

# 课前必读

## 课程讲授的内容

包括：

图灵学院-网络编程和Netty课程表

章节	章节名称
1	深入理解网络通信和TCP/IP协议
2	BIO实战、NIO编程与直接内存、零拷贝深入辨析
3	深入Linux内核理解epoll
4	Netty使用和常用组件辨析
5	Netty实战-手写通信框架与面试难题分析

## 课程编排缘由

网络通信是架构师必须掌握的技能点，从大厂的岗位JD就可以看出来：

美团·社会招聘

首页 社会招聘 美好生活

保险平台-Java技术专家

金融服务平台

发布日期：2022-02-21

北京市 大学本科 3年

美团的使命是“帮大家吃得更好，生活更好”，公司聚焦“零售 + 科技”战略，和广大商户与各类合作伙伴一起，努力为消费者提供品质生活，推动商品零售和服务零售在需求侧和供给侧的数字化转型。

2018年9月20日，美团正式在港交所挂牌上市。美团将始终坚持以客户为中心，不断加大在科技研发方面的投入，更好承担社会责任，更多创造社会价值，与广大合作伙伴一起发展共赢。

岗位职责

岗位职责：

1.负责相关美团保险销售&自有业务服务的建设；

2.提供技术规划、架构设计方案；

3.完成重要业务模块及核心框架的搭建及编码实现；

3.发现和解决系统的技术问题，保证系统的性能和稳定性；

4.协同他人组织跨团队沟通协作，确保系统架构内外设计合理或保障项目质量与进度；

5.能够指导团队其他成员。

岗位基本需求

任职要求：

1.三年以上JAVA研发经验，熟悉Java虚拟机原理，Java高级特性，网络编程及多线程技术；

2.熟悉Spring、MyBatis、SpringBoot、Redis、Zookeeper、Kafka、Es，掌握底层架构、熟悉原理；

## 本地生活-Java后端专家-物流服务商品技术

发布时间:	2022-03-18	工作地点:	杭州	工作年限:	五年以上
所属部门:	本地生活	学 历:	本科	招聘人数:	若干

### 岗位描述:

我们是谁?

我们是一只年轻有活力的团队, 主要提供近场电商的履约仓储物流服务的商业关系, 包括消费商家和履约仓储物流服务商的快速规模化接入。服务对象有饿了么、淘鲜达、同城购、外部订单、天猫超市等业务场景。

我们做什么?

刻画各类业务场景的商业关系, 采用DDD架构设计思想, 抽象统一的商业关系业务领域模型, 支持商家和服务商的规模化接入和开放对接, 达到业财一体的目标。对商家和服务能力分层, 对不同的商家提供差异化的服务。我们有高性能、高并发、高可用等“三高”的技术场景, 有复杂领域业务的系统架构设计, 对技术和架构有追求的小伙伴, 有充足的发展空间。海阔凭鱼跃, 天高任鸟飞。来吧, 我们需要你。

### 岗位要求:

- 1、JAVA基础扎实, 理解IO 多线程、集合等基础框架, 对JVM原理有一定的了解
- 2、3年及以上JAVA开发经验, 对于使用过的开源框架, 能了解到它的原理和机制; 熟悉Spring, ibatis, struts等开源框架

仔细研究这些JD, 对网络通信有要求, 对Netty却没有要求。但是Java生态圈内的大量开源框架比如Dubbo、Zookeeper等等都用到了Netty, 如果简历上说研读过相关的框架源码却不知道和熟悉Netty, 这是说不通的, 所以对于Netty我们必须要知道它的用法和高级特性, 源码却不是必需。

但是如果应聘的是类似于下面的岗位:

美团·社会招聘

首页 社会招聘 美好生活

餐饮SaaS技术部-Java高级开发/Java技术专家

到店事业群

发布日期: 2021-10-05

成都市 大学本科 3年

美团餐饮系统, 为餐饮企业提供一站式IT解决方案, 帮餐饮商户实现从供应链管理、生产管理、前厅管理到外卖的数字化经营。美团餐饮系统不仅实现餐厅和平台的打通, 更帮餐厅连接客人, 帮餐饮商户了解顾客, 有助于做商业决策, 并给顾客带来更好的消费体验。

2020年12月3日, 美团“春风行动”百万商户成长计划增加了“餐饮系统”解决方案, 未来三年内, 通过全新升级的美团餐饮系统一体化软硬件产品, 助力超20万连锁门店, 100万餐饮商户线上化经营, 帮助商户实现新店推广、效率提升、降低成本、会员营销、流量扶持、智慧选址六大能力提升。

美团餐饮系统一体化软硬件产品, 主要包括智能收银一体机、智能打印机、扫码盒子和点菜宝等硬件, 配套餐饮管理软件美团餐饮系统智能版, 升级了会员营销功能、开业引流、连锁管理等方面功能。

岗位职责

1. 负责餐饮系统的需求分析、业务建模、技术调研和选型、系统架构设计、关键模块设计和开发工作;
2. 负责项目管理、实施和落地, 有一定的项目管理和团队协作能力, 通过流程优化和技术手段持续提升研发效能;
3. 负责系统架构重构优化, 建设高并发, 高性能, 高可用系统, 支撑亿级访问量;
4. 难点攻克, 技术输出, 指导初级工程师, 参与团队技术分享, 促进团队共同成长。

岗位基本需求

1. 具有大规模分布式系统应用架构设计经验, 扎实的计算机专业基本功;
2. 强大的写码能力, 精通Java及面向对象设计开发, 对部分Java技术有深入研究, 研究过优秀开源软件的源码并有心得者优先;
3. 较高的技术钻研能力、技术攻关能力, 分析问题解决问题的能力;
4. 研究过HTTP协议、搜索引擎、缓存、JVM调优、序列化、NIO、RPC框架等, 并且有相关实践经验;
5. 熟练掌握MySQL应用开发、数据库原理和常用性能优化和扩展技术, 以及 NoSQL, Queue 的原理、使用场景以

这个时候要求对Netty的源码有研究, 大家就可以学习第四期里诸葛老师关于Netty源码讲解的有关课程, 讲的非常好:

【图灵课堂VIP课程】第四期-分布式框架专题: [https://vip.tulingxueyuan.cn/detail/p\\_6006d2b8e4b0ab9a254a57fc/6](https://vip.tulingxueyuan.cn/detail/p_6006d2b8e4b0ab9a254a57fc/6), 特别是下面两节课:

所以，总的来说，如果时间上来得及、个人感兴趣、或者岗位要求则可以去学习Netty的源码。

## 上课说明

课程前置：Java语言基础知识、SSM、并发编程的基础知识，如线程、线程池等等。

1、一个知识点如果大部分同学明白，不会重复讲解，未明白的同学请看视频、笔记、请教同学或加老师QQ。

2、以上为本课的章节安排，不是课时安排，如果一章内容在一次课内未讲完，则会顺延到后面的课程继续讲解。

# 网络协议

## 计算机网络是什么？

随着计算机技术发展，计算机的体积和价格都在下降，之前计算机多用于研究机构，现阶段逐步进入一般的公司用于办公。原来计算机之间传输数据需要通过软盘等第三方存储介质进行转存，人们需要将数据直接通过通信线路传输，来缩短传输时间，于是计算机网络开始诞生，并逐渐发展为现在巨大的Internet。

## 定义和分类

计算机网络的标准定义是：利用通信线路将地理上分散的、具有独立功能的计算机系统和通信设备按不同的形式连接起来，以功能完善的网络软件及协议实现资源共享和信息传递的系统。

计算机网络从覆盖范围上划分可以分为三类：局域网、城域网、广域网。局域网LAN（作用范围一般为几米到几十公里）、城域网MAN（介于WAN与LAN之间）、广域网WAN（作用范围一般为几十到几千公里）。当然计算机网络划分不止这一种分类方式，可以按拓扑结构分类（总线型、环型、星型、网状）、还可以按信息的交换方式（电路交换、报文交换、报文分组交换）来分等等方式。

## 计算机网络发展简史

1、诞生阶段，20世纪60年代中期之前的第一代计算机网络是以单个计算机为中心的远程联机系统。

2、ARPANET，多个主机通过通信线路互联起来。60年代初。当时，美国国防部为了保证美国本土防卫力量和海外防御武装在受到前苏联第一次核打击以后仍然具有一定的生存和反击能力，认为有必要设计出一种分散的指挥系统；它由一个个分散的指挥点组成，当部分指挥点被摧毁后，其它点仍能正常工作，并且在这些点之间能够绕过那些已被摧毁的指挥点而继续保持联系。这个设计出发点

很重要，理解了它，就能够理解为何后面要学习的TCP要这么设计。为了对这一构思进行验证，1969年，美国国防部国防高级研究计划署(DOD / DARPA)资助建立了一个名为ARPANET(即“阿帕网”)的网络，将多个大学的计算机主机联接起来，位于各个结点的大型计算机采用分组交换技术，通过专门的通信交换机和专门的通信线路相互连接。E-mail、FTP和Telnet在ARPANET上已经诞生。

3、开放性的标准化体系结构，OSI诞生。ARPANET兴起后，计算机网络发展迅猛，各大计算机公司相继推出自己的网络体系结构及实现这些结构的软硬件产品。由于没有统一的标准，不同厂商的产品之间互联很困难，人们迫切需要一种开放性的标准化实用网络环境，这样应运而生了两种国际通用的最重要的体系结构，为了实现网络设备间的互相通讯，ISO和IEEE（电气和电子工程师协会，是世界上最大的非营利性专业技术学会）相继提出了OSI参考模型及其TCP/IP模型。由于TCP/IP尽早地制定了可行性较强的协议，提出了应对技术快速革新的协议，并及时进行后期改良的方案，因此打败了OSI模型，成为了事实上的标准。

#### 4、Internet互联网

20世纪90年代至今的第四代计算机网络，就是我们所熟知的Internet互联网。

既然网络是很多的计算设备（电脑、手机等等）连接在一起的，这些计算设备来自不同的公司，有不同的体系结构，相互之间如何通信呢？这就好比我们的语言，中国地广人多，地方性语言也非常丰富，而且方言之间差距巨大。A地区的方言可能B地区的人根本无法听懂，所以要为全国进行沟通建立一个语言标准，这就是我们的普通话的作用。计算机网络协议同我们的普通话一样，帮助我们的计算机之间进行沟通。

想要成为计算机网络领域高手，掌握计算机网络领域知识就是必备的，而学习计算机网络领域知识过程就是理解网络协议的构成、原理和工作方式的过程。

## 计算机网络体系结构

### OSI七层模型

开放系统互连参考模型（Open System Interconnect 简称OSI）是国际标准化组织(ISO)和国际电话咨询委员会(CCITT)联合制定的开放系统互连参考模型，为开放式互连信息系统提供了一种功能结构的框架。其目的是为异种计算机互连提供一个共同的基础和标准框架，并为保持相关标准的一致性和兼容性提供共同的参考。这里所说的开放系统，实质上指的是遵循OSI参考模型和相关协议能够实现互连的具有各种应用目的的计算机系统。

OSI采用了分层的结构化技术，共分七层，物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

### TCP/IP模型

OSI模型比较复杂且学术化，所以我们实际使用的TCP/IP模型，分5层，物理层、数据链路层（也有TCP/IP模型将物理层、数据链路层合称为网络接口层，与之对应的，协议就被称为TCP/IP四层协议模型）、网络层、传输层、应用层。两个模型之间的对应关系如图所示：

无论什么模型，每一个抽象层建立在低一层提供的服务上，并且为高一层提供服务。大致来说，可以这么理解（只是帮助我们理解，实际上肯定会有点出入），对于我们的PC机来说，物理层可以看成网卡，数据链路层可以看成网卡驱动程序，网络层和传输层由操作负责处理，应用层则是常用的一些网络应用程序和我们自己所编写的网络应用程序。

**面试点：两个模型在一些国企或者学术气氛较浓厚公司的笔试或面试中可能会出现，请务必牢记。**

## TCP/IP协议族

Transmission Control Protocol/Internet Protocol的简写，中译名为传输控制协议/因特网互联协议，是Internet最基本的协议、Internet国际互联网络的基础，由网络层的IP协议和传输层的TCP协议组成。协议采用了5层的层级结构。然而在很多情况下，它是利用 IP 进行通信时所必须用到的协议群的统称。也就是说，它其实是个协议家族，由很多个协议组成，并且是在不同的层，是互联网的基础通信架构。

### IP、TCP和UDP

在上述图形中，网际协议IP是TCP/IP中非常重要的协议，往往用来确定网络中唯一的一台计算设备，它的作用就好比现实生活中的电话号码或者或者通讯地址。所以这层负责对数据加上IP地址（有发送它的主机的地址（源地址）和接收它的主机的地址（目的地址））和其他的数据以确定传输的目标。

而TCP和UDP都是传输层的协议，传输层主要为两台主机上的应用程序提供端到端的通信。

TCP有点类似于我们日常生活中的打电话，电话接通后通过“喂”确认对方身份，听不清会要求对方重说，对方说的太快了会要求对方说慢点，讲完了各说一句“再见”结束通话。TCP提供了一种可靠的数据传输服务，TCP是面向连接的，也就是说，利用TCP通信的两台主机首先要经历一个建立连接的过程，等到连接建立后才开始传输数据，而且传输过程中采用“带重传的肯定确认”技术来实现传输的可靠性。TCP还采用一种称为“滑动窗口”的方式进行流量控制，发送完成后还会关闭连接。

UDP（User Datagram Protocol的简称，中文名是用户数据报协议）有点类似于我们日常生活中通过不靠谱的物流系统寄东西。UDP是把数据直接发出去，而不管对方是不是在接收，也不管对方是否能接收的了，也不需要接收方确认，属于不可靠的传输，可能会出现丢包现象，实际应用中要求程序员编程验证。

所以TCP要比UDP可靠的多。

注意：

我们一些常见的网络应用基本上都是基于TCP和UDP的，这两个协议又会使用网络层的IP协议。但是我们完全可以绕过传输层的TCP和UDP，直接使用IP，比如Linux内核中的LVS就可以直接基于IP层进行负载平衡调度；甚至还可以直接访问链路层，比如tcpdump程序就是直接和链路层进行通信的。

## TCP/IP网络传输中的数据

每个分层中，都会对所发送的数据附加一个首部，在这个首部中包含了该层必要的信息，如发送的目标地址以及协议相关信息。通常，为协议提供的信息为包首部，所要发送的内容为数据。在下一层



的角度看，从上一层收到的包全部都被认为是本层的数据。

网络中传输的数据包由两部分组成：一部分是协议所要用的首部，另一部分是上一层传过来的数据。首部的结构由协议的具体规范详细定义。在数据包的首部，明确标明了协议应该如何读取数据。反过来说，看到首部，也就能够了解该协议必要的信息以及所要处理的数据。我们用用户A发送，用户B接受来说说明：

① 用户A应用程序处理 首先应用程序会进行编码处理产生报文/消息（message）交给下面的TCP层。

② 用户A TCP 模块的处理 TCP 根据应用的指示，负责建立连接、发送数据以及断开连接。TCP 提供将应用层发来的数据顺利发送至对端的可靠传输。为了实现这一功能，需要将应用层数据封装为报文段（segment）并附加一个 TCP 首部然后交给下面的IP层。

③ 用户A IP 模块的处理 IP 将 TCP 传过来的 TCP 首部和 TCP 数据合起来当做自己的数据，并在 TCP 首部的前端加上自己的 IP 首部生成IP数据报（datagram）然后交给下面的数据链路层。

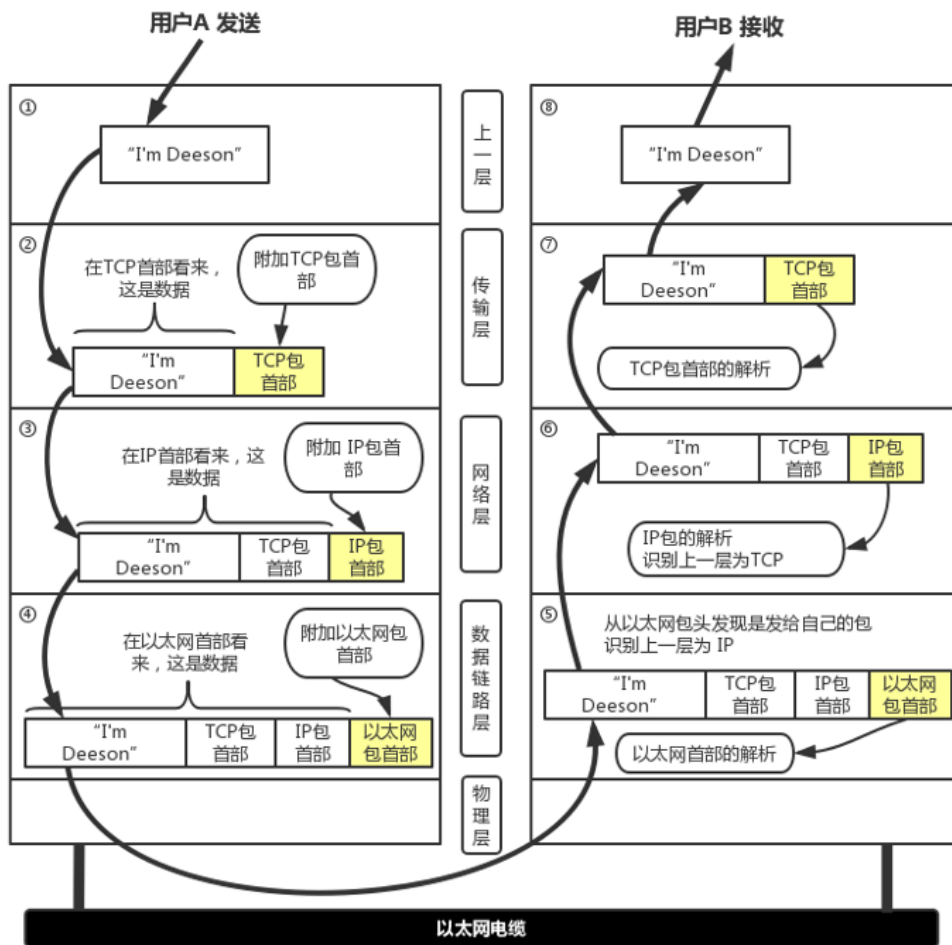
④ 用户A数据链路层的处理 从 IP 传过来的 IP 包对于数据链路层来说就是数据。给这些数据附上链路层首部封装为链路层帧（frame），生成的链路层帧（frame）将通过物理层传输给接收端。

⑤ 用户B数据链路层的处理 用户B主机收到链路层帧（frame）后，首先从链路层帧（frame）首部找到 **MAC 地址** 判断是否为发送给自己的包，若不是则丢弃数据。如果是发送给自己的包，则从以太网包首部中的类型确定数据类型，再传给相应的模块，如 IP、ARP 等。这里的例子则是 IP 。

⑥ 用户B IP 模块的处理 IP 模块接收到 数据后也做类似的处理。从包首部中判断此 IP 地址是否与自己的 IP 地址匹配，如果匹配则根据首部的协议类型将数据发送给对应的模块，如 TCP、UDP。这里的例子则是 TCP。

⑦ 用户B TCP 模块的处理 在 TCP 模块中，首先会计算一下校验和，判断数据是否被破坏。然后检查是否在按照序号接收数据。最后检查端口号，确定具体的应用程序。数据被完整地接收以后，会**传给**由端口号识别的应用程序。

⑧ 用户B 应用程序的处理 接收端应用程序会直接接收发送端发送的数据。通过解析数据，展示相应的内容。



## 地址和端口号

### MAC 地址

我们常听说 MAC 地址和 IP 地址。

MAC地址全称叫做媒体访问控制地址，也称为局域网地址（LAN Address），MAC位址，以太网地址（Ethernet Address）或物理地址（Physical Address），由网络设备制造商生产时写在硬件内部。MAC地址与网络无关，也即无论将带有这个地址的硬件（如网卡、集线器、路由器等）接入到网络的何处，都有相同的MAC地址，它由厂商写在网卡的BIOS里，从理论上讲，除非盗来硬件（网卡），否则是没有办法冒名顶替的。

MAC地址共48位（6个字节）。前24位由IEEE（电气和电子工程师协会）决定如何分配，后24位由实际生产该网络设备的厂商自行制定。例如：FF:FF:FF:FF:FF:FF或FF-FF-FF-FF-FF-FF。

Windows下可用ipconfig -all

以太网适配器 以太网:

```

连接特定的 DNS 后缀 . . . . . :
描述. . . . . : Realtek PCIe GbE Family Controller
物理地址. . . . . : B0-25-AA-35- -BE
DHCP 已启用 . . . . . : 是
自动配置已启用. . . . . : 是
本地链接 IPv6 地址. . . . . : fe80::c85e:d254:bf25:449a%12(首选)
IPv4 地址 . . . . . : 192.168.0.154(首选)
子网掩码 . . . . . : 255.255.255.0
获得租约的时间 . . . . . : 2020年9月15日 10:25:02
租约过期的时间 . . . . . : 2020年9月15日 16:08:59
  
```

Linux 最常用的查看mac地址的方式 有很多种，下面给出4种方式：

```
ifconfig -a  
ip link show  
cat /sys/class/net/eth0/address 查看eth0的mac地址  
dmesg | grep eth0
```

## IP地址

IP地址（Internet Protocol Address）的全称叫作互联网协议地址，它的本义是为互联网上的每一个网络和每一台主机配置一个唯一的逻辑地址，用来与物理地址作区分。

所以IP 地址用来识别 TCP/IP 网络中互连的主机和路由器。IP地址基于逻辑，比较灵活，不受硬件限制，也容易记忆。

IP地址分为：IPv4和IPv6。我们这里着重讲的是IPv4地址，IP地址是由32位的二进制数组成，它们通常被分为4个“8位二进制数”，我们可以把它理解为4个字节，格式表示为：（A.B.C.D）。其中，A，B，C，D这四个英文字母表示为0-255的十进制的整数。例：192.168.1.1

Tips：IP地址和MAC地址之间的区别

1、对于网络中的一些设备，路由器或者是PC及而言，IP地址的设计是出于拓扑设计出来的，只要在不重复IP地址的情况下，它是可以随意更改的；而MAC地址是根据生产厂商烧录好的，它一般不能改动的，一般来说，当一台PC机的网卡坏了之后，更换了网卡之后MAC地址就会变了。

2、在前面的介绍里面，它们最明显的区别就是长度不同，IP地址的长度为32位，而MAC地址为48位。

3、它们的寻址协议层不同。IP地址应用于OSI模型的网络层，而MAC地址应用在OSI模型的数据链路层。数据链路层协议可以使数据从一个节点传递到相同链路的另一个节点上（通过MAC地址），而网络层协议使数据可以从一个网络传递到另一个网络上（ARP根据目的IP地址，找到中间节点的MAC地址，通过中间节点传送，从而最终到达目的网络）。

4、分配依据不同。IP地址的分配是基于我们自身定义的网络拓扑，MAC地址的分配是基于制造商。

## 端口号

在传输层也有这种类似于地址的概念，那就是端口号。端口号用来识别同一台计算机中进行通信的不同应用程序。因此，它也被称为程序地址。

一台计算机上同时可以运行多个程序。传输层协议正是利用这些端口号识别本机中正在进行通信的应用程序，并准确地将数据传输。

### 面试题：为什么端口号有65535个？

因为在TCP、UDP协议报文的开头，会分别有16位二进制来存储源端口号和目标端口号，所以端口个数是  $2^{16}=65536$  个，但是0号端口用来表示所有端口，所以实际可用的端口号是65535个。

### 端口号的确定



- 标准既定的端口号：这种方法也叫静态方法。它是指每个应用程序都有其指定的端口号。但并不是说可以随意使用任何一个端口号。例如 HTTP、FTP、TELNET 等广为使用的应用协议中所使用的端口号就是固定的。这些端口号被称为知名端口号，分布在 0~1023 之间，我们在编写自己的网络应用程序时，尽量不要使用这些端口号。

- 时序分配法：服务器有必要确定监听端口号，以让客户端程序访问服务器上的服务。但是客户端没必要确定端口号。在这种方法下，客户端应用程序完全可以不用自己设置端口号，而全权交给操作系统进行分配，客户端使用的临时端口号，操作系统分配的一般都是大于10000的。

## 观察端口号

Windows下使用netstat -ano 查看所有端口号，netstat -ano|findstr “<端口号>” 查看指定端口号。

Linux下可以用root 用户执行 lsof -i:端口号查看指定端口占用。

lsof -i -U：显示所有打开的UNIX domain和端口文件

我们用的更多的是netstat

netstat -tunlp 用于显示 tcp, udp 的端口和进程等相关情况。

netstat 查看端口占用语法格式：

netstat -tunlp | grep 端口号

-t (tcp) 仅显示tcp相关选项

-u (udp)仅显示udp相关选项

-n 拒绝显示别名，能显示数字的全部转化为数字

-l 仅列出在Listen(监听)的服务状态

-p 显示建立相关链接的程序名

## 综述

所以一般来说，不管计算机中有多少网卡，每个网卡都会有自己的MAC 地址，这个MAC 地址是不会变化的。而每个网卡在正常工作的情况下，都会有一个IP地址，这个IP地址完全是可以变化的。而这台计算机中承载的各种应用程序可以拥有自己的端口号，然后通过服务器的网卡，正确地进行网络通信。

一台服务器上的不同网络应用程序必须有不同的端口号，A程序启动了使用了端口x，B程序启动就不能使用端口x，否则会报错“Address already in use”。

总的来说，操作系统是通过源IP地址、目标IP地址、协议号（协议类型）、源端口号以及目标端口号这五个元素唯一性的识别一个网络上的通信。

## 面试题：一台主机上只能保持最多 65535 个 TCP 连接，对吗？

这个说法不对，我们分服务器和客户端分开讨论，以下的讨论都基于服务器和客户端都只有1个IP地址。

## 服务端

我们已经知道网络通信五元组是由源IP地址、目标IP地址、协议号（协议类型）、源端口号以及目标端口号构成。现在考察的是 TCP 连接，自然五元组中的协议号已经定下来了，于是网络通信五元组就变化为TCP四元组。

那就是说TCP连接四元组是由源IP地址、源端口、目的IP地址和目的端口构成。

很明显当四元组中任意一个元素发生了改变，那么就代表的是一条完全不同的新连接。拿我们常用的MySQL 举例，假设它的 IP 是 X，端口3306。用户A基于IP地址A1，端口PA连接MySQL，于是构成了一个TCP连接四元组(A1, PA, X, 3306)。用户B基于IP地址B1，端口PB连接同一个MySQL，这个时候MySQL需要开启一个新端口来和用户B通信吗？从我们日常的开发就可以知道，MySQL并不需要这么做，所以用户B就和MySQL构成了一个新的TCP连接四元组(B1, PB, X, 3306)。

服务端理论上能达成的最高并发数量是多少？从我们上面的用户A和用户B构成的TCP连接四元组：

(A1, PA, X, 3306)

(B1, PB, X, 3306)

可以看到目的IP地址和目的端口（X, 3306）是不变的，这样就只剩下源IP地址、源端口是可变的。IP 地址是一个 32 位的整数，所以源 IP 最大有 2 的 32 次方这么多个。端口是一个 16 位的整数，所以端口的数量就是 2 的 16 次方。2 的 32 次方（ip数）× 2 的 16 次方（port数）大约等于两百多万亿。所以理论上，我们每个 server 可以接收的连接上限就是两百多万亿。

当然实际上做不到，目前工程实践中可以达到的连接数在千万级别。基于Java的应用程序大概能支持百万级别，具体怎么做会在本课程第五章中详细说明。

## 客户端

前面我们已经说过，“客户端应用程序完全可以不用自己设置端口号，而全权交给操作系统进行分配”，可用的端口号只有6万多，从这个角度考虑，客户端最多只能发起6万多条 TCP 连接。但其实也不是。

从TCP连接四元组来考虑：源IP地址、源端口、目的IP地址和目的端口，目的IP地址和目的端口指的是服务器的IP和端口，源IP地址、源端口自然就是客户端的。

只要服务器的 IP 或者端口不一样，即使客户端的 IP 和端口是一样的。这个四元组也是属于一条完全不同的新连接。比如：

连接1：客户端IP 10000 服务器IP 10000

连接2：客户端IP 10000 服务器IP 20000 虽然客户端的 IP 和端口完全一样，但由于服务器侧的端口不同，所以仍然是两条不同的连接。问题来了，客户端同一个端口可以连接不同的服务器吗？答案是可以的。

客户端只要启动时不显示绑定到某个端口上，内核是可以使用一个端口连不同的服务端，内核会自己进行选择并恰当地复用的，而且完全不会产生数据混乱，因为“源IP地址、目标IP地址、源端口号以及目标端口号就能唯一性确定一个TCP连接”。

那么对客户端来说，四元组里有3个可变，自然客户端能同时支持的连接数比服务器还要大得多。

# TCP特性

在我们上面的讲述中，存在着客户端和服务端两者角色，在网络通信里是怎么区分的？这个就牵涉到了TCP的相关特性。

TCP (Transmission Control Protocol) 是面向连接的通信协议，通过三次握手建立连接，然后才能开始数据的读写，通讯完成时要拆除连接，由于TCP是面向连接的所以只能用于端到端的通讯。

TCP提供的是一种可靠的数据流服务，数据有可能被拆分后发送，那么采用超时重传机制是和应答确认机制是组成TCP可靠传输的关键设计。

而超时重传机制中最最重要的就是重传超时 (RTO, Retransmission TimeOut) 的时间选择，很明显，在工程上和现实中网络环境是十分复杂多变的，有时候可能突然的抽风，有时候可能突然的又很顺畅。在数据发送的过程中，如果用一个固定的值一直作为超时计时器的时长是非常不经济也非常不准确的方法，这样的话，超时的时长就需要根据网络情况动态调整，就需要采样统计一个数据包从发送端发送出去到接收到这个包的回复这段时长来动态设置重传超时值，这个时长就是为RTT，学名 round-trip time，然后再根据这个RTT通过各种算法和公式平滑RTT值后，最终确定重传超时值。

而IP层进行数据传输时，是不能保证数据包按照发送的顺序达到目的机器。当IP将把它们向‘上’传送到TCP层后，TCP将包排序并进行错误检查。TCP数据包中包括序号和确认，所以未按照顺序收到的包可以被排序，而损坏的包可以被重传。

TCP还采用一种称为“滑动窗口”的方式进行流量控制，所谓窗口实际表示接收能力，用以限制发送方的发送速度。

同时TCP还允许在一个TCP连接上，通信的双方可以同时传输数据，也就是所谓的全双工。

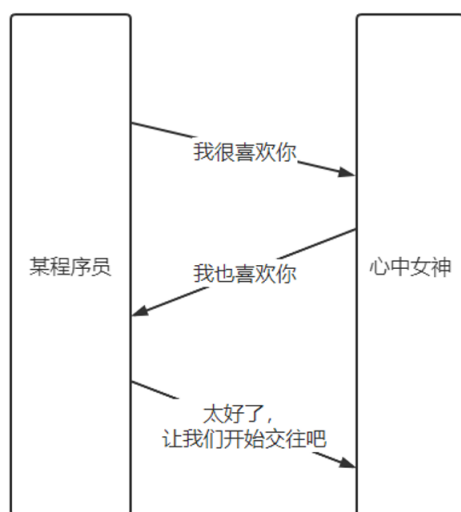
面向连接的服务（例如Telnet、FTP、rlogin、X Windows和SMTP）需要高度的可靠性，所以它们使用了TCP。DNS在某些情况下使用TCP（发送和接收域名数据库），但使用UDP传送有关单个主机的信息。

## TCP三次握手

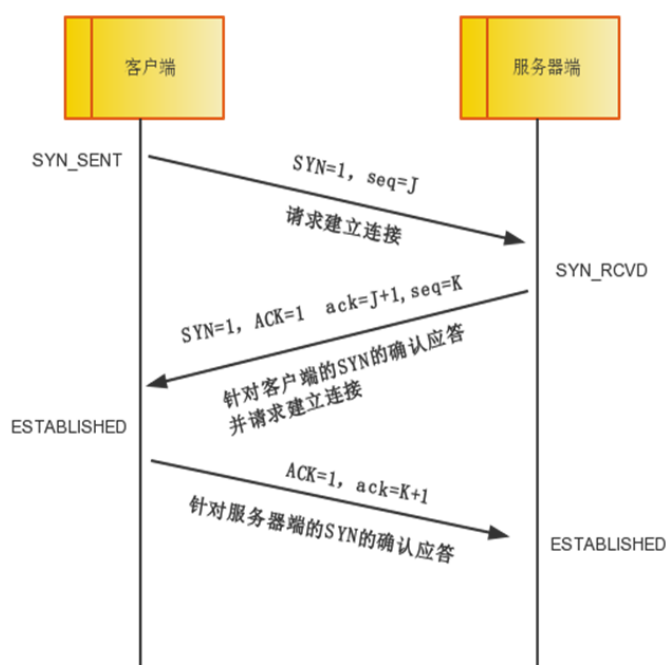
TCP 提供面向有连接的通信传输。面向有连接是指在数据通信开始之前先做好两端之间的准备工作。

所谓三次握手是指建立一个 TCP 连接时需要客户端和服务端总共发送三个包以确认连接的建立。在socket编程中，这一过程由客户端执行connect来触发，所以网络通信中，发起连接的一方我们称为客户端，接收连接的一方我们称之为服务端。

简易速记版



但是简易速记版，在真实的面试中不能让面试官满意，很多公司还会考察握手过程中的具体详情：



第一次握手：客户端将请求报文标志位SYN置为1，请求报文的Sequence Number字段（简称seq）中填入一个随机值J，并将该数据包发送给服务器端，客户端进入SYN\_SENT状态，等待服务器端确认。

第二次握手：服务器端收到数据包后由请求报文标志位SYN=1知道客户端请求建立连接，服务器端将应答报文标志位SYN和ACK都置为1，应答报文的Acknowledgment Number字段（简称ack）中填入ack=J+1，应答报文的seq中填入一个随机值K，并将该数据包发送给客户端以确认连接请求，服务器端进入SYN\_RCVD状态。

第三次握手：客户端收到应答报文后，检查ack是否为J+1，ACK是否为1，如果正确则将第三个报文标志位ACK置为1，ack=K+1，并将该数据包发送给服务器端，服务器端检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，客户端和服务器端进入ESTABLISHED状态，完成三次握手，随后客户端与服务器端之间可以开始传输数据了。

### 为什么TCP握手需要三次？

TCP是可靠的传输控制协议，而三次握手是保证数据可靠传输又能提高传输效率的最小次数。为什么？RFC793，也就是TCP的协议RFC中就谈到了原因，这是因为：

为了实现可靠数据传输，TCP协议的通信双方，都必须维护一个序列号，以标识发送出去的数据包中，哪些是已经被对方收到的。

举例说明：发送方在发送数据包（假设大小为 10 byte）时，同时送上一个序号（假设为 500），那么接收方收到这个数据包以后，就可以回复一个确认号（ $510 = 500 + 10$ ）告诉发送方“我已经收到了你的数据包，你可以发送下一个数据包，序号从 511 开始”。

三次握手的过程即是通信双方相互告知序列号起始值，并确认对方已经收到了序列号起始值的必经步骤。

如果只是两次握手，至多只有连接发起方的起始序列号能被确认，另一方选择的序列号则得不到确认。

至于为什么不是四次，很明显，三次握手后，通信的双方都已经知道了对方序列号起始值，也确认了对方知道自己序列号起始值，第四次握手已经毫无必要了。

### TCP的三次握手的漏洞-SYN洪泛攻击

但是在TCP三次握手中是有一个缺陷，被称为SYN洪泛攻击。三次握手中有一个第二次握手，服务端向客户端应答请求，应答请求是需要客户端IP的，而且因为握手过程没有完成，操作系统使用队列维持这个状态（Linux 2.2以后，这个队列大小参数可以通过`/proc/sys/net/ipv4/tcp_max_syn_backlog`设置）。于是攻击者就伪造这个IP，往服务器端狂发送第一次握手的内容，当然第一次握手时的客户端IP地址是伪造的，从而服务端忙于进行第二次握手，但是第二次握手是不会有应答的，所以导致服务器队列满，而拒绝连接。

面对这种攻击，有以下的解决方案，最好的方案是防火墙。

#### 无效连接监视释放

这种方法不停监视所有的连接，包括三次握手的，还有握手一次的，反正是所有的，当达到一定（与）阈值时拆除这些连接，从而释放系统资源。这种方法对于所有的连接一视同仁，不管是正常的还是攻击的，所以这种方式不推荐。

#### 延缓TCB分配方法

一般的做完第一次握手之后，服务器就需要为该请求分配一个TCB（连接控制资源），通常这个资源需要200多个字节。延迟TCB的分配，当正常连接建立起来后再分配TCB则可以有效地减轻服务器资源的消耗。

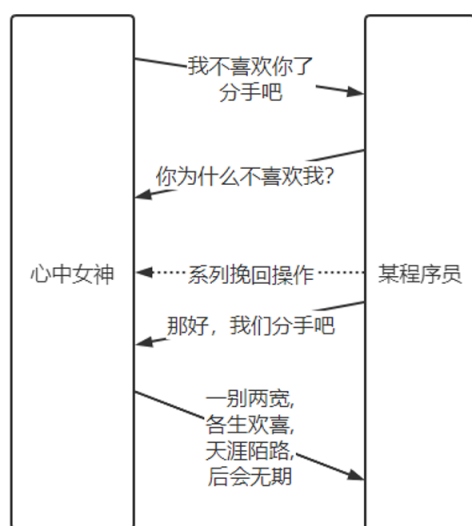
#### 使用防火墙

防火墙在确认了连接的有效性后，才向内部的服务器（Listener）发起SYN请求，

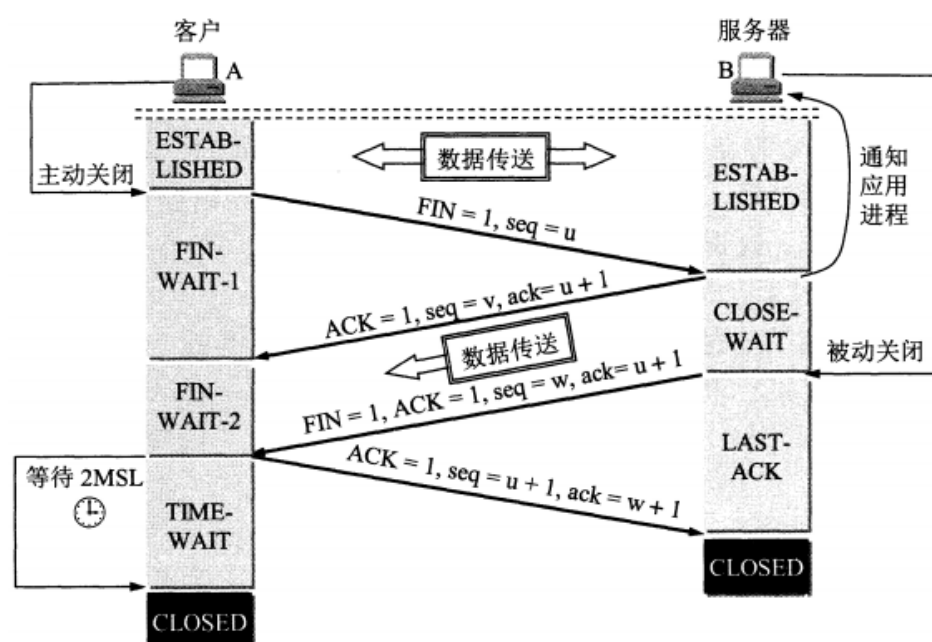
### TCP四次挥手（分手）

四次挥手即终止TCP连接，就是指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开。在socket编程中，这一过程由客户端或服务端任一方执行close来触发。过程简易记忆版：





由于TCP连接是全双工的，因此，每个方向都必须单独进行关闭。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭。



由上图可见，TCP建立一个连接需3个分节，终止一个连接则需4个分节。

(1) 某个应用进程首先调用close，我们称该端执行主动关闭 (active close)。该端的TCP于是发送一个FIN分节，表示数据发送完毕，应用进程进入FIN-WAIT-1（终止等待1）状态。

(2) 接收到这个FIN的对端执行被动关闭 (passive close)，发出确认报文。因为FIN的接收意味着接收端应用进程在相应连接上再无额外数据可接收，接收端进入了CLOSE-WAIT（关闭等待）状态，这时候处于半关闭状态，即主动关闭端已经没有数据要发送了，但是被动关闭端若发送数据，主动关闭端依然要接受。这个状态还要持续一段时间，也就是整个CLOSE-WAIT状态持续的时间。主动关闭端收到确认报文后进入FIN-WAIT-2（终止等待2）状态。

(3) 一段时间后，被动关闭的应用进程将调用close关闭它的套接字。这导致它的TCP也发送一个FIN，表示它也没数据需要发送了。

(4) 接收这个最终FIN的原发送端TCP（即执行主动关闭的那一端）确认这个FIN发出一个确认ACK报文，并进入了TIME-WAIT（时间等待）状态。注意此时TCP连接还没有释放，必须经过 $2 \times \text{MSL}$ （最长报文段寿命/最长分节生命期 max segment lifetime，MSL是任何IP数据报能够在因特网中存活的最长时间，任何TCP实现都必须为MSL选择一个值。RFC 1122[Braden 1989]的建议值是2分钟，不过源自

Berkelcy的实现传统上改用30秒这个值。这意味着TIME\_WAIT状态的持续时间在1分钟到4分钟之间)的时间后,当主动关闭端撤销相应的TCB后,才进入CLOSED状态。(linux kernel 的 tcp time wait的时间,有个 sysctl参数貌似可以使用,它是 /proc/sys/net/ipv4/tcp\_fin\_timeout,缺省值是60,也就是60秒,很多网上的资料都说将这个数值设置低一些就可以减少netstat里面的TIME\_WAIT状态,但是这个说法不是很准确的。经过认真阅读Linux的内核源代码,我们发现这个数值其实是输出用的,修改之后并没有真正的读回内核中进行使用,而内核中真正管用的是一个宏定义,在\$KERNEL/include/net/tcp.h里面,有下面的行: #defineTCP\_TIMEWAIT\_LEN (60\*HZ)

而这个宏是真正控制TCP TIME\_WAIT 状态的超时时间的。如果我们希望减少 TIME\_WAIT状态的数目(从而节省一点点内核操作时间),那么可以把这个数值设置低一些,根据我们的测试,设置为 10秒比较合适,也就是把上面的修改为: #defineTCP\_TIMEWAIT\_LEN (10\*HZ))

(5) 被动关闭端只要收到了客户端发出的确认,立即进入CLOSED状态。同样,撤销TCB后,就结束了这次的TCP连接。可以看到,被动关闭端结束TCP连接的时间要比主动关闭端早一些。

既然每个方向都需要一个FIN和一个ACK,因此通常需要4个分节。我们使用限定词“通常”是因为:某些情形下步骤1的FIN随数据一起发送;另外,步骤2和步骤3发送的分节都出自执行被动关闭那一端,有可能被合并成一个分节。

## 为什么TCP的挥手需要四次?

TCP是全双工的连接,必须两端同时关闭连接,连接才算真正关闭。

如果一方已经准备关闭写,但是它还可以读另一方发送的数据。发送给FIN结束报文给对方,对方收到后,回复ACK报文。当这方也已经写完了准备关闭,发送FIN报文,对方回复ACK。两端都关闭,TCP连接正常关闭。

## 为什么需要TIME-WAIT状态?

TIME\_WAIT状态存在的原因有两点

- 1、可靠的终止TCP连接。
- 2、保证让迟来的TCP报文有足够的时间被识别并丢弃。

根据前面的四次握手的描述,我们知道,客户端收到服务器的连接释放的FIN报文后,必须发出确认。如最后这个ACK确认报文丢失,那么服务器没有收到这个ACK确认报文,就要重发FIN连接释放报文,客户端要在某个状态等待这个FIN连接释放报文段然后回复确认报文段,这样才能可靠的终止TCP连接。

在Linux系统上,一个TCP端口不能被同时打开多次,当一个TCP连接处于TIME\_WAIT状态时,我们无法使用该链接的端口来建立一个新连接。反过来思考,如果不存在TIME\_WAIT状态,则应用程序能过立即建立一个和刚关闭的连接相似的连接(这里的相似,是指他们具有相同的IP地址和端口号)。这个新的、和原来相似的连接被称为原来连接的化身。新的化身可能受到属于原来连接携带应用程序数据的TCP报文段(迟到的报文段),这显然是不该发生的。这是TIME\_WAIT状态存在的第二个原因。

## MYSQL数据库大量TIME\_WAIT的解决方法

## Mysql参数调整

```
mysql> show variables like "time_timeout";
```

`wait_timeout`针对非交互式连接，一般来说通过mysql客户端连接数据库是交互式连接，通过jdbc连接数据库是非交互式连接，这个值代表服务器关闭非交互连接之前等待活动的秒数（也即是控制连接最大空闲时长），默认值:28800，单位秒，即8个小时。调整这个参数。

## 调整内核参数

```
vi /etc/sysctl.conf
```

编辑文件，加入以下内容：

```
net.ipv4.tcp_syncookies = 1
```

```
net.ipv4.tcp_tw_reuse = 1
```

```
net.ipv4.tcp_tw_recycle = 1
```

```
net.ipv4.tcp_fin_timeout = 30
```

然后执行 `/sbin/sysctl -p` 让参数生效。

`net.ipv4.tcp_syncookies = 1` 表示开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击，默认为0，表示关闭；

`net.ipv4.tcp_tw_reuse = 1` 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；

`net.ipv4.tcp_tw_recycle = 1` 表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。

```
net.ipv4.tcp_fin_timeout 修改系统默认的 TIMEOUT 时间
```

## 可能产生的原因

连接Mysql的代码使用短连接来完成，用完后系统自动回收资源。由于访问量巨大，所以产生了很多连接。

另外，程序代码中没有使用`mysql.close()`，导致MySQL空闲判断机制生效

# 实战观察TCP报文

## WireShark

### 为什么要抓包

1、定位网络问题；

大部分场合都可以通过程序调试来定位问题，但有些场景使用抓包来定位接口问题更准确、更方便，如以下场景：

a、你发送数据给后台，但后台没有收到，可以对接口进行抓包分析，看是后台处理有问题，还是没有将数据发出去，或是发送数据格式有误；

b、你和后台接口联调测通，但业务数据对不上，你认为是后台问题，后台认为是你发的问题，可以抓包确认问题所在；

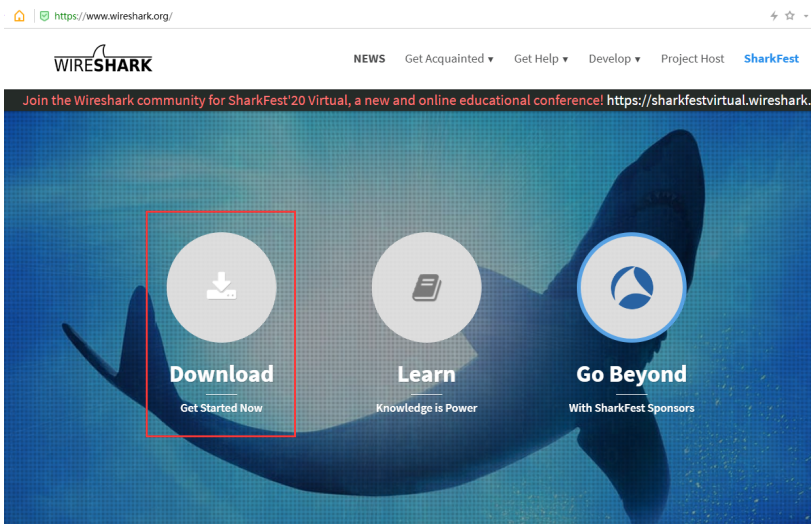
c、线上出现bug需要定位，但你没在公司，没有代码可调试，可直接抓包分析；

d、系统性能不佳，抓包看下接口响应时长，是不是后台出现性能问题。

2、学习网络协议，使用抓包工具分析网络数据更直观。

常用的抓包工具有：F12（浏览器自带的抓包工具）、Fiddler、Charles、Wireshark。而Wireshark在支持的协议，用户友好度、价格（开源）、程序支持、支持的操作系统上都很好，所以Wireshark也是我们最常用的抓包工具和报文分析工具。

## 下载与安装



在<https://www.wireshark.org/>的官方主页点击进入下载页面，选择适合自己的版本下载即可

## 实战：用WireShark看看TCP的三次握手

### 准备

现在我们用平时比较常见的连接mysql服务器来看看，打开一个mysql的客户端，准备连接IP地址为47.112.44.148的云端mysql。

于是我们设定捕获过滤器，只捕获与IP地址47.112.44.148相关的数据包：

host 47.112.44.148

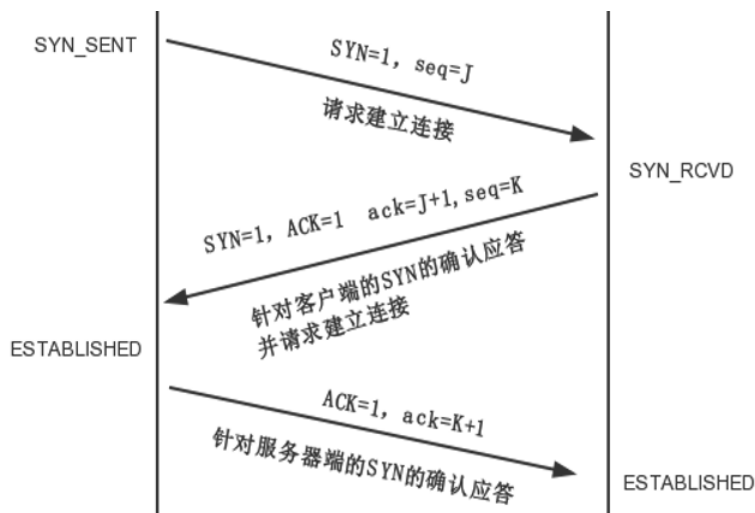
开始捕获，连接数据库：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.154	47.112.44.148	TCP	66	55940 → 3306 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.028876	47.112.44.148	192.168.0.154	TCP	66	3306 → 55940 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1452 SACK_PERM=1 WS=128
3	0.028979	192.168.0.154	47.112.44.148	TCP	54	55940 → 3306 [ACK] Seq=1 Ack=1 Win=132096 Len=0
4	0.050942	47.112.44.148	192.168.0.154	MySQL	136	Server Greeting proto=10 version=5.6.10-log
5	0.051182	192.168.0.154	47.112.44.148	MySQL	238	Login Request user=orderuser
6	0.072580	47.112.44.148	192.168.0.154	TCP	60	3306 → 55940 [ACK] Seq=83 Ack=185 Win=30336 Len=0
7	0.072580	47.112.44.148	192.168.0.154	MySQL	65	Response OK
8	0.072838	192.168.0.154	47.112.44.148	MySQL	76	Request Query
9	0.094361	47.112.44.148	192.168.0.154	MySQL	65	Response OK
10	0.099057	192.168.0.154	47.112.44.148	MySQL	217	Request Query
11	0.120976	47.112.44.148	192.168.0.154	MySQL	306	Response
12	0.121325	47.112.44.148	192.168.0.154	MySQL	302	Response
13	0.121325	47.112.44.148	192.168.0.154	MySQL	120	Response
14	0.121355	192.168.0.154	47.112.44.148	TCP	54	55940 → 3306 [ACK] Seq=370 Ack=671 Win=131328 Len=0
15	0.123777	192.168.0.154	47.112.44.148	MySQL	162	Request Query
16	0.146024	47.112.44.148	192.168.0.154	MySQL	558	Response
17	0.186708	192.168.0.154	47.112.44.148	TCP	54	55940 → 3306 [ACK] Seq=478 Ack=1175 Win=130816 Len=0

很明显，WireShark捕获到了本机所有和 47.112.44.148通信的报文，现在我们只需要观察TCP的三次握手和四次分手的报文，所以，我们在显示过滤器中写上tcp and !mysql

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.154	47.112.44.148	TCP	66	55940 → 3306 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=25
2	0.028876	47.112.44.148	192.168.0.154	TCP	66	3306 → 55940 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=
3	0.028979	192.168.0.154	47.112.44.148	TCP	54	55940 → 3306 [ACK] Seq=1 Ack=1 Win=132096 Len=0
6	0.072580	47.112.44.148	192.168.0.154	TCP	60	3306 → 55940 [ACK] Seq=83 Ack=185 Win=30336 Len=0
14	0.121355	192.168.0.154	47.112.44.148	TCP	54	55940 → 3306 [ACK] Seq=370 Ack=671 Win=131328 Len=0
17	0.186708	192.168.0.154	47.112.44.148	TCP	54	55940 → 3306 [ACK] Seq=478 Ack=1175 Win=130816 Len=0

按照tcp三次握手的规则，三次握手包含一个SYN 包、 一个SYN/ACK 包和 一个ACK包。



就是窗口里最上面的三条记录：

1	0.000000	192.168.0.154	47.112.44.148	TCP	66	55940 → 3306 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
2	0.028876	47.112.44.148	192.168.0.154	TCP	66	3306 → 55940 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0
3	0.028979	192.168.0.154	47.112.44.148	TCP	54	55940 → 3306 [ACK] Seq=1 Ack=1 Win=132096 Len=0

我们一条条看过来，

## 第一次握手

Ethernet II, Src: Private_35:a2:be (b0:25:aa:35:a2:be), Dst: Tp-LinkT_6f:43:9a (9c:a6:15:6f:43:9a)
Destination: Tp-LinkT_6f:43:9a (9c:a6:15:6f:43:9a)
Address: Tp-LinkT_6f:43:9a (9c:a6:15:6f:43:9a)
....0. .... = LG bit: Globally unique address (factory default)
....0. .... = IG bit: Individual address (unicast)
Source: Private_35:a2:be (b0:25:aa:35:a2:be)
Address: Private_35:a2:be (b0:25:aa:35:a2:be)
....0. .... = LG bit: Globally unique address (factory default)
....0. .... = IG bit: Individual address (unicast)
Type: IPv4 (0x0800)




从这里我们可以看出，这是数据链路层相关的信息，source部分的地址和我们机器上的mac地址一模一样

以太网适配器 以太网:
连接特定的 DNS 后缀 . . . . . :
描述. . . . . :
物理地址. . . . . :
DHCP 已启用 . . . . . :
自动配置已启用. . . . . :
本地链接 IPv6 地址. . . . . :
IPv4 地址. . . . . :

接下来，就是IP层的相关信息，其中表明了它的上一层协议是TCP，同时本地和远程服务器的IP地址：







Internet Protocol Version 4, Src: 192.168.0.154, Dst: 47.112.44.148

0100 .... = Version: 4  
.... 0101 = Header Length: 20 bytes (5)  
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
Total Length: 52  
Identification: 0x0622 (1570)  
> Flags: 0x4000, Don't fragment  
Fragment offset: 0  
Time to live: 128  
Protocol: TCP (6)   
Header checksum: 0x0000 [validation disabled]  
[Header checksum status: Unverified]  
Source: 192.168.0.154   
Destination: 47.112.44.148 

接下来，就是TCP层的相关信息，其中包括了本地端口和远程服务端口，既然是syn包，里面当然会带上seq值，本次通信是1979849485，tcp报文格式中的syn字段被设置为1，用来表明这是一个syn包。

Transmission Control Protocol, Src Port: 55940, Dst Port: 3306, Seq: 0, Len: 0

Source Port: 55940   
Destination Port: 3306   
[Stream index: 0]  
[TCP Segment Len: 0]  
Sequence number: 0 (relative sequence number)  
Sequence number (raw): 1979849485   
[Next sequence number: 1 (relative sequence number)]  
Acknowledgment number: 0  
Acknowledgment number (raw): 0  
1000 .... = Header Length: 32 bytes (8)  
v Flags: 0x002 (SYN)  
000. .... = Reserved: Not set  
...0 .... = Nonce: Not set  
... 0... = Congestion Window Reduced (CWR): Not set  
... .0.. = ECN-Echo: Not set  
... ..0. = Urgent: Not set  
... ...0 = Acknowledgment: Not set  
... ....0... = Push: Not set  
... .....0.. = Reset: Not set  
v .... ....1. = Syn: Set 

## 第二次握手

数据链路层和IP层的报文我们不再查阅，直接看TCP层的报文，很明显，这是服务器给客户端的ACK报文，其中依然包括了远程服务端口和本地端口，同时服务器要把自己端的seq值告诉客户端，我们看到实际值是3366556883。同时服务器要把客户端传给自己的seq值做个应答确认，所以我们看见Acknowledgment number 字段值是 1979849486，刚好是第一次握手中，客户端传递给服务端的seq值1979849485加1。同时tcp报文格式中的syn字段被设置为1，Acknowledgment字段被设置为1，用来表明这是一个syn/ack包。

```

Transmission Control Protocol, Src Port: 3306, Dst Port: 55940, Seq: 0, Ack: 1, Len: 0
Source Port: 3306
Destination Port: 55940
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 0 (relative sequence number)
Sequence number (raw): 3366556883
[Next sequence number: 1 (relative sequence number)]
Acknowledgment number: 1 (relative ack number)
Acknowledgment number (raw): 1979849486
1000 .... = Header Length: 32 bytes (8)
Flags: 0x012 (SYN, ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... .... 0... = Push: Not set
.... ..... 0.. = Reset: Not set
.... .... .1. = Syn: Set
> [Expert Info (Chat/Sequence): Connection establish acknowledge (SYN+ACK): server port 3306]
.... .... ...0 = Fin: Not set
[TCP Flags: .....A..S.]

```

### 第三次握手

这次握手的的报文分析，通过前两次的分析，相信同学们能够自行分析出来，这里就不在赘述。

```

Transmission Control Protocol, Src Port: 55940, Dst Port: 3306, Seq: 1, Ack: 1, Len: 0
Source Port: 55940
Destination Port: 3306
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 1 (relative sequence number)
Sequence number (raw): 1979849486
[Next sequence number: 1 (relative sequence number)]
Acknowledgment number: 1 (relative ack number)
Acknowledgment number (raw): 3366556884
0101 .... = Header Length: 20 bytes (5)
Flags: 0x010 (ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... .... 0... = Push: Not set
.... ..... 0.. = Reset: Not set
.... .... .0. = Syn: Not set
.... .... ...0 = Fin: Not set
[TCP Flags: .....A....]

```

## UDP和UDT、QUIC

### UDP概述

我们已经知道UDP（User Datagram Protocol的简称，中文名是用户数据报协议）是把数据直接发出去，而不管对方是不是在接收，也不管对方是否能接收的了，也不需要接收方确认，属于不可靠的传输，可能会出现丢包现象，实际应用中要求程序员编程验证。

### UDP单播和广播

单播的传输模式，定义为发送消息给一个由唯一的地址所标识的单一的网络目的地。面向连接的协议和无连接协议都支持这种模式。

由于通讯不需要连接，所以可以实现广播发送，所谓广播——传输到网络（或者子网）上的所有主机。

UDP因为没有TCP等一系列复杂机制，所以使用也非常广泛，使用UDP的服务包括NTP（网络时间协议）和DNS（DNS也使用TCP），包总量较少的通信（DNS、SNMP等）；2. 视频、音频等多媒体通信（即时通信）；3. 限定于 LAN 等特定网络中的应用通信；4. DHCP等协议就利用了UDP的广播功能。

常用的QQ，就是一个以UDP为主，TCP为辅的通讯协议。

## UDT

基于UDP的数据传输协议（UDP-based Data Transfer Protocol，简称UDT）是一种互联网数据传输协议。UDT的主要目的是支持高速广域网上的海量数据传输，最典型的例子就是建立在光纤广域网上的网格计算，一些研究所在这样的网络上运行他们的分布式的数据密集程式，例如，远程访问仪器、分布式数据挖掘和高分辨率的多媒体流。

而互联网上的标准数据传输协议TCP在高带宽长距离网络上性能很差。顾名思义，UDT建于UDP之上，并引入新的拥塞控制和数据可靠性控制机制。UDT是面向连接的双向的应用层协议。

UDT的特性主要包括在以下几个方面：

基于UDP的应用层协议：有基本网络知识的朋友都知道TCP和UDP的区别和使用场景，但是有没有一种协议能同时兼顾TCP协议的安全可靠和UDP协议的高效，那么UDT就是一种。

面向连接的协议：面向连接意味着两个使用协议的应用在彼此交换数据之前必须先建立一个连接，当然UDT是逻辑上存在的连接通道。这种连接的维护是基于握手、Keep-alive（保活）以及关闭连接。

可靠的协议：依靠包序号机制、接收者的ACK响应和丢包报告、ACK序号机制、重传机制（基于丢包报告和超时处理）来实现数据传输的可靠性。

双工的协议：每个UDT实例包含发送端和接收端的信息。

新的拥塞算法，并且具有可扩展的拥塞控制框架：新的拥塞控制算法不同于基于窗口的TCP拥塞控制算法（慢启动和拥塞避免），是混合的基于窗口的、基于速率的拥塞控制算法。可扩展的拥塞控制框架开源的代码和拥塞控制的C++类架构，可支持开发者派生专用的拥塞控制算法。

带宽估计：UDT使用对包（PP -- Packet pair）的机制来估计带宽值。即每16个包为一组，最后一个是对包，即发送方不用等到下一个发送周期内再发送。接收方接收到对包后对其到达时间进行记录，可结合上次记录的值计算出链路的带宽（计算的方法称为中值过滤法），并在下次ACK中进行反馈。

## QUIC

QUIC代表”快速UDP Internet连接”，基于UDP的传输层协议，它本身就是Google尝试将TCP协议重写为一种结合了HTTP/2、TCP、UDP和TLS（用于加密）等多种技术的改进技术。

谷歌希望QUIC通信技术逐渐取代TCP和UDP，作为在Internet上移动二进制数据的新选择协议，QUIC协议的主要目的，是为了整合 TCP 协议的可靠性和 UDP 协议的速度和效率。

由于 TCP 是在操作系统内核和中间件固件中实现的，因此对 TCP 进行重大更改几乎是不可能的（TCP 协议栈通常由操作系统实现，如 Linux、Windows 内核或者其他移动设备操作系统。修改 TCP 协议是一项浩大的工程，因为每种设备、系统的实现都需要更新）。但是，由于 QUIC 建立在 UDP 之上，因此没有这种限制。

QUIC 的优势在于：

1、采用多路复用 思想，一个连接可以同时承载多个 流（stream），同时发起多个请求。请求间完全独立，某个请求阻塞甚至报文出错均不影响其他请求。

2、QUIC只需要1RTT（Round-Trip Time）的延迟就可以建立可靠安全的连接,相对于TCP+TLS的3次RTT要更加快捷。之后客户端可以在本地缓存加密的认证信息，再次与服务器建立连接时可以实现0-RTT的连接建立延迟。

3、TCP 采用 重传 机制，而 QUIC 采用 纠错 机制。

TCP 发生丢包时，需要一个等待延时判断发生了丢包，然后再启动重传机制，这个过程会造成一定的阻塞，影响传输时间。

而 QUIC 则采用一种更主动的方案，有点类似 RAID5，每 n 个包额外发一个 校验和包。如果这 n 个包中丢了一个包，可以通过其他包和校验和恢复出来，完全不需要重传。

4、QUIC 直接基于客户端(应用进程)实现，而非基于内核，可以快速迭代更新，不需要操作系统层面的改造，部署灵活。

5、连接保持

QUIC 在客户端保存连接标识，当客户端 IP 或者端口发生变化时，可以快速恢复连接——客户端以标识请求服务端，服务端验证标识后感知客户端新地址端口并重新关联，继续通讯。这对于改善移动端应用连接体验意义重大(从 WiFi 切换到流量)。

本文档共享地址：

<https://note.youdao.com/s/Z3GNyDLe>