

HTTP

• HTTP的基本知识

• what's HTTP

• http的全称:

- HyperText Transfer Protocol——**超文本传输协议**

• protocol:

- **两个及以上参与者，对一种行为的规范和约定**

- http就是一种协议:

- 用在计算机世界，用计算机能够理解的语言，确立了计算机之间进行**交流通信**的规范，以及各种控制和错误处理方式——http是使计算机之间能够正常进行交流的一种协议

• transfer:

- **双向协议，允许中间有中转or接力**

- http:

- 就是在计算机之间**两点进行传输数据**的约定和规范

- 举例:

- 上网时，浏览器是请求方A，服务器是请求方B，双方约定用HTTP进行通信，浏览器将请求数据发送给网站，网站处理之后将一些数据返回给浏览器，然后浏览器将数据渲染到屏幕上——双向的

- 可以有**中转/接力**:

- $A \leftrightarrow N \leftrightarrow M \leftrightarrow B$ ，中转/接力，需要遵循http协议，但是在不影响基本数据传输的基础上，可以在传输过程中添加任意额外的东西

• hyperText:

- 超越文本的文本，含有**链接到其他文本的链接点（关键点）**

- 文本概念:

- 传统的字符、文字；图片、视频、压缩包等

- 超文本:

- 超越文本的文本，是文本的混合体，且一定包含有超链接，能从一个超文本跳转到另一个超文本

- 举例:

- **HTML是常见的超文本**，本身是一个纯文字的文件，但是内部使用了很多标签定义了图片等其他文本的链接（eg: `<a>`），经过浏览器的解释，展现出来的就是一个图文并茂的网页

• 总结:

- HTTP就是在**计算机世界**中，在**两点之间传输超文本**的数据该操作的一种**约定和规范**，限定了 范围 -- 对象 -- 操作 -- 内容 -- 性质

- **PS:**

- 两点之间：可以是服务器和服务器之间，服务器到计算机之间等

- **HTTP的常见状态码**

- **1xx：提示信息**

- 表示目前是协议处理的中间状态，还需要后续的操作，用的少

- **2xx：成功**

- 报文已经收到，并且被正确处理了
 - 常见的状态码
 - 【200 OK】一切正常，**如果是非head请求**，那么服务器返回的响应头都会有body数据
 - 【204 No Content】一切正常，但是响应头没有body数据
 - 【206 Partial Content】用于http的**分块下载或断点续传**，响应返回的数据是其中一部分

- **3xx：重定向**

- 资源位置发生变化，需要客户端用新的URL重新发出请求
 - 常见状态码
 - 【301 Moved Permentantly】永久重定向，请求的资源已经迁地方了，需用新的URL去访问

返回301状态码的同时，会返回一个Location，指明新的地址，浏览器会自动拿着这个新的地址去访问

eg: 通过<http://www.baidu.com>去访问，会得到一个301，并且Location得到的是<https://www.baidu.com>，再去访问该URL，从而浏览器最后到达的是后面这个地址
 - 【302 Found】临时重定向，请求的资源还在这个地方，但是需要另一个URL访问

在得到状态码的同时，也会返回一个Location，得到一个临时URL，可通过该URL去访问新的地址
 - 301和302的区别：
 - 301重定向是永久的重定向，搜索引擎在抓取新的内容的同时也将旧的网址**替换**为了重定向之后的网址
 - 302 重定向是临时的，搜索引擎会抓取新的内容而保留旧的地址
 - 【304 Not Modified】没有跳转的含义，表示资源未修改，重定向已存在的缓冲文件——用于缓存控制

即客户端从上次请求后，该网页没有发生变化，客户端已经有对应的缓存文件，那么服务器判断出来之后就返回304，提示本地有缓存，从而减少带宽加快响应速度

- **4xx：客户端错误（主观错误，可改正）**

- **请求报文错误**，服务器无法处理

- 常见的状态码
 - 【400 Bad Request】客户端请求的报文有误，笼统的错误
 - 【403 Forbidden】服务器禁止访问的资源，请求报文未出错，只是访问了不该访问的
 - 【404 Not Found】请求的资源在服务器上找不到

- **5xx：服务器错误（客观错误，不可改正）**

- 服务器在处理请求的时候发生内部错误，说明传输的报文是正确的
- 常见的状态码
 - 【500 Internal Server Error】服务器发生错误，笼统的错误
 - 【501 Not Implemented】客户端请求的功能还不支持
 - 【502 Bad Gateway】服务器作为网关、代理时，向后端服务器发出请求，但是没有得到及时的响应返回的错误码，表示服务器工作正常，但是去访问后端服务器出现错误
 - 【503 Service Unavailable】服务器正忙，暂时无法响应

- **HTTP的常见字段**

- **Host字段：指定服务器域名**

- 格式：Host: www.A.com
- 可以将请求发往同一台服务器上的不同网站

- **Content-Length字段：**

- 服务器返回时，带有该字段，表明本次响应的字段的长度

- **Content-Type字段：**

- 格式：Content-Type: text/html; charset=utf-8（HTML的数据，编码格式为utf-8）
 - 服务器返回时，表明本次响应的数据格式
- 对应客户端发送请求时，指明可以接受的数据格式：**Accept: */***（表明任何格式都可接受）

- **Content-Encoding字段**

- 格式：Content-Encoding: gzip
- 服务器返回时，表明本次响应的数据的压缩方式，告诉客户端需要用此种方法进行解压

客户端在请求时，指明可以接受的数据压缩格式：**Accept-Encoding: gzip, deflate**

- **Connection字段：**

- 格式：Connection: keep-alive
- 最常用于客户端要求和服务器端**用TCP持久连接**，以便其他请求复用，直到客户端或服务器主动关闭连接

HTTP/1.1版本默认的都是持久连接，但是为了兼容老的HTTP版本，需要显式指定Connection首部字段的值为Keep-Alive

- 这个不是标准字段

• HTTP的GET和POST

• GET和POST的概念

- GET：从服务器获取资源，可以是文本（广义的）
- POST：向指定的URL提交数据，数据就放在报文的body中，会修改服务器上的数据（可能增加、可能覆盖）eg：上交作业

• GET和POST方法都是安全和幂等的吗？

- 概念：
 - 安全：请求的方法不会破坏服务器上的资源
 - 幂等：多次执行相同操作，结果一样
- 分析
 - GET方法是安全的且幂等的——因为是只读操作
 - POST方法是不安全的，且不幂等——因为是修改操作

• HTTP的特性

• http的优点

• 简单

- 主要就是格式简单，基本报文格式是**head+body**，而在head里面的字段格式是：**key: value**，易于理解

• 灵活易扩展

- HTTP协议里面的各类请求方法、URL/URI、状态码、头字段等每个组成部分都没有被固定死，可以让开发人员自定义和补充
- HTTP是OSI第七层（应用层）的协议，下层可以随意变化
HTTPS就是在HTTP的基础上，在HTTP层和TCP层之间增加了SSL/TLS安全传输层，HTTP/3把整个TCP层全部替换成基于UDP的QUIC

• 应用广泛和跨平台

• http的缺点

• 无状态：有优有劣

- 优势是：服务器不会记忆HTTP的状态，不需要额外的资源来记录状态信息，**减轻服务器的负担**
- 劣势是：没有记忆，完成关联性问题时很麻烦
eg：登录后，各个页面之间的跳转，都需要知道用户的信息，常见的解决方式——**Cookie技术**
- ps：cookie技术
 - 就是在请求和响应的报文中，写入Cookie信息来传递客户端的状态
 - 形象说明：客户端第一次请求后，服务器会发送一个有客户信息的【通行证】（表明服务器记住了该用户），而后续的客户请求都带上【该

通行证】，那么服务器就能通过检查Cookie去识别

- **明文传输：有优有劣**

- 优势：明文传输，那么传输过程中的数据都是直接可读的，方便调试
可以通过F12或者Wireshark抓包，直接得到传递的数据
- 劣势：所有信息都暴露了，没有隐私，且容易被攻击，如果包含有密码账号等信息，就会被窃取

- **不安全：HTTP的严重缺点**

- 通信使用明文：内容被窃取
eg：账号、密码等被窃取
- 不验证通信方的身份：遭遇伪装
eg：访问假的淘宝
- 无法证明报文的完整性：可能在传输过程中已经被篡改
eg：网页上植入垃圾广告
- 解决方法：可以通过HTTPS解决——通过引入SSL/TLS层，达到安全极致

- **http的性能（针对的是HTTP/1.1）**

- **长连接：good**

- 背景：HTTP/1.0每次发起一个请求，都要创建一个TCP连接（三次握手），并且是**串行请求**（因为每次通信结束都会断开，所以必然是串行执行的），增加了通信开销
- HTTP/1.1提出了长连接，只要一方没有明确提出断开请求，那么就一直保持TCP连接
- 优势是：减少了TCP的重复建立和断开的额外开销，减轻服务器端的负载

- **管道网络传输：good**

- 前提是：HTTP/1.1采用了长连接的方式
- 背景：HTTP/1.0，客户端需要两个资源，那么在同一个TCP连接中，发送请求后，必须要收到回复才能再发送下一个请求
- 概念：在同一个TCP连接中，客户端可以发起多个请求，并且可以**不等前一个请求返回响应，就可以发出第二个请求**，但是服务器还是需要顺序处理的
- 优势是：**减少整体响应时间**

- **队头阻塞：bad**

- 背景：前面的管道传输中，允许客户端发送请求序列
- 概念：一个请求因为某种原因被阻塞（可能是很耗时的执行），那么后面等待处理的请求就全部被阻塞了，而客户端一直得不到数据
- ps：队头——就是当前正在被处理的请求，就是引起阻塞的原因，非队头元素都是在等待执行的
- PS：【请求 - 应答】的模式会影响HTTP的性能


- **HTTPS和HTTP**

• HTTPS和HTTP的区别

- HTTP是明文传输，存在信息泄漏、篡改等问题；HTTPS通过在HTTP和TCP之间增加SSL/TLS安全协议，**对报文进行加密传输**
- HTTP连接只需要TCP三次握手，之后连接就建立了；HTTPS在三次握手之后，还需要SSL/TLS握手，才能加密并传输
- HTTP的端口号：80；HTTPS的端口号是443
- HTTPS协议还需要向CA申请数字证书，来保证服务器的身份是可靠的

CA：证书权威机构，CA提供的数字证书类似于防伪标签，有效的CA证书包含公钥和私钥两部分，二者相互加密、解密，即公钥加密、私钥解密或私钥加密、公钥解密。

• HTTPS弥补了HTTP的哪些问题

- 明确HTTP的问题
 - **窃听问题**：查看通信链路上的报文内容（只读）——缺少内容加密
 - **篡改问题**：对通信链路上的报文添加内容（修改）——缺少内容验证
 - **冒充问题**：假冒一个服务器——缺少身份验证
- 解决了啥：
 - 信息加密：报文内容加密——无法窃听——混合加密
 - 校验机制：对传输的内容进行验证，如果发生篡改就无法正常显示
 - 身份证书：验证对方的真实性
- 如何解决的：**通过增加SSL/TLS协议**
 - 混合加密：信息高级加密，无法简单破解
 - HTTPS采用的是**非对称加密 + 对称加密**结合的
 - 通信建立前，采用非对称加密，交换【会话密钥】，后续不再使用非对称加密 

非对称加密：使用两个密钥，公钥和密钥，公钥可以公开，密钥需要保密，速度慢
发送端通过公钥进行加密，接收端收到后通过密钥进行解密，eg：RSA就能，公钥只能用来加密，而无法通过公钥解密；密钥就能用来对公钥加密的内容进行解密
RSA简易原理：<https://zhuanlan.zhihu.com/p/37738632>，本质上就是利用大整数分解难题，来实现无法破解密钥，只需要数字足够大，那么当前技术就无法破解它
 - 通信建立过程中，使用对称加密的会话密钥，来加密明文数据
对称加密：一个密钥，保密，速度快
 - 摘要算法：实现完整性，为数据生成一个【指纹】，用来校验数据的完整性，防止篡改
 - 客户在发送明文之前，算出明文的指纹，然后将指纹 + 明文进行加密传递到服务器。服务器解密后，用相同方法算出明文的指纹，如果两者指纹一致，那么数据未发生篡改

- 数字证书：服务器公钥放到证书中，解决冒充风险
 - 客户端向服务器要公钥，然后客户端将公钥加密【会话密钥】，发送给服务器，服务器收到后通过密钥解密就得到了【会话密钥】
 - 但是，如何证明该服务器是可信任的？
 - 借助第三方权威机构CA（数字证书认证机构），然后将服务器密钥放在数字证书中，只要证书可靠那么公钥就是可靠的



• 总结：HTTPS和HTTP在操作过程中就是增加了两个步骤：

- HTTPS：在TCP连接之后，还需要SSL/TLS连接（里面用到了非对称加密）
- 在传输过程中，把内容进行对称加密
- ps：TLS（传输层安全协议）和SSL（安全套接层）本质上是同一个东西，是一个东西的两个不同阶段，首先有了SSL，后面由于应用的广泛性，将其标准化为了TLS
 - SSL/TLS 1.2需要4次握手，需要两个RTT时延；SSL/TLS 1.3优化，需要4次握手，需要1个RTT时延

• HTTPS建立连接的过程

主要关注的是TCP三次握手建立通信连接之后，增加了SSL/TLS的握手过程，看这个过程是如何操作的

• SSL/TLS协议的基本流程

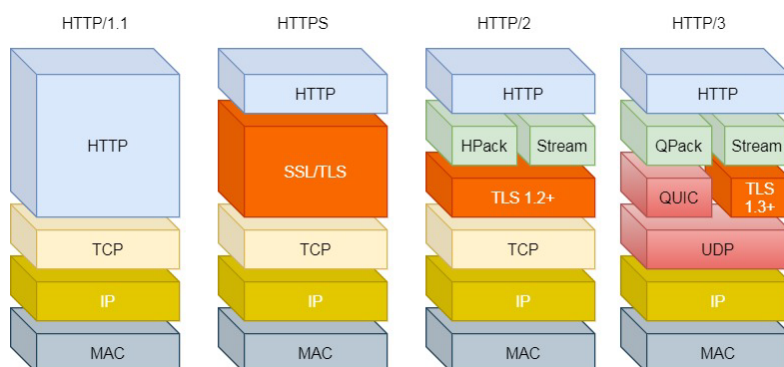
- 客户端向服务器索要公钥，并且验证该公钥的可信性
- 双方协商产生【会话密钥】
- 双方采用【会话密钥】进行加密通信——此时通信已经连接

• 具体SSL/TLS握手阶段——4次握手

- 1. ClientHello：客户端发起加密通信请求
 - 发送的内容：
 - 客户端支持的SSL/TLS版本（eg：TLS 1.2）；

- 客户端支持的密码套件列表（eg：RSA加密算法）；
- 客户端产生的随机数（用来生成会话密钥）
- 2. ServerHello：响应客户端的请求
 - 回应的内容：
 - 确认SSL/TLS协议版本，看是否支持（如果不支持就直接关闭加密通信）；
 - 确认密码套件列表；服务器生成随机数（用来生成会话密钥）；
 - **服务器的数字证书**
- 3. 客户端回应服务器：收到服务器的响应之后，给服务器的响应
 - 操作：
 - 先通过浏览器orOS中存储的CA密钥，去解密数字证书，看服务器数字证书的真实性；
 - 如果证书真实，那么从解密的数字证书中获得服务器的公钥，然后使用它加密报文，然后将该加密报文发送信息
 - 回应的内容：
 - 随机数
 - 加密通信算法改变通知，之后都用【会话密钥】进行加密和解密
 - 客户端握手结束通知，表示握手阶段结束，并且会将之前的所有内容的数据做摘要，来提供服务端校验
- 4. 服务器最后回应客户端：计算本次通信的【会话密钥】
 - 操作：根据前面3次的通信过程中生成的随机数，通过协商的加密算法，计算出【会话密钥】
 - 回应的内容：
 - 加密通信算法改变通知，提示随后的信息都通过【会话密钥】加密
 - 服务器握手结束通知，表示握手阶段结束，并且会将之前的所有内容的数据做摘要，来提供客户端校验

• HTTP/1.1、HTTP/2、HTTP/3的变化过程



- HTTP/1.1比HTTP/1.0
 - 优势
 - TCP长连接

- 管道网络传输
- 未改进
 - 在传输过程中，只对body进行压缩，而不对头部压缩，首部信息越大延迟越大
 - 首部内容过于冗长
 - 队头阻塞
 - 没有请求优先级控制（平均看待）
 - 请求只能从客户端开始，而服务器只能被动响应（服务器不能主动去连接客户端）
- HTTP/2比HTTP/1.1
 - 优势
 - HTTP/2是基于HTTPS，所以是安全传输的
 - 头部压缩
 - 背景：HTTP/1.1的首部不压缩，且内容冗余
 - 概念：同时发送多个请求，如果头是一样or相似的，那么协议会**删除冗余部分**
 - 原理：**HPACK 算法**：在客户端和服务器都维护了一张头信息表，所有字段都存入该表中，然后对应一个索引号，以后只需要发索引号，而不用发相同的字段，从而提高速度
 - 二进制格式
 - 背景：HTTP/1.1是纯文本形式
 - 概念：head和body都是用二进制信息流，统称为**帧——头信息帧和数据帧**
 - 优势：对计算机友好，计算机收到报文后可以直接执行，而不要再转换，增加数据传输效率
 - 多路复用
 - 背景：HTTP/1.1的队头阻塞
 - 概念：在一个连接中，可以并发响应多个请求，而**不需要按照请求顺序一一回应**
 - 优势：移除了HTTP/1.1中的串行请求，不会出现队头阻塞，**降低延迟和提高连接利用率**
 - eg：服务器收到了客户端的A、B请求，如果A比较耗时，那么先响应A的部分结果，接着执行B并且响应B的处理结果，然后再去处理A剩下的部分（类似于OS的时间片轮转机制）
 - 数据流
 - 背景：HTTP/1.1未控制请求的优先级，上面的多路复用
 - 由于服务器响应的数据包不是顺序的，那么需要对数据包做标记，标明是哪个响应的哪部分响应

- 概念：每个请求或响应的所有数据包，就是一个**数据流**，每个数据流都标记一个独有的编号，其中规定**客户端**发出的数据流编号为**奇数**，**服务器**发出的数据流编号为**偶数**。客户端可以指定数据流的优先级，优先级高的请求，服务器可以优先响应
- 服务器推送
 - 背景：HTTP/1.1的服务器是被动响应的
 - 概念：HTTP/2改善了传统的【请求 - 应答模式】，服务器可以主动向客户端发送消息
 - eg：浏览器在初始请求HTML时，服务器会将JS、CSS等静态文件资源主动发给客户端，减少延时等待的问题
- 未改进
 - 主要问题：HTTP的多路复用中，如果发生丢包问题，会触发TCP的重传机制，而HTTP的下层协议TCP，是不了解有多少个请求的，所以一旦发生丢包，那么TCP连接中的所有HTTP请求必须要等待丢了包被重传过来，所以全部被阻塞了
 - 根源：传输层TCP的问题
- HTTP3比HTTP/2
 - 优势：将TCP变成了UDP，**UDP是不管顺序和丢包的**，所以不会出现HTTP/1.1的队头阻塞和HTTP/2的丢包全部重传问题
 - UDP是不可靠的传输，但是**基于UDP的QUIC协议**，可以保证可靠性传输
 - QUIC是在UDP的基础上，伪TCP + TLS + HTTP/2的多路复用协议
 - QUIC，有一套机制可以保证可靠性传输，如果某个数据流发生丢包，**只会阻塞该流**，而其他流不受影响
 - 升级到TLS1.3，头压缩算法QPack
 - HTTPS要建立连接：需要TCP进行3次握手，TLS1.3需要3次握手，而QUIC将该6次握手合并为3次握手——**QUIC三次握手**，从而减少交互次数