**[Java 网络编程（一） 网络基础知识](http://www.cnblogs.com/mengdd/archive/2013/03/09/2951826.html)**

**网络基础知识**

**网络编程的目的**：直接或间接地通过网络协议与其他计算机进行通讯。

　　网络编程中有两个主要的问题：

　　1.如何准确地定位网络上一台或多台主机。

　　2.找到主机后如何可靠高效地进行数据传输。

　　目前较为流行的网络编程模型是**客户端/服务器（C/S）结构**。

　　即通信双方一方作为服务器等待客户提出请求并予以相应。客户则在需要服务时向服务器提出申请。

　　服务器始终运行，监听网络端口，一旦有客户请求，就会启动一个服务线程来响应该客户，同时自己继续监听服务窗口，使后来的客户也能及时得到服务。

**IP地址**

　　IP网络中每台主机都必须有一个唯一的IP地址，IP地址是一个逻辑地址。

　　英特网上的IP地址具有全球唯一性。

　　32位，四个字节，常用点分十进制的格式表示。

　　例如：192.168.0.200

**协议**

**为进行网络中的数据交换（通信）而建立的规则、标准或约定。（=语义+语法+规则）。**

　　不同层具有各自不同的协议。

**ISO/OSI七层参考模型**

　　网络体系结构**解决异质性问题**采用的是分层的方法——**把复杂的网络互联问题划分为若干个较小的、单一的问题，在不同层上予以解决。**

**OSI(Open System Interconnection)**参考模型将网络的不同功能划分为7层：

　　应用层：处理网络应用

　　表示层：数据表示

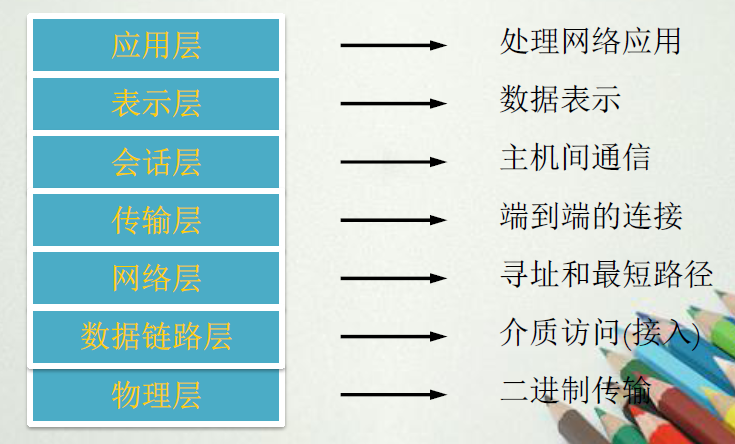
　　会话层：主机间通信

　　传输层：端到端的连接

　　网络层：寻址和最短路径

　　数据链路层：介质访问（接入）

　　物理层：二进制传输



**通信实体的对等层之间不允许直接通信，各层之间是严格的单向依赖**，上层（Service user）使用下层提供的服务，下层（Service provider）向上层提供服务。

**对等层通信的实质**：**对等层实体之间虚拟通信，下层向上层提供服务，实际通信在最底层完成。**

**OSI各层所使用的协议：**

　　应用层：Telnet、FTP、HTTP、DNS、SMTP、POP3

　　传输层：TCP、UDP

　　TCP：面向连接的可靠的传输协议。

　　UDP：是无连接的，不可靠的传输协议。

　　网络层：IP、ICMP、IGMP

**端口**

　　在互联网上传输的数据都包含有用来识别目的地的IP地址和端口号。

**IP地址用来标识网络上的计算机，而端口号用来指明该计算机上的应用程序。**

**端口**是一种抽象的软件结构（包括一些数据结构和I/O缓冲区）。

　　应用程序通过系统调用与某端口建立连接（binding）后，传输层传给该端口的数据都被相应的进程所接收，相应进程发给传输层的数据都通过该端口输出。

**端口用一个整数型标识符来表示，即端口号。**

　　端口号跟协议相关，TCP/IP传输层的两个协议TCP和UDP是完全独立的两个软件模块，因此各自的端口号也相互独立，端口通常称为**协议端口（protocol port）**，简称端口。

**端口使用一个16位的数字来表示，它的范围是0~65535，1024以下的端口号保留给预定义的服务。例如，http使用80端口。**

**数据封装**

　　一台计算机要发送数据到另一台计算机，数据首先必须打包，打包的过程称为**封装**。

**封装就是在数据前面加上特定的协议头部。**

　　OSI参考模型中，对等层协议之间的交换的信息单元称为**协议数据单元（PDU, Protocol Data Unit）**。

　　OSI参考模型中的每一层都要依靠下一层提供的服务。

　　为了提供服务，下层把上层的PDU作为本层的数据封装，然后加入本层的头部（和尾部）。头部中含有完成数据传输所需的控制信息。

　　这样，数据自上而下递交的过程实际上就是不断封装的过程。到达目的地后自下而上递交的过程就是不断拆封的过程。由此可知，在物理线路上传输的数据，其外面实际上被包封了多层“信封”。

# [Java 网络编程（二） 两类传输协议:TCP UDP](http://www.cnblogs.com/mengdd/archive/2013/03/09/2951841.html)

## 

## 两类传输协议：TCP,UDP

## 

## TCP

　　TCP是**Transfer Control Protocol（传输控制协议）**的简称，是一种**面向连接的保证可靠传输的协议**。

　　在TCP/IP协议中，

**IP层主要负责网络主机的定位，数据传输的路由，由IP地址可以唯一确定Internet上的一台主机**。

**而TCP层则提供面向应用的可靠的或非可靠的数据传输机制**，这是网络编程的主要对象，一般不需要关心IP层是如何处理数据的。

　　通过TCP协议传输，得到的是一个顺序的无差错的数据流。

　　发送方和接收方的成对的两个socket之间必须建立连接，以便在TCP协议的基础上进行通信。

　　当一个socket（通常都是server socket）等待建立连接时，另一个socket可以要求进行连接，一旦这两个socket连接起来，它们就可以进行双向数据传输，双方都可以进行发送或接收操作。

**TCP是一个基于连接的协议，它能够提供两台计算机之间的可靠的数据流。**

　　HTTP、FTP、Telnet等应用都需要这种可靠的通信通道。

## 

## UDP

　　UDP是**User Datagram Protocol**的简称，是一种**无连接的协议**。

　　UDP是从一台计算机向另一台计算机发送称为**数据报**的独立数据包的协议，该协议并不保证数据报是否能正确地到达目的地，它是一个非面向连接的协议。

　　每个数据报都是一个独立的信息，包括完整的源地址或目的地址，**它在网络上以任何可能的路径传往目的地，因此能否到达目的地，到达时间以及内容的正确性都是不能保证的。**

## 

## TCP和UDP的比较

　　使用UDP时，每个数据报中都给出了完整的地址信息，因此无需建立发送方和接收方的连接。

　　对于TCP协议，由于它是一个面向连接的协议，在socket之间进行数据传输之前必然要建立连接，所以在TCP中多了一个连接建立的时间。

　　使用UDP传输数据时是有大小限制的，每个被传输的数据报必须限定在**64KB**之内。

　　TCP没有这方面的限制，一旦连接建立起来，双方的socket就可以按统一的格式传输大量的数据。

　　UDP是一个不可靠的协议，发送方所发送的数据报并不一定以相同的次序到达接收方。

　　TCP是一个可靠的协议，它确保接收方完全正确地获取发送方所发送的全部数据。

　　可靠的传输是要付出代价的，对数据内容正确性的检验必然占用计算机的处理时间和网络的带宽。因此TCP传输的效率不如UDP高。

　　TCP在网路通信上有极强的生命力，例如远程连接（Telnet）和文件传输（FTP）都需要不定长度的数据被可靠地传输。

　　相比之下UDP操作简单，而且仅需要较少的监护，因此通常用于局域网高可靠性的分散系统中client/server应用程序。

　　此处推荐书籍《TCP/IP详解》，分三卷。

## TCP/IP模型

　　TCP/IP模型包括四个层次：

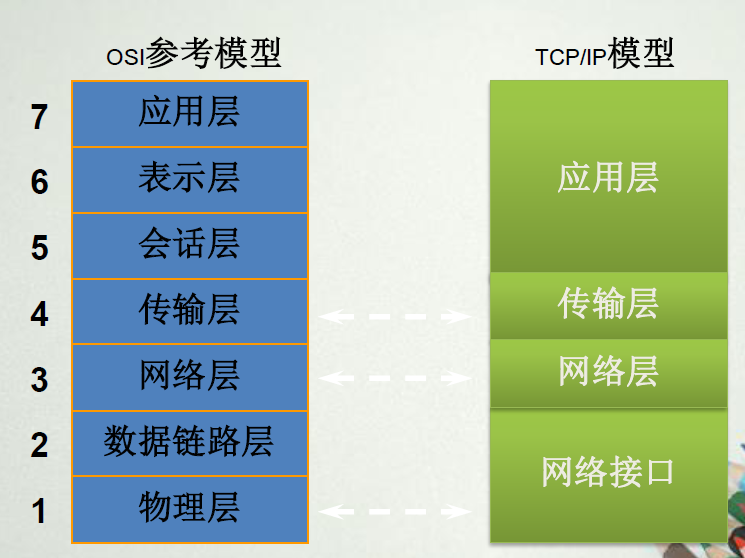
　　应用层

　　传输层

　　网络层

　　网络接口

　　TCP/IP与OSI参考模型的对应关系：



## JDK中的网络类

　　通过java.net包中的类，java程序能够使用TCP或UDP协议在互联网上进行通讯。

　　Java通过扩展已有的流式输入/输出接口和增加在网络上建立输入/输出对象特性这两个方法支持TCP/IP。

　　Java支持TCP和UDP协议族。

　　TCP用于网络的可靠的流式输入/输出。

　　UDP支持更简单的、快速的、点对点的数据报模式。

# **[Java 网络编程（三） 创建和使用URL访问网络上的资源](http://www.cnblogs.com/mengdd/archive/2013/03/09/2951877.html)**

## 

## **创建和使用URL访问网络上的资源**

**URL(Uniform Resource Locator)**是**统一资源定位符**的简称，它表示Internet上某一资源的地址。

　　通过URL我们可以访问Internet上的各种网络资源，比如最常见的WWW, FTP站点。浏览器通过解析给定的URL可以在网络上查找相应的文件或其他资源。

　　在目前使用最为广泛的TCP/IP中对于URL中主机名的解析也是协议的一个标准，即所谓的**域名解析服务**。

　　使用URL进行网络编程，不需要对协议本身有太多的了解，功能也比较弱，相对而言是比较简单的。

## 

## **URL组成**

　　一个URL包括两个主要部分：

**协议标识符**：HTTP, FTP, File等。

**资源名字**：主机名，文件名，端口号，引用。

## 

## **创建URL**

　　在Java程序中，可以创建表示URL地址的URL对象。

**URL对象**表示一个绝对的URL地址，但URL对象可用绝对URL、相对URL和部分URL构建。

　　创建URL的代码如下，如果创建失败会抛出异常：

[IMG_256](http://www.cnblogs.com/mengdd/archive/2013/03/09/javascript:void(0);" \o "复制代码)

try

{

URL myURL = new URL("http://www.google.com.tw/");

}

catch (MalformedURLException e)

{

//exception handler code here

}

[IMG_257](http://www.cnblogs.com/mengdd/archive/2013/03/09/javascript:void(0);" \o "复制代码)

## **获得URL对象的各个属性**

　　URL类中有各种用于获取属性的方法：

　　getProtocol

　　getHost

　　getPort

　　getFile

　　getRef

　　例子程序如下：

IMG_258获取URL对象属性

## **创建和使用URL访问网上资源**

　　为获得URL的实际比特或内容信息，用它的openConnection()方法从它创建一个**URLConnection**对象，与调用URL对象相关，它返回一个URLConnection对象。它可能引发IOException异常。

**URLConnection**是访问远程资源属性的一般用途的类。如果你建立了与远程服务器之间的连接，你可以在传输它到本地之前用URLConnection来检查远程对象的属性。这些属性由HTTP协议规范定义并且仅对用HTTP协议的URL对象有意义。

　　URL和URLConnection类对于希望建立与HTTP服务器的连接来获取信息的简单程序来说是非常好的。

　　例子程序UrlConnection01，建立连接，从连接对象获取输入流，然后读入，再写出到文件中去。

IMG_259程序UrlConnection01

　　也可以直接从URL对象获取输入流，见例子程序UrlConnection02。

IMG_260程序UrlConnection02

 　　查看源代码可以看到内部实现机制是一样的：

public final InputStream openStream() throws java.io.IOException   
　　{

return openConnection().getInputStream();

}

 　　程序代码UrlConnection03用**字符流**的方式读取网站内容显示在控制台上。