## **HTTP /302 redirect: 302 代表暂时性转移(Temporarily Moved )。**

## **HTTP/206 “Partial Content”响应是在客户端表明自己只需要目标URL上的部分资源的时候返回的.这种情况经常发生在客户端继续请求一个未完成的下载的时候(通常是当客户端加载一个体积较大的嵌入文件,比如视屏或PDF文件 ),或者是客户端尝试实现带宽遏流的时候.**

## **请求(Request)**

一个完整的HTTP请求包括：**一个请求行、若干请求头、以及实体内容**

### **请求头字段**

* Accept:用于告诉服务器，**客户机支持的数据类型**
* Accept-Charset:用于告诉服务器，**客户机采用的编码**
* Accept-Encoding:用于告诉服务器，**客户机支持数据压缩格式**
* Accept-Language:**客户机的语言环境**
* Host:客户机通过这个头告诉服务器，**想访问的主机名**
* If-Modified-Since:客户机通过这个头告诉服务器，**资源的缓存时间**
* Refer:客户机通过这个头告诉服务器，它是**从哪个资源访问服务器的**(防盗链)
* User-Agent:客户机通过这个头告诉服务器，**客户机的软件环境**
* Cookie:客户机通过这个头**向服务器带数据**
* Connection:这个请求完了，**是保持连接还是关闭**
* **Range:断点下载**
  + bytes=n1-n2,传输范围n1到n2字节
  + bytes=n-，传输web资源中第n个字节以后的所有内容
  + bytes=n,传输最后n个字节

## **响应(Response)**

一个HTTP响应代表服务器向客户端回送的数据，包括：**一个状态行、若干响应头、以及实体内容**

### **响应状态行**

[](https://camo.githubusercontent.com/11b8dccb06e9b7b2fcbd479107cf068eb9ce711b/687474703a2f2f3778706836642e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f6a6176617765625f48747470526573706f6e73655374617475732e706e67" \t "https://github.com/brianway/java-learning/blob/master/blogs/javaweb/_blank)

详情可参考

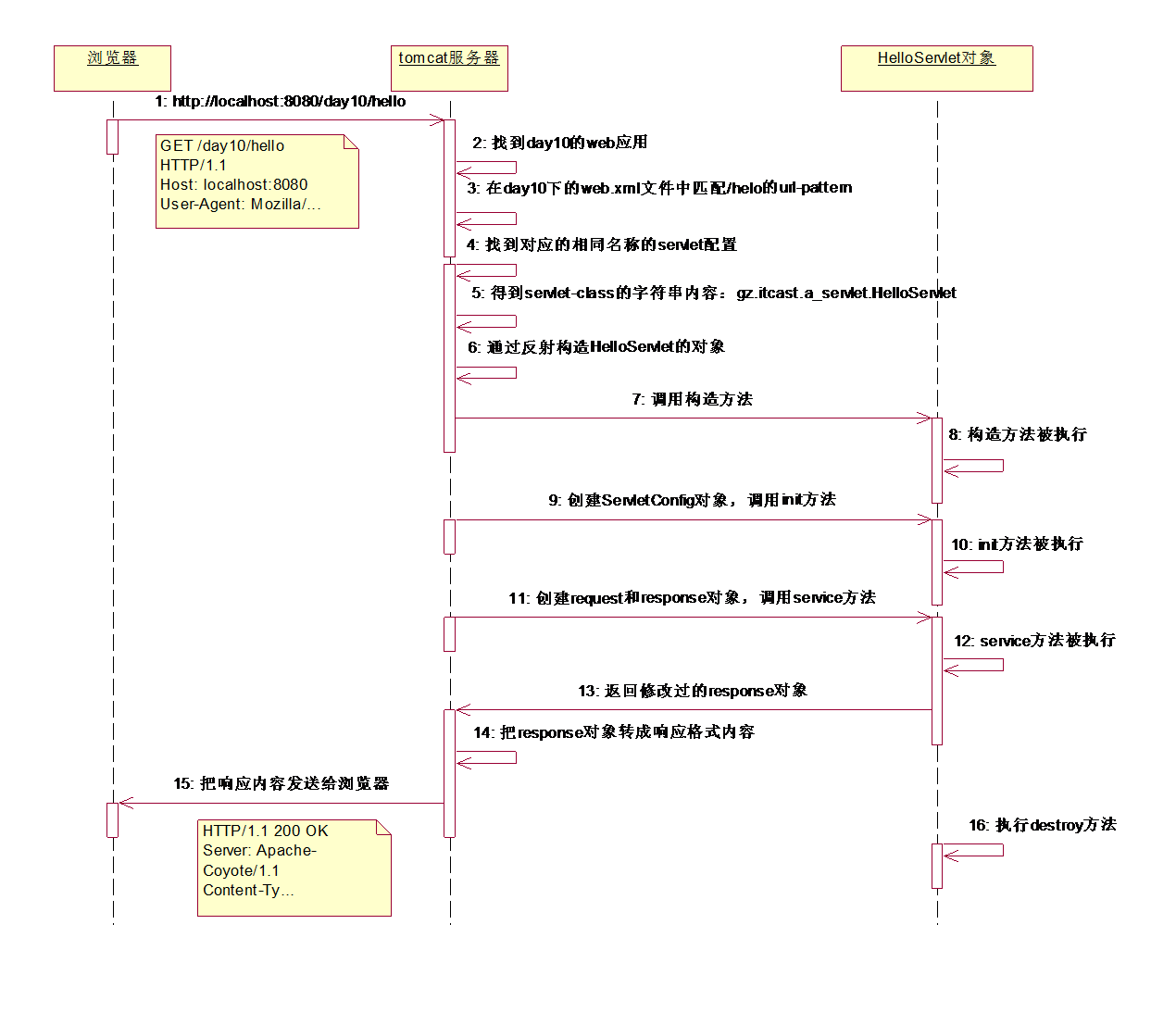
[HTTP 状态消息](http://www.w3school.com.cn/tags/html_ref_httpmessages.asp)

### **响应头字段**

* **Location:这个头配合302状态码使用，用于告诉客户机找谁(location和302实现请求重定向)**
* Server:服务器通过这个头，**告诉浏览器服务器的类型**
* Content-Encoding:服务器通过这个头，**数据的压缩格式** (相关java知识:GZIPOutputStream,包装流/底层流)
* Content-Length:服务器通过这个头，告诉浏览器**回送数据的长度**
* Content-Type:**服务器通过这个头，告诉浏览器回送数据的类型**
* Last-Modified:服务器通过这个头，**告诉浏览器当前资源的缓存时间**
* Refresh:服务器通过这个头，**告诉浏览器隔多长时间刷新一次**
* Content-Disposition:服务器通过这个头，**告诉浏览器以下载方式打开**
* Transfer-Encoding:服务器通过这个头，**告诉浏览器数据的传送格式**
* Etag:缓存相关的头部，用于实时性要求高的系统
* **Expires**:服务器通过这个头，告诉浏览器把**回送的资源缓存多长时间，-1或0则不缓存**
* Cache-Control和Pragma:no-cache,服务器通过这两个头，也是控制浏览器不要缓存数据
* Connection:**断开连接/保持连接**
* Date:**当前时间**
* Accept-Ranges:**用来说明web服务器是否支持range**。支持返回bytes;不支持返回none
* Content-Range:制定了返回web资源的字节范围，格式：n1-n2/n\_total

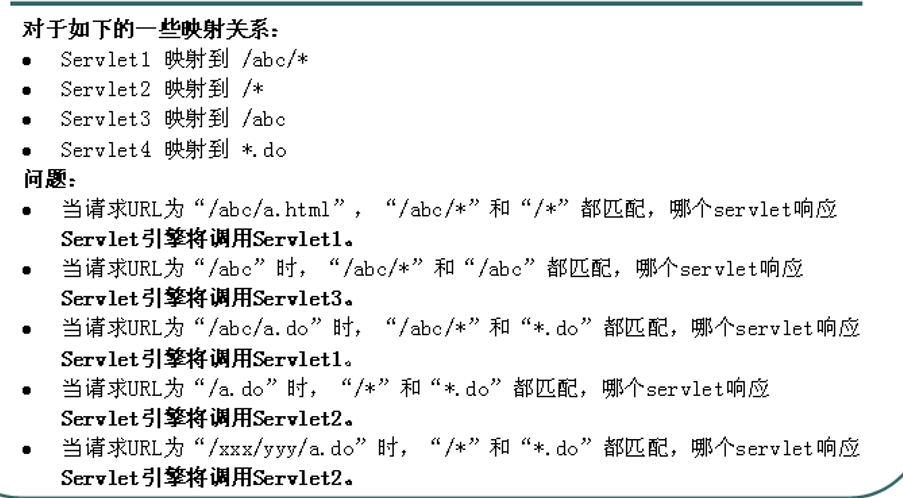
## **servlet的调用过程和生命周期**

时序图

[](https://camo.githubusercontent.com/4add5ddf8c0da7d49f0167902fab4c81f56ee167/687474703a2f2f3778706836642e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f6a6176617765625f736572766c65742d6c6966656379636c652e706e67" \t "https://github.com/brianway/java-learning/blob/master/blogs/javaweb/_blank)

## **servlet开发的一些细节**

* ****标签****:<servlet>包含<servlet-name>和<servlet-class>；<servlet-mapping>包含<servlet-name>和<url-pattern>
* ****映射****:web.xml中一个<servlet>可对应多个<servlet-mapping>
* ****通配符****:<servlet-mapping>的<url-pattern>可以使用通配符，两种固定格式：\*.扩展名；以/开头，以/\*结尾

[](https://camo.githubusercontent.com/19554cb1969db13be8079587b757946d3e847ad8/687474703a2f2f3778706836642e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f6a6176617765625f736572766c65742d75726c2545352538432542392545392538352538442e706e67" \t "https://github.com/brianway/java-learning/blob/master/blogs/javaweb/_blank)

* ****对象****:servlet由servlet引擎调用，不能独立运行。客户端多次请求，服务器只创建一个servlet实例，之后驻留内存中继续服务直至web容器退出才销毁它。
* ****请求****:服务器针对客户端的每一次请求都会创建新的request和response对象(它们的生命周期很短)，传给service方法。
* ****加载****:servlet实例的创建和init方法的调用是在第一次请求时，而非服务器启动时，除非在<servlet>标签配置<load-on-start-up>,数字越小优先级越高
* ****缺省****:映射路径为正斜杠/，则为当前web应用的缺省servlet,不匹配的都交给缺省
* ****线程安全****:访问同一资源会引发线程安全问题; SingleThreadModel标记接口(已弃用)
* ****ServletConfig****:在<servlet>标签配置<init-param>，通过getServletConfig方法获得配置。可配置输出字符集，读哪个配置文件等等。
* ****ServletContext****:代表当前web应用，含有一些web应用全局性方法，实现web资源共享、servlet转发等。通过ServletConfig.getServletContext方法获得，在<context-param>标签配置。

## **ServletContext**

API:[Interface ServletContext](https://tomcat.apache.org/tomcat-8.0-doc/servletapi/index.html?javax/servlet/ServletContext.html)

概念

一个web应用所有servlet共享同一个ServletContext对象，可实现数据共享。ServletContext被称为context****域****对象。

域：作用范围；context域：整个应用程序范围。

ServletContext域： 1. 这是一个容器 2. 说明了这个容器的作用范围，也就是应用程序范围

**转发:客户机一次请求；重定向：客户机两次请求**

作用

* 获取web应用的初始化参数
* 实现servlet转发
* 利用ServletContext对象读取资源文件
  + 获得文件路径
  + 读取资源文件的三种方式
  + .properties文件（属性文件）

配置文件：properties文件和xml文件；数据有关系使用xml文件，没有关系则使用properties文件。

1.通过ServletContext的getResourceAsStream方法，读取properties文件

模板代码(注意文件位置不同写路径会不同)：

InputStream in = this.getServletContext().getResourceAsStream("/WEB-INF/classes/org/iot/servlet/db.properties");

Properties properties = new Properties();//map

properties.load(in);

2.通过servletContext的getRealPath方法得到资源的绝对路径，再通过传统方式(FileInputStream)读取

3.通过类装载器去读，ClassLoader的getResourceAsStream(如果读取资源文件的程序不是servlet)，文件不能太大。****只装载一次****，所以如要读到更新后的数据，通过类装载的方式得到资源文件的位置，再通过传统方式读取资源文件的数据（用getResource得到path，再用FileInputStream）

****HTTP与TCP/IP区别？****

**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**。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。

下面的图表试图显示不同的TCP/IP和其他的协议在最初OSI（Open System Interconnect）模型中的位置：

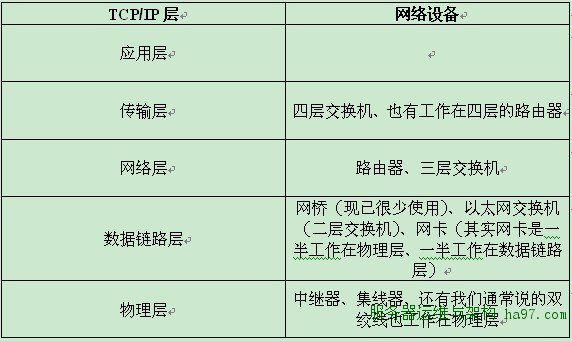
RIP：内部网关协议，一种动态路由协议

OSPF：开放式最短路径优先协议。网络层，作用域一个自治系统

BGP：边界网关协议，基于TCP的动态路由协议，用来连接好几个自治系统







**除了层的数量之外，开放式系统互联（OSI）模型与TCP/IP协议有什么区别？**

开放式系统互联模型是一个参考标准，解释协议相互之间应该如何相互作用。TCP/IP协议是美国国防部发明的，是让互联网成为了目前这个样子的标准之一。开放式系统互联模型中没有清楚地描绘TCP/IP协议，但是在解释TCP/IP协议时很容易想到开放式系统互联模型。两者的主要区别如下：

TCP/IP协议中的应用层处理开放式系统互联模型中的第五层、第六层和第七层的功能。

TCP/IP协议中的传输层并不能总是保证在传输层可靠地传输数据包，而开放式系统互联模型可以做到。TCP/IP协议还提供一项名为UDP（用户数据报协议）的选择。UDP不能保证可靠的数据包传输。

**TCP/UDP协议**

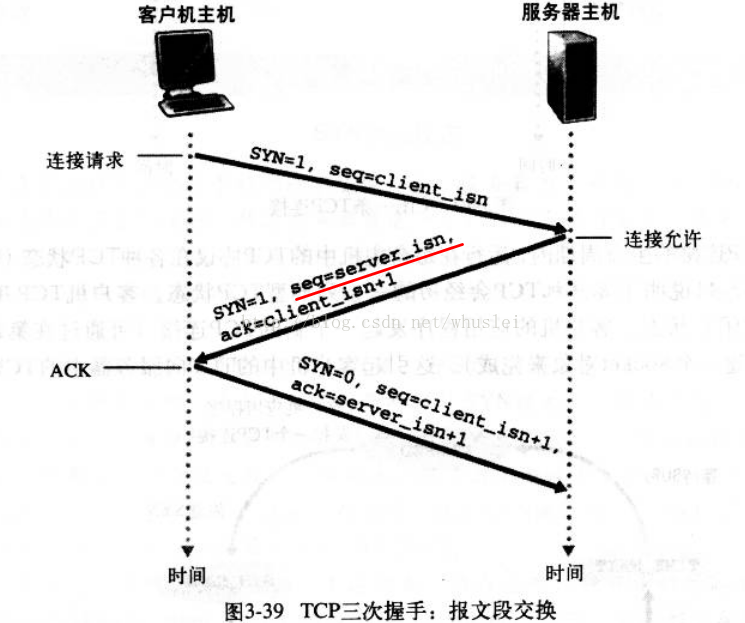
TCP(Transmission Control Protocol)和UDP(User Datagram Protocol)协议属于传输层协议。其中TCP提供IP环境下的数据可靠传输，它提供的服务包括数据流传送、可靠性、有效流控、全双工操作和多路复用。通过面向连接、端到端和可靠的数据包发送。通俗说，它是事先为所发送的数据开辟出连接好的通道，然后再进行数据发送；而UDP则不为IP提供可靠性、流控或差错恢复功能。一般来说，TCP对应的是可靠性要求高的应用，而UDP对应的则是可靠性要求低、传输经济的应用。

TCP支持的应用协议主要 有：Telnet、FTP、SMTP等；UDP支持的应用层协议主要有：NFS（网络文件系统）、SNMP（简单网络管理协议）、DNS（主域名称系 统）、TFTP（通用文件传输协议）等.  
TCP/IP协议与低层的数据链路层和物理层无关，这也是TCP/IP的重要特点。

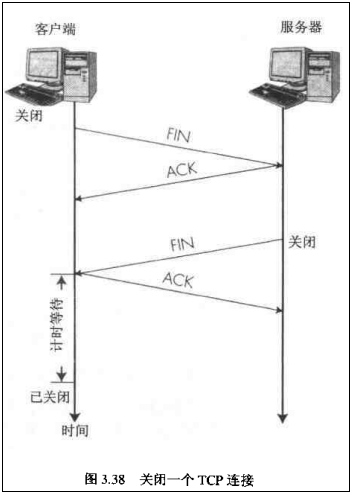
****HTTP三次握手****

HTTP（HyperText Transfer Protocol)超文本传输协议是互联网上应用最为广泛的一种网络协议。由于信息是明文传输，所以被认为是不安全的。而关于HTTP的三次握手，其实就是使用三次TCP握手确认建立一个HTTP连接。

如下图所示，SYN（synchronous）是TCP/IP建立连接时使用的握手信号、Sequence number（序列号）、Acknowledge number（确认号码），三个箭头指向就代表三次握手，完成三次握手，客户端与服务器开始传送数据。



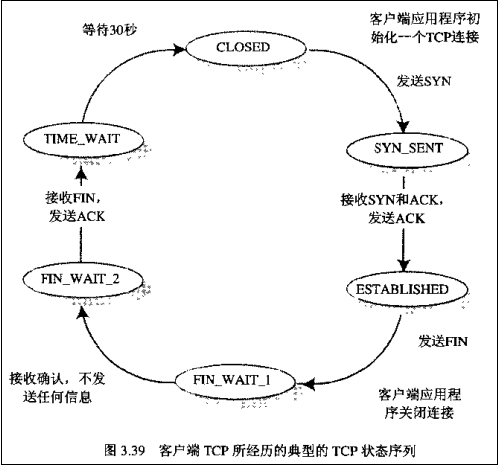
首先Client端发送连接请求报文，Server段接受连接后回复ACK报文，并为这次连接分配资源。Client端接收到ACK报文后也向Server段发生ACK报文，并分配资源，这样TCP连接就建立了。



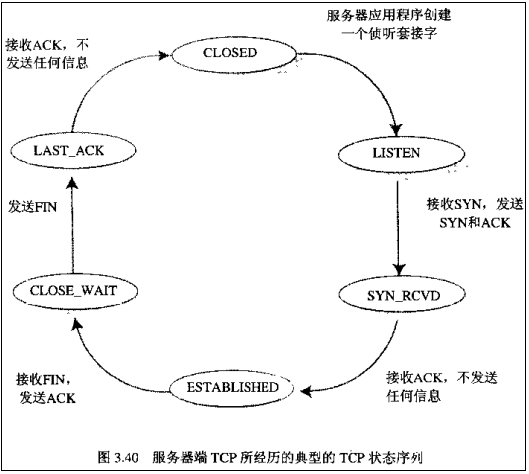
【注意】中断连接端可以是Client端，也可以是Server端。

假设Client端发起中断连接请求，也就是发送FIN报文。Server端接到FIN报文后，意思是说"我Client端没有数据要发给你了"，但是如果你还有数据没有发送完成，则不必急着关闭Socket，可以继续发送数据。所以你先发送ACK，"告诉Client端，你的请求我收到了，但是我还没准备好，请继续你等我的消息"。这个时候Client端就进入FIN\_WAIT状态，继续等待Server端的FIN报文。当Server端确定数据已发送完成，则向Client端发送FIN报文，"告诉Client端，好了，我这边数据发完了，准备好关闭连接了"。Client端收到FIN报文后，"就知道可以关闭连接了，但是他还是不相信网络，怕Server端不知道要关闭，所以发送ACK后进入TIME\_WAIT状态，如果Server端没有收到ACK则可以重传。“，Server端收到ACK后，"就知道可以断开连接了"。Client端等待了2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，我Client端也可以关闭连接了。Ok，TCP连接就这样关闭了！

整个过程Client端所经历的状态如下：



而Server端所经历的过程如下：



【注意】 在TIME\_WAIT状态中，如果TCP client端最后一次发送的ACK丢失了，它将重新发送。TIME\_WAIT状态中所需要的时间是依赖于实现方法的。典型的值为30秒、1分钟和2分钟。等待之后连接正式关闭，并且所有的资源(包括端口号)都被释放。

【问题1】为什么连接的时候是三次握手，关闭的时候却是四次握手？  
答：因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步握手。

【问题2】为什么TIME\_WAIT状态需要经过2MSL**(最大报文段生存时间)**才能返回到CLOSE状态？

答：虽然按道理，四个报文都发送完毕，我们可以直接进入CLOSE状态了，但是我们必须假象网络是不可靠的，有可以最后一个ACK丢失。所以TIME\_WAIT状态就是用来重发可能丢失的ACK报文。

什么是2MSL？MSL即Maximum Segment Lifetime，也就是报文最大生存时间，引用《TCP/IP详解》中的话：“它(MSL)是任何报文段被丢弃前在网络内的最长时间。”那么，2MSL也就是这个时间的2倍。其实我觉得没必要把这个MSL的确切含义搞明白，你所需要明白的是，当TCP连接完成四个报文段的交换时，主动关闭的一方将继续等待一定时间(2-4分钟)，即使两端的应用程序结束。

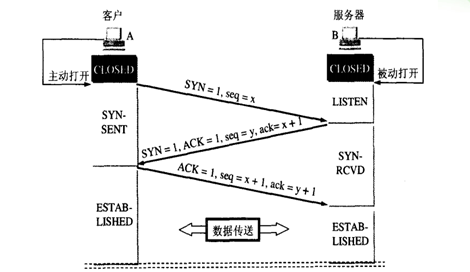
为什么需要2MSL？根据《TCP/IP详解》和《The TCP/IP Guide》中的说法，有两个原因：

   其一，保证发送的ACK会成功发送到对方，如何保证？我觉得可能是通过超时计时器发送。这个就很难用代码演示了。

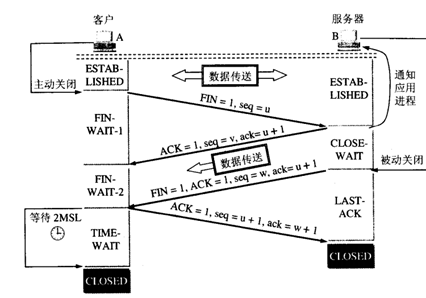
   其二，报文可能会被混淆，意思是说，其他时候的连接可能会被当作本次的连接。直接引用《The TCP/IP Guide》的说法：The second is to provide a “buffering period” between the end of this connection and any subsequent ones. If not for this period, it is possible that packets from different connections could be mixed, creating confusion.

**常见面试题**

* + **TCP协议和UDP协议的区别是什么**
    - **TCP协议是有连接的**，有连接的意思是开始传输实际数据之前TCP的客户端和服务器端必须通过三次握手建立连接，会话结束之后也要结束连接。而UDP是无连接的
    - **TCP协议保证数据按序发送，按序到达，提供超时重传来保证可靠性**，但是UDP不保证按序到达，甚至不保证到达，只是努力交付，即便是按序发送的序列，也不保证按序送到。
    - **TCP协议所需资源多，TCP首部需20个字节（不算可选项），UDP首部字段只需8个字节。**
    - **TCP有流量控制和拥塞控制，UDP没有，网络拥堵不会影响发送端的发送速率**
    - **TCP是一对一的连接，而UDP则可以支持一对一，多对多，一对多的通信。**
    - **TCP面向的是字节流的服务，UDP面向的是报文的服务。**
    - [TCP介绍](http://www.cnblogs.com/newwy/archive/2013/08/02/3232503.html)和UDP介绍
  + 请详细介绍一下TCP协议建立连接和终止连接的过程？
    - 助于理解的[一段话](http://www.cppblog.com/MemoryGarden/archive/2011/07/02/150007.html)
    - 两幅图（[来源](http://www.cnblogs.com/newwy/archive/2013/08/03/3234536.html)）：
      * 建立连接：三次握手

[](http://images.cnitblog.com/blog/385532/201308/30193701-72f027c0ea0f4c69834076466fa5383c.png)

* + - * 关闭连接：四次挥手

[](http://images.cnitblog.com/blog/385532/201308/30193703-63640062b79a4fc8b8ed31e95fd87bd8.png)

* + 三次握手建立连接时，发送方再次发送确认的必要性？

主 要是为了防止已失效的连接请求报文段突然又传到了B,因而产生错误。假定出现一种异常情况，即A发出的第一个连接请求报文段并没有丢失，而是在某些网络结 点长时间滞留了，一直延迟到连接释放以后的某个时间才到达B，本来这是一个早已失效的报文段。但B收到此失效的连接请求报文段后，就误认为是A又发出一次 新的连接请求，于是就向A发出确认报文段，同意建立连接。假定不采用三次握手，那么只要B发出确认，新的连接就建立了，这样一直等待A发来数据，B的许多 资源就这样白白浪费了。

* + **四次挥手释放连接时，等待2MSL的意义？**
    - * 第 一，为了保证A发送的最有一个ACK报文段能够到达B。这个ACK报文段有可能丢失，因而使处在LAST-ACK状态的B收不到对已发送的FIN和ACK 报文段的确认。B会超时重传这个FIN和ACK报文段，而A就能在2MSL时间内收到这个重传的ACK+FIN报文段。接着A重传一次确认。
      * 第二，就是防止上面提到的已失效的连接请求报文段出现在本连接中，A在发送完最有一个ACK报文段后，再经过2MSL，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失。

* + 常见的应用中有哪些是应用TCP协议的，哪些又是应用UDP协议的，为什么它们被如此设计？
    - 以下应用一般或必须用udp实现？
      * **多播的信息一定要用udp实现，因为tcp只支持一对一通信。**
      * **如果一个应用场景中大多是简短的信息，适合用udp实现，因为udp是基于报文段的，它直接对上层应用的数据封装成报文段，然后丢在网络中，如果信息量太大，会在链路层中被分片，影响传输效率。**
      * **如果一个应用场景重性能甚于重完整性和安全性，那么适合于udp**，比如多媒体应用，缺一两帧不影响用户体验，但是需要流媒体到达的速度快，因此比较适合用udp
      * **如果要求快速响应，那么udp听起来比较合适**
      * 如果又要利用udp的快速响应优点，又想可靠传输，那么只能考上层应用自己制定规则了。
      * 常见的使用udp的例子：**ICQ,QQ的聊天模块**。
    - 以qq为例的一个说明（转载自[知乎](http://www.zhihu.com/question/20292749)）

**登陆采用TCP协议和HTTP协议，你和好友之间发送消息，主要采用UDP协议**，内网传文件采用了P2P技术。总来的说：   
1.登陆过程，客户端client 采用TCP协议向服务器server发送信息，HTTP协议下载信息。登陆之后，会有一个TCP连接来保持在线状态。   
2.和好友发消息，客户端client采用UDP协议，但是需要通过服务器转发。**腾讯为了确保传输消息的可靠，采用上层协议来保证可靠传输**。如果消息发送失败，客户端会提示消息发送失败，并可重新发送。   
3.如果是在内网里面的两个客户端传文件，QQ采用的是P2P技术，不需要服务器中转。

[](http://images.cnitblog.com/blog/385532/201308/30193704-d65f24c7ddea474ab3236bd701650059.png)

物理层：通过媒介传输比特,确定机械及电气规范（比特Bit）

数据链路层：将比特组装成帧和点到点的传递（帧Frame）

网络层：负责数据包从源到宿的传递和网际互连（包Packet）

传输层：提供端到端的可靠报文传递和错误恢复（段Segment）

会话层：建立、管理和终止会话（会话协议数据单元SPDU）

表示层：对数据进行翻译、加密和压缩（表示协议数据单元PPDU）

应用层：允许访问OSI环境的手段（应用协议数据单元APDU）

物理层： 透明地传输比特流

数据链路层： 建立逻辑连接、进行硬件地址寻址、差错效验等功能。

网络层： 进行逻辑地址寻址，实现不同网络之间的路径选择。

传输层： 定义传输数据的协议端口号，以及流控和差错效验。

会话层： 建立、管理、终止会话。（在五层模型里面已经合并到了应用层） 对应主机进程，指本地主机与远程主机正在进行的会话

表示层： 数据的表示、安全、压缩。（在五层模型里面已经合并到了应用层）

应用层： 网络服务与最终用户的一个接口。

1、物理层

设备：网线光纤、同轴电缆、集线器(HUB)、Modor

功能：物理拓扑

2、数据链路层

设备：二层交换机、网桥、数据帧

功能：组帧

3、网络层

设备：路由器、防火墙

功能：路由选择

4、传输层

功能：端到端连接

5、会话层

功能：会话控制

6、表示层

功能：数据加密、压缩和转换服务

7、应用层

功能：文件、打印、消息、数据库和应用服务

**错误代码(也称作状态代码)，指为服务器所接收每个请求(网页点击)分配的 3 位数代码。多数有效网页点击都有状态代码 200("正常")。"网页未找到"错误会生产 404 错误。某些常见的代码以粗体显示。**

1xx(临时响应)

用于表示临时响应并需要请求者执行操作才能继续的状态代码。

代码 说明

100(继续) 请求者应当继续提出请求。服务器返回此代码则意味着，服务器已收到了请求的第一部分，现正在等待接收其余部分。

101(切换协议) 请求者已要求服务器切换协议，服务器已确认并准备进行切换。

2xx(成功)

用于表示服务器已成功处理了请求的状态代码。

代码 说明

200(成功) 服务器已成功处理了请求。通常，这表示服务器提供了请求的网页。如果您的 robots.txt文件显示为此状态，那么，这表示 Googlebot 已成功检索到该文件。

201(已创建) 请求成功且服务器已创建了新的资源。

202(已接受) 服务器已接受了请求，但尚未对其进行处理。

203(非授权信息) 服务器已成功处理了请求，但返回了可能来自另一来源的信息。

204(无内容) 服务器成功处理了请求，但未返回任何内容。

205(重置内容) 服务器成功处理了请求，但未返回任何内容。与 204 响应不同，此响应要求请求者重置文档视图(例如清除表单内容以输入新内容)。

206(部分内容) 服务器成功处理了部分 GET 请求。

3xx(已重定向)

要完成请求，您需要进一步进行操作。通常，这些状态代码是永远重定向的。Google 建议您在每次请求时使用的重定向要少于 5 个。您可以使用网站管理员工具来查看 Googlebot 在抓取您已重定向的网页时是否会遇到问题。诊断下的抓取错误页中列出了 Googlebot 由于重定向错误而无法抓取的网址。

代码 说明

300(多种选择) 服务器根据请求可执行多种操作。服务器可根据请求者 (User agent) 来选择一项操作，或提供操作列表供请求者选择。

301(永久移动) 请求的网页已被永久移动到新位置。服务器返回此响应(作为对 GET 或 HEAD 请求的响应)时，会自动将请求者转到新位置。您应使用此代码通知 Googlebot 某个网页或网站已被永久移动到新位置。

302(临时移动) 服务器目前正从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。此代码与响应 GET 和 HEAD 请求的 301 代码类似，会自动将请求者转到不同的位置。但由于 Googlebot 会继续抓取原有位置并将其编入索引，因此您不应使用此代码来通知 Googlebot 某个页面或网站已被移动。

303(查看其他位置) 当请求者应对不同的位置进行单独的 GET 请求以检索响应时，服务器会返回此代码。对于除 HEAD 请求之外的所有请求，服务器会自动转到其他位置。

304(未修改) 自从上次请求后，请求的网页未被修改过。服务器返回此响应时，不会返回网页内容。

如果网页自请求者上次请求后再也没有更改过，您应当将服务器配置为返回此响应(称为 If-Modified-Since HTTP 标头)。由于服务器可以告诉 Googlebot 自从上次抓取后网页没有更改过，因此可节省带宽和开销。

305(使用代理) 请求者只能使用代理访问请求的网页。如果服务器返回此响应，那么，服务器还会指明请求者应当使用的代理。

307(临时重定向) 服务器目前正从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。此代码与响应 GET 和 HEAD 请求的 301 代码类似，会自动将请求者转到不同的位置。但由于 Googlebot 会继续抓取原有位置并将其编入索引，因此您不应使用此代码来通知 Googlebot 某个页面或网站已被移动。

4xx(请求错误)

这些状态代码表示，请求可能出错，已妨碍了服务器对请求的处理。

代码 说明

400(错误请求) 服务器不理解请求的语法。

401(未授权) 请求要求进行身份验证。登录后，服务器可能会返回对页面的此响应。

403(已禁止) 服务器拒绝请求。如果在 Googlebot 尝试抓取您网站上的有效网页时显示此状态代码(您可在 Google 网站管理员工具中诊断下的网络抓取页面上看到此状态代码)，那么，这可能是您的服务器或主机拒绝 Googlebot 对其进行访问。

404(未找到) 服务器找不到请求的网页。例如，如果请求是针对服务器上不存在的网页进行的，那么，服务器通常会返回此代码。

如果您的网站上没有 robots.txt 文件，而您在 Google 网站管理员工具"诊断"标签的 robots.txt 页上发现此状态，那么，这是正确的状态。然而，如果您有 robots.txt 文件而又发现了此状态，那么，这说明您的 robots.txt 文件可能是命名错误或位于错误的位置。(该文件应当位于顶级域名上，且应当名为 robots.txt)。

如果您在 Googlebot 尝试抓取的网址上发现此状态(位于"诊断"标签的 HTTP 错误页上)，那么，这表示 Googlebot 所追踪的可能是另一网页中的无效链接(旧链接或输入有误的链接)。

405(方法禁用) 禁用请求中所指定的方法。

406(不接受) 无法使用请求的内容特性来响应请求的网页。

407(需要代理授权) 此状态代码与 401(未授权)类似，但却指定了请求者应当使用代理进行授权。如果服务器返回此响应，那么，服务器还会指明请求者应当使用的代理。

408(请求超时) 服务器等候请求时超时。

409(冲突) 服务器在完成请求时发生冲突。服务器必须包含有关响应中所发生的冲突的信息。服务器在响应与前一个请求相冲突的 PUT 请求时可能会返回此代码，同时会提供两个请求的差异列表。

410(已删除) 如果请求的资源已被永久删除，那么，服务器会返回此响应。该代码与 404(未找到)代码类似，但在资源以前有但现在已经不复存在的情况下，有时会替代 404 代码出现。如果资源已被永久删除，那么，您应当使用 301 代码指定该资源的新位置。

411(需要有效长度) 服务器不会接受包含无效内容长度标头字段的请求。

412(未满足前提条件) 服务器未满足请求者在请求中设置的其中一个前提条件。

413(请求实体过大) 服务器无法处理请求，因为请求实体过大，已超出服务器的处理能力。

414(请求的 URI 过长) 请求的 URI(通常为网址)过长，服务器无法进行处理。

415(不支持的媒体类型) 请求的格式不受请求页面的支持。

416(请求范围不符合要求) 如果请求是针对网页的无效范围进行的，那么，服务器会返回此状态代码。

417(未满足期望值) 服务器未满足"期望"请求标头字段的要求。

5xx(服务器错误)

这些状态代码表示，服务器在尝试处理请求时发生内部错误。这些错误可能是服务器本身的错误，而不是请求出错。

代码 说明

500(服务器内部错误) 服务器遇到错误，无法完成请求。

501(尚未实施) 服务器不具备完成请求的功能。例如，当服务器无法识别请求方法时，服务器可能会返回此代码。

502(错误网关) 服务器作为网关或代理，从上游服务器收到了无效的响应。

503(服务不可用) 目前无法使用服务器(由于超载或进行停机维护)。通常，这只是一种暂时的状态。

504(网关超时) 服务器作为网关或代理，未及时从上游服务器接收请求。

505(HTTP 版本不受支持) 服务器不支持请求中所使用的 HTTP 协议版本。

# [HTTP 2.0与HTTP 1.1区别](http://www.cnblogs.com/frankyou/p/6145485.html)

### **1、什么是HTTP 2.0**

HTTP/2（超文本传输协议第2版，最初命名为HTTP 2.0），是HTTP协议的的第二个主要版本，使用于万维网。HTTP/2是HTTP协议自1999年HTTP 1.1发布后的首个更新，主要基于**SPDY协议**（是Google开发的基于TCP的应用层协议，用以最小化网络延迟，提升网络速度，优化用户的网络使用体验）。

### **2、与HTTP 1.1相比，主要区别包括**

1. **HTTP/2采用二进制格式而非文本格式**
2. **HTTP/2是完全多路复用的，而非有序并阻塞的——只需一个连接即可实现并行**
3. **使用报头压缩，HTTP/2降低了开销**
4. **HTTP/2让服务器可以将响应主动“推送”到客户端缓存中**
5. **对请求划分优先级；**

### **3、HTTP/2为什么是二进制？**

比起像HTTP/1.x这样的文本协议，二进制协议解析起来更高效、“线上”更紧凑，更重要的是错误更少。

### **4、为什么 HTTP/2 需要多路传输?**

HTTP/1.x 有个问题叫线端阻塞(head-of-line blocking), 它是指一个连接(connection)一次只提交一个请求的效率比较高, 多了就会变慢。 HTTP/1.1 试过用流水线(pipelining)来解决这个问题, 但是效果并不理想(数据量较大或者速度较慢的响应, 会阻碍排在他后面的请求). 此外, 由于网络媒介(intermediary )和服务器不能很好的支持流水线, 导致部署起来困难重重。而多路传输(Multiplexing)能很好的解决这些问题, 因为它能同时处理多个消息的请求和响应; 甚至可以在传输过程中将一个消息跟另外一个掺杂在一起。所以客户端只需要一个连接就能加载一个页面。

### **5、消息头为什么需要压缩?**

假定一个页面有80个资源需要加载（这个数量对于今天的Web而言还是挺保守的）, 而每一次请求都有1400字节的消息头（着同样也并不少见，因为Cookie和引用等东西的存在）, 至少要7到8个来回去“在线”获得这些消息头。这还不包括响应时间——那只是从客户端那里获取到它们所花的时间而已。这全都由于TCP的慢启动机制，它会基于对已知有多少个包，来确定还要来回去获取哪些包 – 这很明显的限制了最初的几个来回可以发送的数据包的数量。相比之下，即使是头部轻微的压缩也可以是让那些请求只需一个来回就能搞定——有时候甚至一个包就可以了。这种开销是可以被节省下来的，特别是当你考虑移动客户端应用的时候，即使是良好条件下，一般也会看到几百毫秒的来回延迟。

### **6、服务器推送的好处是什么？**

当浏览器请求一个网页时，服务器将会发回HTML，在服务器开始发送JavaScript、图片和CSS前，服务器需要等待浏览器解析HTML和发送所有内嵌资源的请求。服务器推送服务通过“推送”那些它认为客户端将会需要的内容到客户端的缓存中，以此来避免往返的延迟。