Proxy Type

静态代理 IP

设置固定的代理 IP 地址来进行网络请求的方式

代理 IP 地址事先确定并配置好，无法动态改变。当需要发送网络请求时，请求会通过配置的代理 IP 进行传递，而不是直接连接到目标服务器。

动态住宅代理

通过代理服务器来中转用户和服务器的网络通信，从而允许用户在访问网站时不暴露自己的 IP 地址。

动态住宅代理的特点在于它可以自动切换所使用的 IP 地址，以避免重复使用相同的 IP 地址，从而导致账号被封禁或其他问题。

动态长效 ISP

可以保持长时间稳定不变的真实住宅 IP，兼有数据中心的高速和住宅代理的高匿

静态住宅代理

独享数据中心代理

是指是使用数据中心拥有并管理 IP 的代理。与住宅代理使用互联网服务提供商（ISP）拥有和分配的 IP 地址，并且还需要多余设备的情况不同。

Proxy

L2TP/Ipsec

L2TP is a tunneling protocol that doesn't provide encryption on its own.
It is often used in combination with IPSec for encryption and authentication.
L2TP/IPSec provides a double layer of security,
where L2TP creates the tunnel, and IPSec encrypts the data within the tunnel.

模拟器因为网络结构，并不支持 PPTP 和 L2TP，有些用户会出现两者使用冲突的现象，是因为存在着兼容和冲撞的问题。

Server

硬件划分

服务器 完整电脑 （可远程 ip 连接） 【安装了服务软件的计算机：Web 服务器，数据库服务器，文件服务器，代理服务器，邮箱服务器】

Vps 完整电脑中分割出一部分空间配置而成的电脑 （可远程 ip 连接）

虚拟主机 vps 中服务器划分出的一部分空间 （仅支持 ftp 文件传输，不支持安装软件，固定服务器）【空间商提供 ftp 连接账号和 mysql 数据库连接】

云服务器 升级版 VPS，更便宜（SFTP over SSH 22 端口，连接云服务器 linux）

软件划分

代理服务器: 主机 <--->代理 <----> 目的地 处于主机 ip 和 目的 ip 之间， 代替主机 ip 请求目的 ip，并帮助返回主机 ip 想要的内容

域名&空间

域名 www. zhihu. com. cn .

四级域 三级域 二级域 顶级域 根域

记录解析 @ 空三级域 beirezx.top www 普通三级域 www.beirezx.top

TTL 域名解析记录在 DNS 服务器中的存留时间

同源和跨域

跨域： 浏览器会监视用户当前访问的网站发送的请求， 请求网址 必须和 当前网页的地址的域名，端口，协议相同。跨域会被禁止并报错（跨域不会携带饼干的）

跨域请求 iframe form a 触发请求（同时修改 当前网页和 iframe 内部网页 的 document.domain 实现跨域，二级域名必须相同 m.baidu.com-->a.qq.com ）

img new 完直接请求（可跨域请求，能够获得图片响应，无法获取其他结果）

link 去掉 rel 不能发送请求，添加到页面上才能发送请求（可跨域请求，能获得 css 文件响应，无法获取其他结果）

script （可跨域 GET 请求，能获得 js 文件响应，和字符串响应） 【css script 响应的内容会被添加 <style></style> <script></script>标签】

同源请求 同一个域名内文件可相互请求（**http://www.koko.com:8080**/xxx） 【ajax 必须域名完全相同才能请求，flash 不能跨域】

jsonp 跨域请求（需要服务器主动提供）：声明函数 var funName=function(data){ console.log(data) }

客户端 发送 声明函数<script src="a.php?callback=funName">中

服务端 响应 字符串--声明函数调用并传入数据 echo "{$_GET['callback']}({$data})"

【Content-Type: application/javascript】 传出函数名，服务端响应 带实参数据的函数执行 字符串

IP

IPv4 网络设备给连接上的电脑分配一个 IP 地址，可手动更改（4 bytes）【每个网卡/适配器都可以绑定一个 IP，多个 IP 连接不同的网络时，电脑就能访问多个网络】

网络地址 192.168.33.0 识别网段（不可用 ip）

主机地址 192.168.33.1-254 主机用地址

广播地址 192.168.33.255 局域网内向 255 地址发送信息是向整个网段广播发送信息【1-254 网关和电脑都可用】

本地回环地址 127.0.0.1 **localhost** 始终指向本机（本机 IP 在不同网络环境下会发生变化）

IPv6 网络设备给连接上的电脑分配一个 IP 地址，可手动更改（8 bytes）

原始地址： 2001:0DB8:0000:0023:0008:0800:200C:417A

前面的 0 可省： 2001:DB8:0:23:8:800:200C:417A

连续的 0 可省： FF01:0:0:0:0:0:0:1101 ==> FF01::1101

子网掩码 交换机分配的网段 （子网网段===交换机 IP & 子网掩码 ）

192.168.100.5/16 ==> （16 表示子网掩码从左到右为 16 个 1 == 11111111.11111111.00000000.00000000 == 255.255.0.0）

主机地址范围：192.168.100.1 - 192.168.100.254 （网络地址+1 即为第一个主机地址，广播地址-1 即为最后一个主机地址）

端口 联网程序启动占用一个端口，叫做监听（0-65535） 【1~1023 用于一些知名的网络服务，用户使用 1024 以上的端口就行了。http **80**, https **443** **ftp 21** **ssh 22**】

网段 网络设备分配的 IP 范围 // 192.168.0.1 到 192.168.255.255 在 192.168.X.X 网段内

NAT 映射 内网地址 192.168.xx.xx 对应一个公网地址，减少对公网地址的占用。（网络设备内设置）

公网 IP ISP 分配的全球唯一地址

网络情况：**单个网络情况** **192.168.0**.106 （192.168.0 网段）

多个网络情况 （同一网络环境机器可以互相访问） 无线网/网线 【不同网络环境下不同的 IP】

重定向：A 重定向到 B

IP 访问 A 时， A 返回带有 B 的地址的响应头

IP 接收到带有 B 地址的响应头时， IP 自动请求 B 的地址 （第二次请求，请求体会失效）

Session：作用时间：自己设置　或者 一直存在直到关闭浏览器

每次访问 A 网址，都返回 sessionID

请求 A 网址所有文件时，都会发送 cookie（带有 sessionID），才能使用 session 的数据

网关机构

全球最大的互联网（众多局域网连接到一起）【使用 TCP/IP 协议】

ISP　　网络服务提供商（分配公网地址连接到互联网）

NAP　　网络交换中心

中国 ISP　　中国移动互联网，中国联通互联网，中国公用计算机互联网，中国教育和科研计算机网，中国科技网，中国网通公用互联网（网通控股）

宽带中国 CHINA169 网（网通集团），中国长城互联网，中国国际经济贸易互联网，中国卫星集团互联网

机房　　服务器托管到运营商的机房

双线机房　　连接了多个 ISP（多个 ISP 内访问此机房都很快）

多层次 ISP 结构互联网　　**第一层 ISP**（接入大公司）<---　[NAP] <--- **第二层 ISP**（国家 ISP）　<---　**本地 ISP**（石家庄地区，或小区）<---　校园网　【高层给低层分配 IP 网段】

网关设备

路由器：将一个网口变成很多个，可以插几条网线供几个人使用。【电脑浏览器输入路由的 IP 地址，登录查看路由器信息】

交换机：扩充端口和转换信号（扩展路由器端口）

光猫：有光纤接口，需要通过网线连接路由器（解调器）

WAN 口　　连接上级网络设备

LAN 口　　通常为 4 个，连接下级用户设备

转发代理：　　内部是客户端，即内网的客户端通过转发代理服务器访问外部网络。（接受客户端发送的任何请求）

反向代理：　　内部是服务器。而外部的用户通过反向代理访问内部的服务器。（只接受到指定服务器的请求）

惊群现象：一个网路连接到来，多个睡眠的进程被同事叫醒，但只有一个进程能获得链接，这样会影响系统性能。

DNS

用域名解析服务器解析域名 返回 IP

寻址顺序：浏览器缓存　操作系统缓存　hosts 文件　DNS 服务器（主机名访问：内网每台机器在 hosts 文件内配置主机名对应专用网内主机 IP）【hosts 文件只能域名指向 ip】

【每个省 ISP 都会提供很多 DNS 服务器，　**8.8.8.8** 为谷歌 DNS 服务器在国内比较慢，　**192.168.x.x** 填写网关地址由网关自动寻找 DNS 服务器】

解析（从后向前）　　越后域级越高，访问越快（ www.a.com　　a.com 是不同的域名）

hosts 文件：　优先使用文件中的 IP 域名解析 -- 本机 DNS 寻址　　C:\Windows\System32\drivers\etc\hosts（第一请求之前操作）

防火墙：　　保护机器的网络入站和出站

Transmission

编码解码

编码：　字符　　转换为　　二进制文件

解码：　二进制文件　转换为　　字符

字符集：编码和解码所采用的规则，我们称为字符集（编码和解码采用的字符集不同=>乱码）

常见码表：　　ANSI（自动以系统默认编码保存 GB2312）

Unicode 万国码，统一了各个语种的 ASCII 码表，一个中文三个字节（支持所有语言，不管是在哪张码表中,中文的第一个字节一定是负数）　　UTF-8　　UTF-8 with BOM　（会在开头添加特殊字符）

ASCII　英文字符　　ISO-8859-1　　英文-字符

GBK　汉字，一个中文两字节　　GB2312　　中文系统浏览器默认汉字编码-支持繁体　BIG5 繁体中文

网络分类

范围分类：

广域网 WAN　　花钱买服务，花钱买带宽　　【外网】

城域网 MAN　　覆盖一个城市

局域网 LAN　　企业或个人组建的网络，自己买设备，自己维护，带宽固定，网线 100 米内　【内网】

个人区域网 PAN　　个人几台电脑相连

内网　Private Network　此网络不能和其他网络连接

公网　Public Network　此网络所有人都可以连接

拓扑结构：　总线型（电脑连接到一根线上）　环形（接到环形线上）　星型（接到路由器）　树型（不断向下延伸连接）　网状（多条路径）

报文传输

分组交换（报文拆开分组发送，路由存储转发）　【报文交换：整个报文发送，路由存储转发，　电路交换：报文直接发送，要先寻找传输线路并且占用线路】

存储转发：　路由器暂存数据包，查找转发表，找到最佳路径和转发端口发送数据包（报文---完整的二进制文件）

发送：地址头+分组数据包　首部+数据包 1　首部+数据包 2。。。。（共用线路）

接收：去掉数据头合并分组

性能指标

速率　单个信道的速率（每秒传输多少 bit）　传输比特：　b/s　kb/s　Mb/s　Gb/s　传输字节：　B/s　KB/s　MB/s　GB/s

带宽　单个信道的最高速率（每秒传输多少 bit）　b/s　kb/s　Mb/s　Gb/s【1kb=2^{10}b，1Mb=2^{10}kb=2^{20}b=1,048,576b，1Gb=2^{10}Mb=2^{20}kb=2^{30}b=1,073,741,824b】

吞吐量　经过某个网络数据速率（多个信道速率和）　b/s　kb/s　Mb/s

时延　发送时延 【数据块长度(b/s)　/　信道带宽(b/s)】　传播时延 【信道长度(m)　/　信号在信道传播速率(m/s)】　处理时延（查看数据包目的地并安排路径）　排队时延（数据包排队时间）

时延=发送时延+传播时延+处理时延+排队时延（ping 的时间）

时延带宽积　=传播时延*带宽（最高速率时，信道内有多少 bit）

往返时间 RTT(Round-Trip Time)　从发送方发送数据开始，到发送收到接收方确认

利用率　信道利用率 = 有数据通过的时间/有和无数据通过的时间　（利用率变高，时延也急剧增高）

数据传输：　单工（单向通信）　半双工（双向通信，但不能同时发送）　全双工（双向通信，可以同时发送）

Internet / Core

Binary

Binary（B）

Octal（O）

Octal numbers can only contain digits from 0 to 7. The digit "8" is not permitted in the octal numbering system.

Convert octal numbers to binary:

1) Convert each octal digit to its 3-bit binary equivalent.

Binary code Example: ()

0217 => 000 010 001 111

Decimal（D）

Convert decimal numbers to binary:

1) Divide the number by 2.
2) Record the remainder.

3) Continue dividing the quotient by 2, recording remainders, until the quotient is 0.

4) The binary representation is the sequence of remainders read from bottom to top.

Binary code Example:

23 => 10111

Hexadecimal（H） 【无论几进制计算，溢出都向前进 1】 【二进制：1001 可以表示 2^4=16 种状态，16 个数】

Convert hexadecimal numbers to binary:

1) Convert each hexadecimal digit to its 4-bit binary equivalent.

0x76 => 01110110

Original, inverse, complement, and binary offset codes

Original Code Represents signed integers directly, with one bit for the sign.

inverse code Represents negative numbers by inverting all the bits of the positive number.

complement code Represents negative numbers by adding 1 to the one's complement of the positive number's binary representation.

Binary Offset Code

Represents signed integers by shifting the entire range of values by a fixed offset, ensuring all numbers are positive.

To find the binary offset representation of a number, you need to:

1) Determine the bias.

In a signed 8-bit system, the bias would be $2^{(n-1)}=2^7=128$

2) Add the bias to the number.

16+128=144

3) Convert the result to binary.

144 in binary is 10010000

The original, inverse, complement, and binary offset codes represent the same number value in different formats.

For positive numbers, directly interpret the binary value as decimal.

For negative numbers, use the two's complement method to calculate the decimal value.

39	0010 0111 (~39 = -40)	
inverse code	1101 1000	
complement code	1101 1001	
-40	1101 1000	(The original code isn't usually used for negative numbers.)
inverse code	0010 0111	
complement code	0010 1000	

Operators

Priority level one

[ ] 数组下标

() 圆括号

. -> 获取成员

Priority level two

- Negative sign (priority operation on the right side at the same level)

~

Inverts the bits

The result 11011000 is a negative number because the most significant bit (MSB) is 1.

```
0010 0111   39
1101 1000  -40
```

++ --　　自增自减（后置时表达式计算完毕再进行计算）　　//TIPS：　(xx++)++　不能二次延迟　　++1　不能修改固定值

i--, pre=nums[i]　Out of Index error.

pre=nums[i], i--　Normal

* &　　地址取变量 取地址符

!　　非

(int)　　强制转换

sizeof

Arithmetic Operators

[* / %] > [+ -]

0%2 = 0

1%2 = 1

2%2 = 0

3%2 = 1

Bitwise Shift Operators

[<< >> >>>]

0 >> 1 = 0

Both >> and >>> are used to shift the bits towards the right.

The difference is that the >> preserve the sign bit while the operator >>> does not preserve the sign bit.

// 7<<1=14 -7<<1=-14 8<<1=16 -8<<1=-16 左移表示乘 二的 n 次方

// 7>>1=3 -7>>1=-4 8>>1=4 -8>>1=-4 右移表示除 二的 n 次方（奇数向下取整，取较小数）

// 1>>1 = 0

Relational Operators

[> >= < <=] > [== !=]

// 结果为 01 （0 为假，非 0 为真）

// 1.00 == 1 true 1.01==1 false

Bitwise Operators

& > ^ > |

0000 0101 & 0000 0111 = 0000 0101

0000 0101 ^ 0000 0111 = 0000 0010

0000 0101 | 0000 0111 = 0000 0111

0b101 & 0b110 = 0b100 (AND operation)

0b100 | 0b110 = 0b110 (OR operation)

-0b100 ^ 0b110 = -0b010 (XOR operation)

~(-0b10) = 1 (Bitwise NOT, two's complement inversion)

a^b^c^d=a^c^b^d

-k

For bitwise operations involving negative numbers, always use the two's complement representation of the negative number.

```
00000000 00101000  40   (16-bit, two's complement)
11111111 11011000  -40 (16-bit, two's complement)
```

k & 1

To check if a number is odd or even using bitwise operations:

100 & 1 = 0 (even)

101 & 1 = 1 (odd)

k & -k

Isolates the least significant set bit (LSB) in the binary representation of b.

Isolates the rightmost set bit.

```
40 & -40
     = 0010 1000  & 1101 1000
     = 0000 1000
b ^= lsb       # Remove the isolated bit from b
```

k -= k & -k or k &= k-1

The operation clears the rightmost set bit (1) in the binary representation of k.

```
11 & 10
     = 1011 & 1010
     = 1010 = 10
48 & 47
     = 0011 0000 & 0010 1111
     = 0010 0000 = 32
11 & 10
     = 1011 & 1010
     = 1010 = 10
```

int highestBit = Integer.highestOneBit(num);

Get the highest significant set bit.

(k+1) & ~k

Set the highest bit in the continuous sequence of rightmost `1` bits to `0`

```
(47 + 1) & ~47 = 0011 0000 & 1101 0000 = 0001 0000
0001 0000 >> 1 = 0000 1000
0010 1111 ^ 0000 1000 = 0010 0111
```

Logical Operators

&& > ||

The result is 0 or 1

`jj&&123&&DD` (If true, take the rightmost value)

`JJ||123||DD` (If true, take the leftmost value)

Priority level three

Ternary Conditional Operator

a>b **?** x **:** y

The right side is calculated first at the same level

In the middle, the function will not be called if the condition is not met, and the ternary will not execute the code that does not trigger true.

`true?false?true? true1:2 :3:4:5` Return 4

Assignment Operators

= *= /= %= += -= <<= >>= &= ^= |=

The right side is calculated first at the same level, the left side can only be a variable

The value of the assignment expression is the leftmost value after the assignment

`null || null || true` null cannot be used as a boolean value

Comma

, The value of the comma expression is the value of the last sub-expression

表达式: 创建和处理变量（一般表达式用于括号内） // int a=（a, b, c+d） --> a=c+d

此逗号表达式内含有算术表达式

位运算: 位运算负进制数都用补码计算，不足 4byte 的在前面补 0（最前方补 1） // -0b101 --> 10000000 00000000 00000000 00000101

<< 左移右移 正数原码计算，负数补码计算

& 位与位或 正负数都原码计算

ASCII 码: 48-56（0-9） 65-90（A-Z） 97-122（a-z） -32 变大写 +32 变小写

转义字符: \u 十六进制 Unicode 字符 \n linux 换行 \r\n windows 换行 \r mac 换行 \r 回车 \'单引号）

\"双引号 \ \反斜线 \? 问号 \0 空（显示为空格==0） \t 制表 \ddd（八进制） \xdd（十六进制）

\u200b \u200c \u200d \uFEFF 零宽字符

（Windows 文件在 Unix/Mac 里打开时，每行的结尾可能会多出一个^M 符号）

（Unix/Mac 文件在 Windows 里打开时，所有文字会变成一行）

时间戳: 1499825149 10 位秒级时间戳

1499825149257 13 位毫秒级时间戳

1499825149257892 16 位微秒级时间戳

Control Flow Statements

条件 if（）{ } { }可以看成整合的一条语句，只有一条语句时{}可省

判断括号内可以调用函数 //

if(func( ))

if（）func() else if( ){ } else func() 三条语句中只执行一条满足条件的语句

switch (xxval){ case 0: xxxx; break; default: xxxx; break; } 根据传入值匹配一个完全相等 case，直到遇到 break 才停止执行，所有 case 都不满足则执行 default（不能在 case 代码块中定义变量）

switch 内部如果用了 return 那么 switch 内部所有的语句将不会执行

循环 while( ){ } while，for 循环条件默认为真

do{ }while( ); 先执行后判断循环

for(; A ; B){ } 三个表达式都可省略，执行顺序为 A-->{ }-->B

for 内有函数时，第三个表达式不能操作与形参名相同的变量

// for(; !func(md,min,max); md+=max;)

第三个表达式可以为不带分号的长代码

// for(;; a++,b=a/(a+100),cout<<"c")

break 跳出当前循环（不包括外部循环）

continue 跳过循环到条件括号（for 循环里跳到第三个表达式，continue 只对循环有效，对 if 无效，）

Wildcard

正则匹配内容: 汉字 字母 数字 特殊符号 _ // _不算特殊符号 正则内部需要的转义字符: {} [] ( ) \ ^ $. | ? * +

匹配规则: 包含则匹配，固定顺序 // /./ 9999a9999 /abc/ 8888abckkkk

左侧正则匹配优先级越高，会让左侧正则尽可能匹配多的字符 // /(abb)|(ab)/ abb

元字符: . 除了换行和行结束符的任意字符，含空白符 (.* 匹配尽可能多的， .*? 匹配尽可能少的，如果不是在两个部分中间，则会什么也不匹配) //EX: /帅/

{n} 前方表达式 必须出现 n 次 // [a-z]{3} ayi

{n,} 出现[n,)次

{n, m} 出现[n, m]次

* 0 次或多次 // [a-z][0-9]* u2405750954 ([a-z][0-9])* a8s7k9z3

+ 1 次或多次 // (a)+ aaaaaa (([A-z]+) | ([A-z]+.*?>))

? 0 次或 1 次 // [0-9a-z]? 8

(?<=exp) 前面首个出现的是 exp，但不返回匹配字符（python 不支持 exp 可变长度 lookbehind）【不能搭配^ 或 $】

// "/project/dependencyManagement/dependencies/dependency".replaceAll("(?<=/)[A-z1-9]*?","p:$0") /p:project/p:dependencyManagement/p:dependencies/p:dependency

(?<! exp)　　前面首个出现的不是 exp

(?=exp)　　后面首个出现的是 exp，但不返回匹配字符　//EX:　(?=.*[A-z])(?=.*[0-9])　必须同时含 字母和数字

(?!exp)　　后面首个出现的不是 exp

(?<xxname>exp)　后面是 exp，分配组名 name，并添加捕获的文本

(?:exp)　　后面是 exp，不分配组名，不捕获匹配的文本（不能使用\1，提高正则匹配的效率）

[]　　包含其中任意一个，范围内可不适用转义字符　ASCII 编码顺序是连续时，可用 - 省略中间的字符　【ASCII 码存储:　48-56（0-9）　65-90（A-Z）　97-122（a-z）　-32 变大写　+32 变小写】

[^exp]　　不匹配所有范围（不匹配必须^开头）　//　[^a-b1-9]　不匹配 a-b 和 1-9　[z^a-b]　匹配 z 或 ^ 或 a-b　[^*|?]* = abcde　^[^(Java|Lua)]　不以 Lua 或者 Java 开头的行　^((?!ArgumentNullException).)*$　匹配不包含 ArgumentNullException 的行

[0-9a-zA-Z]　0 到 9 的数字或字母

[A-z]　大小写字母　//TIPS:　[100-200] [a-Z]　错误，ASCII 编码不连续

[._*]　包含任意一个就匹配　//TIPS:　[zz(sb)zz]　分组不会生效，不会匹配全部 sb，仅仅匹配一个字符】

|　或，优先级最低　//EX:　[0-9]|[a-z]　b|ara|don　b 或 ara 或 don 都可以匹配　/^\s+|\s+$/g;　空白开头或结尾　(
)|(<a.*=\")|(\">)|()　多情况匹配

()　分组，提升优先级，从左向右左括号顺序即分组顺序

^　限定字符串开始，取反（其他全部为包含匹配）　//EX:　^[0-9] 数字开头　|(^,)|(,$)　去掉结尾和开头的符号

$　限定字符串结束　//EX:　[0-9][a-z]$　必须小写字母结束　^[0-9][a-z]$　相当于严格模式 "9z"

\d　数字[0-9]　//EX:　\d{3}　321

\D　非数字

\s　空白符（空格 制表）

\S　非空白符

\w　非特殊符号　//EX:　[a-zA-Z0-9_]　下划线_属于非特殊符号

\W　特殊符号　//EX:　%#$@&*(@)　空白符属于特殊符号

\b　单词边界（/w　/W 之间）

\B　匹配非单词边界

\0　查找 NUL 字符

\n　查找换行符

\f　查找换页符

\r　查找回车符

\t　查找制表符

\v　查找垂直制表符

\141　八进制 ASCII 码

\x61　十六进制 ASCII 码

\u4e00　十六进制 Unicode 字符　//TIPS:　\u4e00-\u9fa5　汉字　(?<![-:\\]) //([^\u4e00-\u9fa5]*?) (\n)　不包含汉字的注释　\};([\n])*?(export)

\p{Nd} 或 \p{Decimal_Digit_Number}:　所有文本中的数字 0 至 9 字符，不含形意符号。

\p{Nl} 或 \p{Letter_Number}:　看起来像字母的符号，包含罗马数字。

\p{No} 或 \p{Other_Number}:　　上角标或下角标数字，或者其他不属于 0 至 9 的数字。不含形意符号。

\p{L}　　所有字母 3
\p{N}　　所有数字，类似于 \d 4
[\p{N}\p{L}]　　所有数字和所有字母，类似于 \w 4
\P{L}　　不是字母，等价于 [^\p{L}]
\P{N}　　不是数字，等价于 [^\p{N}]

表达式　/\d{5}/**g**　/xxx/g 表示全局模式（取出符合条件的所有内容）
　　/[h]/g**i**　/xxx/i 表示忽略大小写
　　/[h]/**m**　多行匹配
子表达式　/(a(b(c)))/　"abc"　--->　"abc","abc","bc","c"　从外向内匹配，全局为（a(b(c))）第一个子表达式为 a(b(c))，当加全局匹配 g 时，不会返回子表达式的结果
　　/(a(b(c)))\1/　"abcabc"　--->　"abcabc","abc","bc","c"　从外向内匹配，\1 为取出 前方匹配的第一分组匹配的值（分组嵌套时，以左括号为顺序区分）

贪婪　*　+　{n,}　??　满足条件的前提下，匹配尽可能多的
　　{n,m}?　{n,}?　{n}?　+?　*?　满足条件的前提下，尽量匹配少的　//　a{2,3}?　aaaa　-->　'aa' 'aa'

邮箱号　(-放到最后防止解析为范围，后缀可能是一级域名可能是二级域名.com.cn @后表达式的.可去掉，防止解析为域名)　[0-9a-zA-Z_.-]+[@][0-9a-zA-Z_.-]+([.][a-zA-Z]+){1,2}
身份证号码　(15 位或 18 位，首位不为 0，末位录入为 xX 保存为 X)　([1-9][0-9]{14})([0-9]{2}[0-9xX])?
座机号码　(010-19876754　0431-87123490)　\d{3,4}[-]\d{7,8}
qq 号码　(首位不为 0　5 位到 11 位)
手机号码　(首位 1　11 位)
邮编　(6 位)
密码　不为空，4-12 位，英文、数字或下划线组成　/([a-zA-Z0-9]|[._]|\W){4,12}/

CRON

秒 分钟 小时 日期 月份 星期 年　(spring 4.x 的 spring task 中只支持前 6 种时间元素)
0 0 9-18 * *　- 表示时间段
0 0 14 L * ?　每个最后一天，下午 2 点发工资的时间，没有具体说明是星期几，通常用问号代替
0/10 * * * * ?
0 0/3 * * * ?　每小时的第 0 分 0 秒开始，每三分钟触发一次　（ / 符号前表示开始时间，符号后表示每次递增的值）
0 0 1 * * Sun　?　代表一天何时会发生的时间:8:05
0 0 1 * * *　每天凌晨 1:00 运行一次

0 0 3 * * ? ：每天 3 点执行;
0 5 3 * * ?: 每天 3 点 5 分执行;
0 5 3 ? * *: 每天 3 点 5 分执行，与上面作用相同;
0 5/10 3 * * ?: 每天 3 点的 5 分、15 分、25 分、35 分、45 分、55 分这几个时间点执行;
0 10 3 ? * 1:　每周星期天，3 点 10 分执行，注，1 表示星期天;
0 10 3 ? * 1#3: 每个月的第三个星期，星期天执行，# 号只能出现在星期的位置。

Process

并发：在同一时刻，有多个指令在单个 CPU 上交替执行

并行：在同一时刻，有多个指令在多个 CPU 上同时执行。

进程：代表正在运行的程序，可以同其他进程一起并发执行，是系统分配资源的基本单位（内核中的任务调度，实际上的调度对象是线程；而进程只是给线程提供了虚拟内存、全局变量等资源）

线程：是进程中的一条执行路径，是 CPU 调度的基本单位（应用程序中做的事情，如 306 软件杀毒，扫描木马，清理垃圾）

单线程：一个进程如果只有一条执行路径，则称为单线程程序

多线程：一个进程如果有多条执行路径，则称为多线程程序

线程调度：计算机中的 CPU，在任意时刻只能执行一条机器指令。每个线程只有获得 CPU 的使用权才能执行代码。

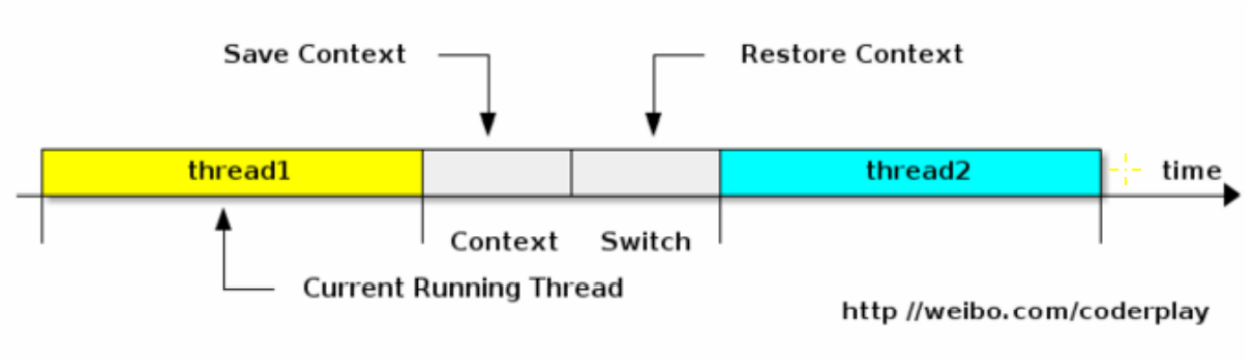
分时调度模型：所有线程轮流使用 CPU 的使用权，平均分配每个线程占用 CPU 的时间片

抢占式调度模型：优先让优先级高的线程使用 CPU，如果线程的优先级相同，那么会随机选择一个，优先级高的线程获取的 CPU 时间片相对多一些

守护线程：也就是后台线程，为用户线程提供公共服务，当程序终止也就是所有非守护线程都执行完毕时，杀死进程中的所有守护线程。

线程上下文切换：使用时间片轮换的方式， CPU 让每个任务都执行一定的时间，到时间后将当前任务的状态保存下来，再加载下一个任务的状态并执行（可能在执行任何一句代码时，失去执行权）

时间片轮转的方式使多个任务在同一个 CPU 上执行变成了可能



上下文：是指某一时间点 CPU 寄存器和程序计数器的状态

寄存器：是 CPU 内部的数量较少但是速度很快的内存（与之对应的是 CPU 外部相对较慢的 RAM 运行内存）。寄存器通过对常用值（通常是运算的中间值）的快速访问来提高计算机程序运行的速 度

程序计数器：是一个专用的寄存器，用于表明指令序列中 CPU 正在执行的位置，存的值为正在执行的指令 的位置或者下一个将要被执行的指令的位置，具体依赖于特定的系统。

切换桢（PCB）：存储线程的上下文状态。

上下文切换过程：

挂起一个线程，将这个线程的上下文存储到 PCB 中。

在 PCB 中查找下一个线程的上下文，并恢复 CPU 的寄存器中的状态，跳转到之前程序计数器所指向的位置，继续执行。

引起线程上下文切换的原因：

线程时间片全部用完

线程提前执行完毕。

线程主动放弃本次时间片的执行权。

线程进入 IO 阻塞状态。

多个任务抢占锁资源，当前任务没有抢到锁资源，被调度器挂起，继续下一任务；

硬件中断；

指令重排：在程序执行过程中，为了性能考虑，编译器和 CPU 可能会对指令重新排序.

```
public void writer() {
    // 以下两句执行顺序可能会在指令重排等场景下发生变化
    a = 1;
    flag = true;
}
```

System

内核：内核是操作系统的核心，可以控制计算机的硬件设备，也可以管理内存，文件，创建和销毁进程

伪共享：CPU 缓存系统中是以缓存行（cache line）为单位存储的。目前主流的 CPU Cache 的 Cache Line 大小都是 64Bytes。
一个缓存行可以存储多个变量（存满当前缓存行的字节数）；而 CPU 对缓存的修改又是以缓存行为最小单位的
在多线程情况下，如果需要修改"共享同一个缓存行的变量"，就会无意中影响彼此的性能，这就是伪共享（False Sharing）

缓存行：Cache Line 可以简单的理解为 CPU Cache 中的最小缓存单位，今天的 CPU 不再是按字节访问内存，而是以 64 字节为单位的块(chunk)拿取，称为一个缓存行(cache line)。
当你读一个特定的内存地址，整个缓存行将从主存换入缓存，并且访问同一个缓存行内的其它值的开销是很小的。

静态库和动态库如何制作及使用：静态库在程序的链接阶段被复制到了程序中；动态库在链接阶段没有被复制到程序中，而是程序在运行时由系统动态加载到内存中供程序调用。

磁盘接口速度排序： SAS > NL-SAS > SATA > SCSI > IDE

Timezone

UTC 世界协调时间
GMT 格林尼治时间(UTC and GMT are identical)

CST（Central Standard Time） 中央标准时间，中国、澳大利亚、古巴、美国 的标准时间
中国标准时间：China Standard Time UT+8:00
古巴标准时间：Cuba Standard Time UT-4:00
美国中部时间：Central Standard Time (USA) UT-6:00
澳大利亚中部时间：Central Standard Time (Australia) UT+9:30

UTC/GMT+8 东八区（中华人民共和国、蒙古、马来西亚、菲律宾、新加坡、印尼、文莱、澳大利亚、俄罗斯）

XPath

Find

nodename Selct current node.
/ Select nodes from the root node.
// Select nodes in the document from the current node selected by matching, regardless of their position.
//*[@id="recaptcha-anchor"]/div[1] 或者 .//*[@href and @lmv]

. Select the current node.
.. Select the parent node of the current node.
@ Select attributes

Select unknown node

* Match any element node.
/bookstore/* Select all child nodes of the bookstore element.
//* Select all elements in the document.
@* Match any attribute node.

//title[@*] Select all title elements with attributes.

node() Match any type of node.

Select associated node

ancestor 选取当前节点的所有先辈（父、祖父等）。

ancestor-or-self 选取当前节点的所有先辈（父、祖父等）以及当前节点本身。

attribute 选取当前节点的所有属性。

child 选取当前节点的所有子元素。

descendant 选取当前节点的所有后代元素（子、孙等）。

descendant-or-self 选取当前节点的所有后代元素（子、孙等）以及当前节点本身。

following 选取文档中当前节点的结束标签之后的所有节点。

namespace 选取当前节点的所有命名空间节点。

parent 选取当前节点的父节点。

preceding 选取文档中当前节点的开始标签之前的所有节点。

preceding-sibling 选取当前节点之前的所有同级节点。

self 选取当前节点。

//div[@id="content" and @id="ul"]/ul[@id="ul"]/li/text() 虚拟路径 使用“@标签属性”获取 a 便签的 href 属性值

//table[2] 当前目录第二个 table

//div[contains(@style,"xxx")][@type!="submit"]') 属性 style 包含 xxx type 不等于 submit

//a[last()-1] 倒数第二个

./preceding-sibling::td[2] （当前节点之前的节点）或者 following-sibling（当前节点之后的节点）

//div[@class='el-tab-pane' and not(contains(@style,'none'))]//button[./span[text()='确定']] 不包含

//tr[not(@id) and not(@class)] 不包含属性

./ div[@class='el-tab-pane' and not(contains(@style,'none'))]

System

ROM 种类

PROM, 只能写入一次 已淘汰

EPROM 用紫外线清除数据

EEPROM 用电 清除数据

FLASH 闪存（允许多次擦写）

IO 口输出模式

开漏输出 输出低电平接地，输出高电平时，电压会拉到上拉电阻的电源电压（外接上拉电阻）

推挽输出 输出低电平接地，输出高电平时为单片机电源电压。（可不接上拉电阻）

上拉电阻（在输出线上接电阻或三极管 增大输出电压）

烧录方式

ISP 整个程序的擦除和写入，（而 BOOT0 接 V3.3,BOOT1 接 GND.则是 ISP 模式，也就是串口更新代码）【TTL：TXD 接 PA10 RXD 接 PA9 BOOT0 接 3V3 后再供电】

IAP 部分程序的擦除和写入，烧写期间程序可运行（BOOT0 接 GND,BOOT1 接 GND.那就是正常的启动模式，从 flash 加载代码.）

ICP 在电路编程

存储器（4 个独立存储器空间）

1. 只读存储 ROM 片内 ROM (4KB 0000H - 0FFFH EA 端选择)

MOVC

外部可扩展 ROM I/O（最大 64KB　1000H - 0FFFFH）　【和片内 ROM 统一编址】

2. 随机存储 RAM　片内 RAM　（128B　00H - 7FH 可以直接赋值　80H - 0FFH 特殊功能寄存器 SFR）　MOV

外部可扩展 RAM　（最大 64KB　1000H - 0FFFFH）

MOVX

What is the difference between ROM and RAM?

RAM (Random Access Memory) and ROM (Read Only Memory) are both types of computer memory.
RAM is volatile memory that temporarily stores data that is currently being worked on,
while ROM is non-volatile memory that permanently stores instructions for the computer.

Internet / Network Model

OSI Model

The OSI model **(Open Systems Interconnection Model)** is a conceptual framework that standardizes the functions of a telecommunication or computing system into seven distinct layers.
Each layer is responsible for specific tasks and interacts with adjacent layers to facilitate communication between devices.
The OSI model layers are:

Physical Layer

Deals with the physical transmission of data over a communication channel.

Ethernet (IEEE 802.3)

Description: Widely used for wired LANs.
Functionality: Specifies physical medium, signal encoding, and network topology.

Wi-Fi (IEEE 802.11)

Description: Standard for wireless LANs.
Functionality: Defines radio frequencies, signal modulation, and transmission power.

DSL (Digital Subscriber Line)

Description: Used for high-speed internet over telephone lines.
Functionality: Utilizes higher frequency bands for data transmission, separate from voice frequencies.

Optical Fiber

Description: Used for high-speed data transmission using light.
Functionality: Uses light pulses to transmit data over long distances with minimal signal loss.

Bluetooth (IEEE 802.15.1)

Description: Standard for short-range wireless communication.
Functionality: Operates in the 2.4 GHz ISM band, using frequency hopping spread spectrum.

USB (Universal Serial Bus)

Description: Standard for wired communication and power supply between computers and devices.
Functionality: Defines connectors, cables, and protocols for data transfer and power delivery.

Data Link Layer

Manages the physical addressing of devices on the network and ensures reliable data transfer between adjacent nodes.

Ethernet (IEEE 802.3):

One of the most widely used LAN technologies.
Defines the standards for wired Ethernet networks, including Ethernet frames, MAC addressing, and collision detection.

Wi-Fi (IEEE 802.11):

Wireless LAN technology that enables wireless communication between devices.

Defines protocols for wireless networking, including frame formats, media access control (MAC), and security mechanisms.

PPP (Point-to-Point Protocol):

Used for establishing a direct connection between two nodes over a serial link, such as a dial-up connection or a leased line.

Supports authentication, error detection, and multilink connections.

HDLC (High-Level Data Link Control):

A bit-oriented synchronous data link layer protocol used for communication between two directly connected devices.

Widely used in WAN environments and forms the basis for other protocols like LAPB (Link Access Procedure, Balanced) and SDLC (Synchronous Data Link Control).

ARP (Address Resolution Protocol):

Used to map IP addresses to MAC addresses in a local network.

Allows devices to discover the MAC address of another device on the same subnet when its IP address is known.

SLIP (Serial Line Internet Protocol):

A simple protocol used for transmitting IP packets over serial connections.

Provides a basic method for encapsulating IP packets for transmission over serial lines.

PPPoE (Point-to-Point Protocol over Ethernet):

Extends PPP over Ethernet networks, allowing ISPs to provide broadband services over DSL and cable modem connections.

Enables the establishment of PPP sessions over Ethernet links, typically used for DSL internet connections.

MAC (Media Access Control) Protocols:

Various MAC protocols govern access to the shared communication medium in local networks.

Examples include CSMA/CD (Carrier Sense Multiple Access with Collision Detection) used in Ethernet networks and CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) used in Wi-Fi networks.

Network Layer

Handles routing and forwarding of data packets across different networks.

IPv4 (Internet Protocol version 4)

The most widely used network layer protocol for routing packets across the internet.

Uses 32-bit addresses to uniquely identify devices on a network and facilitate packet delivery.

IPv6 (Internet Protocol version 6)

The next-generation internet protocol designed to address the limitations of IPv4, such as address exhaustion and lack of security features.

Uses 128-bit addresses and provides improved scalability, security, and support for emerging technologies.

ICMP (Internet Control Message Protocol):

A network layer protocol used for exchanging control and error messages between network devices.

Provides mechanisms for diagnosing network connectivity issues, such as ping and traceroute.

ARP (Address Resolution Protocol):

A protocol used to map IP addresses to MAC addresses in local area networks (LANs).

Allows devices to discover the MAC address corresponding to an IP address and facilitate communication within the same network segment.

RARP (Reverse Address Resolution Protocol):

A protocol used to map MAC addresses to IP addresses.

Primarily used in legacy environments to enable diskless workstations to obtain their IP addresses dynamically.

OSPF (Open Shortest Path First):

A link-state routing protocol used to determine the best path for routing packets within an autonomous system (AS).

Provides dynamic routing capabilities and supports features such as load balancing and route summarization.

BGP (Border Gateway Protocol):

A path vector routing protocol used to exchange routing information between autonomous systems (ASes) on the internet.

Enables internet service providers (ISPs) to make routing decisions based on policies and network conditions.

IPsec (Internet Protocol Security):

A suite of protocols used to secure IP communications by providing authentication, integrity, and confidentiality.

Used to establish secure VPN connections and protect data transmitted over IP networks.

Transport Layer

Provides end-to-end communication between devices, ensuring data integrity and flow control.

TCP (Transmission Control Protocol):

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite, providing reliable, ordered, and error-checked delivery of data between applications running on hosts communicating via an IP network.

Key Features of TCP

- Connection-Oriented
  TCP establishes a connection between the sender and receiver before data transmission begins, ensuring that both parties are ready for communication.
- Reliable Data Transfer
  TCP guarantees the delivery of data packets to their destination. If packets are lost or corrupted, they are retransmitted.
- Ordered Delivery
  TCP ensures that packets are delivered in the same order in which they were sent.
- Error Checking
  TCP uses checksums to verify the integrity of the data.
- Flow Control
  TCP uses flow control mechanisms to prevent overwhelming the receiver with too much data at once.
- Congestion Control
  TCP employs congestion control algorithms to avoid congesting the network.

Structure of a TCP Segment

A TCP segment consists of a header and data. The header includes several fields such as:

- Source Port (16 bits)
  The port of the sender.
- Destination Port (16 bits)
  The port of the receiver.
- Sequence Number (32 bits)
  Used to order the data packets.
- Acknowledgment Number (32 bits)
  Indicates the next expected byte from the sender.
- Data Offset (4 bits)

Indicates the size of the TCP header.

- Flags (9 bits)
  Includes control flags such as SYN, ACK, FIN, etc.
- Window Size (16 bits)
  Specifies the size of the sender's receive window.
- Checksum (16 bits)
  Used for error-checking of the header and data.
- Urgent Pointer (16 bits)
  Indicates if any data is urgent.
- Options (variable length)
  May include additional options for the connection.

How TCP Works

1) Connection Establishment (Three-Way Handshake)

SYN The client sends a synchronization (SYN) packet to the server to initiate a connection.

Flags: SYN
Sequence Number: 1000 (arbitrary starting point)
Acknowledgment Number: Not set (0 or empty)
Packet:

```
SYN=1, ACK=0, Seq=1000, Ack=0
```

SYN-ACK The server responds with a synchronization-acknowledgment (SYN-ACK) packet.

Flags: SYN, ACK
Sequence Number: 2000 (arbitrary starting point)
Acknowledgment Number: 1001 (client's sequence number + 1)
Packet:

```
SYN=1, ACK=1, Seq=2000, Ack=1001
```

ACK The client sends an acknowledgment (ACK) packet, completing the handshake and establishing the connection.

Flags: ACK
Sequence Number: 1001 (client's sequence number + 1)
Acknowledgment Number: 2001 (server's sequence number + 1)
Packet:

```
SYN=0, ACK=1, Seq=1001, Ack=2001
```

2) Data Transfer

Segmentation Data is divided into segments and sent over the network.
Sequencing Each segment is numbered to ensure ordered delivery.
ACK The receiver sends an acknowledgment for received segments.
Retransmission Lost or corrupted segments are retransmitted based on acknowledgments and timeouts.

3) Connection Termination

FIN The client or server sends a finish (FIN) packet to terminate the connection.

Flags: FIN
Sequence Number: 1500 (arbitrary current sequence number)
Acknowledgment Number: Current acknowledgment number (e.g., 3000)
Packet:

```
FIN=1, ACK=0, Seq=1500, Ack=3000
```

ACK The other party acknowledges the FIN packet.

Flags: ACK
Sequence Number: 3000 (arbitrary current sequence number)
Acknowledgment Number: 1501 (client's sequence number + 1)

Packet:

```
FIN=0, ACK=1, Seq=3000, Ack=1501
```

FIN — The other party sends its own FIN packet.

Flags: FIN

Sequence Number: 3000 (current sequence number)

Acknowledgment Number: 1501 (client's sequence number + 1)

Packet:

```
FIN=1, ACK=0, Seq=3000, Ack=1501
```

ACK — The original sender acknowledges the FIN packet, completing the termination.

Flags: ACK

Sequence Number: 1501 (client's sequence number + 1)

Acknowledgment Number: 3001 (server's sequence number + 1)

Packet:

```
FIN=0, ACK=1, Seq=1501, Ack=3001
```

Advantages of TCP

Reliability	Ensures that data is delivered accurately and in order.
Error Handling	Detects and corrects errors through retransmissions.
Flow Control	Manages the rate of data transmission based on the receiver's capacity.
Congestion Control	Adjusts the transmission rate based on network conditions to avoid congestion.

UDP (User Datagram Protocol):

The User Datagram Protocol (UDP) is a core protocol of the Internet Protocol Suite used for transmitting data over a network.

Unlike its counterpart, Transmission Control Protocol (TCP), UDP is connectionless and provides a minimal service for exchanging messages without guaranteeing delivery, ordering, or error checking.

Key Features of UDP

- Connectionless
  UDP does not establish a connection before data transmission. Each packet (datagram) is sent independently, and no handshake is required.
- Unreliable
  There is no guarantee that packets will reach their destination. Packets may be lost, duplicated, or arrive out of order.
- Low Overhead
  UDP has a smaller header compared to TCP, leading to less overhead and faster data transmission.
- No Congestion Control
  UDP does not implement congestion control mechanisms, which means it can be faster but also more prone to overwhelming the network.

Structure of a UDP Datagram

A UDP datagram consists of a simple header and payload. The header includes:

- Source Port (16 bits) — The port of the sender.
- Destination Port (16 bits) — The port of the receiver.
- Length (16 bits) — The length of the UDP header and data.
- Checksum (16 bits) — Used for error-checking of the header and data.

How UDP Works

- Data Transmission
  The sender creates a datagram, attaches the necessary headers, and sends it to the recipient's IP address and port.
- Reception
  The recipient's application reads the datagram from the network buffer, processes it, and extracts the

data.

- No Acknowledgement
  The recipient does not send an acknowledgment to the sender, reducing the protocol overhead but also removing reliability guarantees.

Advantages of UDP

- Speed
  The lack of connection setup and minimal protocol overhead makes UDP faster than TCP.
- Efficiency
  Suitable for applications where speed and efficiency are more critical than reliability.
- Broadcast and Multicast
  Supports broadcasting (sending data to all devices in a network) and multicasting (sending data to multiple specified devices).

SCTP (Stream Control Transmission Protocol):

A transport layer protocol that combines features of both TCP and UDP.
Provides reliable, ordered, and flow-controlled delivery of messages in streams, but with support for multi-homing and message-oriented communication.

DCCP (Datagram Congestion Control Protocol):

A transport layer protocol designed for streaming media applications and other real-time communication services.
Provides congestion control and support for unreliable and unreliable message delivery.

RTP (Real-time Transport Protocol):

A protocol used for delivering real-time multimedia data, such as audio and video streams, over IP networks.
Provides mechanisms for timing synchronization, loss detection, and compensation for network jitter.

RTSP (Real-time Streaming Protocol):

A network control protocol used for controlling streaming media servers and clients.
Allows users to control the playback of streaming media content, such as starting, stopping, pausing, and seeking.

SPX (Sequenced Packet Exchange):

A transport layer protocol used in Novell NetWare networks for reliable communication between networked devices.
Provides connection-oriented, reliable, and sequenced delivery of data packets.

QUIC (Quick UDP Internet Connections):

A modern transport layer protocol developed by Google that combines features of TCP and UDP.
Designed to improve the performance of web applications by reducing latency and connection setup time.

Session Layer

Establishes, maintains, and terminates connections between devices.

RPC (Remote Procedure Call):

A protocol that allows a program to call functions or procedures on a remote server as if they were local.
Facilitates inter-process communication between distributed systems.

NetBIOS (Network Basic Input/Output System):

A networking protocol suite that provides communication services and naming conventions for computers on a local area network (LAN).
Used for sharing resources, such as files and printers, and for establishing sessions between networked devices.

PPTP (Point-to-Point Tunneling Protocol):

A protocol used to establish and manage virtual private network (VPN) connections over the internet.
Allows users to securely access private networks from remote locations.

NFS (Network File System):

A distributed file system protocol that allows clients to access files and directories stored on remote servers over a network.
Provides transparent remote access to files, enabling seamless file sharing and data management across multiple systems.

SMB (Server Message Block):

A network protocol used for providing shared access to files, printers, and other resources on a network.
Used primarily in Windows-based environments for file and print sharing.

AFP (Apple Filing Protocol):

A proprietary network protocol developed by Apple Inc. for sharing files, directories, and other resources between Macintosh computers.
Facilitates communication between Mac clients and Apple Filing Protocol servers.

SDP (Session Description Protocol):

A protocol used to describe multimedia session information, such as session timing, media types, and transport protocols.
Used in conjunction with other protocols like RTP (Real-time Transport Protocol) for establishing and managing multimedia sessions over IP networks.

SIP (Session Initiation Protocol):

A signaling protocol used for initiating, modifying, and terminating real-time communication sessions, such as voice and video calls, over IP networks.
Widely used in Voice over IP (VoIP) and Unified Communications (UC) systems.

Presentation Layer

Handles data translation, encryption, and compression to ensure compatibility between different systems.

JPEG (Joint Photographic Experts Group)

A compression standard for digital images commonly used for storing and transmitting photographs and other continuous-tone images.
Enables efficient storage and transmission of image data by reducing file size without significant loss of image quality.

GIF (Graphics Interchange Format):

A bitmap image format that supports animation and transparency.
Frequently used for web graphics and simple animations.

TIFF (Tagged Image File Format):

A flexible format for storing raster graphics images and data.
Supports multiple layers, various color depths, and compression algorithms.

MPEG (Moving Picture Experts Group):

A set of standards for audio and video compression and transmission.
Includes formats such as MPEG-1, MPEG-2, and MPEG-4, used for digital video broadcasting, DVDs, and streaming media.

ASCII (American Standard Code for Information Interchange):

A character encoding standard for representing text in computers and communication devices.

Assigns numeric codes to characters, allowing them to be represented as binary data.

Unicode:

A character encoding standard that aims to represent all characters from all writing systems in the world.
Supports a vast range of characters, including those from different languages, symbols, and emoji.

SSL/TLS (Secure Sockets Layer/Transport Layer Security):

SSL (Secure Sockets Layer)

SSL was developed by Netscape in the mid-1990s to secure data transmitted over the internet.
It ensures that the data sent between a client (e.g., a web browser) and a server (e.g., a web server) is encrypted and private.

Versions

SSL has had several versions, with SSL 3.0 being the most notable.
However, SSL 2.0 and SSL 3.0 are both considered insecure today due to various vulnerabilities.

TLS (Transport Layer Security)

TLS was developed as an improvement upon SSL.
The first version, TLS 1.0, was essentially SSL 3.1. Since then, there have been several versions, with TLS 1.2 and TLS 1.3 being widely used today.

TLS 1.2

Introduced in 2008, it improved security and performance over previous versions.

TLS 1.3

Released in 2018, it further enhanced security and efficiency, reducing the handshake process and deprecating outdated cryptographic algorithms.

Key Components of SSL/TLS

Encryption

Encrypts data to ensure privacy. Symmetric encryption is used for the actual data transfer, while asymmetric encryption is used during the handshake process.

Authentication

Uses certificates (typically issued by Certificate Authorities, or CAs) to authenticate the server, and optionally the client, to ensure the communicating parties are who they claim to be.

Integrity

Ensures data integrity using message authentication codes (MACs) to prevent data from being tampered with during transmission.

How SSL/TLS Works

Handshake Process

1) Client Hello

The client initiates the TLS handshake by sending a "Client Hello" message to the server. This message includes:

- The TLS version supported by the client.
- A list of supported cipher suites (which include the key exchange algorithms).
- A randomly generated number (Client Random).
- Other options and extensions.

2) Server Hello

The server responds with a "Server Hello" message, which includes:

- The TLS version selected by the server.
- The chosen cipher suite (including the key exchange algorithm).
- A randomly generated number (Server Random).
- The server's digital certificate (containing its public key).

This certificate contains the server's public key, which the client will use for key exchange.

If the chosen cipher suite involves an additional step (such as in Diffie-Hellman or Elliptic Curve Diffie-Hellman),
the server will also send parameters necessary for key exchange (e.g., Diffie-Hellman parameters).

3) Certificate Verification

The client verifies the server's certificate against trusted CAs. If valid, the process continues.
In some cases (e.g., mutual authentication), the server may request the client's certificate. The client would then provide its certificate for the server to authenticate.

4) Key Exchange

Depending on the chosen cipher suite, a key exchange mechanism is used (e.g., RSA, DH, ECDHE) to securely share the encryption keys.
The specific key exchange mechanism is executed based on the selected cipher suite. Common key exchange methods include:

RSA Key Exchange:

The client generates a "pre-master secret," encrypts it with the server's public key (from the certificate), and sends it to the server. The server decrypts this pre-master secret with its private key.

Diffie-Hellman (DH) Key Exchange:

Both client and server exchange public values based on pre-agreed parameters, and each party calculates the shared secret key using their private value and the other party's public value.

Elliptic Curve Diffie-Hellman (ECDHE) Key Exchange:

Similar to DH but uses elliptic curve cryptography for better security and performance.

5) Session Keys

Both the client and server use the agreed-upon key exchange mechanism to derive a shared "pre-master secret."
This pre-master secret, along with the Client Random and Server Random values, is used to generate the session keys:

Encryption keys: For encrypting the data.
MAC keys: For ensuring the integrity of the data.
Initialization vectors (IVs): For some cipher modes.

6) Finished Messages

Both parties send "Finished" messages to each other. These messages are encrypted with the session keys.
The "Finished" message includes a hash of all the previous handshake messages, ensuring the integrity and success of the handshake.

Data Transfer

1) Encryption

- Symmetric Encryption

Data exchanged between the client and server is encrypted using the session keys. Common algorithms include AES (Advanced Encryption Standard).

2) Integrity

Each message is accompanied by a MAC to ensure integrity.

- Message Authentication Code (MAC)

Each encrypted message is accompanied by a MAC to ensure data integrity.

- MAC Calculation

Uses a secure hash function (e.g., HMAC-SHA256) and the session keys to verify that the data has not been tampered with.

Termination

1) Session Resumption

If supported, the client and server can resume a previous session without performing a full handshake, using session identifiers or session tickets.

2) Session Closure

When either the client or server decides to end the session, they exchange "Goodbye" messages and close the connection.

SSH (Secure Shell)

SSH (Secure Shell) is a cryptographic network protocol used to secure network services over an unsecured network.
It provides a secure channel over an unsecured network by using a client-server architecture, enabling secure remote login and other network services.
SSH is widely used for secure access to servers, remote command execution, and secure file transfer.

How SSH Works

1. Connection Establishment

   The SSH client initiates a connection to the SSH server on a specific port (typically port 22).
2. Handshake and Key Exchange
   - Protocol Version Exchange

     The client and server exchange protocol versions to ensure compatibility.
   - Key Exchange

     The client and server perform a key exchange to securely share encryption keys.

     Common methods include Diffie-Hellman (DH) and Elliptic Curve Diffie-Hellman (ECDH).
3. Server Authentication

   The server sends its public host key to the client.

   The client verifies the server's host key against a list of known hosts to ensure it is connecting to the intended server.
4. Client Authentication

   The client authenticates itself to the server using one of the supported methods (e.g., public key, password).
5. Session Establishment:

   Once the client is authenticated, an encrypted session is established.

   Both parties negotiate encryption and integrity algorithms to secure the session.
6. Data Transfer:

   Encrypted Communication

   All data transferred between the client and server is encrypted using symmetric encryption algorithms (e.g., AES).

   Integrity Checks

   Message authentication codes (MACs) are used to ensure data integrity and detect tampering.
7. Connection Closure:

   The client or server can terminate the connection at any time.

   The termination involves securely closing the encrypted session and terminating the TCP connection.

Application Layer

Provides network services directly to end-users and applications.

HTTPS (Hypertext Transfer Protocol Secure)**:**

A secure version of HTTP that uses encryption (SSL/TLS) to provide secure communication over the internet.

Used for transmitting sensitive information, such as login credentials and payment details, securely.

How HTTPS Works

1. Establishing a Secure Connection (SSL/TLS Handshake)

   1) Client Hello

      The client (e.g., web browser) sends a "Client Hello" message to the server to initiate the handshake process.

This message includes:

The SSL/TLS version the client supports.

A list of supported cipher suites (encryption algorithms).

A randomly generated number (nonce).

2) Server Hello

The server responds with a "Server Hello" message.

This message includes:

The SSL/TLS version and cipher suite chosen by the server.

A randomly generated number (nonce).

The server's digital certificate, which contains its public key and is signed by a trusted Certificate Authority (CA).

3) Certificate Verification

The client verifies the server's digital certificate.

This involves checking the certificate's validity, ensuring it was issued by a trusted CA, and confirming that it matches the server's domain name.

If the certificate is valid, the process continues. If not, the connection is terminated.

4) Key Exchange

Depending on the chosen cipher suite, the client and server perform a key exchange to securely share encryption keys.

RSA

The client encrypts a pre-master secret with the server's public key and sends it to the server.

Diffie-Hellman (DH)/Ephemeral DH (DHE)/Elliptic Curve DH (ECDH)

Both parties exchange public keys and use them to compute a shared secret.

5) Session Keys

Both the client and server use the shared secret and the nonces to generate session keys. These keys are symmetric encryption keys used to encrypt and decrypt data during the session.

6) Finished Messages

Both parties send a "Finished" message encrypted with the session keys to verify that the handshake was successful and the keys match.

These messages contain a hash of all previous handshake messages.

2. Secure Data Transfer

1) Encryption

Once the secure connection is established, all data exchanged between the client and server is encrypted using the session keys.

Symmetric Encryption

Common algorithms include AES (Advanced Encryption Standard) for fast and secure encryption.

2) Data Integrity

Each encrypted message is accompanied by a Message Authentication Code (MAC) to ensure data integrity.

MAC Calculation

Uses a secure hash function (e.g., HMAC-SHA256) and the session keys to verify that the data has not been tampered with.

3. Termination

1) Session Resumption

If supported, the client and server can resume a previous session without performing a full handshake, using session identifiers or session tickets.

2) Session Closure

When either the client or server decides to end the session, they exchange "Goodbye" messages and close the

connection.

FTP (File Transfer Protocol):

Used for transferring files between a client and a server on a computer network.

Supports operations like uploading, downloading, renaming, and deleting files.

SMTP (Simple Mail Transfer Protocol):

Used for sending and receiving email messages between email servers.

Handles the transmission of email messages from the sender's email client to the recipient's email server.

DNS (Domain Name System):

Translates domain names (e.g., www.example.com) into IP addresses (e.g., 192.0.2.1) and vice versa.

Enables users to access websites using human-readable domain names instead of numerical IP addresses.

Key Components

1) Domain Names

   Hierarchical and readable names assigned to resources on the internet. Example: www.example.com.

2) IP Addresses

   Numerical labels assigned to devices participating in a network. Example: 192.0.2.1.

3) DNS Servers

   Specialized servers that store DNS records and respond to queries from clients. These include:

   - Root Servers

     The top-level DNS servers that direct queries to appropriate top-level domain (TLD) servers.

   - TLD Servers

     Servers responsible for top-level domains like .com, .org, .net, etc.

   - Authoritative Name Servers

     Servers that hold the actual DNS records for a domain.

   - Recursive Resolvers

     Intermediary servers that handle the query process on behalf of the client, contacting multiple DNS servers to resolve a domain name to an IP address.

DNS Records

- A Record — Maps a domain name to an IPv4 address.
- AAAA Record — Maps a domain name to an IPv6 address.
- CNAME Record — Canonical Name record, aliasing one domain name to another.
- MX Record — Mail Exchange record, specifies mail servers for a domain.
- TXT Record — Holds arbitrary text data, often used for verification and security purposes.
- NS Record — Specifies the authoritative name servers for a domain.

How DNS Works

1) DNS Query

   When a user enters a domain name into their browser, a DNS query is initiated.

2) Recursive Resolution

   - The request first goes to a recursive resolver.
   - The resolver checks its cache for a recent answer. If not found, it contacts a root server.
   - The root server responds with the address of a TLD server for the domain's extension (e.g., .com).
   - The resolver then queries the TLD server, which responds with the authoritative name server for the domain.
   - The resolver queries the authoritative server, which responds with the IP address for the domain.

3) **Response**

   The resolver returns the IP address to the client, allowing the browser to contact the web server directly.

4) **Caching**

Both recursive resolvers and clients cache responses for a period (defined by the TTL, or Time To Live) to speed up future queries.

POP3 (Post Office Protocol version 3):

Used by email clients to retrieve email messages from a mail server.
Allows users to download emails from the server to their local device.

IMAP (Internet Message Access Protocol):

Another protocol used by email clients to access and manage email messages stored on a mail server.
Provides more advanced features compared to POP3, such as folder management and synchronization across multiple devices.

SNMP (Simple Network Management Protocol):

Used for managing and monitoring network devices and systems.
Allows network administrators to collect information, configure devices, and detect and resolve network issues.

HTTP (Hypertext Transfer Protocol)

The Hypertext Transfer Protocol (HTTP) is used for transmitting hypermedia documents between a client (such as a web browser) and a server., such as HTML pages and images, over the World Wide Web.
It defines how messages are formatted and transmitted, and how web servers and browsers should respond to various commands.

Key Features

Request-Response Model

- **Client**: Initiates the communication by sending an HTTP request.
- **Server**: Responds with an HTTP response.

Stateless

Each request from a client to a server is treated as an independent transaction, unrelated to any previous request.
To maintain state (e.g., user sessions), other mechanisms like cookies or session storage are used.

Methods

- **GET**: Requests a representation of a specified resource.
- **POST**: Submits data to be processed to a specified resource.
- **PUT**: Uploads a representation of a specified resource.
- **DELETE**: Deletes a specified resource.
- **HEAD**: Similar to GET, but only retrieves the headers and not the body.
- **OPTIONS**: Describes the communication options for the target resource.
- **PATCH**: Applies partial modifications to a resource.

HTTP Versions

HTTP/0.9

The original version with simple request-response format.

HTTP/1.0

HTTP/1.0, released in 1996, was the first standardized version of the Hypertext Transfer Protocol.

Key Features

It introduced the concept of headers and methods (GET, POST, etc.), allowing for structured communication between clients and servers.

Limitations

Each request-response pair typically required a new TCP connection, leading to slower performance, especially for complex web pages with multiple resources.

HTTP/1.1

Released in 1997, HTTP/1.1 aimed to improve upon HTTP/1.0's limitations.

- Persistent Connections (Keep-Alive):

  Persistent connections allow multiple requests and responses to be sent over the same TCP connection.
  This reduces the overhead of establishing and tearing down TCP connections for each request, improving latency and overall performance.

  HTTP Persistent Connection

  Primarily used to reduce the overhead of creating new connections for each HTTP request,
  improving the performance of traditional web applications by reusing the same connection for multiple requests and responses.

  WebSocket

  Designed for real-time, bidirectional communication, where both client and server can send and receive messages independently,
  making it ideal for interactive applications that require low latency and continuous data exchange.
  WebSockets use the existing HTTP connection for the initial handshake and then upgrade that connection to a WebSocket connection.
  They do not open a new connection but rather upgrade the existing HTTP connection to use the WebSocket protocol.

- Chunked Transfer Encoding:

  Chunked transfer encoding enables HTTP messages to be sent as a series of data chunks, rather than as a single block.
  It allows the server to send data progressively without knowing the total size beforehand, which is useful for dynamically generated content or large files.

HTTP/2

Released in 2015, HTTP/2 brought substantial improvements in performance and efficiency.

- Multiplexing

  HTTP/2 allows multiple requests and responses to be sent in parallel over a single TCP connection.
  This improves efficiency by avoiding the head-of-line blocking problem of HTTP/1.1, where one slow request could delay others.
  It also reduces the number of connections needed, saving server and client resources.

- Header Compression

  HTTP/2 uses HPACK compression to reduce overhead by compressing HTTP header fields.
  This significantly reduces the amount of data transmitted, especially beneficial for requests with large headers or multiple cookies, improving overall latency and bandwidth utilization.

- Binary Framing

  HTTP/2 encapsulates HTTP messages into smaller binary frames, allowing more efficient parsing and processing.
  This simplifies how data is transmitted and parsed, enhancing performance and reducing complexity compared to text-based protocols like HTTP/1.1.

- Server Push

  Allows the server to push resources proactively to the client before they are requested.

HTTP/3

HTTP/3, expected to be standardized in 2022, represents the next evolution, focusing on even faster and more reliable connections.

- QUIC (Quick UDP Internet Connections)

  HTTP/3 uses QUIC as its transport protocol instead of TCP used in previous versions.
  QUIC offers several advantages such as faster connection establishment, improved congestion control, and

reduced latency due to its integration of transport and encryption protocols.
It also handles packet loss and network changes more efficiently.

- Stream-based Transmission
  HTTP/3 uses streams for data transmission, where each stream is independent and can be multiplexed over a single connection.
  This method does not require chunked transfer encoding to segment data.

HTTP Request Structure

Request Line
Method (e.g., GET, POST)
Resource path (e.g., /index.html)
HTTP version (e.g., HTTP/1.1)

```
GET /index.html HTTP/1.1
```

Headers
Provide additional information about the request.

```
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html
```

Body: (Optional)
Contains data sent to the server (e.g., form data in a POST request).

HTTP Response Structure

Status Line
HTTP version (e.g., HTTP/1.1)
Status code (e.g., 200, 404)
Status message (e.g., OK, Not Found)

```
HTTP/1.1 200 OK
```

Headers
Provide additional information about the response.

```
Content-Type: text/html
Content-Length: 137
```

Body
Contains the requested resource (e.g., HTML content, JSON data).

HTTP Workflow

Client Request
Initiation
A client (like a web browser) initiates an HTTP request to a server by specifying a Uniform Resource Identifier (URI) or URL.
For example, when you type a website address (e.g., https://www.example.com) in your browser and hit enter, the browser sends an HTTP request to the server hosting that website.

Prepare Server Response
Processing
The server processes the incoming HTTP request. It retrieves the requested resource (like an HTML page, image, CSS file, etc.) and prepares an HTTP response.
Response Creation
The server constructs an HTTP response that includes

- **Status Code:** Indicates whether the request was successful, redirected, or encountered an error (e.g., 200 OK, 404 Not Found).
- **Headers:** Provide metadata about the response (e.g., content type, caching directives).
- **Body:** Contains the actual content being sent back to the client (e.g., HTML markup, image data).

Transmission
Data Transfer

The server sends the HTTP response back to the client over the network.
This transfer typically occurs over TCP/IP (Transmission Control Protocol/Internet Protocol), which ensures reliable data delivery.

Client Processing

Reception

The client (browser) receives the HTTP response.

Rendering

If the response contains HTML, CSS, and JavaScript, the browser interprets these resources to render the web page for the user.

Additional Requests

The client may parse the received HTML to discover and request additional resources referenced within the page (e.g., images, scripts, stylesheets).

Statelessness and Connection Handling

Statelessness

HTTP is stateless, meaning each request-response cycle is independent.
The server does not retain information about previous requests from the same client unless explicitly managed (e.g., using cookies or sessions).

Connection Management

Modern implementations of HTTP (like HTTP/1.1 and HTTP/2) support persistent connections,
where multiple requests and responses can be sent over the same TCP connection, improving performance by reducing connection setup overhead.

Security Considerations

Encryption

HTTPS (HTTP Secure) uses SSL/TLS encryption to secure HTTP communications, protecting the integrity and confidentiality of data exchanged between clients and servers.

Cross-domain communication

Cross-domain communication in the context of HTTP refers to the ability for a web page from one domain to make requests to a resource on a different domain.

This is a common scenario in modern web applications but is restricted by the same-origin policy (SOP) for security reasons. To facilitate cross-domain requests while maintaining security, Cross-Origin Resource Sharing (CORS) is used.

Same-Origin Policy (SOP)

The same-origin policy is a security measure implemented by web browsers to prevent malicious websites from accessing sensitive data on other websites.
It restricts web pages from making requests to a different origin than the one that served the web page.

Origin: Defined by the combination of protocol, domain, and port.
Example: https://example.com:443 is different from http://example.com:80.

Cross-Origin Resource Sharing (CORS)

CORS allows a web server to grant access to restricted resources from a different domain, bypassing SOP restrictions when properly configured.
It provides a way for servers to specify who can access their resources and how.

Preflight Request

Before making a cross-origin request, the browser sends an HTTP OPTIONS request (preflight request) to the server to check if the actual request is safe to send.
The server responds with the allowed methods and headers.

Cross-Origin Response Headers

Access-Control-Allow-Origin

Access-Control-Allow-Methods

Access-Control-Allow-Headers

Access-Control-Allow-Credentials

Access-Control-Expose-Headers

Access-Control-Max-Age:

The CORS headers must be present in both:

- OPTIONS (Preflight) Response

  Needed for requests with custom headers, credentials, or methods like PUT/DELETE.

- Normal (Actual) Response

  Ensures the browser allows the main request after the preflight check.

If either one is missing, the browser will block the request.

Example:

UI: 80

Nginx: 8090

Server: 8099

```
server {
    listen 8090;

    location / {
        proxy_pass http://localhost:8099;

        # Ensure CORS headers are included in the response
        add_header Access-Control-Allow-Origin *;
        add_header Access-Control-Allow-Methods GET, POST, OPTIONS;
        add_header Access-Control-Allow-Headers Content-Type, Authorization;
    }

    # Handle preflight OPTIONS requests directly
    if ($request_method = OPTIONS) {
        add_header Access-Control-Allow-Origin *;
        add_header Access-Control-Allow-Methods GET, POST, OPTIONS;
        add_header Access-Control-Allow-Headers Content-Type, Authorization;
        return 204;
    }
}
```

WebSocket Protocol

WebSocket Handshake:

The WebSocket protocol begins with a handshake using the HTTP protocol, which allows it to leverage existing HTTP infrastructure (such as ports and security mechanisms).

The client sends an HTTP Upgrade request to change the protocol from HTTP to WebSocket.

If the server supports WebSockets, it responds with a status code 101 Switching Protocols, completing the handshake.

Full-Duplex Communication:

After the handshake, the connection is upgraded to use the WebSocket protocol, which allows for full-duplex communication (i.e., data can be sent and received simultaneously).

Frame-Based Communication:

WebSockets use a frame-based messaging system, where data is sent in frames that can be either text or binary. Control frames manage the connection (e.g., to close the connection or handle ping/pong keep-alives), while data frames carry the actual payload.

Structure

How Browser Caching Works

HTTP Headers:

- Cache-Control
  - This header specifies directives for caching mechanisms in both requests and responses.
  - For example, Cache-Control: max-age=3600 means the resource is considered fresh for 3600 seconds (1 hour).
- Expires
  - This header provides an absolute date/time after which the response is considered stale. For example, Expires: Wed, 21 Oct 2021 07:28:00 GMT.
- Etag
  - This header is used to determine if the resource has changed. It is a unique identifier for a specific version of a resource.
- Last-Modified
  - This header indicates the last time the resource was modified. For example, Last-Modified: Wed, 21 Oct 2021 07:28:00 GMT.

Cache Storage:

- The browser stores cached resources in a cache storage, which can be in-memory or on-disk storage. This includes HTML files, CSS, JavaScript, images, and other web resources.

Validation:

- When a user revisits a web page, the browser checks the cache to see if a valid copy of the resource exists. If it does, the resource is served from the cache.
- If the resource has expired or the browser is instructed to revalidate the resource (using Cache-Control: no-cache or Cache-Control: must-revalidate),
- the browser sends a request to the server with If-Modified-Since or If-None-Match headers to check if the resource has changed.

Revalidation:

- If the server responds with a 304 Not Modified status, the cached resource is still valid and can be used.
- If the server responds with a new version of the resource, the browser updates the cache and serves the new resource.

Uniform Resource Locator (URL)

A Uniform Resource Locator (URL) is a reference or address used to access resources on the internet.
It specifies the location of a resource and the protocol used to retrieve it.
URLs are essential for navigating the web, enabling users to access web pages, download files, and interact with various online services.

Key Components of a URL

1) Scheme (Protocol):
   - Indicates the protocol used to access the resource.
   - Common schemes include http, https, ftp, mailto, and file.
   - Example: https in `https://www.example.com`
2) Host (Domain Name or IP Address)
   - Specifies the server where the resource is located.
   - Typically a domain name (e.g., www.example.com) or an IP address (e.g., 192.168.1.1).
   - Example: `www.example.com` in `https://www.example.com`
3) Port (Optional)
   - Specifies the port number on the server to which the request is directed.
   - If omitted, the default port for the specified scheme is used (e.g., 80 for http, 443 for https).
   - Example: `:8080` in `https://www.example.com:8080`
4) Path

Specifies the specific resource within the host that the client wants to access.
Often corresponds to a file or directory structure on the server.
Example: `/path/to/resource` in `https://www.example.com/path/to/resource`

5) Query (Optional)

Contains additional parameters for the request, usually in the form of key-value pairs.
Begins with a ? and parameters are separated by &.

- Single Resource Request:

  Typically seen in URLs like " `https://www.example.com/path?key1=value1&key2=value2`".
  Here, "`?key1=value1&key2=value2`" specifies two query parameters.

- Multiple Resource Request (Double Question Mark ??

  Used to request multiple resources within a single URL request.
  Syntax: https://example.com/path/to/resources/??resource1.js,resource2.css,resource3.html.
  Benefits

  Reduced HTTP Requests: Combining multiple resource requests into one reduces the number of HTTP connections needed, improving page load times.
  Simplified Management: Easier to manage and maintain asset loading across web applications.
  Efficient Caching: Enhances caching efficiency as resources can be cached together.

  Considerations

  Browser Support: Most modern browsers support this syntax, but ensure compatibility with older browsers if needed.
  Server Configuration: Servers must be configured to handle concatenated resource requests appropriately.

6) Fragment (Optional)

Refers to a specific section within a resource, often used in web pages to navigate to a particular section.
Begins with a #.
Example: `#section1` in `https://www.example.com/path#section1`

Restful API

GET

Purpose:

Retrieves information about a resource or a collection of resources.

Request Body:

Generally not used.

Response:

Should return the resource representation in the body, with a successful status code (e.g., 200 OK). For collections, it may return an array of resources.

Example:

Returns the user with ID 123.

```
GET /api/users/123
```

POST

Purpose:

Creates a new resource or performs an action that does not fit into the other HTTP methods.

Request Body:

Contains the data for the new resource to be created or the action to be performed.

Response:

Typically returns the newly created resource or the result of the action, often with a status code like 201 Created.

Example:

Creates a new user and returns the user resource with a status of 201 Created.

```
POST /api/users
Content-Type: application/json

{
    "name": "Jane Doe",
    "email": "jane.doe@example.com"
}
```

PUT

Purpose:

Updates an existing resource or creates a resource at a specific URI if it doesn't exist.

Request Body:

Contains the complete representation of the resource to be updated or created.

Response:

Returns the updated resource or a status code like 200 OK if updated successfully, or 201 Created if a new resource was created.

Example:

Updates the user with ID 123. If the user did not exist, it creates the user at that ID.

```
PUT /api/users/123
Content-Type: application/json

{
    "name": "John Doe",
    "email": "john.doe@example.com"
}
```

DELETE

Purpose:

Deletes a resource identified by the URI.

Request Body:

Generally not used, though some implementations may accept a body for additional context.

Response:

Should return a status code like 204 No Content if the deletion is successful, indicating that the resource has been removed.

Example:

Deletes the user with ID 123. A successful deletion typically returns a 204 No Content response.

```
DELETE /api/users/123
```

PATCH

Purpose:

Partially updates an existing resource. This method is used when only specific fields need to be modified.

Request Body:

Contains only the fields that need to be updated.

Response:

Typically returns the updated resource or a status code like 200 OK if the update is successful.

Example:

Updates only the email field of the user with ID 123.

```
PATCH /api/users/123
Content-Type: application/json

{
    "email": "new.email@example.com"
}
```

HEAD

Purpose:

Similar to GET but retrieves only the headers, not the body. It is used to check metadata or existence of a resource.

Request Body:

Not used.

Response:

Returns headers similar to a GET request but without the body.

Example:

Checks if the user with ID 123 exists and retrieves headers without the full resource body.

```
HEAD /api/users/123
```

OPTIONS

Purpose:

Describes the communication options for the target resource.

It can be used to query the allowed HTTP methods and other options.

Request Body:

Not used.

Response:

Returns headers that describe the allowed methods and other options for the resource.

Example:

Returns the allowed methods and other options for the /api/users resource.

```
OPTIONS /api/users
```

MIME Type

Main MIME Types for HTTP Request Bodies

application/json

Data is formatted as a JSON object. Commonly used in APIs for data exchange.

Example:

```
{
  "key1": "value1",
  "key2": "value2"
}
```

multipart/form-data

Used for forms that include file uploads. Encodes each form field as a separate part within a MIME multipart message.

Example:

```
--boundary
Content-Disposition: form-data; name="field1"

value1
--boundary
Content-Disposition: form-data; name="file"; filename="example.txt"
Content-Type: text/plain

(file content here)
--boundary--
```

application/x-www-form-urlencoded

Default encoding for HTML forms. Data is encoded as key-value pairs, with keys and values URL-encoded and joined by `'&'`.

Example:

```
key1=value1&key2=value2
```

application/octet-stream

Used for arbitrary binary data. Often used for file uploads when the file type is unknown or not specifically handled.

Example:

```
(binary data)
```

application/xml

Data is formatted as an XML document. Often used in web services and configuration files.

Example:

```
<root>
  <key1>value1</key1>
  <key2>value2</key2>
</root>
```

Application

application/graphql

Data is formatted as a GraphQL query. Used in GraphQL APIs to send queries and mutations.

Example:

```
{
  user(id: "1") {
    name
    email
  }
}
```

application/x-protobuf

Data is serialized using Protocol Buffers (Protobuf) format. Commonly used in gRPC and other high-performance APIs.

Example:

(binary data `serialized` in Protobuf format)

application/x-yaml

Data is formatted as a YAML document. Used for configuration files and data serialization.

Example:

```
key1: value1
key2: value2
```

Multipart

multipart/related

Used to represent a message with multiple parts where each part has its own content type.

Example: Commonly used in email and HTTP protocols for attaching resources related to the main content.

multipart/mixed

Used for combining multiple parts of different types into a single message.

Example: Allows sending different types of content (e.g., text and attachments) in a single HTTP response or email.

multipart/alternative

Used to indicate multiple representations of the same information.

Example: Often used in email to provide plain text and HTML versions of the same message.

Image

image/x-icon

MIME type for ICO (Icon) files used in Windows.

Example: Represents small graphical icons used to represent files, folders, or applications.

image/jpeg

MIME type for JPEG (Joint Photographic Experts Group) image files.

Example: Common format for photographic images on the web.

image/svg+xml

MIME type for Scalable Vector Graphics (SVG) files.

Example: XML-based vector image format used for scalable graphics on the web.

image/gif

MIME type for Graphics Interchange Format (GIF) image files.

Example: Supports animated images and simple graphics.

Audio

audio/mpeg

MIME type for MP3 audio files.

Example: Common format for compressed audio files used for music and audio recordings.

Text

text/plain

MIME type for plain text files.

Example: Used for simple text documents without any formatting.

Request Headers

Cache-Control

Directives for caching mechanisms in both requests and responses.

Controls how caching is applied by intermediate caches (proxies).

Example: `Cache-Control: max-age=3600`

Connection

Controls whether the network connection should stay open after the current transaction finishes.

Example: `Connection: keep-alive`

Keep-Alive

Control the keep-alive parameters of the HTTP persistent connection.

This header can specify the maximum number of requests that can be sent over the connection and the timeout duration in seconds.

Example: `Keep-Alive: timeout=5, max=100`

Content-Length

Indicates the size of the entity-body in bytes sent in the request. It tells the server the exact length of the request body, allowing the server to read the correct amount of data.

Example: `Content-Length: 348`

Content-Type

Specifies the media type (MIME type) of the body of the request. It is used to inform the server about the type of data being sent by the client.

Example: `Content-Type: application/json`

Accept

Specifies the media types (MIME types) that the client can understand.

Allows servers to send responses in the preferred format.

Example: `Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8`

Accept-Language

Specifies the natural languages that the client prefers for the response.

Helps servers localize content based on user preferences.

Example: `Accept-Language: en-US,en;q=0.9`

Accept-Encoding

Specifies the content encoding methods that the client can understand.

Allows servers to compress responses before sending them to reduce data transfer time.

Example: `Accept-Encoding: gzip, deflate, br`

Authorization

Provides credentials (e.g., username and password) for accessing protected resources.

Used in requests requiring authentication.

Example: `Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l`

Content-Disposition

Suggests how the user agent should handle the content attachment sent in the response.

Example: `Content-Disposition: attachment; filename="example.pdf"`

Content-Length

Indicates the size of the entity-body in bytes sent in the request or response.

Example: `Content-Length: 12345`

Content-Type

Specifies the media type (MIME type) of the request body sent to the server.

Required for POST, PUT, and PATCH requests that include a body.

Example: `Content-Type: application/json`

Cookie

Contains previously set cookies that the server sent to the client.

Helps servers maintain session state and track user interactions.

Example: `Cookie: session-id=1234567890abcdef`

Date

Provides the date and time at which the request was originated.

Example: `Date: Tue, 15 Nov 2022 08:12:31 GMT`

Expect

Indicates expectations that need to be fulfilled by the server in order to properly handle the request.

Example: `Expect: 100-continue`

Expires

Description: Specifies a date/time after which the response is considered stale and should no longer be cached.

Example: `Expires: Thu, 01 Dec 2023 16:00:00 GMT`

Host

Specifies the domain name of the server (e.g., Host: `www.example.com`).

Required in HTTP/1.1 requests.

If-Match

Makes the request conditional, ensuring that the request is processed only if the client's specified ETag matches the resource's current ETag.

Example: `If-Match: "etag123"`

If-None-Match

Makes the request conditional, ensuring that the request is processed only if the client's specified ETag does not match the resource's current ETag.

Example: `If-None-Match: "etag456"`

If-Modified-Since

Makes the request conditional, ensuring that the request is processed only if the resource has been modified since the specified date/time.

Example: `If-Modified-Since: Tue, 01 Jan 2023 00:00:00 GMT`

If-Range

Makes the request conditional, requesting only the specified range of the resource if the entity has not been modified.

Example: `If-Range: "etag123"`

If-Unmodified-Since

Makes the request conditional, ensuring that the request is processed only if the resource has not been modified since the specified date/time.

Example: `If-Unmodified-Since: Tue, 01 Jan 2023 00:00:00 GMT`

Max-Forwards

Indicates the maximum number of intermediaries (proxies or gateways) that can forward the request.

Example: `Max-Forwards: 10`

Pragma

Used to include implementation-specific directives that might apply to any recipient along the request/response chain.

Example: `Pragma: no-cache`

Proxy-Authorization

Contains the credentials for authenticating to a proxy server.

Example: `Proxy-Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l`

Referer (sic)

Indicates the URL of the resource from which the current request was initiated.

Helps servers understand the context of the request.

Example: `Referer: https://www.example.com/previous-page`

Range

Requests only part of a resource, specifying the range of bytes desired.

Example: `Range: bytes=0-499`

User-Agent

Identifies the client software (e.g., web browser or application) initiating the request.

Helps servers tailor responses based on the client capabilities.

Example: `User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.9999.999 Safari/537.36`

IP

X-Forwarded-For

Specifies the original IP address of the client when communicating with a proxy or load balancer.

Example: `X-Forwarded-For: 192.168.1.1`

Proxy-Client-IP

Specifies the IP address of the client when communicating with a proxy server.

Example: `Proxy-Client-IP: 192.168.1.1`

WL-Proxy-Client-IP

Specifies the IP address of the client when communicating with Oracle WebLogic Server through a proxy or load balancer.

Example: `WL-Proxy-Client-IP: 192.168.1.1`

Response Headers

Cache-Control

Directives for caching mechanisms in both requests and responses.

Example: `Cache-Control: no-cache, no-store, must-revalidate`

Connection

Controls whether the network connection should stay open after the current transaction finishes.

Example: `Connection: close`

Content-Length

Indicates the size of the entity-body in bytes sent to the client.

Example: `Content-Length: 348`

Content-Type

Specifies the media type (MIME type) of the body sent to the client.

Example: `Content-Type: application/json`

Accept-Ranges

Indicates that the server accepts range requests for a resource.

Example: Accept-Ranges: bytes

Age

The time in seconds the object has been in a proxy cache.

Example: Age: 1200

Allow

Lists the set of methods supported by the resource.

Example: Allow: GET, POST, HEAD

Content-Disposition
: Suggests how the user agent should handle the content attachment sent in the response.
: Example: Content-Disposition: attachment; filename="example.pdf"

Content-Encoding
: Specifies the encoding used on the data sent to the client.
: Example: Content-Encoding: gzip

Content-Language
: Describes the natural language(s) of the intended audience for the enclosed content.
: Example: Content-Language: en-US

Content-Location
: Indicates an alternate location for the returned data.
: Example: Content-Location: /index.html

Content-Range
: Indicates where in a full body message a partial message belongs.
: Example: Content-Range: bytes 200-1000/67589

Date
: Indicates the date and time at which the message was originated.
: Example: Date: Tue, 15 Nov 2022 08:12:31 GMT

ETag
: Provides a unique identifier for a specific version of a resource. Useful for caching and conditional requests.
: Example: ETag: "etag123"

Expires
: Specifies the date/time after which the response is considered stale.
: Example: Expires: Thu, 01 Dec 2023 16:00:00 GMT

Last-Modified
: Indicates the date and time at which the resource was last modified.
: Example: Last-Modified: Tue, 15 Nov 2022 12:45:26 GMT

Link
: Used to define relationships between the current document and other documents.
: Example: Link: <https://example.com/api>; rel="alternate"

Location
: Indicates the URL to redirect a page to.
: Example: Location: https://www.example.com/newpage.html

Pragma
: Used for implementation-specific directives that might apply to any recipient along the request/response chain. Often used to control caching.
: Example: Pragma: no-cache

Proxy-Authenticate
: Specifies the authentication method that should be used to access a resource behind a proxy server.
: Example: Proxy-Authenticate: Basic realm="Access to the staging site"

Retry-After
: Indicates how long the user agent should wait before making a follow-up request.
: Example: Retry-After: 120

Server
: Contains information about the software used by the origin server to handle the request.
: Example: Server: Apache/2.4.1 (Unix)

Set-Cookie
: Sends cookies from the server to the user agent.

Example: `Set-Cookie: sessionId=abc123; Path=/; HttpOnly`

Strict-Transport-Security

A security feature that tells browsers to only interact with the server using secure HTTPS connections.

Example: Strict-Transport-Security: max-age=31536000; includeSubDomains

Transfer-Encoding

Specifies the form of encoding used to safely transfer the payload body to the user.

Example: Transfer-Encoding: chunked

Vary

Indicates which headers the server used to select the response, often used for content negotiation.

Example: Vary: Accept-Encoding

WWW-Authenticate

Defines the authentication method that should be used to access a resource.

Example: WWW-Authenticate: Basic realm="Access to the staging site"

Corsss-domain

Access-Control-Allow-Origin

The server specifies which origins are allowed to access the resource using this header.

Example:

```
Access-Control-Allow-Origin: https://example.com
```

Access-Control-Allow-Methods

Specifies which HTTP methods are allowed when accessing the resource.

Example

```
Access-Control-Allow-Methods: GET, POST, PUT
```

Access-Control-Allow-Headers

Specifies which headers can be used during the actual request.

Example

```
Access-Control-Allow-Headers: Content-Type, Authorization
```

Access-Control-Allow-Credentials

Indicates whether the browser should include credentials (cookies, authorization headers) in the request.

Example

```
Access-Control-Allow-Credentials: true
```

Access-Control-Expose-Headers

Allows the server to expose certain headers to the client.

Example

```
Access-Control-Expose-Headers: Content-Length
```

Access-Control-Max-Age:

Indicates how long the results of a preflight request can be cached.

Example

```
Access-Control-Max-Age: 600 (600 seconds)
```

Status Codes

Informational Responses (1xx)

100 Continue

The server has received the initial part of the request and the client should proceed with the rest of the request or ignore it if it's already completed.

101 Switching Protocols

The server is acknowledging the client's upgrade request to switch protocols, such as switching from HTTP to WebSocket.

102 Processing

The server has received and is processing the request, but no response is available yet.

103 Early Hints

Used to return some response headers before the final HTTP message.

Successful Responses (2xx)

200 OK

The request has succeeded. The response typically includes the requested resource.

201 Created

The request has been fulfilled, and a new resource has been created as a result.

202 Accepted

The request has been accepted for processing, but the processing has not been completed.

203 Non-Authoritative Information

The server is a transforming proxy that received a 200 OK from its origin, but is returning a modified version of the response.

204 No Content

The server successfully processed the request but is not returning any content.

205 Reset Content

Informs the client to reset the document view, often used after form submission.

206 Partial Content

The server is delivering only part of the resource due to a range header sent by the client.

Redirection Messages (3xx)

300 Multiple Choices

The requested resource has multiple representations, each with its own specific location, and the client should choose one.

301 Moved Permanently

The requested resource has been permanently moved to a new URL.

302 Found (Moved Temporarily)

The requested resource resides temporarily under a different URL. Often used for temporary redirects.

303 See Other

The response to the request can be found under a different URI, and should be retrieved using a GET method on that resource.

304 Not Modified

Indicates that the resource has not been modified since the version specified by the request headers.

305 Use Proxy

The requested resource must be accessed through the proxy given by the Location field.

306 Switch Proxy

No longer used, but reserved.

307 Temporary Redirect

The requested resource resides temporarily under a different URL. The client should continue to use the original URL for future requests.

308 Permanent Redirect

The requested resource has been permanently moved to a different URL. Similar to 301, but HTTP method and body must not change.

Client Error Responses (4xx)

400 Bad Request

The server cannot process the request due to a client error, such as malformed syntax or invalid parameters.

401 Unauthorized

The request requires user authentication. The client needs to authenticate itself to get the requested response.

402 Payment Required

Reserved for future use.

403 Forbidden

The server understood the request but refuses to authorize it. Access is not allowed.

404 Not Found

The server cannot find the requested resource. This is the most common status code to indicate a missing resource.

405 Method Not Allowed

The method specified in the request is not allowed for the resource identified by the request URI.

406 Not Acceptable

The server cannot generate a response that meets the criteria specified in the request headers.

407 Proxy Authentication Required

Similar to 401, but specifically for use when authenticating through a proxy.

408 Request Timeout

The client did not produce a request within the time that the server was prepared to wait.

409 Conflict

Indicates that the request could not be completed due to a conflict with the current state of the resource.

410 Gone

Indicates that the resource requested is no longer available and will not be available again.

411 Length Required

The server requires a content-length header to be specified in the request.

412 Precondition Failed

One or more conditions specified in the request header fields evaluated to false when tested on the server.

413 Payload Too Large

The server is refusing to process a request because the request payload is larger than the server is willing or able to process.

414 URI Too Long

The server is refusing to service the request because the request-target (URI) is longer than the server is willing to interpret.

415 Unsupported Media Type

The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

416 Range Not Satisfiable

The client has asked for a portion of the file, but the server cannot supply that portion.

417 Expectation Failed

The server cannot meet the requirements of the Expect request-header field.

418 I'm a teapot

This code was defined in 1998 as one of the traditional IETF April Fools' jokes, not to be implemented or used.

421 Misdirected Request

The request was directed at a server that is not able to produce a response.

422 Unprocessable Entity

The request was well-formed but was unable to be followed due to semantic errors.

423 Locked

The resource that is being accessed is locked.

424 Failed Dependency

The request failed due to failure of a previous request.

425 Too Early

Indicates that the server is unwilling to risk processing a request that might be replayed.

426 Upgrade Required

The server refuses to perform the request using the current protocol but might be willing to do so after the client

upgrades to a different protocol.

428 Precondition Required

The server requires the request to be conditional.

429 Too Many Requests

The user has sent too many requests in a given amount of time.

431 Request Header Fields Too Large

The server is unwilling to process the request because its header fields are too large.

451 Unavailable For Legal Reasons

The server is denying access to the resource as a consequence of a legal demand.

Server Error Responses (5xx)

500 Internal Server Error

A generic error message indicating that the server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented

The server does not support the functionality required to fulfill the request.

502 Bad Gateway

The server received an invalid response from an upstream server while attempting to fulfill the request.

503 Service Unavailable

The server is currently unable to handle the request due to temporary overloading or maintenance of the server.

504 Gateway Timeout

The server did not receive a timely response from an upstream server it accessed to process the request.

505 HTTP Version Not Supported

The server does not support the HTTP protocol version used in the request.

506 Variant Also Negotiates

Transparent content negotiation for the request results in a circular reference.

507 Insufficient Storage

The server is unable to store the representation needed to complete the request.

508 Loop Detected

The server detected an infinite loop while processing the request.

510 Not Extended

Further extensions to the request are required for the server to fulfill it.

511 Network Authentication Required

The client needs to authenticate to gain network access.

TCP/IP Model

The TCP/IP model **(Transmission Control Protocol/Internet Protocol Model)** is a simpler and more widely used network model that combines the functionalities of the OSI model into four layers:

Application Layer

Corresponds to the OSI Application, Presentation, and Session layers, providing network services to end-users and applications.

Transport Layer

Equivalent to the OSI Transport layer, responsible for end-to-end communication and data transfer reliability.

Internet Layer

Equivalent to the OSI Network layer, handles addressing, routing, and packet forwarding across different networks.

Link Layer

Corresponds to the OSI Data Link and Physical layers, deals with physical addressing and data transmission over the local

network medium.