面试真题

【美团】HTTP常见的方法有哪些？

1. **GET**
2. **POST**：用于向服务器提交数据，通常用于创建新资源或提交表单。参数包含在请求体中，可以处理较大的数据量。
3. **PUT**：用于更新资源，将请求体中的数据完全替换目标资源的内容。如果资源不存在，可能会创建新资源（取决于服务器配置）。
4. **DELETE**
5. **HEAD**：类似GET，但不返回响应体，只返回响应头，常用于检查缓存有效性。

HEAD 方法就好像你去商店问店员某个商品的情况，但是你只关心商品的一些基本信息，比如价格标签、生产日期等，而不想要把商品本身拿回家看。  
当你想检查一个网页是否更新了，但又不想把整个网页内容都下载下来看时，就可以用 HEAD 方法。浏览器或者一些工具发送一个 HEAD 请求到服务器，服务器就会只返回响应头信息。通过查看响应头中的 “Last-Modified” 字段，就能知道这个网页最后修改的时间，以此判断是否有更新。如果缓存里有这个网页，还可以根据响应头中的缓存相关字段，比如 “Expires” 或 “Cache-Control”，来确定缓存是否还有效。

1. **OPTIONS**：用于查询某个资源支持的HTTP方法，通常用于跨域请求的预检请求。  
   OPTIONS 方法就好比你要去一个新的游乐场玩，但是不确定这个游乐场里的每个项目你能不能玩，于是你先去问工作人员每个项目都有什么要求，有哪些项目可以玩。在网络中，当你的浏览器要向一个不同域名的服务器发送请求（比如从你的网站请求另一个公司网站的数据，这就是跨域请求）时，它会先发送一个 OPTIONS 请求，去问服务器这个资源支持哪些 HTTP 方法，比如支不支持 GET、POST 等。服务器收到这个 OPTIONS 请求后，就会告诉浏览器它支持哪些方法，这样浏览器就知道后续可以用什么方法来请求数据了。

在使用 JavaScript 进行跨域的 AJAX 请求时，浏览器会自动先发送 OPTIONS 请求进行预检。比如你在自己的网页上写了一段代码，想要从另一个域名的服务器获取数据，浏览器会先发送 OPTIONS 请求到目标服务器，询问 “我能不能用 GET 方法获取这个数据呀” 等信息。服务器会返回一个响应，告诉浏览器它支持的方法，可能会在响应头中通过 “Access-Control-Allow-Methods” 字段来列出支持的方法，像 “GET, POST, PUT, DELETE” 等，浏览器收到后，就知道后续该怎么正确地发送请求了。

1. **PATCH**：用于对指定资源进行部分修改，客户端只需要提交更改的部分数据，而不是替换整个资源。  
   通俗解释：PATCH 方法就好像你要修改一幅画，但是你只需要修改其中的一部分，比如把画里的一朵花颜色改一下，而不用把整幅画重新画一遍。在网络请求中，如果你要修改服务器上的某个资源，比如一个文档、一条数据库记录等，不需要把整个资源的所有内容都重新提交一遍，只需要把你要修改的那部分内容告诉服务器就行。

假设你在一个网站上有自己的个人资料，你只想修改其中的电话号码，就可以使用 PATCH 方法。你通过客户端（比如手机 APP 或者网页）发送一个 PATCH 请求到服务器，请求中只包含你要修改的电话号码这部分数据，服务器收到后，就会根据你提供的数据，只修改数据库中你个人资料里的电话号码字段，而其他信息，像姓名、地址等都保持不变。

【美团】get和post方法的区别？

* 用途：GET获取数据；POST提交数据
* 数据传输方式和安全性：GET通过URL传少量数据，较不安全；POST通过请求体传大量数据，相对更安全
* 幂等性：GET幂等，执行多次结果相同；POST非幂等，执行多次会改变服务器上的数据。

面向简历

TCP、UDP

TCP（Transmission Control Protocol）和UDP（User Datagram Protocol）是两种常用的传输层协议，它们在网络通信中有一些重要的区别：

1. 连接性：

TCP： 提供面向连接的通信。在数据传输之前，必须先建立连接，然后进行可靠的数据传输，最后再释放连接。

UDP： 是面向无连接的通信。每个数据包都是独立的，不需要先建立连接，直接发送数据。因此，UDP通信更为轻量，但不保证可靠性。

2. 可靠性：

TCP： 提供可靠的数据传输，通过序列号、确认和重传机制确保数据的完整性和顺序性。如果数据包丢失，TCP会进行重传。

UDP： 不提供可靠性保证，数据包可能会丢失，也不保证数据包的顺序性。对于某些实时性要求高的应用，可以容忍少量的丢包。

6. 连接和断开：

TCP： 建立连接时有三次握手的过程，断开连接有四次挥手的过程。

UDP： 无连接的特性，不需要建立和断开连接的过程。

5. 适用场景：

TCP： 适用于对数据完整性和顺序性有要求的应用，如文件传输、网页访问等。

UDP： 适用于实时性要求高、可以容忍少量数据丢失的应用，如语音通话、视频直播等。

3. 数据传输方式：

TCP： 数据以流的方式进行传输，被划分为小的数据段，然后通过序列号进行重新组装。数据传输是可靠而有序的。

UDP： 数据以数据包的形式传输，每个数据包都是独立的。UDP更适用于需要快速传输的场景，如实时音视频。

4. 头部开销：

TCP： TCP头部相对较大，包含序列号、确认号、窗口大小等字段，用于保证可靠性和流控制。

UDP： UDP头部较小，只包含源端口、目标端口、长度和校验和等基本信息，开销较小。

总体来说，选择TCP还是UDP取决于具体的应用需求。TCP适用于要求可靠性和有序性的场景，而UDP适用于实时性要求高、可以容忍少量数据丢失的场景。在某些应用中，也可以结合使用TCP和UDP，根据不同的数据传输需求选择合适的协议。

HTTP1.0和HTTP1.1的区别是什么？

**持久连接**、**管道化**、**缓存控制**、**Host头部**、**错误代码**

**1. 持久连接**

* **HTTP1.0**：每次请求都需要建立一个新的TCP连接，响应之后关闭连接。
* **HTTP1.1**：默认启用持久连接（即Connection: keep-alive），允许多次请求/响应复用同一连接，减少了建立和关闭连接的开销。

**2. 管道化**

* **HTTP1.0**：不支持请求管道化，每次请求必须等待前一个请求完成。
* **HTTP1.1**：支持请求管道化，客户端可以在等待响应的同时发送多个请求，但服务器端必须按顺序返回响应。

**3. 缓存控制**

* **HTTP1.0**：缓存控制较为简单，仅依赖于Expires头部。
* **HTTP1.1**：新增了Cache-Control头部，提供了更细粒度的缓存控制，例如no-cache、no-store、max-age等。

**4. Host头部**

* **HTTP1.0**：不包含Host头部，无法在多个域名共享同一IP时进行区分。
* **HTTP1.1**：强制要求在请求中包含Host头部，允许同一IP托管多个虚拟主机（即多个域名）。

**5. 错误代码**

* **HTTP1.0**：错误码较少，错误响应不够细致。
* **HTTP1.1**：新增了多个错误码，如：
  + 409 Conflict：表示请求与当前状态冲突。
  + 410 Gone：表示资源已永久删除等。

HTTP2.0与HTTP1.1相比有哪些主要改进？

**多路复用**、**头部压缩**、**流量控制**、**服务器推送、二进制协议**

**1. 多路复用**

* **HTTP2.0：使用一个连接复用多个请求和响应，消除了HTTP1.1中的队头阻塞问题，使得请求和响应可以并行传输。**
* **HTTP1.1：虽然支持管道化，但请求和响应仍然是串行化的，且如果其中一个请求被阻塞，其他请求也会受到影响。**

**2. 头部压缩**

* **HTTP2.0：采用HPACK算法对HTTP头进行压缩，减少了冗余数据的传输，尤其对于重复的请求头，如Cookies等，效果显著。**
* **HTTP1.1：头部信息未压缩，且请求头较为冗长，这增加了网络带宽的消耗。**

**3. 流量控制**

* **HTTP2.0：支持对每个流进行独立的流量控制，服务器和客户端可以通过调整流的大小来管理带宽使用。**
* **HTTP1.1：没有内建流量控制，主要依赖TCP的拥塞控制，可能导致带宽浪费。**

**4. 服务器推送**

* **HTTP2.0：服务器可以在客户端请求之前主动推送资源，这对于加载复杂页面时有显著的性能提升。例如，在请求HTML文件的同时，服务器可以主动推送该页面依赖的CSS、JavaScript文件等。**
* **HTTP1.1：只有客户端可以发起请求，服务器只能被动响应，无法主动推送资源。**

**5. 二进制协议**

* **HTTP2.0：采用二进制协议，而不是文本协议，数据以帧的形式传输，解析更高效。**
* **HTTP1.1：是基于文本的协议，易于人类阅读和调试，但由于解析过程繁琐，性能不如二进制协议。**

HTTP3.0有了解过吗？它与之前的版本有哪些主要不同？

**传输协议、连接建立**、**多路复用**、**加密**、**头部压缩**：

**1. 基于QUIC协议**

* **HTTP3.0**：HTTP3不再依赖TCP，而是基于QUIC协议（Quick UDP Internet Connections）。QUIC是Google提出的一个新协议，建立在UDP之上，具有低延迟和更好的多路复用性能。
* **HTTP1.x 和 HTTP2**：两者都依赖于TCP协议，TCP的连接建立和管理较为复杂且容易受到延迟影响。

**2. 零RTT连接建立**

* **HTTP3.0**：QUIC协议支持零RTT连接建立（0-RTT），即客户端在首次发送请求时，可以立即开始数据传输，而无需等到完整的三次握手过程结束。通过保存会话信息，可以减少延迟。
* **HTTP1.x 和 HTTP2**：这两个版本都使用基于TCP的三次握手过程，每次建立新连接都需要等待完整的握手。

**3. 避免队头阻塞**

* **HTTP3.0**：QUIC协议为每个HTTP请求分配独立的流，所有流之间互不阻塞。即使某个流发生丢包或延迟，其他流依然能够继续传输数据，从而避免队头阻塞。
* **HTTP2**：虽然支持多路复用，但仍然受TCP队头阻塞的影响。当一个流的数据包丢失时，整个连接的延迟会增加，影响其他流的传输。

**4. 内建加密**

* **HTTP3.0**：QUIC将TLS加密内建于协议中，每个QUIC数据包都会自动加密，不需要额外的TLS握手过程。
* **HTTP2**：虽然支持TLS加密，但并没有内建，使用时需要显式配置TLS层。

**5. 头部压缩**

* **HTTP3.0**：与HTTP2类似，HTTP3继续使用HPACK进行头部压缩，提高了带宽效率。
* **HTTP1.1**：HTTP1.1没有内建头部压缩功能，导致大量重复头部信息的传输，浪费带宽。

HTTPS和HTTP有哪些区别？

**协议类型**、**端口号**、**加密方式**、**性能**、**安全性**

**1. 协议类型**

* **HTTP（超文本传输协议）**：是一种用于从Web服务器传输超文本到本地浏览器的协议。HTTP协议是无状态的，不提供数据加密和认证，所有信息以明文形式传输。
* **HTTPS（安全超文本传输协议）**：是在HTTP协议基础上加入SSL/TLS加密层，保证数据的保密性和完整性，防止中间人攻击。

**2. 端口号**

* **HTTP**：默认为80端口。
* **HTTPS**：默认为443端口，通信是加密的，增加了一个安全层。

**3. 加密和传输层**

* **HTTP**：数据在传输过程中是未加密的，容易被拦截、篡改。
* **HTTPS**：通过SSL/TLS协议进行加密，使得数据在传输过程中即使被截获，也无法解密。

**4. 性能**

* **HTTP**：由于没有加密过程，性能较好，传输速度较快。
* **HTTPS**：虽然由于加密和解密过程稍有性能损失，但现如今的硬件和加密算法优化已经让其性能差距不再显著。

**5. 安全性**

* **HTTP**：数据传输过程中不进行加密，容易被窃听和篡改，存在中间人攻击风险。
* **HTTPS**：通过SSL/TLS协议提供数据的加密、完整性验证及认证机制，有效防止中间人攻击、数据篡改及窃取。

HTTPS工作原理是什么？它是如何实现数据加密的？

**1. SSL/TLS协议**

* **SSL/TLS**协议是实现HTTPS加密通信的核心技术，TLS是SSL的继任版本。SSL/TLS用于加密数据传输、验证身份并确保数据完整性。TLS提供了对称加密、非对称加密和哈希算法的结合，确保数据在传输中的机密性、完整性和认证。

**2. 加密过程**

* **对称加密**：HTTPS使用对称加密算法（如AES）对传输的数据进行加密。对称加密的优点是加密和解密速度快，但问题是密钥的安全性。因为双方使用相同的密钥加密和解密数据，必须确保密钥的安全传输。
* **非对称加密**：用于密钥交换和身份认证。非对称加密使用公钥和私钥对数据进行加解密，公钥加密的数据只能用对应的私钥解密。服务器通过数字证书将公钥发送给客户端，客户端用该公钥加密会话密钥（对称密钥），然后发送给服务器。服务器用自己的私钥解密该密钥。

**3. 握手过程**

* **SSL/TLS握手**：HTTPS通信开始时，客户端与服务器会进行SSL/TLS握手，以下是握手的基本步骤：
  1. **客户端Hello**：客户端向服务器发送支持的加密算法、生成的随机数等信息。
  2. **服务器Hello**：服务器从客户端提供的算法中选择一个并返回自己的证书（包含公钥），以及用于生成会话密钥的随机数。
  3. **密钥交换**：客户端生成一个会话密钥并使用服务器的公钥加密该密钥，然后发送给服务器。服务器使用自己的私钥解密，获得会话密钥。
  4. **验证身份**：客户端验证服务器的数字证书，以确保与预期的服务器通信，防止中间人攻击。
  5. **完成握手**：客户端和服务器使用协商的会话密钥进行加密通信。

**4. 数据传输**

* **加密数据传输**：一旦握手完成，客户端和服务器就开始使用对称加密算法（如AES）加密通信数据。此时，通信双方使用的是会话密钥（由握手过程中的密钥交换生成），确保数据在传输过程中既不会被窃取，也不会被篡改。
* **数据完整性**：在数据传输过程中，SSL/TLS协议使用哈希算法（如SHA-256）对数据进行完整性验证，确保数据在传输过程中没有被篡改。

知识框架

浏览器键入baidu.com所有用到的协议

当你在浏览器中键入baidu.com并按下回车时，浏览器会根据输入的网址进行一系列操作，涉及到多个协议。以下是在浏览器键入baidu.com时可能用到的主要协议：

DNS 协议（Domain Name System）：

浏览器首先会将域名 baidu.com 转换成对应的 IP 地址。这个过程使用 DNS 协议进行域名解析。

HTTP 协议（Hypertext Transfer Protocol）：

一般情况下，浏览器会默认使用 HTTP 协议进行网页的请求和响应。浏览器向服务器发送 HTTP 请求，请求获取 baidu.com 的网页内容。

HTTPS 协议（Hypertext Transfer Protocol Secure）：

如果网站启用了 HTTPS，浏览器会使用 HTTPS 协议进行加密的通信。这涉及到 SSL/TLS 协议用于加密数据传输，确保数据的安全性。

TCP 协议（Transmission Control Protocol）：

HTTP 和 HTTPS 协议都是基于 TCP 协议的，它负责在网络上建立可靠的连接，确保数据的可靠传输。

IP 协议（Internet Protocol）：

IP 协议用于在网络上进行数据包的寻址和路由。它是 TCP 和 UDP 的底层协议，负责在网络上进行数据包的传递。

ICMP 协议（Internet Control Message Protocol）：

在进行网络通信时，可能涉及 ICMP 协议，用于处理网络错误、诊断问题等。

以上是在浏览器键入 baidu.com 时可能用到的主要协议。实际情况中，根据网站的具体配置和浏览器的支持情况，可能还涉及到其他协议和技术，如 QUIC 协议、WebSocket 等。

TCP和UDP区别

TCP（Transmission Control Protocol）和UDP（User Datagram Protocol）是两种常用的传输层协议，它们在网络通信中有一些重要的区别：

1. 连接性：

TCP： 提供面向连接的通信。在数据传输之前，必须先建立连接，然后进行可靠的数据传输，最后再释放连接。

UDP： 是面向无连接的通信。每个数据包都是独立的，不需要先建立连接，直接发送数据。因此，UDP通信更为轻量，但不保证可靠性。

2. 可靠性：

TCP： 提供可靠的数据传输，通过序列号、确认和重传机制确保数据的完整性和顺序性。如果数据包丢失，TCP会进行重传。

UDP： 不提供可靠性保证，数据包可能会丢失，也不保证数据包的顺序性。对于某些实时性要求高的应用，可以容忍少量的丢包。

3. 数据传输方式：

TCP： 数据以流的方式进行传输，被划分为小的数据段，然后通过序列号进行重新组装。数据传输是可靠而有序的。

UDP： 数据以数据包的形式传输，每个数据包都是独立的。UDP更适用于需要快速传输的场景，如实时音视频。

4. 头部开销：

TCP： TCP头部相对较大，包含序列号、确认号、窗口大小等字段，用于保证可靠性和流控制。

UDP： UDP头部较小，只包含源端口、目标端口、长度和校验和等基本信息，开销较小。

5. 适用场景：

TCP： 适用于对数据完整性和顺序性有要求的应用，如文件传输、网页访问等。

UDP： 适用于实时性要求高、可以容忍少量数据丢失的应用，如语音通话、视频直播等。

6. 连接和断开：

TCP： 建立连接时有三次握手的过程，断开连接有四次挥手的过程。

UDP： 无连接的特性，不需要建立和断开连接的过程。

总体来说，选择TCP还是UDP取决于具体的应用需求。TCP适用于要求可靠性和有序性的场景，而UDP适用于实时性要求高、可以容忍少量数据丢失的场景。在某些应用中，也可以结合使用TCP和UDP，根据不同的数据传输需求选择合适的协议。

三次握手

第一次: 客户端发出**同步请求**报文。

第二次：服务器发出**同步确认**报文。

第三次: 客户端发出**确认**报文。



四次挥手

第一次：客户端发出**连接释放**报文。

第二次：服务器发出**确认**报文。

第三次：服务器发出**连接释放确认**报文。

第四次：客户端发出**确认**报文。



MSL（Maximum Segment Lifetime）

任何报文在网络上的存在的最长时间超过这个时间报文将被丢弃。

RFC 规定为2分钟，常用的有30秒、1分钟、2分钟等。

OSI/RM七层模型

**①** 应**用层：**在应用与网络间作为接口

**②** 表**示层：**对数据编码、加密、压缩、转换

**③** 会**话层**

**④** 传**输层**（通信子网与资源子网的分界层）：提供端到端的可靠透明数据传输

**⑤** 网**络层**（通信子网的最高层）：逻辑地址寻址、路由选择

**⑥ 数据**链**路层：**提供可靠数据传输链路

**⑦** 物**理层**：屏蔽具体设备的差异，使链路层不必关心具体传输介质。

TCP/IP网络四层模型

**应用层**：提供网络服务，比如 FTP、DNS等

**传输层**：提供端到端可靠数据流传输服务

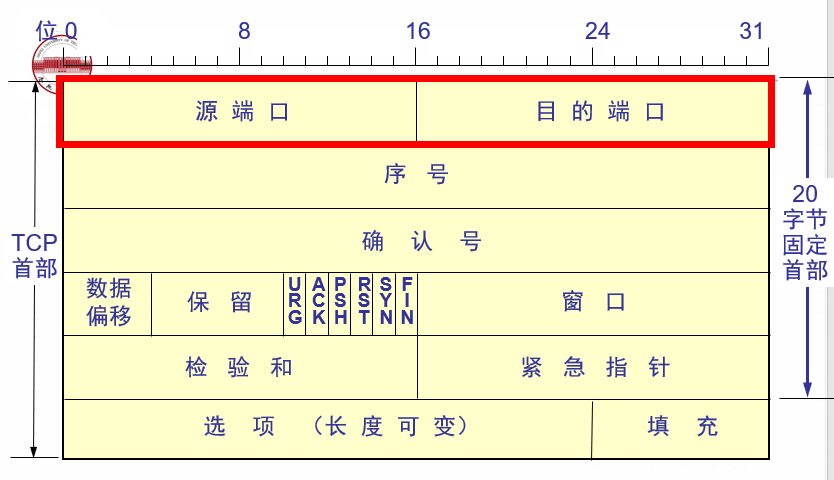
**网际层**：负责异构网或同构网的进程间通信,

将传输层分组封装为数据报格式进行传送,

每个数据报必须包含目的地址和源地址。

**网络接口层**：是网络访问层，负责与物理网络的连接。

TCP报文格式



端口是传输层与应用层的服务接口

序号seq：数据流按字节编号

确认号ack：期望收到的下一个报文段的序号

数据偏移/首部长度：指出数据距离起始处有多远

* 紧急URG：紧急数据 尽快传送
* 确认ACK
* 推送 PSH：不等缓存都填满，尽快向上交付
* 复位 RST：主机崩溃或其他原因出现严重差错，必须释放重连
* 同步 SYN
* 终止 FIN：数据已发送完毕，并要求释放连接。

窗口：用来让对方设置发送窗口的依据

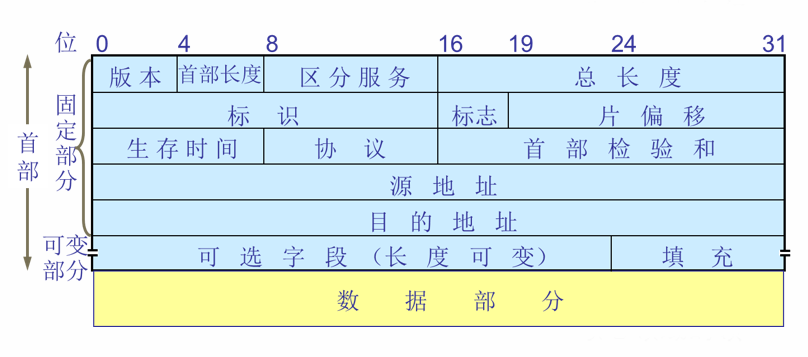
检验和：检验范围包括首部和数据。计算时，要加上 12 字节的伪首部。

紧急指针：指出紧急数据字节数（放在本报文段数据的最前面）

* MSS选项：加上首部 整个TCP报文段
* 窗口扩大选项：新窗口增大到(16 + S)
* 时间戳选项：用作乱序的时间判断依据，以及计算RTT
* 选择确认选项：有选择的确认收到的某些报文，以提高效率。

填充：使整个首部长度是 4 字节的整数倍。

IP报文格式



**版本**：4位，表示**IP协议版本**，通常为0100（v4），若为0110( v6)

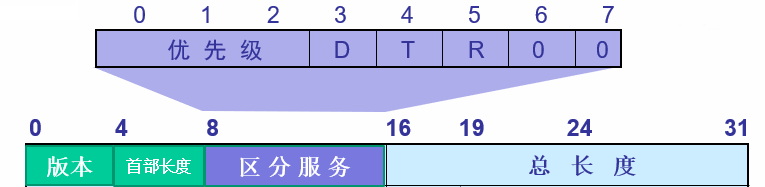
**首部长度**：4位，表示**IP包首部长度**，最短20字节，最长60字节

**区分服务**：8位，以前称为**服务类型**，从未使用过。1998年改称**区分服务**。

用于**指明要求网络提供的服务**，

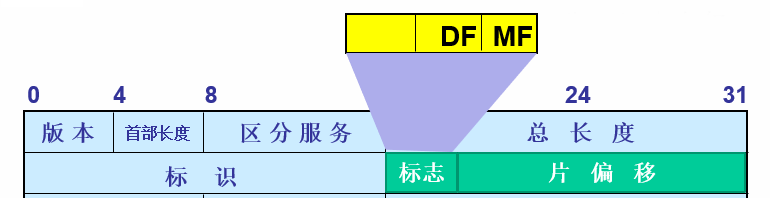
目前主要包括**D**延迟**、T**吞吐量**、R**可靠性等三种。

即使如此，依然没有真正使用。



**总长度**：16位，包括了**首部长度和数据长度**，最长为65535个字节。

**标识**：16位，**数据报计数器**，用于**区分数据报**的唯一标识符。在数据报传向目的地址时，如果将数据报**被分为多个片段**，那么**每个片段**都有**相同的标识符**。



**标志**：3 位，

- 最高位**保留**；

- 中间位是**不分片（Don‘t Fragment，DF）**标志，DF=**1**则**不允许**分片。

- 最低位是**有更多分片（More Fragment，MF）**标志

除**最后一个分片MF=0**以外外其余都是MF=1

**片偏移**：12 位，表示分片后，该片在原分组中的**相对位置**。

片偏移以 **8 个字节**为**偏移单位**。

**生存时间**：8 位，一般记为 **TTL (Time To Live)**。

表示数据报在网络中**可通过的路由器数**的**最大值**。

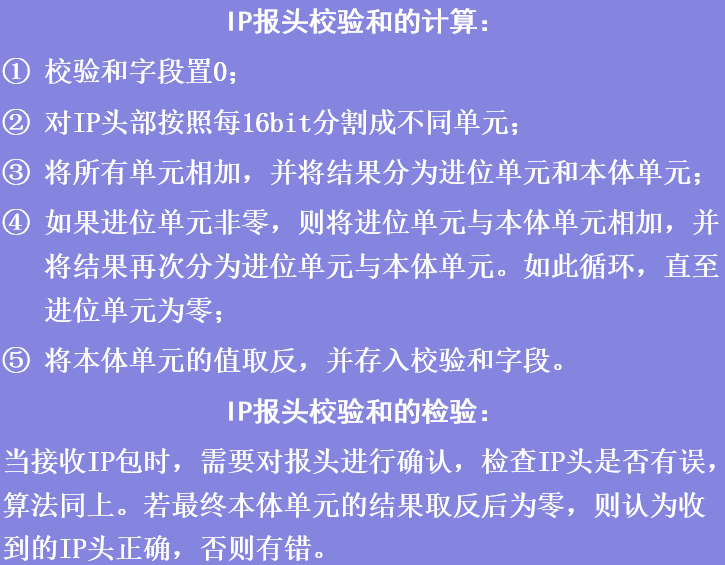
若**超出**最大值，则**丢弃**数据包，并返回“目标不可达”。

**协议**：8位，指出此数据报**使用何种协议**，

以便目的主机的 IP 层将数据部分上交给哪个处理过程。

如TCP、UDP、ICMP、IGMP、OSPF等。

**首部检验和**：16位，**只检验**数据报**首部**，**不检验数据**部分。

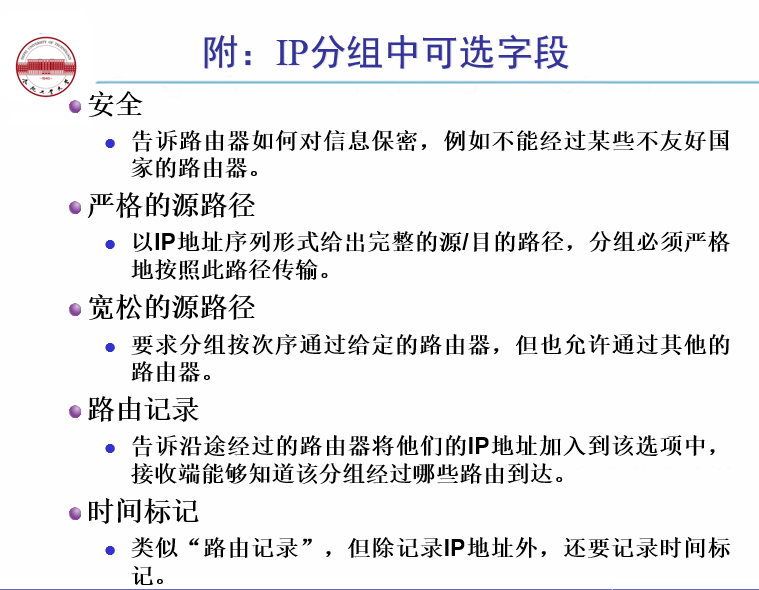


**源地址**：32位，发送端主机IP地址。

**目的地址**：32位，接收端主机IP地址。

**可选字段**：在整个包头长度**不超过60个字节**的情况下，可选字段的**长度可变**，

主要用来进行一些**测试工作**。主要**包括**：安全性（security）、松散源路由（Loose source routing）、 严格源路由（Strict source routing）、路由记录（Record route）、时间戳（Timestamps）等。



**填充**：根据可选字段的长度，填充**若干个0**，使得包头长度为32的整倍数

如何流量控制

1、滑动窗口实现

**发送方**根据接收方给出的窗口值，构造发送窗口尺寸不大于对方的窗口值。

**发送缓存**暂时存放：准备发送的数据和已发出但未确认的数据。

**接收缓存**暂时存放：按序到达但未被读取的数据和不按序到达的数据。

2、坚持计时器

面向非零窗口确认报文丢失导致的死锁。

发送方收到零窗口确认报文时，启动坚持计时器：

* 若设定时间内收到非零窗口确认报文，则**撤销**。
* 若设定时间内未收到，则发出只有**一个字节**数据的探测报文，若截止时间后没有收到发送**另一个探测报文**，并**加倍设置**坚持计时器值，如此**反复**直到收到接收方的非零窗口确认报文为止。（最大值限制60秒）

3、糊涂窗口综合症

糊涂窗口综合症：接收端和发送端速率不匹配时产生很多小于MSS的数据报文的小包，造成网络拥塞。

Nagle算法治疗“发送端糊涂”：尽可能避免发送小数据报文，一个TCP连接上最多只能有一个未被确认的小数据报文。持续接受上层小数据报文，并在确认到来时以一个较大的报文发送出去

Clark + 延迟ACK治疗“接收端糊涂”：只要有数据到达就发送确认，但宣布的窗口大小为零，直到或者缓存空间已能放入具有最大长度的报文段

如何差错控制

1、检验和

如果检验和无效则认为损伤，丢弃。

2、确认ACK和选择确认SACK选项

（1）确认（ACK）：**期望接收**的下一个字节序号

（2）选择确认（SACK）：报告**失序**以及**重复**的数据块。

3、超时重传时间RTO

当**无法快速交付**数据报文时**减小网络负担**。

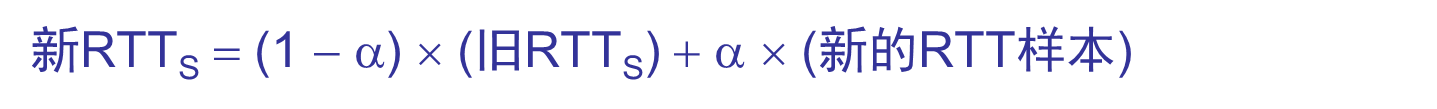
（1）**RTO 小**于 RTT，则会造成很多**不必要的重传**；

**RTO 远大**于 RTT，则会**降低整体网络利用率**。

（2）

每发生一次**重传**，RTO加倍，直到收到重传数据包的应答，退避因子恢复1。

（2）加权平均往返时间RTTS



若 α 更接近于**0**，表示 RTT 值更新较**慢**。

若选择 α 更接近于 **1**，则表示 RTT 值更新较**快**。

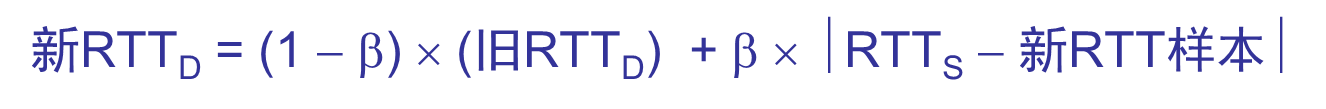
RFC 2988 推荐的 α 值为 **1/8，即 0.125**。

（3）超时重传时间



其中：RTTD 是 RTT 的**偏差加权平均值**。

第一次测量，RTTD = RTT / 2



推荐值是 **1/4，即 0.25**。

（4）RTT测量的复杂性和困难

① 重传后收到了上一个未确认报文的ACK

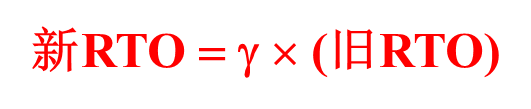
② IP数据报所选择的路由变化很大。因而运输层的往返时间的方差也很大。

（5）**Karn 算法**：不采用重传报文段的往返时间样本。

① 理由：当**重传发生**时，**RTO**已经采用了**指数退避**。下一次的传输将按照这个**已经增大的RTO**计时，不应该再将其纳入到RTT的统计样本中。

② 问题：当**发生重传**时，时延**增大**或**出错**，可能会带来**连续重传**，从而使超时重传时间 **RTO一直得不到更新**，无法反映当前现状。

③ 修正：每发生一次**重传**，即**增大一些RTO值**

系数 γ 的典型值是 2

（6）指数避退

**超时而发生连续多次重传**时，**超时阈值**遵循指数避退：

下一次是这次的2倍，2为退避因子

直到收到重传数据包的应答，RTO退避因子回复为1。

目的：为了当网络处于**无法快速交付**数据报文状态时**减小网络负担**。

慢开始和拥塞避免

1、慢开始和拥塞避免

（1）发送方维持**拥塞窗口 cwnd**.窗口大小**取决于网络的拥塞程度**，**动态变化**。

原则：

网络**没有拥塞**，拥塞窗口就再**增大**一些，以便**把更多的分组发送出去**。

网络**出现拥塞**，拥塞窗口就**减小**一些，以**减少注入到网络中的分组数**。

（2）慢开始算法的原理

**刚刚开始**发送时设置拥塞窗口 **cwnd = 1**，即一个MSS的数值。（一倍）

在**每收到**一个**确认**后，将**拥塞窗口加 1**，即增加一个 MSS 的数值。（加倍）

（3）传输轮次

每经过**一个传输轮次**，拥塞窗口 cwnd 就**加倍**。

一个传输轮次所经历的时间其实就是**往返时间 RTT**。

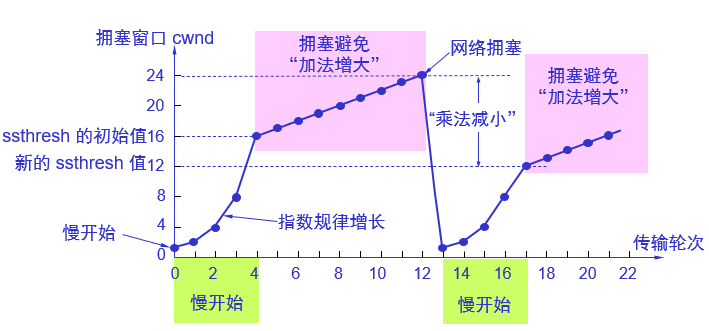
（4）门限状态变量（ssthresh）

当 cwnd < ssthresh 时，使用**慢开始算法**。

当 cwnd > ssthresh 时，**改用拥塞避免算法**。

**拥塞避免算法**：

是让拥塞窗口 cwnd **缓慢地增大**，即每经过一个往返时间 RTT 就把发送方的拥塞窗口 cwnd **加 1**，使拥塞窗口 cwnd 按线性规律缓慢增长。



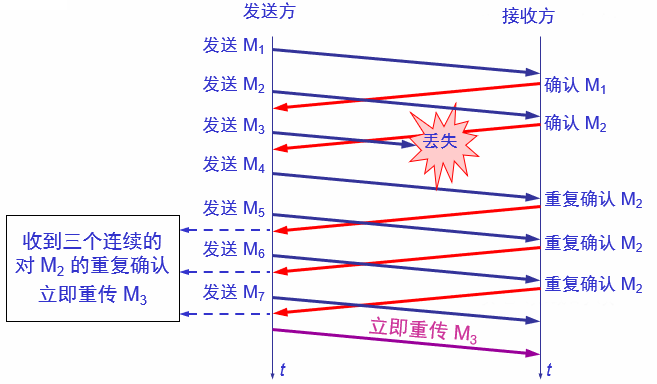
① 加法增大：执行**拥塞避免算法**后，经过一个往返时间，就把拥塞窗口**增加一个 MSS 大小**，使拥塞窗口**缓慢增大**，以**防止网络过早出现拥塞**。

② 乘法减小：只要出现一次超时（网络拥塞），就把**门限值 ssthresh** 设置为当前的**拥塞窗口值乘以 0.5**。

2、快重传和快恢复

（1）快重传

接收方收到一个**失序的报文段后**就**立即发出重复确认**。让发送方**及早知道**。发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段。



（2）快恢复

当发送端**收到连续三个重复**的确认时，

执行“乘法减小”算法：把慢开始**门限 ssthresh 减半**。

**拥塞窗口**不设置为 1，而是**设置为ssthresh 减半**，

然后开始执行**拥塞避免算法（“加法增大”）**，使拥塞窗口缓慢地**线性增大**。