# 即时通信相关技术分享

## 网络协议

|  |  |  |
| --- | --- | --- |
| OSI中的层 | 功能 | TCP/IP协议族 |
| 应用层 | 文件传输，电子邮件，文件服务，虚拟终端 | TFTP，HTTP，SNMP，FTP，SMTP，DNS，Telnet |
| 表示层 | 数据格式化，代码转换，数据加密 | 没有协议 |
| 会话层 | 解除或建立与别的接点的联系 | 没有协议 |
| 传输层 | 提供端对端的接口 | TCP，UDP |
| 网络层 | 为数据包选择路由 | IP，ICMP，RIP，OSPF，BGP，IGMP |
| 数据链路层 | 传输有地址的帧以及错误检测功能 | SLIP，CSLIP，PPP，ARP，RARP，MTU |
| 物理层 | 以二进制数据形式在物理媒体上传输数据 | ISO2110，IEEE802。IEEE802.2 |

<http://jingyan.baidu.com/article/08b6a591e07ecc14a80922f1.html>

### TCP/IP协议栈

通常人们认为OSI模型的最上面三层（应用层、表示层和会话层）在TCP/IP组中是一个应用层。由于TCP/IP有一个相对较弱的会话层，由TCP和RTP下的打开和关闭连接组成，并且在TCP和UDP下的各种应用提供不同的端口号，这些功能能够被单个的应用程序（或者那些应用程序所使用的库）增加。与此相似的是，IP是按照将它下面的网络当作一个黑盒子的思想设计的，这样在讨论TCP/IP的时候就可以把它当作一个独立的层。

|  |  |  |
| --- | --- | --- |
| 4 | **应用层** *（OSI 5到7层）* | 例如[HTTP](http://zh.wikipedia.org/wiki/HTTP)、[FTP](http://zh.wikipedia.org/wiki/FTP)、[DNS](http://zh.wikipedia.org/wiki/DNS) *（如*[*BGP*](http://zh.wikipedia.org/wiki/BGP)*和*[*RIP*](http://zh.wikipedia.org/wiki/RIP)*这样的路由协议，尽管由于各种各样的原因它们分别运行在TCP和UDP上，仍然可以将它们看作网络层的一部分）* |
| 3 | **传输层** *（OSI 4层）* | 例如[TCP](http://zh.wikipedia.org/wiki/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE)、[UDP](http://zh.wikipedia.org/wiki/UDP)、[RTP](http://zh.wikipedia.org/wiki/RTP)、[SCTP](http://zh.wikipedia.org/wiki/SCTP) *（如*[*OSPF*](http://zh.wikipedia.org/wiki/OSPF)*这样的路由协议，尽管运行在IP上也可以看作是网络层的一部分）* |
| 2 | **网络互连层** *（OSI 3层）* | 对于TCP/IP来说这是[因特网协议](http://zh.wikipedia.org/wiki/%E5%9B%A0%E7%89%B9%E7%BD%91%E5%8D%8F%E8%AE%AE)（IP） *（如*[*ICMP*](http://zh.wikipedia.org/wiki/ICMP)*和*[*IGMP*](http://zh.wikipedia.org/wiki/IGMP)*这样的必须协议尽管运行在IP上，也仍然可以看作是网络互连层的一部分；*[*ARP*](http://zh.wikipedia.org/wiki/ARP)*不运行在IP上）* |
| 1 | **网络接口层** *（OSI 1和2层）* | 例如[以太网](http://zh.wikipedia.org/wiki/%E4%BB%A5%E5%A4%AA%E7%BD%91)、[Wi-Fi](http://zh.wikipedia.org/wiki/Wi-Fi)、[MPLS](http://zh.wikipedia.org/wiki/MPLS)等。 |

### 应用层

该层包括所有和应用程序协同工作，利用基础网络交换应用程序专用的数据的协议。 [应用层](http://zh.wikipedia.org/wiki/%E5%BA%94%E7%94%A8%E5%B1%82)是大多数普通与网络相关的程序为了通过网络与其他程序通信所使用的层。这个层的处理过程是应用特有的；数据从网络相关的程序以这种应用内部使用的格式进行传送，然后被编码成标准协议的格式。

一旦从应用程序来的数据被编码成一个标准的应用层协议，它将被传送到IP栈的下一层。

在传输层，应用程序最常用的是TCP或者UDP，并且服务器应用程序经常与一个[公开的端口号](http://zh.wikipedia.org/wiki/TCP/UDP%E7%AB%AF%E5%8F%A3%E5%88%97%E8%A1%A8)相联系。服务器应用程序的端口由[互联网号码分配局](http://zh.wikipedia.org/wiki/%E4%BA%92%E8%81%94%E7%BD%91%E5%8F%B7%E7%A0%81%E5%88%86%E9%85%8D%E5%B1%80)（IANA）正式地分配，但是现今一些新协议的开发者经常选择它们自己的端口号。由于在同一个系统上很少超过少数几个的服务器应用，端口冲突引起的问题很少。应用软件通常也允许用户强制性地指定端口号作为[运行](http://zh.wikipedia.org/w/index.php?title=%E8%BF%90%E8%A1%8C&action=edit&redlink=1)[参数](http://zh.wikipedia.org/wiki/%E5%8F%82%E6%95%B0)

连结外部的客户端程序通常使用系统分配的一个随机端口号。监听一个端口并且通过服务器将那个端口发送到应用的另外一个副本以建立对等连结（如[IRC](http://zh.wikipedia.org/wiki/IRC)上的[dcc](http://zh.wikipedia.org/w/index.php?title=Dcc&action=edit&redlink=1)文件传输）的应用也可以使用一个随机端口，但是应用程序通常允许定义一个特定的端口范围的规范以允许端口能够通过实现[网络地址转换](http://zh.wikipedia.org/wiki/%E7%BD%91%E7%BB%9C%E5%9C%B0%E5%9D%80%E8%BD%AC%E6%8D%A2)（NAT）的路由器映射到内部。

每一个应用层（[TCP/IP参考模型](http://zh.wikipedia.org/wiki/TCP/IP%E5%8F%82%E8%80%83%E6%A8%A1%E5%9E%8B)的最高层）协议一般都会使用到两个传输层协议之一： 面向连接的[TCP传输控制协议](http://zh.wikipedia.org/wiki/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE)和无连接的包传输的[UDP](http://zh.wikipedia.org/wiki/UDP)[用户数据报文协议](http://zh.wikipedia.org/wiki/%E7%94%A8%E6%88%B7%E6%95%B0%E6%8D%AE%E6%8A%A5%E6%96%87%E5%8D%8F%E8%AE%AE)。

常用的应用层协议有：

运行在[TCP](http://zh.wikipedia.org/wiki/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE)协议上的协议：

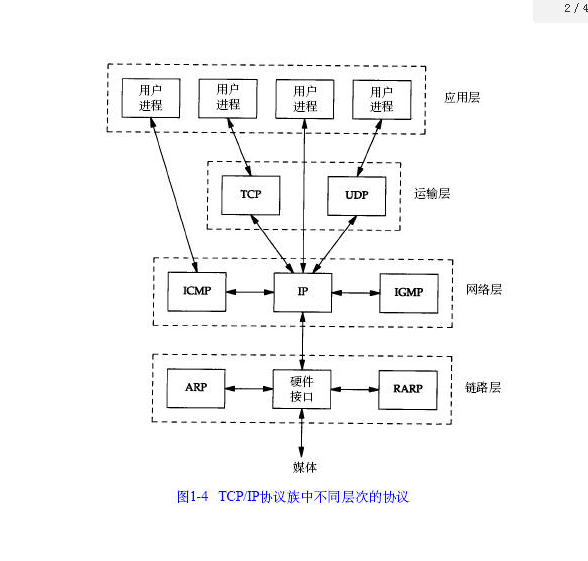
* [HTTP](http://zh.wikipedia.org/wiki/HTTP)（Hypertext Transfer Protocol，超文本传输协议），主要用于普通浏览。
* [HTTPS](http://zh.wikipedia.org/wiki/HTTPS)（Hypertext Transfer Protocol over Secure Socket Layer, or HTTP over SSL，安全超文本传输协议）,HTTP协议的安全版本。
* [FTP](http://zh.wikipedia.org/wiki/FTP)（File Transfer Protocol，文件传输协议），由名知义，用于文件传输。
* [POP3](http://zh.wikipedia.org/wiki/POP3)（Post Office Protocol, version 3，邮局协议），收邮件用。
* [SMTP](http://zh.wikipedia.org/wiki/SMTP)（Simple Mail Transfer Protocol，简单邮件传输协议），用来发送电子邮件。
* [TELNET](http://zh.wikipedia.org/wiki/TELNET)（Teletype over the Network，网络电传），通过一个终端（terminal）登陆到网络。
* [SSH](http://zh.wikipedia.org/wiki/SSH)（Secure Shell，用于替代安全性差的[TELNET](http://zh.wikipedia.org/wiki/TELNET)），用于加密安全登陆用。

运行在[UDP](http://zh.wikipedia.org/wiki/UDP)协议上的协议：

* [BOOTP](http://zh.wikipedia.org/wiki/BOOTP)（Boot Protocol，启动协议），应用于无盘设备。
* [NTP](http://zh.wikipedia.org/wiki/NTP)（Network Time Protocol，网络时间协议），用于网络同步。

其他：

* [DNS](http://zh.wikipedia.org/wiki/%E5%9F%9F%E5%90%8D%E7%B3%BB%E7%BB%9F)（Domain Name Service，域名服务），用于完成地址查找，邮件转发等工作（运行在[TCP](http://zh.wikipedia.org/wiki/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE)和[UDP](http://zh.wikipedia.org/wiki/UDP)协议上）。
* [ECHO](http://zh.wikipedia.org/w/index.php?title=ECHO&action=edit&redlink=1)（Echo Protocol，回绕协议），用于查错及测量应答时间（运行在[TCP](http://zh.wikipedia.org/wiki/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE)和[UDP](http://zh.wikipedia.org/wiki/UDP)协议上）。
* [SNMP](http://zh.wikipedia.org/wiki/SNMP)（Simple Network Management Protocol，简单网络管理协议），用于网络信息的收集和网络管理。
* [DHCP](http://zh.wikipedia.org/wiki/DHCP)（Dynamic Host Configuration Protocol，动态主机配置协议），动态配置IP地址。
* [ARP](http://zh.wikipedia.org/wiki/ARP)（Address Resolution Protocol，地址解析协议），用于动态解析以太网硬件的地址。



TCP/IP（TransmissionControlProtocol/InternetProtocol的简写，中文译名为传输控制协议/互联网络协议）协议是Internet最基本的协议，简单地说，就是由底层的IP协议和TCP协议组成的。TCP/IP协议的开发工作始于70年代，是用于[互联网](http://www.baike.com/wiki/%E4%BA%92%E8%81%94%E7%BD%91)的第一套协议。

TCP/IP的通讯协议

简要介绍一下TCP/IP的内部结构，TCP/IP协议组之所以流行，部分原因是因为它可以用在各种各样的信道和底层协议（例如T1和X.25、以太网以及RS-232串行接口）之上。确切地说，TCP/IP协议是一组包括TCP协议和IP协议，[UDP](http://www.baike.com/wiki/UDP)（UserDatagramProtocol）协议、[ICMP](http://www.baike.com/wiki/ICMP)（InternetControlMessageProtocol）协议和其他一些协议的协议组

## HTTP协议

### 协议概述

http是一种应用层的协议。

### Socket

<http://baike.baidu.com/view/13870.htm>

socket的英文原义是“孔”或“插座”。作为BSD UNIX的[进程通信](http://baike.baidu.com/view/549640.htm)机制，取后一种意思。通常也称作"[套接字](http://baike.baidu.com/view/538713.htm)"，用于描述IP[地址](http://baike.baidu.com/view/494802.htm)和端口，是一个通信链的句柄。在Internet上的[主机](http://baike.baidu.com/view/23880.htm)一般运行了多个服务软件，同时提供几种服务。每种服务都打开一个Socket，并绑定到一个端口上，不同的端口对应于不同的服务。Socket正如其英文原意那样，象一个多孔插座。一台[主机](http://baike.baidu.com/view/23880.htm)犹如布满各种插座的房间，每个插座有一个编号，有的插座提供220伏交流电， 有的提供110伏交流电，有的则提供有线电视节目。 客户软件将插头插到不同编号的插座，就可以得到不同的服务。

【是TCP/IP协议的封装】

#### 什么是socket

所谓socket通常也称作"套接字"，应用程序通常通过"套接字"向网络发出请求或者应答网络请求。以J2SDK-1.3为例，Socket和ServerSocket类库位于java .net包中。ServerSocket用于[服务器](http://baike.baidu.com/view/899.htm)端，Socket是建立网络连接时使用的。在连接成功时，应用[程序](http://baike.baidu.com/view/17674.htm)两端都会产生一个Socket实例，操作这个实例，完成所需的会话。对于一个网络连接来说，[套接字](http://baike.baidu.com/view/538713.htm)是平等的，并没有差别，不因为在服务器端或在[客户端](http://baike.baidu.com/view/930.htm)而产生不同级别。不管是Socket还是ServerSocket它们的工作都是通过SocketImpl类及其子类完成的。

#### SOCKET连接过程

根据连接启动的方式以及本地[套接字](http://baike.baidu.com/view/538713.htm)要连接的目标，[套接字](http://baike.baidu.com/view/538713.htm)之间的连接过程可以分为三个步骤：[服务器](http://baike.baidu.com/view/899.htm)监听，[客户端](http://baike.baidu.com/view/930.htm)请求，连接确认。

[服务器](http://baike.baidu.com/view/899.htm)监听：是[服务器](http://baike.baidu.com/view/899.htm)端[套接字](http://baike.baidu.com/view/538713.htm)并不定位具体的[客户端](http://baike.baidu.com/view/930.htm)[套接字](http://baike.baidu.com/view/538713.htm)，而是处于等待连接的状态，实时监控网络状态。

[客户端](http://baike.baidu.com/view/930.htm)请求：是指由[客户端](http://baike.baidu.com/view/930.htm)的[套接字](http://baike.baidu.com/view/538713.htm)提出连接请求，要连接的目标是[服务器](http://baike.baidu.com/view/899.htm)端的套接字。为此，[客户端](http://baike.baidu.com/view/930.htm)的[套接字](http://baike.baidu.com/view/538713.htm)必须首先描述它要连接的[服务器](http://baike.baidu.com/view/899.htm)的套接字，指出[服务器](http://baike.baidu.com/view/899.htm)端套接字的[地址](http://baike.baidu.com/view/494802.htm)和[端口号](http://baike.baidu.com/view/642103.htm)，然后就向[服务器](http://baike.baidu.com/view/899.htm)端套接字提出连接请求。

连接确认：是指当[服务器](http://baike.baidu.com/view/899.htm)端[套接字](http://baike.baidu.com/view/538713.htm)监听到或者说接收到[客户端](http://baike.baidu.com/view/930.htm)套接字的连接请求，它就响应[客户端](http://baike.baidu.com/view/930.htm)套接字的请求，建立一个新的线程，把[服务器](http://baike.baidu.com/view/899.htm)端套接字的描述发给[客户端](http://baike.baidu.com/view/930.htm)，一旦客户端确认了此描述，连接就建立好了。而[服务器](http://baike.baidu.com/view/899.htm)端[套接字](http://baike.baidu.com/view/538713.htm)继续处于[监听状态](http://baike.baidu.com/view/1624776.htm)，继续接收其他[客户端](http://baike.baidu.com/view/930.htm)套接字的连接请求。

#### 如何开发一个Server-Client模型的程序

开发原理：

[服务器](http://baike.baidu.com/view/899.htm)，使用ServerSocket监听指定的端口，端口可以随意指定（由于1024以下的端口通常属于保留端口，在一些[操作系统](http://baike.baidu.com/view/880.htm)中不可以随意使用，所以建议使用大于1024的端口），等待客户连接请求，客户连接后，会话产生；在完成会话后，关闭连接。

[客户端](http://baike.baidu.com/view/930.htm)，使用Socket对网络上某一个[服务器](http://baike.baidu.com/view/899.htm)的某一个端口发出连接请求，一旦连接成功，打开会话；会话完成后，关闭Socket。[客户端](http://baike.baidu.com/view/930.htm)不需要指定打开的端口，通常临时的、动态的分配一个1024以上的端口。

Socket接口是TCP/IP网络的API，Socket接口定义了许多函数或例程，[程序员](http://baike.baidu.com/view/39175.htm)可以用它们来开发TCP/IP网络上的应用程序。要学Internet上的[TCP/IP网络编程](http://baike.baidu.com/view/3058831.htm)，必须理解Socket接口。Socket接口设计者最先是将接口放在Unix[操作系统](http://baike.baidu.com/view/880.htm)里面的。如果了解Unix系统的输入和输出的话，就很容易了解Socket了。网络的Socket数据传输是一种特殊的I/O，Socket也是一种[文件描述符](http://baike.baidu.com/view/1303430.htm)。Socket也具有一个类似于打开文件的[函数调用](http://baike.baidu.com/view/2369016.htm)Socket（），该函数返回一个[整型](http://baike.baidu.com/view/1311503.htm)的Socket描述符，随后的连接建立、数据传输等操作都是通过该Socket实现的。

#### 常用的Socket类型

有两种：流式Socket（SOCK\_STREAM）和数据报式Socket（SOCK\_DGRAM）。流式是一种面向连接的Socket，针对于面向连接的TCP服务应用；数据报式Socket是一种无连接的Socket，对应于无连接的[UDP](http://baike.baidu.com/view/30509.htm)服务应用。Socket为了建立Socket，[程序](http://baike.baidu.com/view/17674.htm)可以调用Socket函数，该函数返回一个类似于[文件描述符](http://baike.baidu.com/view/1303430.htm)的句柄。socket函数原型为：int socket(int domain,int type,int[protocol](http://baike.baidu.com/view/286872.htm));[domain](http://baike.baidu.com/view/655135.htm)指明所使用的协议族，通常为PF\_INET(其与addrinfo 里的 AF\_INET在现在看来是相同的。只是历史上人们曾构想将AF（[地址](http://baike.baidu.com/view/494802.htm)家族address family）与PF(protocol family 协议家族)分开），但实际上这种区分并未真正推广，所以现在AF\_INET和PF\_INET具有相同的意义。其中AF\_INET是基于[IPv4](http://baike.baidu.com/view/21992.htm)而PF\_INET基于[IPv6](http://baike.baidu.com/view/5228.htm)）表示[互联网协议](http://baike.baidu.com/view/1087323.htm)族（[TCP/IP协议族](http://baike.baidu.com/view/2221037.htm)）；type[参数](http://baike.baidu.com/view/327406.htm)指定socket的类型：SOCK\_STREAM 或SOCK\_DGRAM，Socket接口还定义了原始Socket（SOCK\_RAW），允许[程序](http://baike.baidu.com/view/17674.htm)使用低层协议；protocol通常赋值0。Socket（）调用返回一个[整型](http://baike.baidu.com/view/1311503.htm)socket描述符，你可以在后面的调用使用它。Socket描述符是一个指向内部[数据结构](http://baike.baidu.com/view/9900.htm)的[指针](http://baike.baidu.com/view/159417.htm)，它指向描述符表入口。调用Socket函数时，socket执行体将建立一个Socket，实际上"建立一个Socket"意味着为一个Socket数据结构分配[存储空间](http://baike.baidu.com/view/1375172.htm)。Socket执行体为你管理描述符表。两个网络[程序](http://baike.baidu.com/view/17674.htm)之间的一个网络连接包括五种信息：[通信协议](http://baike.baidu.com/view/185322.htm)、本地[协议地址](http://baike.baidu.com/view/1252611.htm)、[本地主机](http://baike.baidu.com/view/544310.htm)端口、远端[主机地址](http://baike.baidu.com/view/547482.htm)和远端协议端口。Socket[数据结构](http://baike.baidu.com/view/9900.htm)中包含这五种信息。socket在测量软件中的使用也很广泛。

现在越来越多企业采用这种，当然也有人曾经想将[地址](http://baike.baidu.com/view/494802.htm)与[协议](http://baike.baidu.com/view/36190.htm)分开，但实际的意义上并未实行到，现在还在使用原始的类型，它是指向低层协议的一种。

以下摘书来自

<http://zh.wikipedia.org/wiki/%E8%B6%85%E6%96%87%E6%9C%AC%E4%BC%A0%E8%BE%93%E5%8D%8F%E8%AE%AE>

HTTP是一个客户端终端（用户）和服务器端（网站）请求和应答的标准（[TCP](http://zh.wikipedia.org/wiki/TCP)）。通过使用Web浏览器、网络爬虫或者其它的工具，客户端发起一个HTTP请求到服务器上指定端口（默认[端口](http://zh.wikipedia.org/wiki/%E7%AB%AF%E5%8F%A3)为80）。我们称这个客户端为用户代理程序（user agent）。应答的服务器上存储着一些资源，比如HTML文件和图像。我们称这个应答服务器为源服务器（origin server）。在用户代理和源服务器中间可能存在多个“中间层”，比如[代理](http://zh.wikipedia.org/wiki/%E4%BB%A3%E7%90%86)、[网关](http://zh.wikipedia.org/wiki/%E7%BD%91%E5%85%B3)或者[隧道](http://zh.wikipedia.org/wiki/%E9%9A%A7%E9%81%93)（tunnel）。

尽管[TCP/IP](http://zh.wikipedia.org/wiki/TCP/IP)协议是互联网上最流行的应用，HTTP协议中，并没有规定必须使用它或它支持的层。事实上，HTTP可以在任何互联网协议上，或其他网络上实现。HTTP假定其下层协议提供可靠的传输。因此，任何能够提供这种保证的协议都可以被其使用。因此也就是其在TCP/IP协议族使用TCP作为其传输层。

通常，由HTTP客户端发起一个请求，创建一个到服务器指定端口（默认是80端口）的TCP连接。HTTP服务器则在那个端口监听客户端的请求。一旦收到请求，服务器会向客户端返回一个状态，比如"HTTP/1.1 200 OK"，以及返回的内容，如请求的文件、错误消息、或者其它信息。

### HTTP服务器

#### Tomcat

<http://tomcat.apache.org/>

#### 编译Tomcat的源码

官方源码

<http://tomcat.apache.org/svn.html>

1：从svn 检出源代码  [http://svn.apache.org/repos/asf/tomcat](http://svn.eu.apache.org/repos/asf/tomcat/trunk/)

假设位于：E:\workspaces\MyEclipse\_9\TOMCAT\_7\_0\_19

2：编辑：

 build.properties.default  
把base.path=/usr/share/java   
改成base.path=E:/workspaces/MyEclipse\_9/TOMCAT\_7\_0\_19

**3:**用ant编译Tomcat7

在myeclipse中右击build.xml ---> run build

开始编译：

--------------------------------------------------

BUILD SUCCESSFUL   
Total time: 5 minutes 16 seconds   
**第三步**：  
修改E:/。。/output/build/conf/server.xml   [注意并不是根目录下的/conf下的]

【运行org.apache.catalina.startup.Bootstrap时候，则修改conf的server.xml.. 】

 将

**注**意**:目录结构： 反斜杠 / 或者 \\**

**<Connector** port="8080" protocol="HTTP/1.1"    connectionTimeout="20000"      redirectPort="8443" **/>**

改成

**<Connector** port="8080" protocol="org.apache.coyote.http11.Http11NioProtocol" connectionTimeout="20000"

               redirectPort="8443" **/>**

7.运行:   
。。。。。/output/build/bin>startup

注意： 如果直接双击 startup.bat  需要 配置 catalina\_home 环境变量

【此时的catalina\_home为E:/workspaces/MyEclipse\_9/TOMCAT\_7\_0\_19/o**utput/build** 】

[Http://localhost:8080/excmples/](http://localhost:8080/excmples/)

.....................

  ok  tomcat 检出并编译成功

 然后接下来是运行tomcat 源代码 ：

 新建一个 java 工程 【注意不是web工程 】 test

 然后把 trunk 中的 java 下的源文件 导入test 下的src

 然后把 /output/build   下的 conf    webapps  俩文件夹copy到

 test 的根目录下 : 目录结构如下

 test --

       -- src

       -- conf

       -- webapps

 其他的非必须。。。。。。。。

 运行 org.apache.catalina.startup.Bootstrap    搞定。。。。

=====================

  其中需要四个jar

1：  ant.jar  ：  下载 Ant    找到  lib  目录

 2：  eclipse :org.eclipse.jtd.core\_xxxx.jar   :  去 eclipse 官网 下载 ：

 3：  jaxrpc.jar    |     wsdl4j-1.5.1.jar

 去 apache 下载 axis-1.4   ： 解压  找到 lib  目录

 注意： axis-1.4  与 其他的版本 尤其是 1.5.x 由很大的不同

 其他版本中 不存在 jaxrpc.jar

[653](http://baike.baidu.com/view/13870.htm)