

其他协议

@M了个J
李明杰

<https://github.com/CoderMJLee>

<https://space.bilibili.com/325538782>

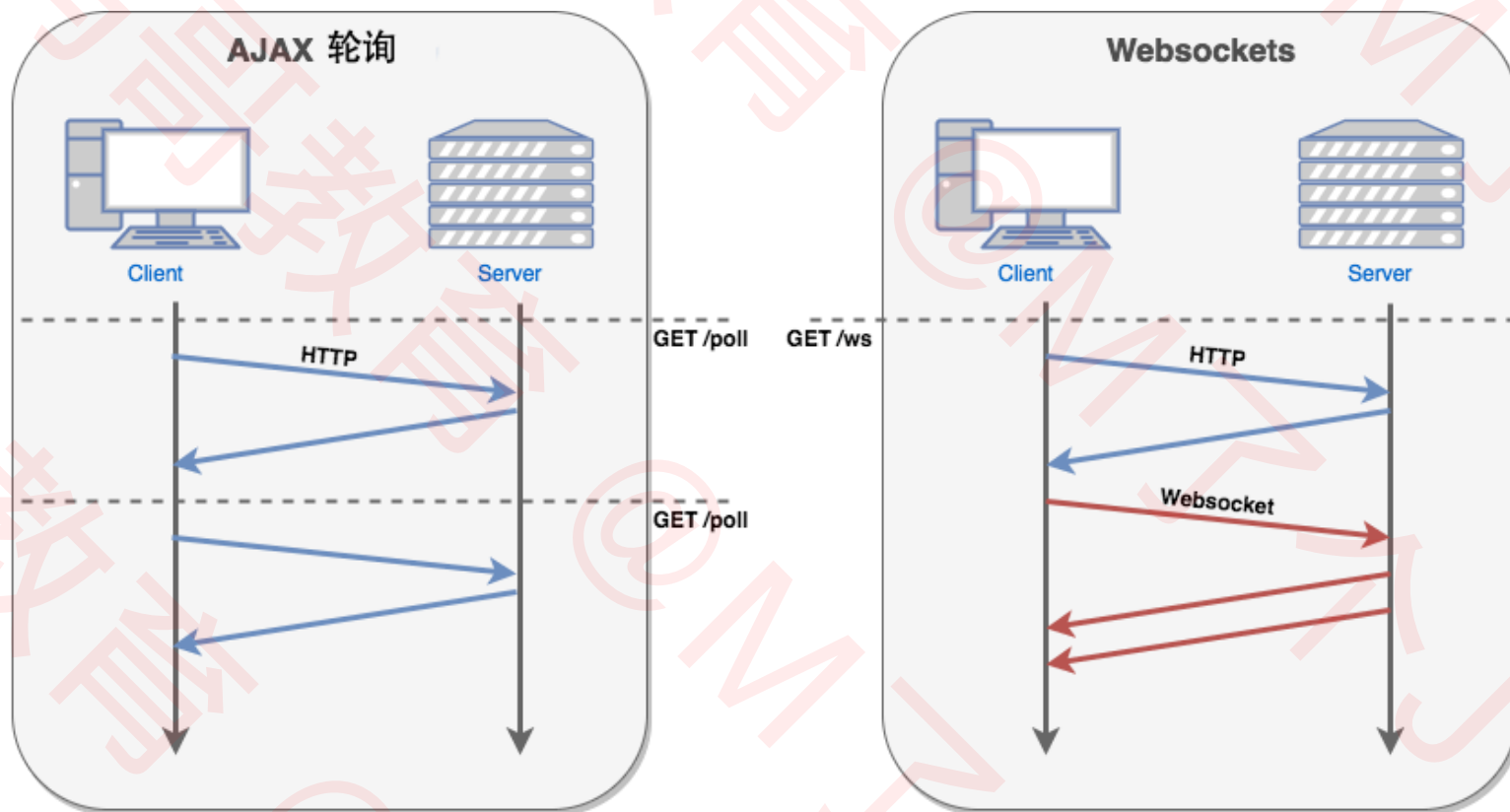


实力IT教育 www.520it.com



HTTP vs WebSocket

- HTTP请求的特点：通信只能由客户端发起。所以，早期很多网站为了实现推送技术，所用的技术都是轮询
- 轮询是指由浏览器每隔一段时间（如每秒）向服务器发出HTTP请求，然后服务器返回最新的数据给客户端
- 为了能更好的节省服务器资源和带宽，并且能够更实时地进行通讯，HTML5规范中出现了WebSocket协议



WebSocket

- WebSocket，是基于TCP的支持**全双工**通信的应用层协议
 - 在2011年由IETF标准化为[RFC 6455](#)，后由[RFC 7936](#)补充规范
 - 客户端、服务器，任何一方都可以主动发消息给对方
-
- WebSocket的应用场景很多
 - 社交订阅、股票基金报价、体育实况更新、多媒体聊天、多玩家游戏等

HTTP vs WebSocket

- WebSocket和HTTP属于平级关系，都是应用层的协议
 - 其实TCP本身就是支持全双工通信的（客户端、服务器均可主动发消息给对方）
 - 只是HTTP的“请求-应答模式”限制了TCP的能力
- WebSocket使用80（ws://）、443（wss://）端口，可以绕过大多数防火墙的限制
 - ws://example.com/wsapi
 - wss://secure.example.com/wsapi
- 与HTTP不同的是，WebSocket需要先建立连接
 - 这就使得WebSocket成为一种有状态的协议，之后通信时可以省略部分状态信息
 - 而HTTP请求可能需要在每个请求都额外携带状态信息（如身份认证等）

WebSocket – 建立连接

- WebSocket需要借助HTTP协议来建立连接（也叫作握手，[Handshake](#)）
- 由客户端（浏览器）主动发出握手请求

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=
Sec-WebSocket-Protocol: chat
```

- **Connection**必须设置Upgrade
- 表示客户端希望连接升级
- **Upgrade**必须设置websocket
- 表示希望升级到WebSocket协议
- **Sec-WebSocket-Version**
- 表示支持的Websocket版本
- [RFC 6455](#)要求使用的版本是13

WebSocket – 建立连接

■ **Sec-WebSocket-Key**是客户端生成的随机字符串，比如例子中的dGh1IHNhbXBsZSBub25jZQ==

■ 服务器接收到客户端的**Sec-WebSocket-Key**后，会进行以下操作

① **Sec-WebSocket-Key**加上一个固定的**GUID**值（258EAF55-E914-47DA-95CA-C5AB0DC85B11）

□ dGh1IHNhbXBsZSBub25jZQ==258EAF55-E914-47DA-95CA-C5AB0DC85B11

② 将①的结果进行**SHA-1摘要计算**

□ b37a4f2cc0624f1690f64606cf385945b2bec4ea

③ 将②的结果进行**Hex To Base64编码**

□ s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

④ 将③的结果做为**Sec-WebSocket-Accept**响应头的值，返回给客户端

■ 如此操作，可以尽量避免普通HTTP请求被误认为WebSocket协议

WebSocket — 使用

- WebSocket体验和演示

- <https://www.websocket.org/echo.html>

- W3C标准化了一套WebSocket API，可以直接使用JS调用

```
let ws = new WebSocket('wss://example.com')
```

WebService

- WebService，译为：Web服务，是一种跨编程语言和跨操作系统平台的远程调用技术标准
- WebService使用场景举例
 - 天气预报、手机归属地查询、航班信息查询、物流信息查询等
 - 比如天气预报，是气象局把自己的服务以WebService形式暴露出来，让第三方可调用这些服务功能
 - http://www.webxml.com.cn/zh_cn/index.aspx
- 事实上，WebService完全可以用普通的Web API取代（比如HTTP + JSON）
 - 现在很多企业的开放平台都是直接采用Web API

WebService – 核心概念

- SOAP (Simple Object Access Protocol)，译为：简单对象访问协议
- 很多时候，SOAP = HTTP + XML
- WebService使用SOAP协议来封装传递数据



- WSDL (Web Services Description Language)，译为：Web服务描述语言
- 一个XML文档，用以描述WebService接口的细节（比如参数、返回值等）
- 一般在WebService的URL后面跟上?wsdl获取WSDL信息
- ✓ 比如：<http://ws.webxml.com.cn/WebServices/WeatherWS.asmx?wsdl>

RESTful — 简介

- REST的全称是：REpresentational State Transfer
 - 译为“表现层状态转移”
- REST是一种互联网软件架构设计风格
 - 定义了一组用于创建Web服务的约束
 - 符合REST架构的Web服务，称为RESTful Web服务

RESTful — 实践建议

■ URL中使用名词（建议用复数形式），不使用动词

□推荐：/users、/users/6

□不推荐：/listUsers、/getUser?id=6、/user/list、/user/get?id=6

■ 使用HTTP的方法表达动作

GET：查询		POST：创建	PUT：更新	DELETE：删除
/users	查询所有的用户	创建一个用户	更新所有用户的信息	删除所有的用户
/users/6	查询id为6的用户	405 Method Not Allowed	更新id为6的用户的信息	删除id为6的用户

■ 一个资源连接到其他资源，使用子资源的形式

□GET /users/6/cars/88

□POST /users/8/cars

RESTful — 实践建议

- API版本化

- `mj.com/v1/users`

- `mj.com/v2/users/66`

- 返回JSON格式的数据

- 发生错误时，不要返回200状态码

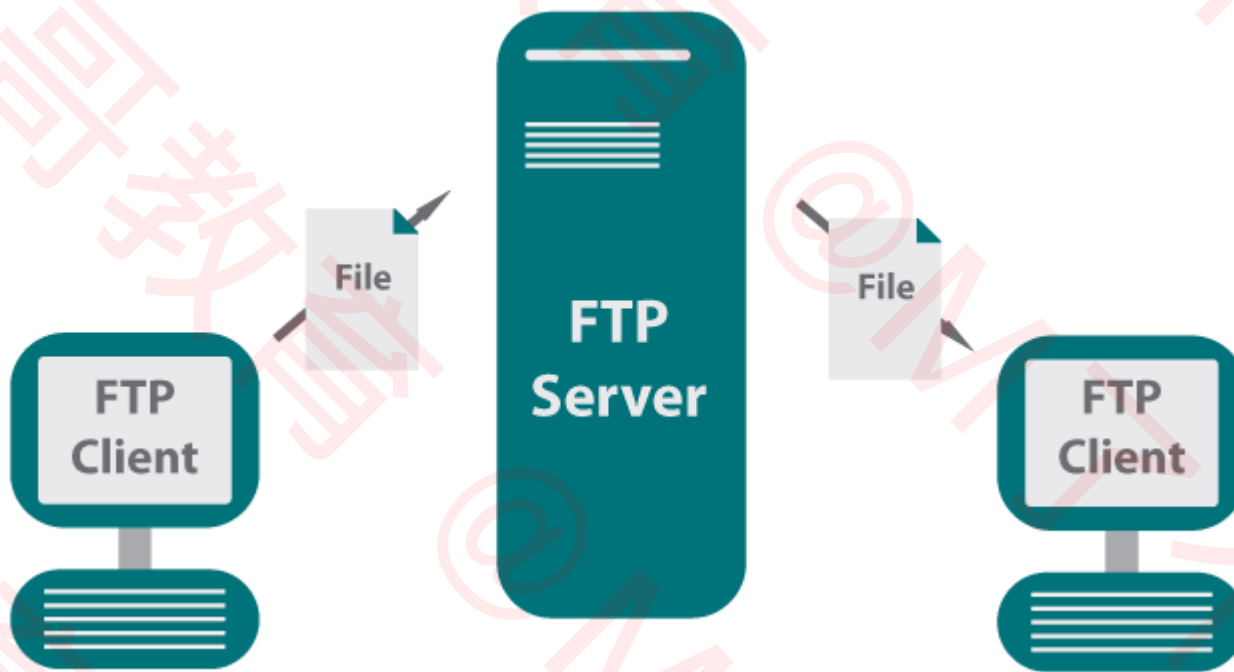
- HTTPDNS是基于HTTP协议向DNS服务器发送域名解析请求
- 替代了基于DNS协议向运营商Local DNS发起解析请求的传统方式
- 可以避免Local DNS造成的域名劫持和跨网访问问题
- 常用在**移动互联网**中（比如在Android、iOS开发中）

优势点	XX云HTTPDNS	运营商Local DNS
高速	高速访问 接入节点覆盖国内Top17运营商、东南亚及北美，解析精准，访问迅速	访问缓慢 用户跨网访问、解析异常问题
安全	防劫持 绕过运营商Local DNS，无劫持，防止DNS被污染拦截	广告劫持 域名解析结果被指向广告页面、插入第三方广告
智能	精准调度 精确识别来源请求，访问导向最准确节点	解析转发 自身不进行域名递归解析，而把请求转发给其他运营商
可靠	负载均衡 一个IP三地集群容灾，秒级自动故障切换，服务提供99%以上的SLA	异常故障 缓存服务器运维环境参差不齐，时有故障

HTTPDNS — 使用

- 市面上已经有现成的解决方案
 - 腾讯云: <https://cloud.tencent.com/product/httpdns>
 - 阿里云: <https://help.aliyun.com/product/30100.html>
- 移动端集成相关的SDK即可使用HTTPDNS服务

- FTP (File Transport Protocol) , 译为: 文件传输协议, [RFC 959](#)定义了此规范, 是基于TCP的应用层协议
- 在[RFC 1738](#)中有定义, FTP的URL格式为: **ftp://[user[:password]@]host[:port]/url-path**



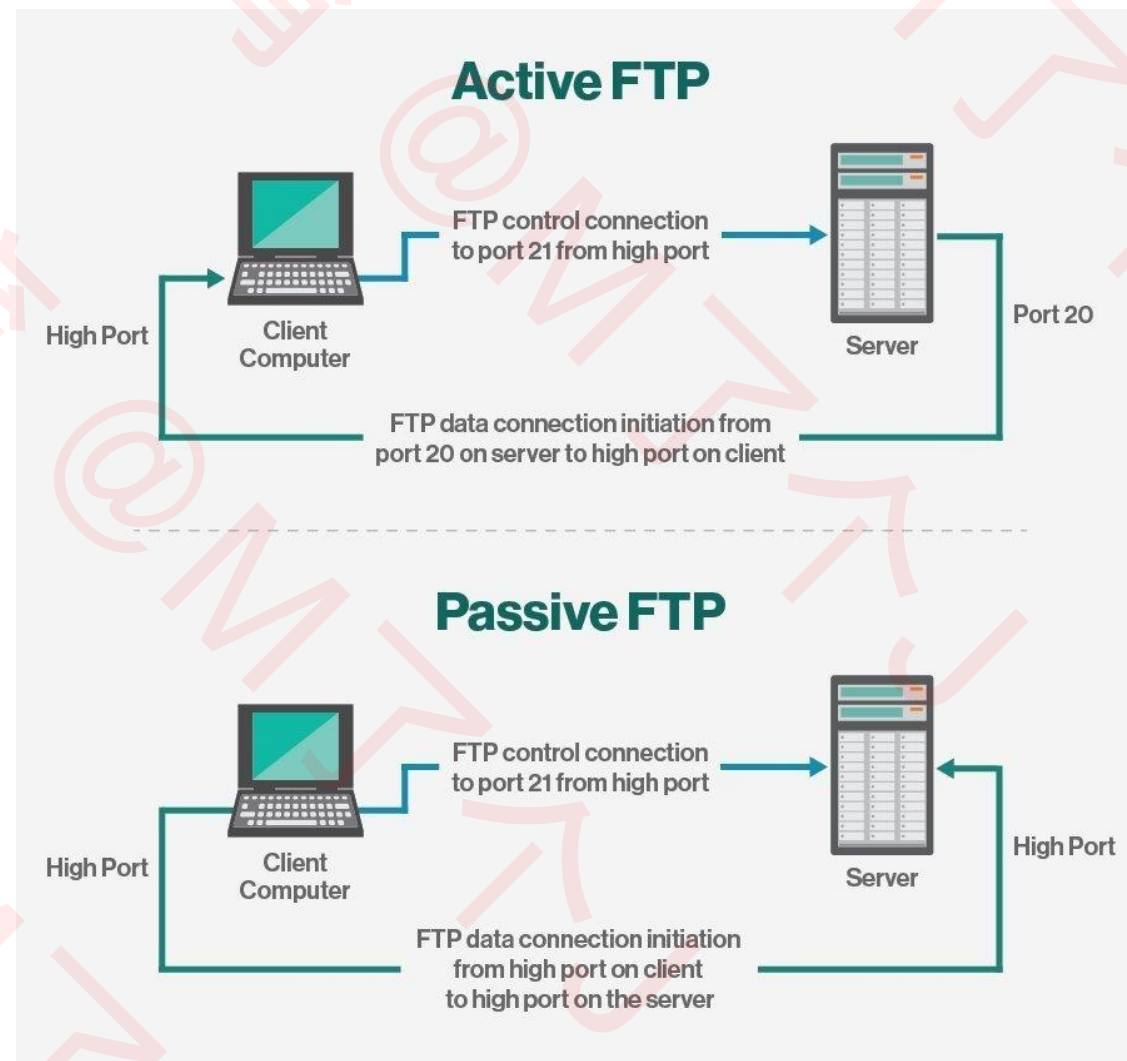
FTP — 连接模式

■ FTP有2种连接模式：主动（Active）和被动（Passive）

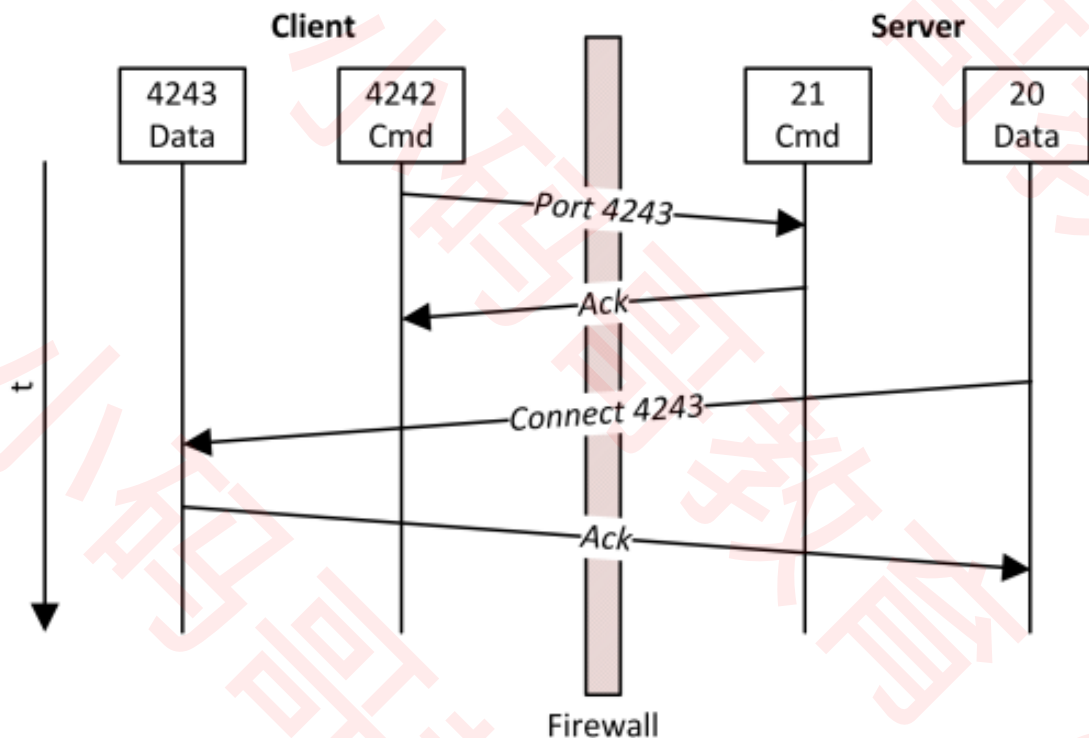
■ 不管是哪种模式，都需要客户端和服务端建立2个连接

① 控制连接：用于传输状态信息（命令，cmd）

② 数据连接：用于传输文件和目录信息（data）



FTP – 主动模式



① 客户端打开一个随机的命令端口

▣ 端口号大于1024，假设为N

▣ 同时连接至服务器的命令端口21

② 客户端开始监听N+1数据端口

▣ 同时向服务器发送一个Port命令给服务器的命令端口21

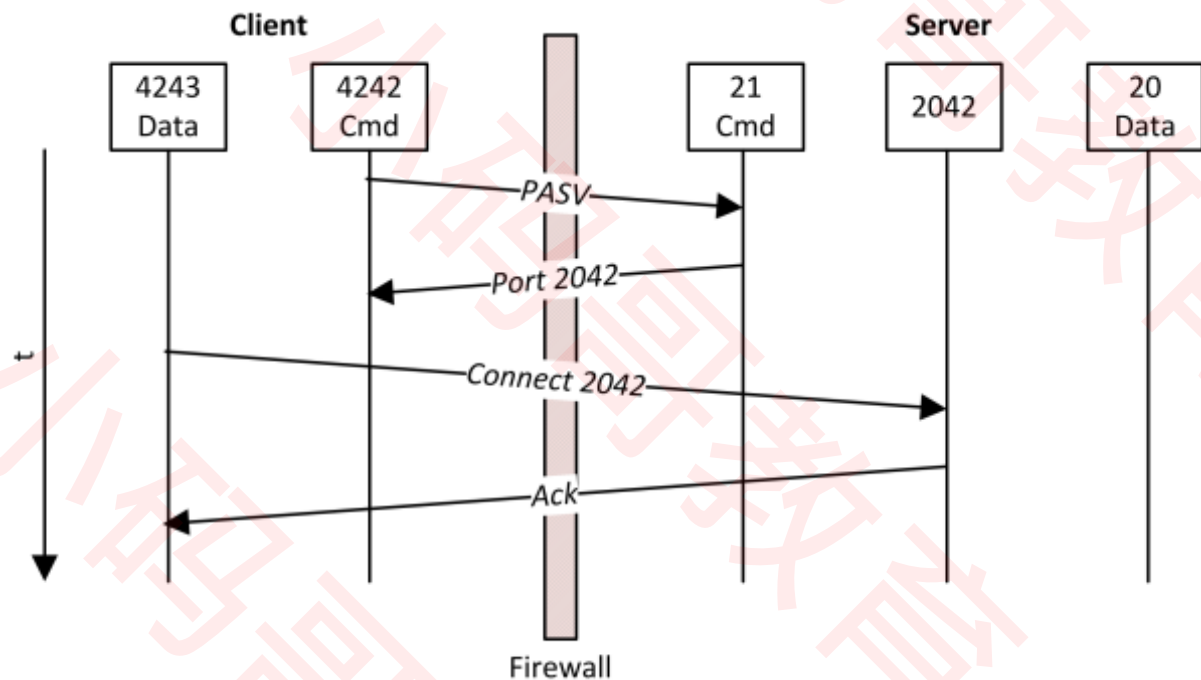
▣ 此命令告诉服务器

✓ 客户端正在监听的数据端口N+1

✓ 并且已准备好从此端口接收数据

③ 服务器打开20号数据端口，并且创建和客户端数据端口（N+1）的连接

FTP – 被动模式



- ① 客户端的命令端口N用于连接服务器的命令端口21
- ② 客户端通过命令端口N发送PASV命令给服务器的命令端口21
- ③ 服务器打开一个随机的数据端口P，并告知客户端该端口号P
- ④ 客户端数据端口N+1发起与服务器数据端口P的连接

邮件相关的协议

■ 发邮件使用的协议

□ SMTP (**S**imple **M**ail **T**ransfer **P**rotocol) , 译为: 简单邮件传输协议

✓ 基于TCP, 标准参考[RFC 5321](#)

✓ 服务器默认使用25端口, SSL/TLS使用465端口

■ 收邮件使用的协议

□ POP (**P**ost **O**ffice **P**rotocol) , 译为: 邮局协议

✓ 基于TCP, 最新版是POP3, 标准参考[RFC 1939](#)

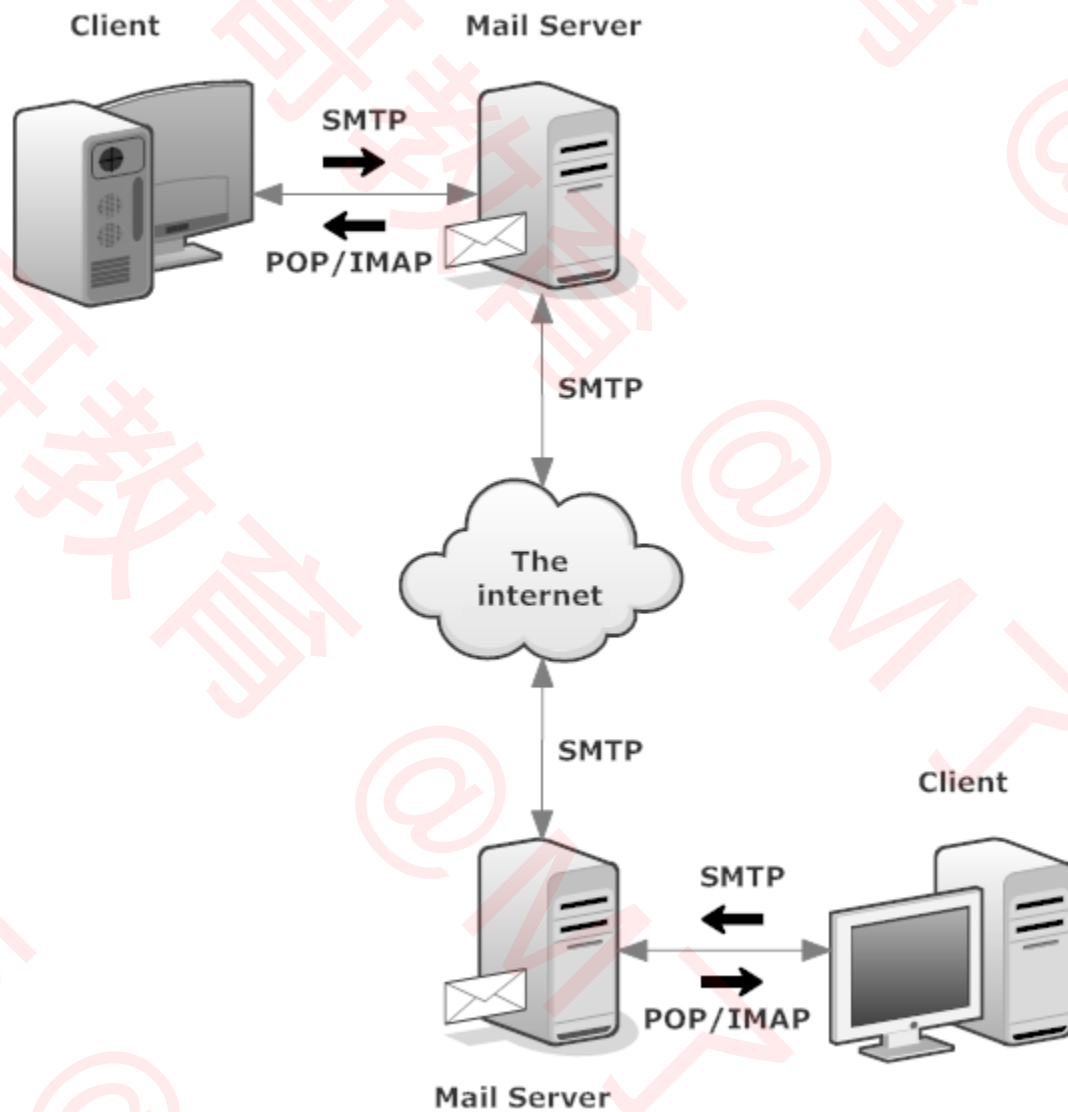
✓ 服务器默认使用110端口, SSL/TLS使用995端口

□ IMAP (**I**nternet **M**essage **A**ccess **P**rotocol) , 译为: 因特网信息访问协议

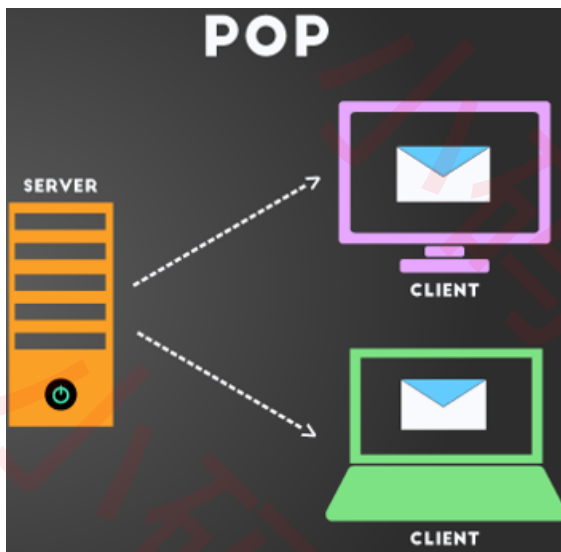
✓ 基于TCP, 最新版是IMAP4, 标准参考[RFC 3501](#)

✓ 服务器默认使用143端口, SSL/TLS使用993端口

收发邮件的过程

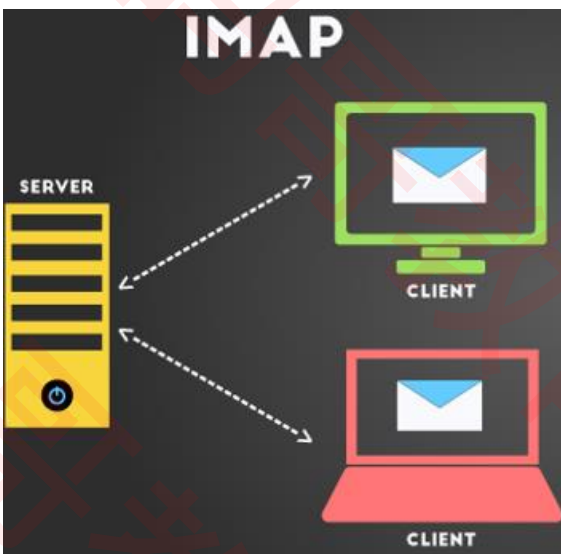


POP vs IMAP



■ POP的特点

- ❑ 客户端连接服务器时，将会从服务器下载所有邮件
- ✓ 可以设置下载完后，立即或一段时间后删除服务器邮件
- ❑ 客户端的操作（比如删除邮件、移动到文件夹）**不会**跟服务器同步
- ❑ 每个客户端都是独立的，都可以获得其自己的电子邮件副本



■ IMAP的特点

- ❑ 客户端连接服务器时，获取的是服务器上邮件的基本信息，并不会下载邮件
- ✓ 等打开邮件时，才开始下载邮件
- ❑ 客户端的操作（比如删除邮件、移动到文件夹）**会**跟服务器同步
- ❑ 所有客户端始终会看到相同的邮件和相同的文件夹

IPv6

- IPv6 (Internet Protocol version 6)，译为：网际协议第6版
 - 用它来取代IPv4主要是为了解决IPv4地址枯竭问题，同时它也在其他方面对于IPv4有许多改进
 - 然而长期以来IPv4在互联网流量中仍占据主要地位，IPv6的使用增长缓慢
 - 在2019年12月，通过IPv6使用Google服务的用户百分率首次超过30%
 - ✓ 因为需要设备、操作系统内核升级支持IPv6
-
- IPv6采用128位的地址，而IPv4使用的是32位
 - 支持 2^{128} (约 $3.4 * 10^{38}$) 个地址
 - 就以地球人口70亿人计算，每人平均可分得约 $4.86 * 10^{28}$ 个IPv6地址

IPv6 – 地址格式

- IPv6地址为128bit，每16bit一组，共8组
- 每组以冒号 “:” 隔开，每组以4位十六进制方式表示
 - 例如2001:0db8:86a3:08d3:1319:8a2e:0370:7344
- 类似于IPv4的点分十进制，同样也存在点分十六进制的写法
 - 2.0.0.1.0.d.b.8.8.5.a.3.0.8.d.3.1.3.1.9.8.a.2.e.0.3.7.0.7.3.4.4

IPv6 – 地址格式

■ 每组前面连续的0可以省略。下面的IPv6地址是等价的

□ 2001:0db8:02de:0000:0000:0000:0000:0e13

□ 2001:db8:2de:0:0:0:0:e13

■ 可以用双冒号 “::” 表示一组0或多组连续的0，但只能出现一次。下面的IPv6地址是等价的

□ 2001:db8:2de:0:0:0:0:e13

□ 2001:db8:2de::e13

■ 2001::25de::cade是非法的，因为双冒号出现了两次，会造成歧义

□ 2001:0000:0000:0000:0000:25de:0000:cade

□ 2001:0000:25de:0000:0000:0000:0000:cade

■ ::1是本地环回地址 (0:0:0:0:0:0:0:1)

IPv6 - 首部格式

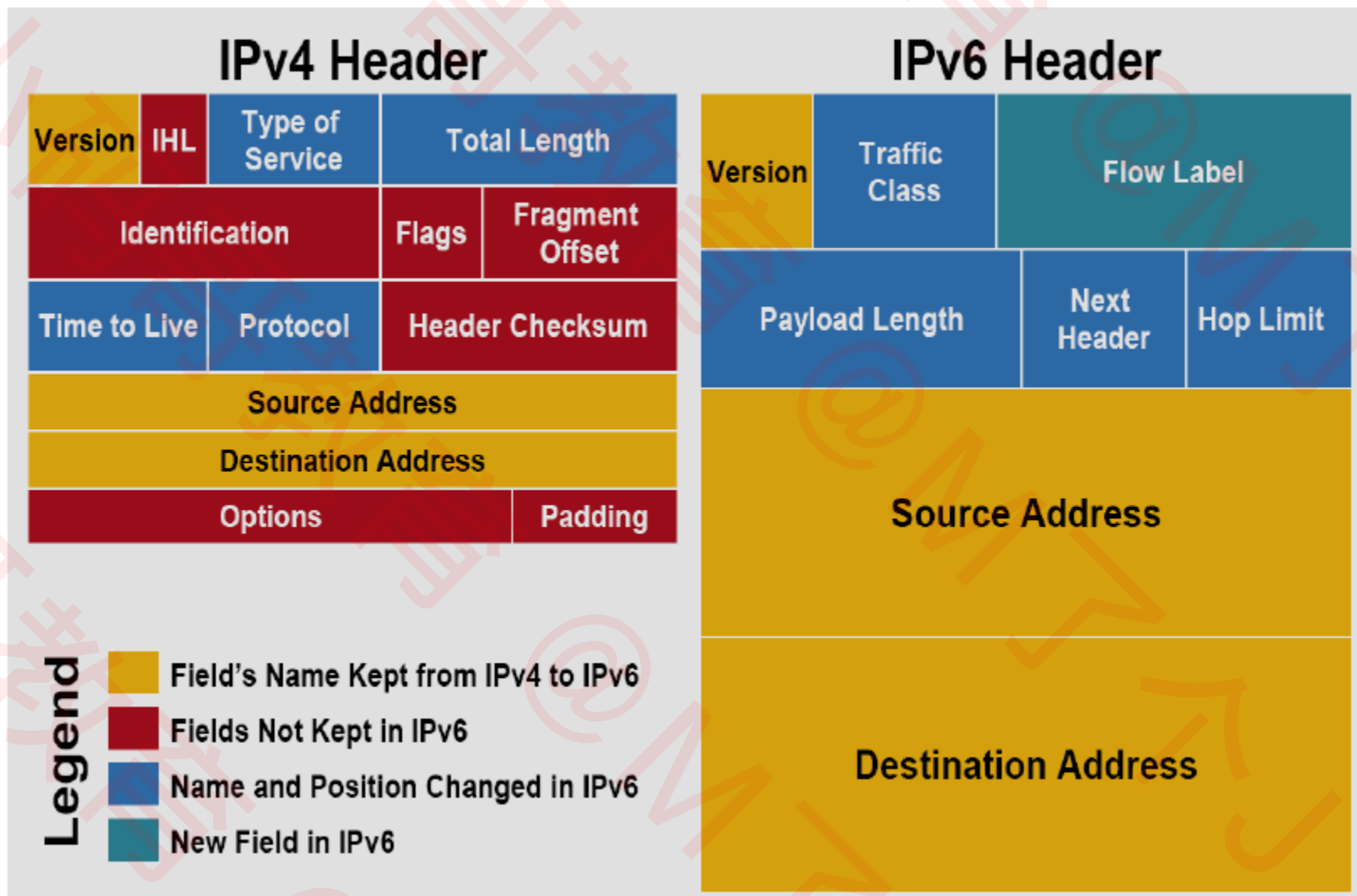
IPv6 Fixed header format																																	
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				Traffic Class								Flow Label																			
4	32	Payload Length																Next Header								Hop Limit							
8	64	Source Address																															
12	96																																
16	128																																
20	160																																
24	192	Destination Address																															
28	224																																
32	256																																
36	288																																

Legend:

	Field's name kept from IPv4 to IPv6
	Fields not kept in IPv6
	Name and position changed in IPv6
	New field in IPv6

- 有40字节的固定首部

IPv6 – 首部格式



IPv6 – 首部格式

■ **Version** (占4bit, 0110) : 版本号

■ **Traffic Class** (占8bit) : 交通类别

- 指示数据包的类别或优先级, 可以帮助路由器根据数据包的优先级处理流量
- 如果路由器发生拥塞, 则优先级最低的数据包将被丢弃

■ **Payload Length** (占16bit) : 有效负载长度

- 最大值65535字节
- 包括了扩展头部、上层(传输层)数据的长度

IPv6 – 首部格式

■ **Hop Limit** (占8bit) : 跳数限制

□ 与IPv4数据包中的TTL相同

■ **Source Address** (占128bit) : 源IPv6地址

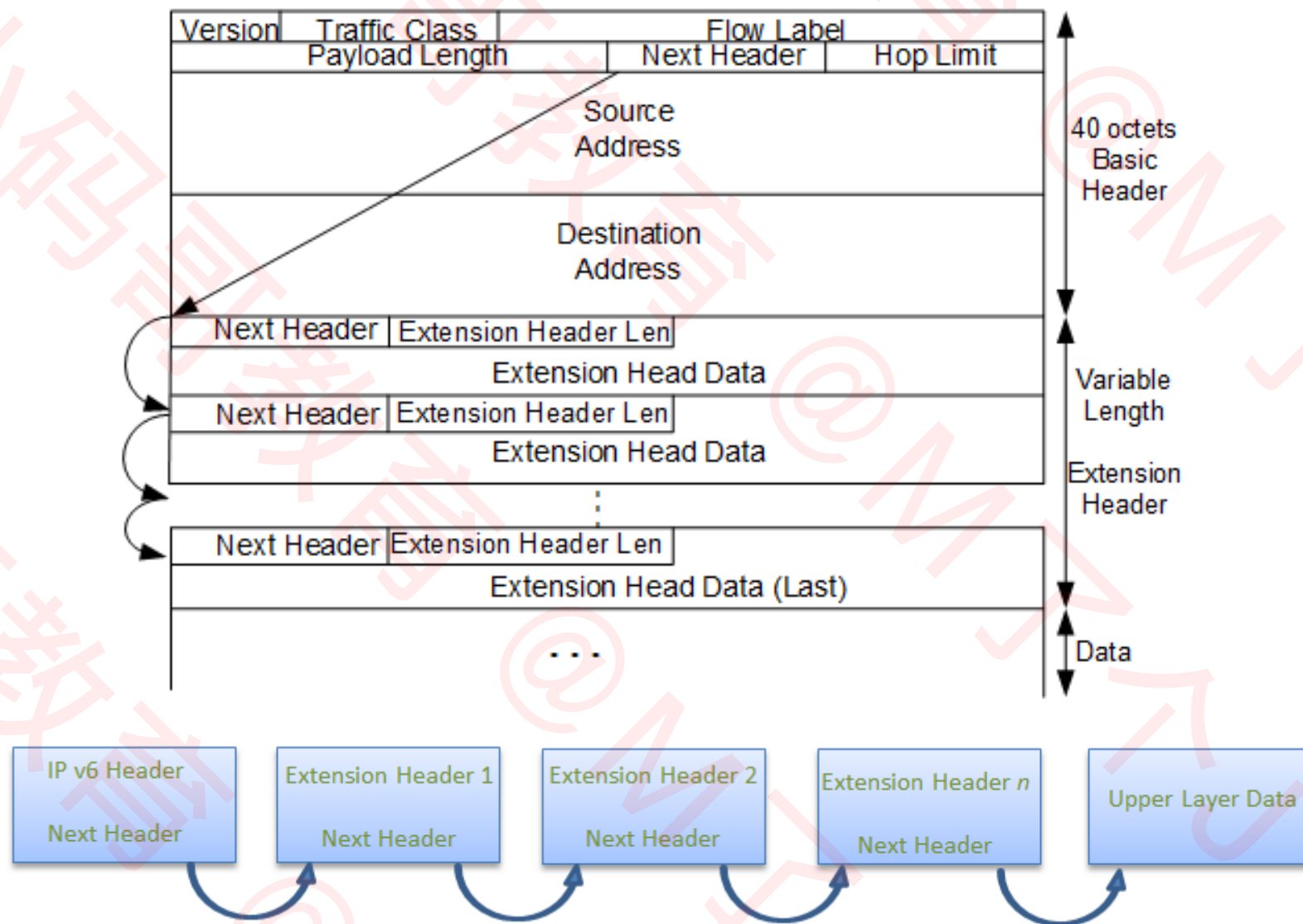
■ **Destination Address** (占128bit) : 目的IPv6地址

■ **Flow Label** (占20bit) : 流标签

□ 指示数据包属于哪个特定序列 (流)

□ 用数据包的源地址、目的地址、流标签标识一个流

IPv6 - 扩展头部



IPv6 - 扩展头部

■ **Next Header** (占8bit) : 下一个头部

□ 指示扩展头部 (如果存在) 的类型、上层数据包的协议类型 (例如TCP、UDP、ICMPv6)

Order	Header Type	Next Header Code
1	Basic IPv6 Header	-
2	Hop-by-Hop Options	0
3	Destination Options (with Routing Options)	60
4	Routing Header	43
5	Fragment Header	44
6	Authentication Header	51
7	Encapsulation Security Payload Header	50
8	Destination Options	60
9	Mobility Header	135
	No next header	59
Upper Layer	TCP	6
Upper Layer	UDP	17
Upper Layer	ICMPv6	58

Example: *TCP is used in IPv6 packet*

Next Header= 6	TCP header	TCP data
----------------	------------	----------

Example2:

Next Header= 43	Routing Extension Header	TCP header	TCP data
	Next Header= 6		

即时通信

- 即时通信（Instant Messaging，简称IM），平时用的QQ、微信，都属于典型的IM应用
- 国内的IM开发者社区
 - <http://www.52im.net/>
- IM云服务
 - [网易云信](#)、[腾讯云](#)、[环信](#)等
- 常用的协议
 - XMPP、MQTT、自定义协议

即时通信 — XMPP

- [XMPP](#) (Extensible Messaging and Presence Protocol)

- 译为：可扩展消息与存在协议，前身是Jabber

- 基于TCP，默认端口5222、5269

- 特点

- 使用XML格式进行传输，体积较大

- 比较成熟的IM协议，开发者接入方便



即时通信 — MQTT

■ [MQTT](#) (Message Queuing Telemetry Transport) , 译为: 消息队列遥测传输

□ 基于TCP, 默认端口1883、8883 (带SSL/TLS)

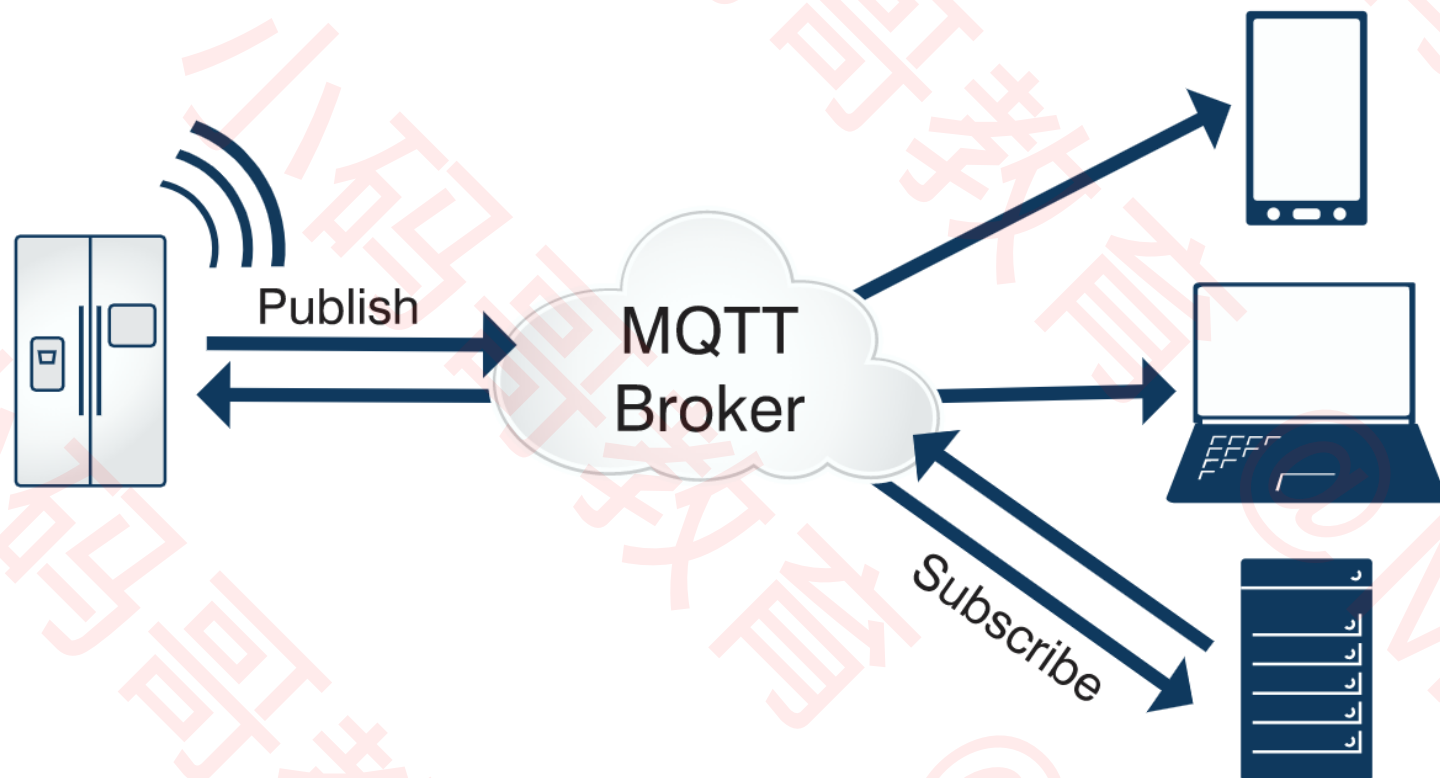
■ 特点

□ 开销很小, 以降低网络流量, 信息冗余远小于XMPP

□ 不是专门为IM设计的协议, 很多功能需要自己实现

□ 很多人认为MQTT是最适合物联网 (IoT, Internet of Things) 的网络协议

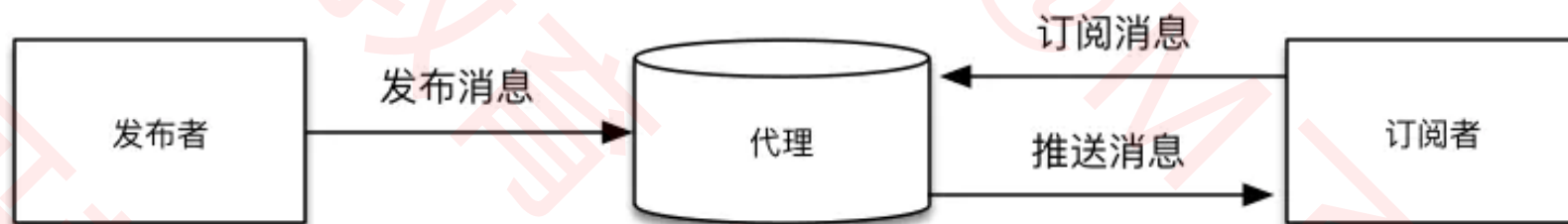
即时通信 — MQTT



■ 发布者：客户端

■ 代理：服务器

■ 订阅者：客户端



流媒体

- 流媒体 (Streaming Media) , 又叫流式媒体
- 是指将一连串的多媒体数据压缩后, 经过互联网分段发送数据, 在互联网上即时传输影音以供观赏的一种技术
- 此技术使得资料数据包得以像流水一样发送, 不使用此技术, 就必须在使用前下载整个媒体文件

流媒体 — 常见协议

■ RTP (**R**eal-Time **T**ransport **P**rotocol) , 译为: 实时传输协议

□ 参考: [RFC 3550](#)、[RFC 3551](#), 基于UDP

■ RTCP (**R**eal-Time **T**ransport **C**ontrol **P**rotocol) , 译为: 实时传输控制协议

□ 参考: [RFC 3550](#), 基于UDP, 使用RTP的下一个端口

■ RTSP (**R**eal-Time **S**treaming **P**rotocol) , 译为: 实时流协议, 参考: [RFC 7820](#)

□ 基于TCP、UDP的554端口

■ RTMP (**R**eal-Time **M**essaging **P**rotocol) , 译为: 实时消息传输协议, 由Adobe公司出品

□ 默认基于TCP的1935端口

■ HLS (**H**TT**P** **L**ive **S**treaming) , 基于HTTP的流媒体网络传输协议, 苹果公司出品, 参考: [RFC 8216](#)

