TCP/IP协议簇

- TCP/IP协议簇
  - TCP/IP协议群
  - TCP/IP的含义
    - TCP/IP 指的是协议簇, 它是利用 IP 进行通信时所必须用到的协议群的统称
  - 结构分层
    - 通信链路层

      通信链路层也可以分为 物理层 和 数据链路层
      - 通信物理层

        EX: 运输工具，比如火车、汽车
        - 该层作用：定义物理设备标准。以二进制数据形式传输数据
        - 物理设备：中继器/集线器/双绞线
        - 涉及协议：ISO2110/IEEE802
      - 数据链路层

        EX: 相当于货物核对单，标明里面有些什么东西，接受的时候确认一下是否正确 数据链路层在不可靠的物理介质上提供可靠的传输。 该层的作用包括：物理地址寻址、数据的成帧、流量控制、数据的检错、重发等
        - 该层作用：传输有地址的帧, 保证数据的准确性

          最小的传输单位——帧, 帧里面是有目标主机的MAC地址 如果传送数据的过程中，接收点检测到数据有错误，就通知发送方重新发送一帧
        - 物理设备：交换机/网卡
        - 涉及协议：PPP（点对点）/HDLC（高级数据链路协议）
    - 网络层

      EX: 相当于邮政局或快递公司地址（IP地址），能正确到达对方 网络层的任务就是选择合适的网间路由和交换结点， 确保数据及时传送。网络层将数据链路层提供的帧组成数据包。
      - 该层作用：地址解析, 路由选择
      - 物理设备： 路由器/三层交换机
      - 涉及协议： IP/ARP/ICMP

        IP协议： 将数据包发送到目标主机。不可靠，无状态 ARP（地址解析协议）: 通过目标设备的IP地址，查询目标设备的MAC地址 ICMP（控制消息协议）： 测网络通信故障和实现链路追踪 IP 在数据包的发送过程中可能会出现异常，当 IP 数据包因为异常而无法到达目标地址时， 需要给发送端发送一个异常通知
    - 传输层
      - 该层作用： 定义传输数据的协议和端口，维护端到端的连接
      - 物理设备： 四层交换机/四层路由器
      - 涉及协议： TCP/UDP协议
    - 应用层
      - 会话层
        - 该层作用：建立,维持和终止节点间的通信
        - 涉及协议： SMTP/DNS
      - 表示层
        - 该层作用：数据格式转换。压缩和解压缩， 加密和解密
        - 涉及协议：Telnet/Rlogin
      - 应用层
        - 该层作用：为应用程序提供服务
        - 涉及协议： HTTP/SMTP/FTP
  - 模型汇总
    - 五层/七层模型
    - 各层的作用
    - 各层的协议
- 数据处理流程
  - 数据包结构

    每个分层中，都会对所发送的数据增加一个首部，这个首部中包含了该层必要的信息 [avatar](https://mmbiz.qpic.cn/mmbiz_png/A3ibcic1Xe0iaTicuU6kBxklSjlPjd8lZLl4ibeianUVm2oBCsuRfzDo8bWh2ic9wQw84lwiaia5xX42OeeGwpRuIm9nP0A/640)
  - 数据包发送流程
    - 应用层：数据编码,格式化处理

      主机 A 也就是用户点击了某个应用或者打开了一个聊天窗口输入了cxuan，然后点击了发送， 应用层还需要对这个数据包进行处理，包括字符编码、格式化等等。这一层其实是 OSI 中表现层做的工作， 只不过在 TCP/IP 协议中都归为了应用层 数据包在发送的那一刻建立TCP/UDP连接，这个连接相当于通道，在这之后其他数据包也会使用通道传输数据
    - 传输层: 添加首部字段（源端口号和目的端口号）

      为了描述信息能准确的到达另一方，我们使用 TCP 协议来进行描述。TCP 会根据应用的指示，负责建立连接、发送数据和断开连接。 TCP 会在应用数据层的前端附加一个 TCP 首部字段，TCP 首部包含了源端口号 和 目的端口号， 这两个端口号用于表明数据包是从哪里发出的，需要发送到哪个应用程序上； TCP 首部还包含序号，用以表示该包中数据是发送端整个数据中第几个字节的序列号； TCP 首部还包含校验和，用于判断数据是否损坏，随后将 TCP 头部附加在数据包的首部发送给 IP。
    - 网络层：添加首部字段（发送端IP地址+接收端IP地址+传输层协议类型）

      网络层主要负责处理数据包的是 IP 协议，IP 协议将 TCP 传过来的 TCP 首部和数据结合当作自己的数据， 并在 TCP 首部的前端加上自己的 IP 首部， IP 首部包含目的和源地址，紧随在 IP 首部的还有用来判断后面是 TCP 还是 UDP 的信息 IP 包生成后，会由路由控制表判断应该发送至哪个主机，IP 修饰后的数据包继续向下发送给路由器或者网络接口的驱动程序，从而实现真正的数据传输。
    - 通信链路层：添加首部字段（发送端MAC地址+接收端MAC地址+以太网类型）

      经由 IP 传过来的数据包，以太网会给数据附上以太网首部并进行发送处理。以太网首部包含接收端的 MAC 地址、 发送端的 MAC 地址以及标志以太网类型的以太网数据协议
  - 数据包解析流程
    - 通信链路层：检查MAC 地址

      目标主机收到数据包后，从以太网的首部找到 MAC 地址判断是否是发给自己的数据包，如果不是发给自己的数据包则会丢弃该数据包。 如果收是发送给自己的，就会查找以太网类型判断是哪种协议， 如果是 IP 协议就会扔给 IP 协议进行处理， 如果是 ARP 协议就会扔给 ARP 协议进行处理。 如果协议类型是一种无法识别的协议，就会将该数据包直接丢弃。
    - 网络层：检查IP地址

      经过以太网处理后的数据包扔给网络层进行处理，我们假设协议类型是 IP 协议，那么，在IP收到数据包后就会解析IP首部， 判断 IP 首部中的 IP 地址是否和自己的 IP 地址匹配，如果匹配则接收数据并判断上一层协议是 TCP 还是 UDP； 如果不匹配则直接丢弃。
    - 传输层：检查数据完整型

      在传输层中，我们默认使用 TCP 协议，在 TCP 处理过程中 首先会计算一下校验和，判断数据是否被损坏。然后检查是否按照序号接收数据，最后检查端口号，确定具体是哪个应用程序。 数据被完整的识别后，会传递给由端口号识别的应用程序进行处理。
    - 应用层：解码数据

      接收端指定的应用程序会处理发送方传递过来的数据，通过解码等操作识别出数据的内容，然后把对应的数据存储在磁盘上。 返回一个保存成功的消息给发送方，如果保存失败，则返回错误消息。
  - 包首部图

    [avatar](https://mmbiz.qpic.cn/mmbiz_png/A3ibcic1Xe0iaTicuU6kBxklSjlPjd8lZLl4oicgLcibARhfxUxpyXsyYELOLQJTqZSEP2A8K21WfwgpTXWyQPmtrAxA/640)
- HTTPS（应用层）
  - HTTP的风险
    - 窃听风险：获取通信内容
    - 篡改风险：植入垃圾广告
    - 冒充风险： 冒充官方网站
  - HTTPS的概念
    - HTTPS = HTTP + SSL/TLS

      SSL（安全套接字层）: 位于 TCP/IP 和应用层之间，为数据通讯提供安全支持 TLS（传输层安全）： 前身是 SSL
  - HTTPS的安全性
    - 混合加密：实现机密性

      通过混合加密的方式可以保证信息的机密性，解决了窃听的风险 HTTPS 采用的是对称加密和非对称加密结合的「混合加密」方式： 在通信建立前采用非对称加密的方式交换「会话秘钥」，后续就不再使用非对称加密。 在通信过程中全部使用对称加密的「会话秘钥」的方式加密明文数据。 采用「混合加密」的方式的原因： 对称加密只使用一个密钥，运算速度快，密钥必须保密，无法做到安全的密钥交换。 非对称加密使用两个密钥： 公钥和私钥，公钥可以任意分发而私钥保密，解决了密钥交换问题但速度慢。 [avatar](https://mmbiz.qpic.cn/mmbiz_jpg/J0g14CUwaZfXG1113Sjm0iaOXfoOv0tlUYNGEmfY95A74GR3xicqXKZCDI7Q4icgQu7CuSSx9QiaFlr4Py49RHonjw/640?wx_fmt=jpeg&tp=webp&wxfrom=5&wx_lazy=1&wx_co=1)
    - 摘要算法：实现完整性

      摘要算法用来实现完整性，能够为数据生成独一无二的「指纹」，用于校验数据的完整性，解决了篡改的风险。 原理： 客户端在发送明文之前会通过摘要算法算出明文的「指纹」，发送的时候把「指纹 + 明文」一同加密成密文后，发送给服务器， 服务器解密后，用相同的摘要算法算出发送过来的明文，通过比较客户端携带的「指纹」和当前算出的「指纹」做比较， 若「指纹」相同，说明数据是完整的。 [avatar](https://mmbiz.qpic.cn/mmbiz_jpg/J0g14CUwaZfXG1113Sjm0iaOXfoOv0tlUicIliaBcr2XAXpMdeibLG4MMticpkX0e6xZHbXeiavMu7faJcL2TdVj0Udw/640?wx_fmt=jpeg&tp=webp&wxfrom=5&wx_lazy=1&wx_co=1)
    - 数字证书：实现真实性

      客户端先向服务器端索要公钥，然后用公钥加密信息，服务器收到密文后，用自己的私钥解密。 这就存在些问题，如何保证公钥不被篡改和信任度？ 所以这里就需要借助第三方权威机构 CA （数字证书认证机构），将服务器公钥放在数字证书（由数字证书认证机构颁发）中，只要证书是可信的，公钥就是可信的。 [avatar](https://mmbiz.qpic.cn/mmbiz_jpg/J0g14CUwaZfXG1113Sjm0iaOXfoOv0tlUibyiaEab7NMrTn632LZmYQe5qaibibT0xsOs7ic6u98ypWJBjbPMzOUCb2g/640?wx_fmt=jpeg&tp=webp&wxfrom=5&wx_lazy=1&wx_co=1)
  - SSL/TLS 详细流程
    - ClientHello

      首先，由客户端向服务器发起加密通信请求，也就是 ClientHello 请求。 在这一步，客户端主要向服务器发送以下信息： （1）客户端支持的 SSL/TLS 协议版本，如 TLS 1.2 版本。 （2）客户端生产的随机数（Client Random），后面用于生产「会话秘钥」。 （3）客户端支持的密码套件列表，如 RSA 加密算法。
    - SeverHello

      服务器收到客户端请求后，向客户端发出响应，也就是 SeverHello。服务器回应的内容有如下内容： （1）确认 SSL/ TLS 协议版本，如果浏览器不支持，则关闭加密通信。 （2）服务器生产的随机数（Server Random），后面用于生产「会话秘钥」。 （3）确认的密码套件列表，如 RSA 加密算法。 （4）服务器的数字证书。
    - 客户端回应

      1. 确认服务器的数字证书 客户端收到服务器的回应之后，首先通过浏览器或者操作系统中的 CA 公钥，确认服务器的数字证书的真实性。 如果证书没有问题，客户端会从数字证书中取出服务器的公钥，然后使用它加密报文 2. 向服务器发送如下信息： （1）一个随机数（pre-master key）。该随机数会被服务器公钥加密。 （2）加密通信算法改变通知，表示随后的信息都将用「会话秘钥」加密通信。 （3）客户端握手结束通知，表示客户端的握手阶段已经结束。这一项同时把之前所有内容的发生的数据做个摘要，用来供服务端校验。 上面第一项的随机数是整个握手阶段的第三个随机数，这样服务器和客户端就同时有三个随机数，接着就用双方协商的加密算法，各自生成本次通信的「会话秘钥」。
    - 服务器回应

      服务器收到客户端的第三个随机数（pre-master key）之后，通过协商的加密算法，计算出本次通信的「会话秘钥」。然后，向客户端发生最后的信息： （1）加密通信算法改变通知，表示随后的信息都将用「会话秘钥」加密通信。 （2）服务器握手结束通知，表示服务器的握手阶段已经结束。这一项同时把之前所有内容的发生的数据做个摘要，用来供客户端校验。 至此，整个 SSL/TLS 的握手阶段全部结束。接下来，客户端与服务器进入加密通信，就完全是使用普通的 HTTP 协议，只不过用「会话秘钥」加密内容。
  - HTTP与HTTPS的区别
    - 安全性

      HTTP 是超文本传输协议，信息是明文传输，存在安全风险的问题 HTTPS 则解决 HTTP 不安全的缺陷，在 TCP 和 HTTP 网络层之间加入了 SSL/TLS 安全协议，使得报文能够加密传输。
    - 建立连接

      HTTP: TCP三次握手 HTTPS: TCP三次握手 + SSL/TLS握手
    - 端口号

      HTTP端口号是80，HTTPS的端口号是443
    - 数字证书

      HTTPS 协议需要向 CA（证书权威机构）申请数字证书，来保证服务器的身份是可信的
- HTTP（应用层）
  - HTTP的概念

    HTTP的名字「超文本协议传输」， 是一个在计算机世界里专门在「两点」之间「传输」文字、图片、音频、视频等「超文本」数据的「约定和规范」
    - 协议

      HTTP 是一个用在计算机世界里的协议。它使用计算机能够理解的语言确立了一种计算机之间交流通信的规范 以及相关的各种控制和错误处理方式（行为约定和规范）
    - 传输

      HTTP 协议是一个双向协议, 专门用来在两点之间传输数据的约定和规范
    - 超文本

      它就是超越了普通文本的文本，它是文字、图片、视频等的混合体最关键有超链接，能从一个超文本跳转到另外一个超文本
  - HTTP常见状态码
    - 1xx：中间状态
    - 2xx：成功处理

      「200 OK」是最常见的成功状态码，表示一切正常 「204 No Content」也是常见的成功状态码，与 200 OK 基本相同，但响应头没有 body 数据 「206 Partial Content」是应用于 HTTP 分块下载或断电续传，表示响应返回的 body 数据并不是资源的全部，而是其中的一部分
    - 3xx：重定向

      「301 Moved Permanently」表示永久重定向，说明请求的资源已经不存在了，需改用新的 URL 再次访问 「302 Moved Permanently」表示临时重定向，说明请求的资源还在，但暂时需要用另一个 URL 来访问 「304 Not Modified」不具有跳转的含义，表示资源未修改，重定向已存在的缓冲文件，也称缓存重定向，用于缓存控制
    - 4xx：客户端错误

      「400 Bad Request」表示客户端请求的报文有错误，但只是个笼统的错误。 「403 Forbidden」表示服务器禁止访问资源，并不是客户端的请求出错。 「404 Not Found」表示请求的资源在服务器上不存在或未找到，所以无法提供给客户端
    - 5xx：服务器错误

      「500 Internal Server Error」 笼统通用的错误码，服务器发生了错误 「501 Not Implemented」表示客户端请求的功能还不支持 「502 Bad Gateway」通常是服务器作为网关或代理时返回的错误码，表示服务器自身工作正常，访问后端服务器发生了错误 「503 Service Unavailable」表示服务器当前很忙，暂时无法响应服务器
  - HTTP请求和响应
    - HTTP请求

      请求由客户端向服务器端发出, 由4部分内容组成： 请求头\r\n\r\n请求体
      - 请求方法

        GET 请求指定的页面信息，并返回实体主体。 HEAD 类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头 POST 向指定资源提交数据进行处理请求（例如提交表单或者上传文件) PUT 从客户端向服务器传送的数据取代指定的文档的内容 PATCH 是对 PUT 方法的补充，用来对已知资源进行局部更新 DELETE 请求服务器删除指定的页面。 CONNECT 把服务器当作跳板机 OPTIONS 允许客户端查看服务器的性能。 TRACE 回显服务器收到的请求，主要用于测试或诊断。
      - 请求头

        协议头 说明 示例 Accept 可接受的响应内容类型 Accept: text/plain Accept-Charset 可接受的字符集 Accept-Charset: utf-8 Accept-Encoding 可接受的响应内容的编码方式 Accept-Encoding: gzip, deflate Accept-Language 可接受的响应内容语言列表 Accept-Language: zh-CN Connection 客户端想要优先使用的连接类型 Connection: keep-alive Connection: Upgrade Cookie Host 表示服务器的域名以及服务器所监听的端口号 host: www.itbilu.com:80 Content-Length 以8进制表示的请求体的长度 Content-Length: 348 Referer 标识这个请求是哪个页面发来的 Referer: http://itbilu.com/nodejs User-Agent 浏览器的身份标识字符串 Content-Type 具体请求中的媒体类型信息 Content-Type: application/x-www-form-urlencoded
      - 空行(\r\n\r\n)
      - 请求体
    - HTTP响应

      响应由服务器向客户端端发出, 由4部分内容组成： 响应头\r\n\r\n响应体
      - 响应状态码
      - 响应头

        协议头 说明 示例 Allow 服务器支持哪些请求方法 Allow: GET, HEAD Content-Encoding 响应内容的编码方式 Content-Encoding: gzip, deflate Content-Length 内容长度 Content-Length: 68 Content-Language 响应内容所使用的语言 Content-Language: zh-cn Content-Type 当前内容的MIME类型 Content-Type: text/html; charset=utf-8 Date： 此条消息被发送时的日期和时间 Date: Tue, 15 Nov 1994 08:12:31 GMT Expires 响应过期时间 Expires: Thu, 01 Dec 1994 16:00:00 GMT Set-Cookie 告诉浏览器需要将此内容放到Cookie中 Set-Cookie: UserID=itbilu; Max-Age=3600; Version=1 Server 服务器的名称 Server: nginx/1.6.3
      - 空行(\r\n\r\n)
      - 响应体
  - GET和POST

    GET和POST本质上就是TCP链接，并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同
    - 本质无区别（TCP链接）
      - GET/POST都是TCP链接

        GET和POST是HTTP协议中的两种发送请求的方法，HTTP的底层是TCP/IP， 所以GET和POST的底层也是TCP/IP， 也就是说，GET/POST都是TCP链接。GET和POST能做的事情是一样一样的。要给GET加上request body，给POST带上url参数， 技术上是完全行的通的
      - 汽车（TCP）/ 交通规则（HTTP）

        TCP就像汽车，我们用TCP来运输数据，但是如果路上跑的全是看起来一模一样的汽车，那这个世界看起来是一团混乱， 送急件的汽车可能被前面满载货物的汽车拦堵在路上，整个交通系统一定会瘫痪。为了避免这种情况发生，交通规则HTTP诞生了 HTTP给汽车运输设定了好几个服务类别，有GET, POST, PUT, DELETE等等，HTTP规定，当执行GET请求的时候， 要给汽车贴上GET的标签（设置method为GET），而且要求把传送的数据放在车顶上（url中）以方便记录。 如果是POST请求，就要在车上贴上POST的标签，并把货物放在车厢里。 当然，也可以在GET的时候往车厢内偷偷藏点货物，但是这是很不光彩；也可以在POST的时候在车顶上也放一些数据，让人觉得傻乎乎的。
      - 运输公司（浏览器/服务器）

        不同的浏览器（发起http请求）和服务器（接受http请求）就是不同的运输公司。 虽然理论上，你可以在车顶上无限的堆货物（url中无限加参数） 但是运输公司可不傻，装货和卸货也是有很大成本的， 他们会限制单次运输量来控制风险，数据量太大对浏览器和服务器都是很大负担。 （大多数）浏览器通常都会限制url长度在2K个字节，而（大多数）服务器最多处理64K大小的url。超过的部分，恕不处理 如果你用GET服务，在request body偷偷藏了数据，不同服务器的处理方式也是不同的，有些服务器会帮你卸货，读出数据，有些服务器直接忽略，所以，虽然GET可以带request body，也不能保证一定能被接收到
    - 应用过程上的区别
      - 传参方式： GET在URL中传参，POST在BODY中传参
      - 参数长度： GET传送的参数有长度限制，而POST没有
      - 参数类型： GET只能接受ASCII字符，而POST没有限制
      - 安全性能：GET比POST不安全，GET参数直接暴露在URL上
      - 编码方式： GET只能进行url编码，而POST支持多种编码方式
      - 参数缓存：GET请求参数会被浏览器主动Cache，而POST不会
    - 重大区别（数据包）
      - GET产生一个TCP数据包， POST产生两个TCP数据包

        GET请求： 浏览器会把http header和data一并发送出去，服务器响应200 POST请求： 浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok 两次数据包的优点： POST需要两步，时间上消耗的要多一点，看起来GET比POST更有效 但是在网络环境好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。 而在网络环境差的情况下，两次包的TCP在验证数据包完整性上，有非常大的优点。 并不是所有浏览器都会在POST中发送两次包，Firefox就只发送一次
  - HTTP特性
    - HTTP的优点
      - 简单

        HTTP 基本的报文格式就是 header + body，头部信息也是 key-value 简单文本的形式
      - 灵活和易于扩展

        HTTP协议里的各类请求方法、URI/URL、状态码、头字段等每个组成要求都没有被固定死，都允许开发人员自定义和扩充
      - 应用广泛和跨平台
    - HTTP 的缺点
      - 无状态、明文传输（Cookie）
      - 不安全（Https）
  - HTTP性能
    - 短连接(HTTP/1.0)

      每发起一个请求，都要新建一次 TCP 连接，而且是串行请求
    - 长连接(HTTP/1.1)

      要任意一端没有明确提出断开连接，则保持 TCP 连接状态
      - 分块传送

        分块传输编码只在HTTP/1.1中提供, 允许服务器发送给客户端的数据可以分成多个部分。 优点： 1. 分块传输编码允许服务器在最后发送消息头字段。 2. HTTP服务器有时使用压缩以缩短传输花费的时间。分块编码有利于一边进行压缩一边发送数据
    - 管道网络传输

      同一个 TCP 连接里面，客户端可以发起多个请求，只要第一个请求发出去了，不必等其回来，就可以发第二个请求出去，可以减少整体的响应时间
    - HTTP/1.1 VS HTTP/1.0

      改进： 1. 使用 TCP 长连接的方式改善性能开销 2. 支持管道（pipeline）网络传输, 减少整体的响应时间 HTTP/1.1瓶颈: 1. 请求 / 响应头部（Header）未经压缩就发送，首部信息越多延迟越大。只能压缩 Body 的部分； 2. 发送冗长的首部。每次互相发送相同的首部造成的浪费较多； 3. 服务器是按请求的顺序响应的，如果服务器响应慢，会招致客户端一直请求不到数据，也就是队头阻塞； 4. 没有请求优先级控制； 5. 请求只能从客户端开始，服务器只能被动响应。
    - HTTP/1.2 VS HTTP/1.1

      1. 头部压缩 如果同时发出多个请求，他们的头是一样的或是相似的，那么，协议会帮你消除重复的分。 2. 二进制格式 HTTP/2 不再像 HTTP/1.1 里的纯文本形式的报文，而是全面采用了二进制格式。 3. 数据流 HTTP/2 的数据包不是按顺序发送的，同一个连接里面连续的数据包，可能属于不同的回应。 因此，必须要对数据包做标记，指出它属于哪个回应。 4. 多路复用 HTTP/2 是可以在一个连接中并发多个请求或回应，而不用按照顺序一一对应 5. 服务器推送 HTTP/2 还在一定程度上改善了传统的「请求 - 应答」工作模式，服务不再是被动地响应，也可以主动向客户端发送消息。
  - Http一次连接的全过程

    1. 域名解析 2. 发起TCP的3次握手，建立连接 3. 基于TCP发起HTTP请求 4. 服务器响应http请求,并返回数据 5. 浏览器解析html代码 6. 浏览器对页面进行渲染呈现给用户
  - Content-Type与POST
    - text/xxx：文本数据
    - application/json：json数据
    - multipart/form-data：文件数据
    - application/x-www-form-urlencode：表单数据用&连接
  - session和cookie的区别
    - 存储位置：cookie保存在客户端； session保存在服务器
    - 存储方式：cookie只保存ASCII字符串； session无限制
    - 存储容量：一个站最多保存20个Cookie； session没限制
    - 隐私策略：cookie对客户端是可见的； session是透明的