户端的请求，则认为客户端“下线”，若客户端长时间无法收到服务器的回复，则认为网络已经断开。

**（5）TCP/IP协议和Http协议之间的关系**：

**TPC/IP协议是传输层协议，主要解决数据 如何在网络中传输，而HTTP是应用层协议，主要解决如何包装数据，而socket则是对TCP/IP协议的封装和应用（程序员层面上）。实际上http协议就是建立在tcp/ip协议之上的。关于TCP/IP和HTTP协议的关系：**

“我们在传输数据时，可以只使用（传输层）TCP/IP协议，但是那样的话，如 果没有应用层，便无法识别数据内容，如果想要使传输的数据有意义，则必须使用到应用层协议，应用层协议有很多，比如HTTP、FTP、TELNET等，也 可以自己定义应用层协议。WEB使用HTTP协议作应用层协议，以封装HTTP文本信息，然后使用TCP/IP做传输层协议将它发到网络上。”

**（6）Socket和TCP/IP协议之间的关系**：

socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口（API），通过Socket，我们才能使用TCP/IP协议。 实际上，Socket跟TCP/IP协议没有必然的联系。Socket编程接口在设计的时候，就希望也能适应其他的网络协议。所以说，Socket的出现 只是使得程序员更方便地使用TCP/IP协议栈而已，是对TCP/IP协议的抽象，从而形成了我们知道的一些最基本的函数接口，比如create、 listen、connect、accept、send、read和write等等。网络有一段关于socket和TCP/IP协议关系的说法比较容易理 解：

“TCP/IP只是一个协议栈，就像操作系统的运行机制一样，必须要具体实现，同时还要提供对外的操作接口。这个就像操作系统会提供标准的编程接口，比如win32编程接口一样，TCP/IP也要提供可供程序员做网络开发所用的接口，这就是Socket编程接口。”

**（7）tcp/ip协议, http协议，socket三者之间的关系**：

实际上，传输层的TCP是基于网络层的IP协议的，而应用层的HTTP协议又是基于传输层的TCP协议的，而Socket本身不算是协议，它只是提供了一个针对TCP或者UDP编程的接口。

**（8）tcp协议和udp协议之间的区别**：

TCP --- 传输控制协议,提供的是面向连接、可靠的字节流服务。当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。 理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连接前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP连接的请求

UDP --- 用户数据报协议，是一个无连接的简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快

TCP发送的包有序号，对方收到包后要给一个反馈，如果超过一定时间还没收到反馈就自动执行超时重发，因此**TCP最大的优点是可靠**。一般网页（http）、邮件（SMTP)、远程连接(Telnet)、文件(FTP)传送就用TCP

UDP是面向消息的协议，通信时不需要建立连接，数据的传输自然是不可靠的，**UDP一般用于多点通信和实时的数据业务**，比如语音广播、视频、QQ、TFTP(简单文件传送）、SNMP（简单网络管理协议）、RTP（实时传送协议）RIP（路由信息协议，如报告股票市场，航空信息）、DNS(域名解释）。注重速度流畅。

**（9）FTP协议**：

文件传输协议（File Transfer Protocol, FTP）是[TCP/IP](http://baike.baidu.com/view/7729.htm)网络上两台计算机传送文件的协议，FTP是在TCP/IP网络和INTERNET上最早使用的协议之一，它属于网络协议组的[应用层](http://baike.baidu.com/view/239619.htm)。FTP客户机可以给[服务器](http://baike.baidu.com/view/899.htm)发出命令来下载文件，上载文件，创建或改变服务器上的目录。

**android的网络编程分为2种：基于socket的，和基于http协议的。**

**一、基于socket的用法**

　　服务器端：

　　先启动一个服务器端的socket     ServerSocket svr = new ServerSocket(8989);

　　开始侦听请求　　　　　　　　　Socket s = svr.accept();

　　取得输入和输出　　　　　　　　DataInputStream dis = new DataInputStream(s.getInputStream());

　　　　　　　　　　　　　　　　　DataOutputStream dos = new DataOutputStream(s.getOutputStream());

　　Socket 的交互通过流来完成，即是说传送的字节流，因此任何文件都可以在上面传送。谁打开的记得要关上。

　　用DataInputStream/DataOutputStream来进行包装是因为我们想要他们对基本数据类型的读写功能readInt(),writeInt(),readUTF(),writeUTF()等等。

　　客户端：

　　发起一个socket连接　　　　　　Socket s = new Socket("192.168.1.200",8989);

　　取得输入和输出　　　　　　　　DataInputStream dis = new DataInputStream(s.getInputStream());

　　　　　　　　　　　　　　　　　DataOutputStream dos = new DataOutputStream(s.getOutputStream());

　　之后就可以相互通信了。谁打开的记得要关上。

**二、基于Http协议**

Android中提供的HttpURLConnection和HttpClient接口可以用来开发HTTP程序。以下是本人在学习中的总结与归纳。

**1. HttpURLConnection接口**

      首先需要明确的是，Http通信中的POST和GET请求方式的不同。GET可以获得静态页面，也可以把参数放在URL字符串后面，传递给服务器。而POST方法的参数是放在Http请求中。因此，在编程之前，应当首先明确使用的请求方法，然后再根据所使用的方式选择相应的编程方式。

      HttpURLConnection是继承于URLConnection类，二者都是抽象类。其对象主要通过URL的openConnection方法获得。创建方法如下代码所示：

Java代码

URL url = new URL("http://www.51cto.com/index.jsp?par=123456");  

**HttpURLConnection urlConn=(HttpURLConnection)url.openConnection();**

**通过以下方法可以对请求的属性进行一些设置，如下所示**

**Java代码**

**//设置输入和输出流**

**urlConn.setDoOutput(true);**

**urlConn.setDoInput(true);**

**//设置请求方式为POST**

**urlConn.setRequestMethod("POST");**

**//POST请求不能使用缓存**

**urlConn.setUseCaches(false);**

**//关闭连接**

**urlConn.disConnection();**

Manifest文件中权限的设定：

Xml代码

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

**HttpURLConnection默认使用GET方式，例如下面代码所示：**

Java代码

//以Get方式上传参数

public class Activity03 extends Activity

{

 private final String DEBUG\_TAG = "Activity03";

 

 @Override

 public void onCreate(Bundle savedInstanceState)

 {

   super.onCreate(savedInstanceState);

   setContentView(R.layout.http);  

   TextView mTextView = (TextView)this.findViewById(R.id.TextView\_HTTP);

   //http地址"?par=abcdefg"是我们上传的参数

   String httpUrl = "http://192.168.1.110:8080/httpget.jsp?par=abcdefg";

   //获得的数据

   String resultData = "";

   URL url = null;

   try

   {

     //构造一个URL对象

     url = new URL(httpUrl);

   }

   catch (MalformedURLException e)

   {

     Log.e(DEBUG\_TAG, "MalformedURLException");

   }

   if (url != null)

   {

     try

     {

**// 使用HttpURLConnection打开连接**

**HttpURLConnection urlConn = (HttpURLConnection) url.openConnection();**

       //得到读取的内容(流)

       InputStreamReader in = new InputStreamReader(urlConn.getInputStream());

       // 为输出创建BufferedReader

BufferedReader buffer = new BufferedReader(in);

       String inputLine = null;

       //使用循环来读取获得的数据

       while (((inputLine = buffer.readLine()) != null))

       {

         //我们在每一行后面加上一个"\n"来换行

         resultData += inputLine + "\n";

       }     

       //关闭InputStreamReader

       in.close();

       //关闭http连接

       urlConn.disconnect();

       //设置显示取得的内容

       if ( resultData != null )

       {

         mTextView.setText(resultData);

       }

       else

       {

         mTextView.setText("读取的内容为NULL");

       }

     }

     catch (IOException e)

     {

       Log.e(DEBUG\_TAG, "IOException");

     }

   }

   else

   {

     Log.e(DEBUG\_TAG, "Url NULL");

   }

}

**如果需要使用POST方式，则需要setRequestMethod设置。代码如下：**

**Java代码**

//以post方式上传参数

public class Activity04  extends Activity

{

 private final String DEBUG\_TAG = "Activity04";

 

 @Override

 public void onCreate(Bundle savedInstanceState)

 {

   super.onCreate(savedInstanceState);

   setContentView(R.layout.http);

   

   TextView mTextView = (TextView)this.findViewById(R.id.TextView\_HTTP);

   //http地址"?par=abcdefg"是我们上传的参数

   String httpUrl = "http://192.168.1.110:8080/httpget.jsp";

   //获得的数据

   String resultData = "";

   URL url = null;

   try

   {

     //构造一个URL对象

     url = new URL(httpUrl);

   }

   catch (MalformedURLException e)

   {

     Log.e(DEBUG\_TAG, "MalformedURLException");

   }

   if (url != null)

   {

     try

**{**

**// 使用HttpURLConnection打开连接**

**HttpURLConnection urlConn = (HttpURLConnection) url.openConnection();**

**//因为这个是post请求,设立需要设置为true**

**urlConn.setDoOutput(true);**

**urlConn.setDoInput(true);**

**// 设置以POST方式**

**urlConn.setRequestMethod("POST");**

**// Post 请求不能使用缓存**

**urlConn.setUseCaches(false);**

**urlConn.setInstanceFollowRedirects(tr**ue);

       // 配置本次连接的Content-type，配置为application/x-www-form-urlencoded的

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

       // 连接，从postUrl.openConnection()至此的配置必须要在connect之前完成，

       // 要注意的是connection.getOutputStream会隐含的进行connect。

**urlConn.connect();**

**//DataOutputStream流**

**DataOutputStream out = new DataOutputStream(urlConn.getOutputStream());**

**//要上传的参数**

**String content = "par=" + URLEncoder.encode("ABCDEFG", "gb2312");**

**//将要上传的内容写入流中**

**out.writeBytes(content);**

       //刷新、关闭

       out.flush();

       out.close();

       //获取数据

       BufferedReader reader = new BufferedReader(new InputStreamReader(urlConn.getInputStream()));

       String inputLine = null;

       //使用循环来读取获得的数据

       while (((inputLine = reader.readLine()) != null))

       {

         //我们在每一行后面加上一个"\n"来换行

         resultData += inputLine + "\n";

       }     

       reader.close();

       //关闭http连接

       urlConn.disconnect();

       //设置显示取得的内容

       if ( resultData != null )

       {

         mTextView.setText(resultData);

       }

       else

       {

         mTextView.setText("读取的内容为NULL");

       }

     }

     catch (IOException e)

     {

       Log.e(DEBUG\_TAG, "IOException");

     }

   }

   else

   {

     Log.e(DEBUG\_TAG, "Url NULL");

   }

 }

}

**2. HttpClient接口**

 使用Apache提供的HttpClient接口同样可以进行HTTP操作。

 对于GET和POST请求方法的操作有所不同。GET方法的操作代码示例如下：

Java代码

public class Activity02 extends Activity

{

 

 @Override

 public void onCreate(Bundle savedInstanceState)

 {

   super.onCreate(savedInstanceState);

   setContentView(R.layout.http);

   TextView mTextView = (TextView) this.findViewById(R.id.TextView\_HTTP);

   // http地址

   String httpUrl = "**http://192.168.1.110:8080/httpget.jsp?par=HttpClient\_android\_Get";**

   //HttpGet连接对象

   HttpGet httpRequest = new HttpGet(httpUrl);

   try

   {

     //取得HttpClient对象

     **HttpClient httpclient = new DefaultHttpClient();**

     //请求HttpClient，取得HttpResponse

     **HttpResponse httpResponse = httpclient.execute(httpRequest);**

     //请求成功

     if (httpResponse.getStatusLine().getStatusCode() == HttpStatus.SC\_OK)

     {

       //取得返回的字符串

       String strResult = EntityUtils.toString(httpResponse.getEntity());

       mTextView.setText(strResult);

     }

     else

     {

       mTextView.setText("请求错误!");

     }

   }

   catch (ClientProtocolException e)

   {

     mTextView.setText(e.getMessage().toString());

   }

   catch (IOException e)

   {

     mTextView.setText(e.getMessage().toString());

   }

   catch (Exception e)

   {

     mTextView.setText(e.getMessage().toString());

   }  

     }

}

**使用POST方法进行参数传递时，需要使用NameValuePair来保存要传递的参数，另外，还需要设置所使用的字符集。代码如下所示：**

Java代码

public class Activity03 extends Activity

{

 

 @Override

 public void onCreate(Bundle savedInstanceState)

 {

   super.onCreate(savedInstanceState);

   setContentView(R.layout.http);

   TextView mTextView = (TextView) this.findViewById(R.id.TextView\_HTTP);

   // http地址

   String httpUrl = "http://192.168.1.110:8080/httpget.jsp";

   //HttpPost连接对象

   HttpPost httpRequest = new HttpPost(httpUrl);

   //使用NameValuePair来保存要传递的Post参数

   **List<NameValuePair> params = new ArrayList<NameValuePair>();**

   //添加要传递的参数

   params.add(new BasicNameValuePair("par", "HttpClient\_android\_Post"));

   try

   {

     //设置字符集

     HttpEntity httpentity = new UrlEncodedFormEntity(params, "gb2312");

     //请求httpRequest

     httpRequest.setEntity(httpentity);

     //取得默认的HttpClient

     HttpClient httpclient = new DefaultHttpClient();

     //取得HttpResponse

     HttpResponse httpResponse = httpclient.execute(httpRequest);

     //HttpStatus.SC\_OK表示连接成功

     if (httpResponse.getStatusLine().getStatusCode() == HttpStatus.SC\_OK)

     {

       //取得返回的字符串

       String strResult = EntityUtils.toString(httpResponse.getEntity());

       mTextView.setText(strResult);

     }

     else

     {

       mTextView.setText("请求错误!");

     }

   }

   catch (ClientProtocolException e)

   {

     mTextView.setText(e.getMessage().toString());

   }

   catch (IOException e)

   {

     mTextView.setText(e.getMessage().toString());

   }

   catch (Exception e)

   {

     mTextView.setText(e.getMessage().toString());

   }  

 }

}

     HttpClient实际上是对Java提供方法的一些封装，在HttpURLConnection中的输入输出流操作，在这个接口中被统一封装成了HttpPost(HttpGet)和HttpResponse，这样，就减少了操作的繁琐性。

    另外，在使用POST方式进行传输时，需要进行字符编码。

**2.说下android下的多线程**

**在Android应用中为什么要用多线程?为了解决哪些问题？或者为了实现哪些功能？有哪些好处？**

1. 提高用户体验或者避免ANR  
   在事件处理代码中需要使用多线程，否则会出现ANR(Application is not responding)

b) 异步  
应用中有些情况下并不一定需要同步阻塞去等待返回结果，可以通过多线程来实现异步，例如：上一点中提到的，你的应用中的某个Activity需要从云端获取一些图片，加载图片比较耗时，这时需要使用异步加载，加载完成一个图片刷新一个

c) 多任务  
例如多线程下载。

**Main线程通讯的方法：Handler的使用。**  
  
2) 使用Handler  
这里需要了解Android SDK提供的几个线程间通讯的类。  
2.1 **Handler**  
**Handler在android里负责发送和处理消息，通过它可以实现其他线程与Main线程之间的消息通讯。**

2.2 **Looper**  
Looper负责管理线程的消息队列和消息循环

Looper扮演着一个Handler和消息队列之间通讯桥梁的角色。程序组件首先通过Handler把消息传递给Looper，Looper把消息放入队列。Looper也把消息队列里的消息广播给所有的Handler，Handler接受到消息后调用handleMessage进行处理。  
2.3 **Message**  
Message是线程间通讯的消息载体。两个码头之间运输货物，Message充当集装箱的功能，里面可以存放任何你想要传递的消息。  
2.4 **MessageQueue**  
MessageQueue是消息队列，先进先出，它的作用是保存有待线程处理的消息。

Message Queue是一个消息队列，用来存放通过Handler发布的消息

**它们四者之间的关系是，在其他线程中调用Handler.sendMsg()方法(参数是Message对象)，将需要Main线程处理的事件添加到Main线程的MessageQueue中，Main线程通过MainLooper从消息队列中取出Handler发过来的这个消息时，会回调Handler的handlerMessage()方法。**

**1）android系统是单线程系统，为了实现多线程的效果，采用了message queue。**

**2）为实现多线程，可采用runOnUiThread，post，handle，AsyncTask技术实现**

**二、多线程的实现**

有以下几种方式：    
  
1)Activity.runOnUiThread(Runnable)    
  
2)View.post(Runnable) ;View.postDelay(Runnable , long)    
  
3)Handler    
  
4)AsyncTask

 对于第一种方法：

activity.runOnUiThread(new Runnable() {

@Override

public void run() {

// TODO Auto-generated method stub

int i = 30;

textView.setText(""+i+" s");

}

}

});

　　Handler主要有两个作用：

* 在工作线程中发送消息。
* 在UI线程中获取、处理消息。

上面介绍到Handler可以把一个Message对象或者Runnable对象压入到消息队列中，进而在UI线程中获取Message或者执行Runnable对象，所以Handler把压入消息队列有两大体系，Post和sendMessage：

* Post：Post允许把一个Runnable对象入队到消息队列中。它的方法有：post(Runnable)、postAtTime(Runnable,long)、postDelayed(Runnable,long)。
* sendMessage：sendMessage允许把一个包含消息数据的Message对象压入到消息队列中。它的方法 有：sendEmptyMessage(int)、sendMessage(Message)、 sendMessageAtTime(Message,long)、sendMessageDelayed(Message,long)。

　　从上面的各种方法可以看出，不管是post还是sendMessage都具有多种方法，它们可以设定Runnable对象和Message对象被入队到消息队列中，是立即执行还是延迟执行。

Demo ----------------------------------

**package** com.example.multithreaddemo;

**import** java.net.URL;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**import** android.app.Activity;

**import** android.content.Intent;

**import** android.graphics.drawable.Drawable;

**import** android.os.Bundle;

**import** android.os.Handler;

**import** android.os.SystemClock;

**import** android.view.View;

**import** android.view.View.OnClickListener;

**import** android.widget.Button;

**import** android.widget.ImageView;

**public** **class** MainActivity **extends** Activity {

**private** Handler handler = **new** Handler();

**private** ExecutorService executorService = Executors.*newFixedThreadPool*(5);

**private** Button button;

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

loadImage3("http://www.baidu.com/img/baidu\_logo.gif", R.id.*imageView1*);

loadImage3("http://www.chinatelecom.com.cn/images/logo\_new.gif",

R.id.*imageView2*);

loadImage3("http://cache.soso.com/30d/img/web/logo.gif",

R.id.*imageView3*);

loadImage3("http://csdnimg.cn/www/images/csdnindex\_logo.gif",

R.id.*imageView4*);

loadImage3("http://images.cnblogs.com/logo\_small.gif", R.id.*imageView5*);

button = (Button) findViewById(R.id.*imageView6*);

button.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View v) {

People people = **new** People();

people.setNameString("ABD");

Intent intent = **new** Intent(getApplicationContext(),

MainActivity2.**class**);

Bundle bundle = **new** Bundle();

bundle.putSerializable("test", people);

intent.putExtras(bundle);

startActivity(intent);

}

});

}

// 引入线程池来管理多线程

**private** **void** loadImage3(**final** String url, **final** **int** id) {

executorService.submit(**new** Runnable() {

**public** **void** run() {

**try** {

**final** Drawable drawable = Drawable.*createFromStream*(

**new** URL(url).openStream(), "image.png");

// 模拟网络延时

SystemClock.*sleep*(2000);

handler.post(**new** Runnable() {

**public** **void** run() {

((ImageView) MainActivity.**this**.findViewById(id))

.setImageDrawable(drawable);

}

});

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

}

});

}

}

**//Handler+ExecutorService(线程池)+MessageQueue+缓存模式**

//下面比起前一个做了几个改造:

//把整个代码封装在一个类中

//为了避免出现同时多次下载同一幅图的问题,使用了本地缓存

//封装的类:

// 为了加快速度，在内存中开启缓存（主要应用于重复图片较多时，或者同一个图片要多次被访问，比如在ListView时来回滚动）

// 固定五个线程来执行任务

/\*\*

\*

\* **@param** imageUrl

\* 图像url地址

\* **@param** callback

\* 回调接口

\* **@return** 返回内存中缓存的图像，第一次加载返回null

\*/

// 如果缓存过就从缓存中取出数据

// 缓存中没有图像，则从网络上取出数据，并将取出的数据缓存到内存中

// 从网络上取数据方法

// 测试时，模拟网络延时，实际时这行代码不能有

// 对外界开放的回调接口

// 注意 此方法是用来设置目标对象的图像资源

**public** **class** MainActivity3 **extends** Activity {

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

loadImage4("http://www.baidu.com/img/baidu\_logo.gif", R.id.*imageView1*);

loadImage4("http://www.chinatelecom.com.cn/images/logo\_new.gif",

R.id.*imageView2*);

loadImage4("http://cache.soso.com/30d/img/web/logo.gif",

R.id.*imageView3*);

loadImage4("http://csdnimg.cn/www/images/csdnindex\_logo.gif",

R.id.*imageView4*);

loadImage4("http://images.cnblogs.com/logo\_small.gif",

R.id.*imageView5*);

}

**private** AsyncImageLoader3 asyncImageLoader3 = **new** AsyncImageLoader3();

// 引入线程池，并引入内存缓存功能,并对外部调用封装了接口，简化调用过程

**private** **void** loadImage4(**final** String url, **final** **int** id) {

// 如果缓存过就会从缓存中取出图像，ImageCallback接口中方法也不会被执行

Drawable cacheImage = asyncImageLoader3.loadDrawable(url,

**new** AsyncImageLoader3.ImageCallback() {

// 请参见实现：如果第一次加载url时下面方法会执行

**public** **void** imageLoaded(Drawable imageDrawable) {

((ImageView) findViewById(id))

.setImageDrawable(imageDrawable);

}

});

**if** (cacheImage != **null**) {

((ImageView) findViewById(id)).setImageDrawable(cacheImage);

}

}

}

**public** **class** AsyncImageLoader3 {

// 为了加快速度，在内存中开启缓存（主要应用于重复图片较多时，或者同一个图片要多次被访问，比如在ListView时来回滚动）

**public** Map<String, SoftReference<Drawable>> imageCache = **new** HashMap<String, SoftReference<Drawable>>();

**private** ExecutorService executorService = Executors.*newFixedThreadPool*(5); // 固定五个线程来执行任务

**private** **final** Handler handler = **new** Handler();

/\*\*

\*

\* **@param** imageUrl

\* 图像url地址

\* **@param** callback

\* 回调接口

\* **@return** 返回内存中缓存的图像，第一次加载返回null

\*/

**public** Drawable loadDrawable(**final** String imageUrl,

**final** ImageCallback callback) {

// 如果缓存过就从缓存中取出数据

**if** (imageCache.containsKey(imageUrl)) {

SoftReference<Drawable> softReference = imageCache.get(imageUrl);

**if** (softReference.get() != **null**) {

**return** softReference.get();

}

}

// 缓存中没有图像，则从网络上取出数据，并将取出的数据缓存到内存中

executorService.submit(**new** Runnable() {

**public** **void** run() {

**try** {

**final** Drawable drawable = loadImageFromUrl(imageUrl);

imageCache.put(imageUrl, **new** SoftReference<Drawable>(

drawable));

handler.post(**new** Runnable() {

**public** **void** run() {

callback.imageLoaded(drawable);

}

});

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

}

});

**return** **null**;

}

// 从网络上取数据方法

**protected** Drawable loadImageFromUrl(String imageUrl) {

**try** {

// 测试时，模拟网络延时，实际时这行代码不能有

SystemClock.*sleep*(2000);

**return** Drawable.*createFromStream*(**new** URL(imageUrl).openStream(),

"image.png");

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

}

// 对外界开放的回调接口

**public** **interface** ImageCallback {

// 注意 此方法是用来设置目标对象的图像资源

**public** **void** imageLoaded(Drawable imageDrawable);

}

}

**4.Json 和xml的解析数据的使用**

(1).XML定义  
扩展标记语言 (Extensible Markup Language, XML) ，用于标记电子文件使其具有结构性的标记语言，可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。 XML使用DTD(document type definition)文档类型定义来组织数据;格式统一，跨平台和语言，早已成为业界公认的标准。  
XML是标准通用标记语言 (SGML) 的子集，非常适合 Web 传输。XML 提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。

(2).JSON定义  
JSON(JavaScript Object Notation)一种轻量级的数据交换格式，具有良好的可读和便于快速编写的特性。可在不同平台之间进行数据交换。JSON采用兼容性很高的、完全独立于语言文本格式，同时也具备类似于C语言的习惯(包括C, C++, C#, Java, JavaScript, Perl, Python等)体系的行为。这些特性使JSON成为理想的数据交换语言。  
JSON基于JavaScript Programming Language , Standard ECMA-262 3rd Edition - December 1999 的一个子集。

2.XML和JSON优缺点

(1).XML的优缺点  
<1>.XML的优点  
　　A.格式统一，符合标准；  
　　B.容易与其他系统进行远程交互，数据共享比较方便。  
<2>.XML的缺点  
　　A.XML文件庞大，文件格式复杂，传输占带宽；  
　　B.服务器端和客户端都需要花费大量代码来解析XML，导致服务器端和客户端代码变得异常复杂且不易维护；  
　　C.客户端不同浏览器之间解析XML的方式不一致，需要重复编写很多代码；  
　　D.服务器端和客户端解析XML花费较多的资源和时间。

(2).JSON的优缺点  
<1>.JSON的优点：  
　　A.数据格式比较简单，易于读写，格式都是压缩的，占用带宽小；  
　　B.易于解析，客户端JavaScript可以简单的通过eval()进行JSON数据的读取；  
　　C.支持多种语言，包括ActionScript, C, C#, ColdFusion, Java, JavaScript, Perl, PHP, Python, Ruby等服务器端语言，便于服务器端的解析；  
　　D.在PHP世界，已经有PHP-JSON和JSON-PHP出现了，偏于PHP序列化后的程序直接调用，PHP服务器端的对象、数组等能直接生成JSON格式，便于客户端的访问提取；  
　　E.因为JSON格式能直接为服务器端代码使用，大大简化了服务器端和客户端的代码开发量，且完成任务不变，并且易于维护。  
<2>.JSON的缺点  
　　A.没有XML格式这么推广的深入人心和喜用广泛，没有XML那么通用性；  
　　B.JSON格式目前在Web Service中推广还属于初级阶段。

//////////////////////////////////////////////////////////////////////////////

1) JSON和XML可读性基本相同  
2）JSON和XML通用拥有丰富的解析手段  
3）JSON相对XML来讲体积更小  
4）JSON与JavaScript的交互性更加方便  
5） JSON对数据的描述性相对较差

三 JSON实战  
1. 解析json格式的数据 用gson 或者jackson解析  
比如说有一个User对象，他有两个属性:name和age,当然实际情况可能有更多的属性，  
那么用json格式表示就是：{name:"Ian", age: "20"},我相信你懂的。  
1）自定义JSON帮助类--JsonUtils  
Java代码   
public class JsonUtils {   
  
public void parseJson(String jsonData) {   
  
try {   
JsonReader reader = new JsonReader(new StringReader(jsonData));   
reader.beginArray();   
while (reader.hasNext()) {   
reader.beginObject();   
while(reader.hasNext()) {   
String tagName = reader.nextName();   
if(tagName.equals("name")) {   
System.out.println("name--->" + reader.nextString());   
} else if(tagName.equals("age")) {   
System.out.println("age--->" + reader.nextString());   
}   
}   
reader.endObject();   
}   
reader.endArray();   
} catch (IOException e) {   
e.printStackTrace();   
}   
  
}   
}   
  
  
2） Activity代码  
  
Java代码   
public class MainActivity extends Activity {   
private Button btnStart;   
private String jsonDataUser = "{\"name\":\"Ian\",\"age\":20}";   
  
  
public void onCreate(Bundle savedInstanceState) {   
super.onCreate(savedInstanceState);   
setContentView(R.layout.main);   
  
btnStart = (Button)findViewById(R.id.btnStart);   
btnStart.setOnClickListener(new BtnStartSetOnClickListener());   
  
  
}   
  
private class BtnStartSetOnClickListener implements OnClickListener {   
@Override   
public void onClick(View v) {   
JsonUtils jsonUtils = new JsonUtils();   
jsonUtils.parseJson(jsonData);   
}   
}   
}   
就这么几行代码，在Console上就可以看到解析输出的信息了，是不是很简单呢？  
  
  
2. 解析对象  
1）上面解析的是字符串形式，如果我们定义一个User对象会怎么样呢？  
  
Java代码   
public class User {   
private String name;   
private int age;   
public String getName() {   
return name;   
}   
public void setName(String name) {   
this.name = name;   
}   
public int getAge() {   
return age;   
}   
public void setAge(int age) {   
this.age = age;   
}   
}   
  
2） 解析方法  
  
Java代码   
public void parseUserFromJson(String jsonData) {   
  
Gson gson =new Gson();   
User user = gson.fromJson(jsonData, User.class);   
System.out.println("name---------->" + user.getName());   
System.out.println("age---------->" + user.getAge());   
  
}   
  
是不是更简单了呢？  
  
3. 解析集合  
1）如果不是单个User，而是ArrayList<User>,那么该怎么解析啊，因为这个太常见了。  
要解决这个问题，得先弄明白集合用json该怎么表示，简单来讲“{}”表示一个对象，“[]”表示一个集合，集合里面可以有多个对象，这样表示[{},{},{}]。OK，那么看实战代码，  
json数据:  
  
Java代码   
private String jsonDataUsers = "[{\"name\":\"Jack\",\"age\":20},{\"name\":\"Tom\",\"age\":22},{\"name\":\"Ian\",\"age\":20}]";   
解析代码：  
  
Java代码   
public void parseUserFromJson(String jsonData) {   
  
Type typeList = new TypeToken<ArrayList<User>>(){}.getType();   
Gson gson = new Gson();   
LinkedList<User> users = gson.fromJson(jsonData, typeList);   
for(Iterator<User> iterator = users.iterator(); iterator.hasNext();) {   
User user = iterator.next();   
System.out.println("name---------->" + user.getName());   
System.out.println("age---------->" + user.getAge());   
}   
}