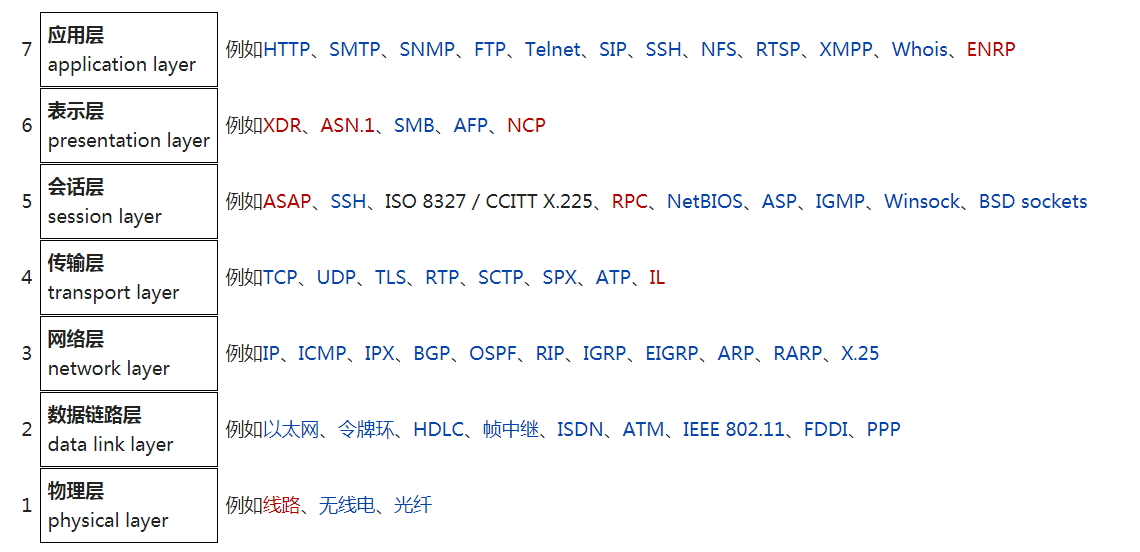
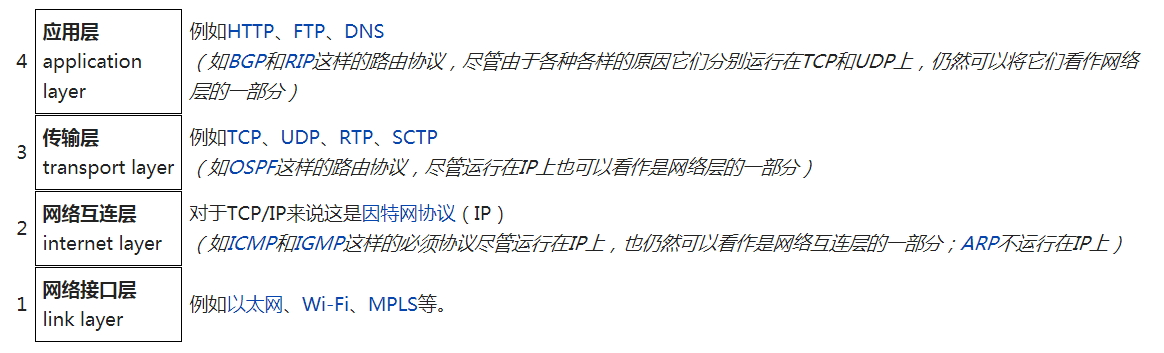
# OSI七层模型



# TCP/IP协议簇四层模型



# TCP和UDP

TCP/IP协议是一个协议族。里面包括很多的协议。UDP只是其中的一个。之所以命名为TCP/IP协议，因为TCP/IP协议是两个很重要的协议，就用他两命名了。

在TCP/IP模型中，**UDP和TCP为网络层以上和应用层以下（传输层）提供接口。**

## TCP

<https://zh.wikipedia.org/wiki/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE>

Transmission Control Protocol，传输控制协议，是一种**面向连接的、可靠的、基于字节流的传输层通信协议。**

TCP协议的运行可划分为三个阶段：**连接创建、数据传送、连接终止。**

应用层向TCP层（传输层）发送**用于网间传输的、用8位字节（一个字符）表示的数据流**，然后TCP把数据流分区成适当长度的**报文段**。之后TCP把结果包传给IP层（网络互联层），由它来通过网络将包传送给接收端实体的TCP层。

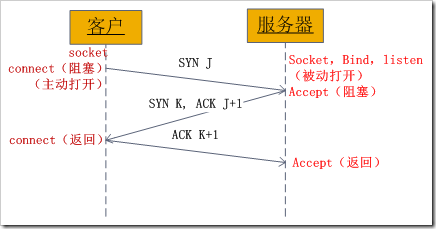
TCP为了保证不发生丢包，就给**每个包一个序号**，同时序号也保证了传送到接收端实体的包的**按序接收**。然后接收端实体对已成功收到的包发回一个相应的**确认（ACK）**；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包就被假设为已丢失将会进行**丢失（发送端）重传**。TCP用一个**校验和函数来检验数据是否有错误**；在发送和接收时都要计算校验和。

### TCP的三次握手建立连接

* **SYN**表示建立连接
* **FIN**表示关闭连接
* **ACK**表示响应
* **PSH**表示有DATA数据传输
* **RST**表示连接重置

**三次握手**：

1. 客户端向服务器发送一个SYN J （客户端请求）
2. 服务器向客户端响应一个SYN K，并对SYN J进行确认ACK J+1（服务端响应请求）
3. 客户端再向服务器发一个确认ACK K+1（客户端确认收到服务端响应）



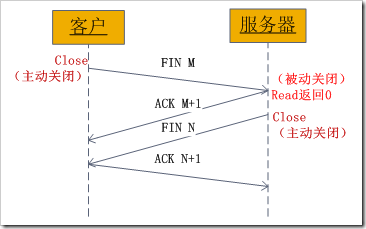
当客户端调用connect时，触发了连接请求，向服务器发送了SYN J包，这时connect进入阻塞状态；服务器监听到连接请求，即收到SYN J包，调用accept函数接收请求向客户端发送SYN K ，ACK J+1，这时accept进入阻塞状态；客户端收到服务器的SYN K ，ACK J+1之后，这时connect返回，并对SYN K进行确认；服务器收到ACK K+1时，accept返回，至此三次握手完毕，连接建立。

**客syn服 -> 服syn+ack客 –> 客ack服**

### TCP的四次挥手释放连接

**四次挥手：**

1. 某个应用进程首先调用close主动关闭连接，这时TCP发送一个FIN M；
2. 另一端接收到FIN M之后，执行被动关闭，对这个FIN进行确认。它的接收也作为文件结束符传递给应用进程，因为FIN的接收意味着应用进程在相应的连接上再也接收不到额外数据；
3. 一段时间之后，接收到文件结束符的应用进程调用close关闭它的socket。这导致它的TCP也发送一个FIN N；
4. 接收到这个FIN的源发送端TCP对它进行确认。



**客fin服务 -> 服ack客 -> 服fin客 -> 客ack服务**

## UDP

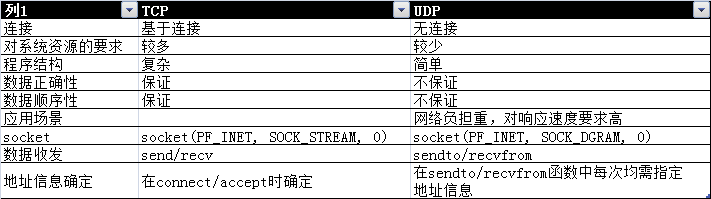
<https://zh.wikipedia.org/wiki/%E7%94%A8%E6%88%B7%E6%95%B0%E6%8D%AE%E6%8A%A5%E5%8D%8F%E8%AE%AE>

User Datagram Protocol，缩写为UDP，又称**用户数据报文协议**，是一个简单的**面向数据报的传输层协议。**

**UDP只提供数据的不可靠传递**，它一旦把应用程序发给网络层的数据发送出去，就**不保留数据备份**（所以UDP有时候也被认为是不可靠的数据报协议）。

UDP在IP数据报的头部仅仅加入了复用和数据校验（字段）。由于缺乏可靠性且属于非连接导向协议，UDP应用一般必须允许一定量的丢包、出错。流媒体（流技术）、即时多媒体游戏和IP电话（VoIP）一定就是典型的UDP应用。如果某个应用需要很高的可靠性，那么可以用传输控制协议（TCP协议）来代替UDP。

## 区别



# Socket

## 网络中进程之间如何通信？

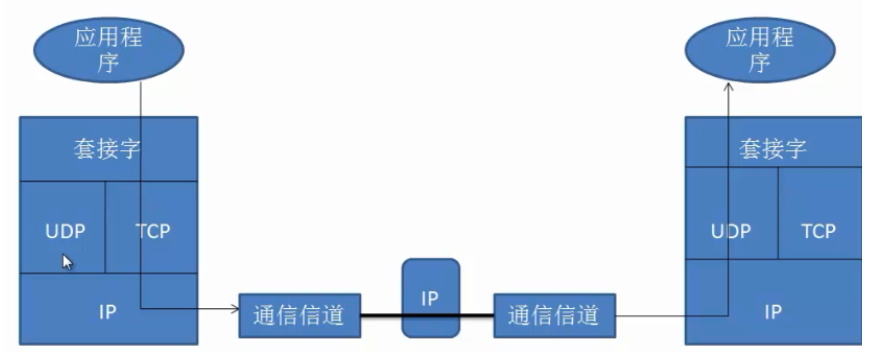
**网络层**的“ip地址”可以唯一标识网络中的主机;

**传输层**的“协议+端口”可以唯一标识主机中的应用程序（进程）。

这样利用**三元组（ip地址，协议，端口）就可以标识网络的进程**了，网络中的进程通信就可以利用这个标志与其它进程进行交互。

就目前而言，几乎所有的应用程序都是采用socket，而现在又是网络时代，网络中进程通信是无处不在，这就是我为什么说“一切皆socket”。

## Socket通信模型



**什么是Socket？**

上面我们已经知道网络中的进程是通过socket来通信的，那什么是socket呢？

socket起源于Unix，而Unix/Linux基本哲学之一就是“一切皆文件”，都可以用“打开open –> 读写write/read –> 关闭close”模式来操作。Socket就是该模式的一个实现，socket即是一种特殊的文件，一些socket函数就是对其进行的操作（读/写IO、打开、关闭）

## Scoket常用函数（使用步骤）

1. int **socket**(int domain, int type, int protocol);

创建socket

1. int **bind**(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

把一个地址族中的特定地址赋给socket

1. int **listen**(int sockfd, int backlog);

如果作为一个服务器，在调用socket()、bind()之后就会调用listen()来监听这个socket，如果客户端这时调用connect()发出连接请求，服务器端就会接收到这个请求。

1. int **connect**(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

客户端通过调用connect函数来建立与TCP服务器的连接。

1. int **accept**(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

TCP服务器端依次调用socket()、bind()、listen()之后，就会监听指定的socket地址了。

TCP客户端依次调用socket()、connect()之后就向TCP服务器发送了一个连接请求。

TCP服务器监听到这个请求之后，就会调用accept()函数接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了，即类同于普通文件的读写I/O操作

1. read()/write() 、recv()/send()、readv()/writev()、recvmsg()/sendmsg()、recvfrom()/sendto()

至此服务器与客户已经建立好连接了。可以调用网络I/O进行读写操作了，即实现了**网络中不同进程之间的通信**，开发语言不同可能读写函数也就不同，只要把自己想要发送的消息，以字节流的方式写入Socket或者从Socket读出来即可实现网络的I/O操作。

1. int **close**(int fd);

close一个TCP socket的缺省行为时**把该socket标记为以关闭**，然后立即返回到调用进程。该描述字不能再由调用进程使用，也就是说不能再作为read或write的第一个参数。注意：close操作只是使相应socket描述字的引用计数-1，只有当引用计数为0的时候，才会触发TCP客户端向服务器发送终止连接请求。

## Socket在TCP和UDP上的区别

**TCP编程的服务器端一般步骤是：**

1、创建一个socket，用函数socket()；

2、设置socket属性，用函数setsockopt(); \* 可选

3、绑定IP地址、端口等信息到socket上，用函数bind();

4、开启监听，用函数listen()；

5、接收客户端上来的连接，用函数**accept**()；

6、收发数据，用函数send()和recv()，或者read()和write();

7、关闭网络连接；

8、关闭监听；

**TCP编程的客户端一般步骤是：**

1、创建一个socket，用函数socket()；

2、设置socket属性，用函数setsockopt();\* 可选

3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

4、设置要连接的对方的IP地址和端口等属性；

5、连接服务器，用函数**connect**()；

6、收发数据，用函数send()和recv()，或者read()和write();

7、关闭网络连接；

与之对应的UDP编程步骤要简单许多，分别如下：

**UDP编程的服务器端一般步骤是：**

1、创建一个socket，用函数socket()；

2、设置socket属性，用函数setsockopt();\* 可选

3、绑定IP地址、端口等信息到socket上，用函数bind();

4、循环接收数据，用函数**recvfrom**();

5、关闭网络连接；

**UDP编程的客户端一般步骤是：**

1、创建一个socket，用函数socket()；

2、设置socket属性，用函数setsockopt();\* 可选

3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

4、设置对方的IP地址和端口等属性;

5、发送数据，用函数**sendto**();

6、关闭网络连接；

# 常见HTTP错误代码

<http://blog.csdn.net/xinxin19881112/article/details/6565823>

200 服务器成功返回网页

403 从缓存中获取页面，而不是从服务器中获取

404 请求的页面不存在

500 服务器内部错误

503 服务不可用

1XX 临时响应

2XX 成功处理了请求

3XX 重定向

4XX 请求错误

5XX 服务器错误

## 304

<https://www.douban.com/note/161120791/>

304 的标准解释是：Not Modified **客户端有缓冲的文档并发出了一个条件性的请求**（一般是提供If-Modified-Since头表示客户只想比指定日期更新的文档）。然后服务器告诉客户，文档自指定时间以来并没有修改，使用原来缓冲的文档即可。

如果客户端发起一个文件请求的时候，发现缓存的文件有Last modified，那么在请求中会包含if modified since，这个时间就是缓存文件的最后修改时间。因此，如果请求中包含 If Modified Since，就说明已经有缓存在客户端。只要判断这个时间和当前请求的文件的修改时间就可以确定是返回 304 还是 200 。

对于**静态文件**，如图片，css等，服务器会自动完成Last modified和if modified since的比较，完成缓存或者更新。

# HTTP请求报文

## HTTP请求行、请求头、请求体详解

<http://blog.csdn.net/u010256388/article/details/68491509>

http请求由三部分组成：请求行+请求头+请求体





①是请求方法，GET和POST是最常见的HTTP方法，除此以外还包括DELETE、HEAD、OPTIONS、PUT、TRACE。不过，当前的大多数浏览器只支持GET和POST，Spring 3.0提供了一个HiddenHttpMethodFilter，允许你通过“\_method”的表单参数指定这些特殊的HTTP方法（实际上还是通过POST提交表单）。服务端配置了HiddenHttpMethodFilter后，Spring会根据\_method参数指定的值模拟出相应的HTTP方法，这样，就可以使用这些HTTP方法对处理方法进行映射了。

②为请求对应的URL地址，它和报文头的Host属性组成完整的请求URL

③是协议名称及版本号。

④是HTTP的报文头，报文头包含若干个属性，格式为“属性名:属性值”，服务端据此获取客户端的信息。

⑤是报文体，它将一个页面表单中的组件值通过param1=value1&param2=value2的键值对形式编码成一个格式化串，它承载多个请求参数的数据。不但报文体可以传递请求参数，请求URL也可以通过类似于“/chapter15/user.html? param1=value1&param2=value2”的方式传递请求参数。

## 常见HTTP请求报文头属性

### Accept

请求报文可通过一个“Accept”报文头属性告诉服务端 客户端接受什么类型的响应。

Accept属性的值可为一个或多个MIME类型（MIME：<http://en.wikipedia.org/wiki/MIME_type>  ）

### Cookie

客户端的Cookie就是通过这个报文头属性传给服务端的，如：

Cookie: $Version=1; Skin=new;**jsessionid=5F4771183629C9834F8382E23BE13C4C**

服务端怎么知道客户端的请求属于哪一个session，就是通过jsessionid判断的。

### Referer

表示这个请求是从哪个URL过来的

### Cache-Control

对缓存进行控制，如一个请求希望响应返回的内容在客户端要被缓存一年，或不希望被缓存就可以通过这个报文头达到目的。如：Cache-Control：no-cache （服务端将对请求返回的响应内容不再客户端进行缓存）

### Accept-Language

客户端本地化信息

### Content-Length

请求报文体长度

### Content-Type

客户端发送的数据格式

### User-Agent

产生请求的浏览器类型

HttpServletRequest.getSession()的内部机制就是获得请求报文头中Cookie属性的JSESSIONID值。

# HTTP响应报文

由三部分组成：响应行+响应头+响应体



①报文协议及版本；

②状态码及状态描述；

③响应报文头，也是由多个属性组成；

④响应报文体，即我们真正要的“干货”。

HttpServletResponse.setStatus(int sc)：可设置响应状态码。

## 常见的HTTP响应报文头属性

<http://www.51testing.com/html/28/116228-238337.html>

### Expires

**过期时间**：属性是HTTP控制缓存的基本手段，这个属性告诉缓存器：相关副本在多长时间内是新鲜的。过了这个时间，缓存器就会向源服务器发送请求，检查文档是否被修改。

过期时间头信息属性值只能是HTTP格式的日期时间，其他的都会被解析成当前时间“之前”，副本会过期，记住：HTTP的日期时间必须是格林威治时 间（GMT），而不是本地时间。举例：

Expires: Fri, 30 Oct 1998 14:19:41

### Cache-Control

响应输出到客户端后，服务端通过该报文头属告诉客户端如何控制响应内容的缓存。

给静态资源(HTML文件，图片文件等）的Repsone加上Expires/Cache-Control Header是很有效的一招。如果HTTP Response中有Expires这样的Header的话，浏览器会Cache这个资源，理想状况下。

Expires/Cache-Control Header是控制浏览器是否直接从浏览器缓存取数据还是重新发请求到服务器取数据。只是Cache-Control比Expires可以控制的多一些， 而且Cache-Control会重写Expires的规则。

### Etag /If-None-Match

ETag 是实现与最近修改数据检查同样的功能的另一种方法：没有变化时不重新下载数据。其工作方式是：服务器发送你所请求的数据的同时，发送某种数据的 hash (在 ETag 头信息中给出)。hash 的确定完全取决于服务器。当第二次请求相同的数据时，你需要在 If-None-Match: 头信息中包含 ETag hash，如果数据没有改变，服务器将返回 304 状态代码。与最近修改数据检查相同，服务器仅仅 发送 304 状态代码；第二次将不为你发送相同的数据。在第二次请求时，通过包含 ETag hash，你告诉服务器：如果 hash 仍旧匹配就没有必要重新发送相同的数据，因为你还有上一次访问过的数据。

### Location

我们在JSP中让页面Redirect到一个某个A页面中，其实是让客户端再发一个请求到A页面，这个需要Redirect到的A页面的URL，其实就是通过**响应报文头的Location属性**告知客户端的，如下的报文头属性，将使客户端redirect到iteye的首页中，如：Location: <http://www.iteye.com>，这个响应报文属性将使客户端再次发送请求到指定url（即重定向）

### Set-Cookie

服务端可以设置客户端的Cookie，其原理就是通过这个响应报文头属性实现的，如：Set-Cookie: UserID=JohnDoe; Max-Age=3600; Version=1

HttpServletResponse.setHeader(String name,String value)：设置响应报文头属性。

addCookie，sendRedirect，方法内部都是修改了响应报文头。

### Last-Modified/If-Modified-Since

通常服务器知道你所请求的数据的最后修改时间，并且 HTTP 为服务器提供了一种将最近修改数据连同你请求的数据一同发送的方法。

如果你第二次 (或第三次，或第四次) 请求相同的数据，你可以告诉服务器你上一次获得的最后修改日期：在你的请求中发送一个 If-Modified-Since 头信息，它包含了上一次从服务器连同数据所获得的日期。如果数据从那时起没有改变，服务器将返回一个特殊的 HTTP 状态代码 304: Not Modified (没有修改)，这意味着 “从上一次请求后这个数据没有改变”。这一点有何进步呢？当服务器发送状态编码 304 时，不再重新发送数据。您仅仅获得了这个状态代码。所以当数据没有更新时，你不需要一次又一次地下载相同的数据；服务器假定你有本地的缓存数据。

# 四种常见的 POST 提交数据方式

<https://imququ.com/post/four-ways-to-post-data-in-http.html>

HTTP/1.1 协议规定的 HTTP 请求方法有 OPTIONS、**GET**、HEAD、**POST**、PUT、DELETE、TRACE、CONNECT 这几种。其中 POST 一般用来向服务端提交数据.

## application/x-www-form-urlencoded

最常见的 POST 提交数据的方式了。浏览器的原生 <form> 表单，如果不设置 enctype 属性，那么最终就会以 application/x-www-form-urlencoded 方式提交数据。

请求类似于下面这样（无关的请求头在本文中都省略掉了）：

POST http://www.example.com HTTP/1.1

Content-Type: **application/x-www-form-urlencoded**;charset=utf-8

**title=test&sub%5B%5D=1&sub%5B%5D=2&sub%5B%5D=3**

提交的数据按照 key1=val1&key2=val2 的方式进行编码，key 和 val 都进行了 URL 转码。

很多时候，我们用 Ajax 提交数据时，也是使用这种方式。

## multipart/form-data

使用表单上传文件时，必须让 <form> 表单的 enctype 等于 multipart/form-data。

POST http://www.example.com HTTP/1.1

Content-Type:**multipart/form-data**; boundary=----WebKitFormBoundaryrGKCBY7qhFd3TrwA

------WebKitFormBoundaryrGKCBY7qhFd3TrwA

Content-Disposition: form-data; name="text"

title

------WebKitFormBoundaryrGKCBY7qhFd3TrwA

Content-Disposition: form-data; name="file"; filename="chrome.png"

Content-Type: image/png

PNG ... content of chrome.png ...

------WebKitFormBoundaryrGKCBY7qhFd3TrwA—

首先生成了一个 boundary 用于**分割**不同的字段，为了避免与正文内容重复，boundary 很长很复杂。然后 Content-Type 里指明了数据是以 **multipart/form-data** 来编码，本次请求的 boundary 是什么内容。消息主体里按照字段个数又分为多个结构类似的部分，每部分都是以 --boundary 开始，紧接着是**内容描述信息**，然后是回车，最后是字段具体内容（**文本或二进制**）。如果传输的是文件，还要包含文件名和文件类型信息。消息主体最后以 --boundary-- 标示结束。

**这种方式一般用来上传文件。**

上面提到的这两种 POST 数据的方式，都是浏览器原生支持的，而且现阶段标准中原生 <form> 表单也只支持这两种方式（通过 <form> 元素的 enctype 属性指定，默认为 application/x-www-form-urlencoded。其实 enctype 还支持 text/plain，不过用得非常少）。

## applation/json

JSON 格式支持比键值对复杂得多的结构化数据，这一点也很有用。可以方便的提交复杂的结构化数据，特别适合 RESTful 的接口。

POST http://www.example.com HTTP/1.1

Content-Type: **application/json**;charset=utf-8

{"title":"test","sub":[1,2,3]}

## test/xml

XML-RPC（XML Remote Procedure Call）。它是一种使用 HTTP 作为传输协议，XML 作为编码方式的远程调用规范。