**周志垒，从网络到分布式（集群内高并发）、、、**



**+++++++++++++++++++高并发负载均衡：网络协议原理 +++++++++++++++++++++++++++++++**

# 一、前言

未来几天讲的是：高并发，负载均衡，高可用。

其实这个知识点适用于架构师也适用于大数据，为什么要聊这个事情，不要因为技术而技术。

不要因为有这个技术了就要研究这个技术，我觉得这个技术怎么怎么样。你更应该弄明白为什么要有这个技术，以及这个技术是什么样的原理，我怎么样去使用它。这样才是一个正确饿的学习方式。

中国网民基数大，互联网项目带来高并发问题。

高并发怎么去实现？网上对高并发的解释一抓一大把，怎么学？一切从最基本的基础讲起，什么是网络。因为在解决负载均衡、高并发的时候，其实有多种解决方案，最常规的有基于四层的 ，和基于七层的解决方案。Nginx是属于几层？ 最终自己要知道，4层里面可以做什么事情，7层里面可以做什么事情。

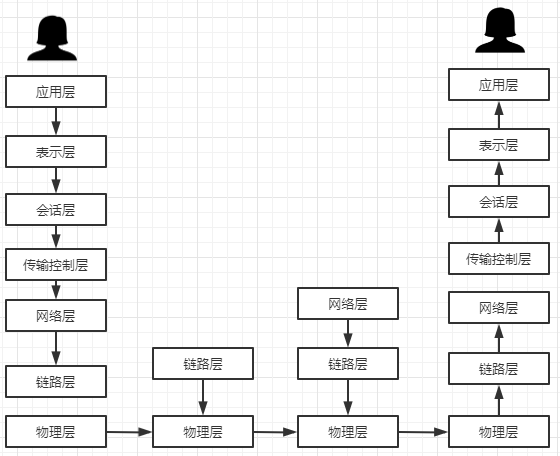
4层 7层，描述的是网络的术语。

# OSI七层参考模型、TCP/IP协议

## OSI七层参考模型

“参考模型”这个词，意思就是没有给出具体的层的定义，分了7个层，干七个事。7个层串起来就是个通信的过程。

OSI七层参考模型，哪七层呢？回归本质，本质是通信。大白话就是俩人一人拿一电脑，在这通信，通信可以写一软件从头到尾写完。你做的是软件工程学，是个工程，是有一定学问的，最重要的是 “分层解耦”



分层解耦，接口不变，每一层实现可以变



## TCP/IP协议

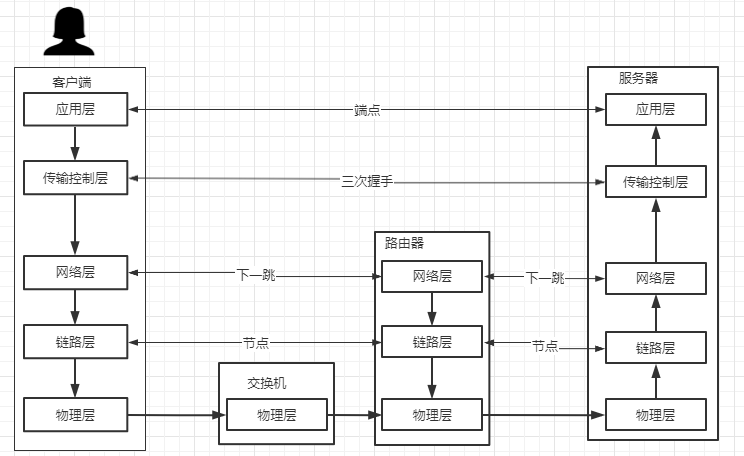
tcp/ip五层或tcp/ip四层

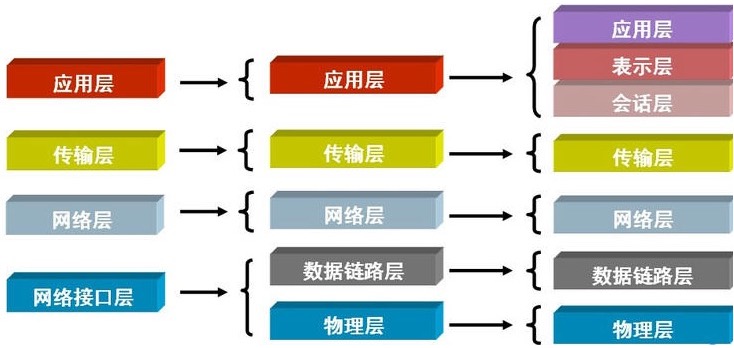
### 1定义：

相对于的参考模型，TCP/IP协议，协议：明确定义了双方怎么具体去实现。协议相对于参考模型就是一个具体的方案，具体的定义。所以接下来讨论的是TCP/IP精简完的4层或5层。

每一层都有自己的协议。

### 2图示





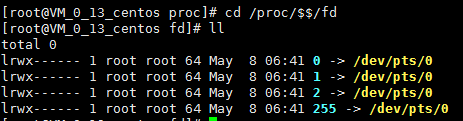
真正的网络这一块是从传输控制层往下走的（不带应用层），每一层有具体的协议。

应用层的协议有：http协议，ssh协议，smtp协议，ftp协议等。

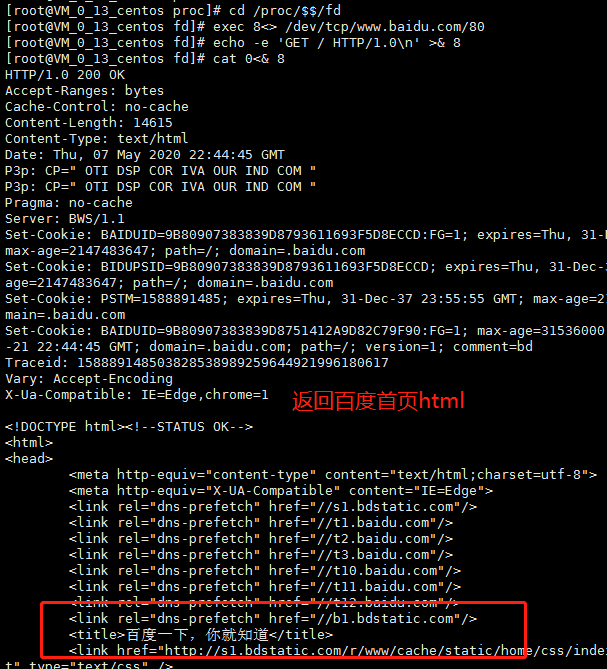
### 3实验演示http协议

实验：通过linux命令行模拟应用层的软件（浏览器）演示http协议







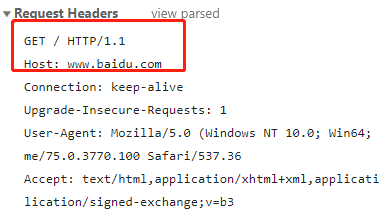


第一步：建立连接

第二步：传输数据（http协议：规范标准）

最终展现的是一个应用层协议

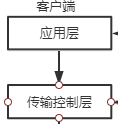
浏览器访问百度：http请求



http响应：



协议规范完的是数据，浏览器准备好数据最终是要发给对方的，不是浏览器把数据发出去的，浏览器只需要按http协议把数据规范好，OSI模型是分层的，接下来传输控制不归我管，浏览器（应用层）不管，它接着往下调用。



### 4应用层（http协议）

【应用层】访问百度过程：（命令行或者浏览器为例）

1. 浏览器先调用【传输控制层】，告诉他我想和百度传输数据
2. 传输控制层和百度建立连接（Socket）。然后我把请求头给百度，你在帮我传过去。对方再给我输送回来，我在渲染图形。

备注：访问百度之前，需要先和百度建立了一个Socket连接（就行java一样，第一步先new 一个Socket，拿到一个IO，才能把你的数据发出去）

每一层都有协议，在传输控制层就会涉及到Socket等概念了。

#### 用户态内核态解释

|  |
| --- |
| 为什么叫应用层：  这和计算机的一个术语有关系，计算机内存中分为两个区域：内核区-内核态，用户空间-用户态。浏览器、tomcat都是应用程序，传输控制层往下，都是内核的东西。  一个计算机为什么要有内核： 比方说你放着直播，登着QQ，很多程序都需要联网，这么多软件要是都控制网卡一定会控制乱。这些软件不需要关注怎么去控制网卡，怎么去传输数据包，他们只需要调用一下内核，内核里面有一套传输的标准，轮流会让他们把数据发出去或接收回来。  浏览器、Tomcat都是应用程序，都是应用层。传输控制层、网络层往下都是操作系统内核的东西了。  一个JVM说想建立连接的话，调用JVM所谓的Socket，new了一个对象，socket底层一定是native的，最终JVM是掉了操作系统内核的系统调用，等着操作系统跑完从  传输控制层----》网络层----》链路层-----》物理层 ---》链路层-----》网络层-----》控制层，这几层之后，建立连接之后，返回连接建立成功。之后才能走下一步代码，代码在new Socket这一步会出现小小的阻塞。 |

### 5 传输控制层

传输控制层是和某一个具体的应用无关的。

你的qq可以调传输控制层，你的远程登录可以调传输控制层，什么都可以调传输控制层。

每一层都有自己的协议，应用层是http协议，传输控制层必须知道的是

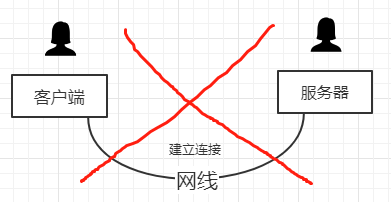
#### TCP协议

是一种面向连接的、可靠的、基于字节流的传输层通信协议

##### TCP的面向连接

###### 错误理解：

一个计算机想使用TCP去访问其它节点的时候，不是说你这边拿着一根网线Duang 顶上，Duang插到那边，给你一个屋里的连接么？不是这样的。



###### 正确理解：

有一首歌很火，确认过眼神 你是我想要的人。什么叫确认过眼神？有一美女在大马路上走着呢，你怎么知道她对你有感觉，你就能追到手呢？ ，一定是 你看她了一眼，她回看了你一眼，俩人眼睛一放电，她心里也这么想的。总之，**面向连接就是**：我发一个，你给我确认一个，我发你一个你给我确认一个。这个确认的过程，保证他它的连接是可靠的传输方式。

这只是说了什么是“面向连接”，但是还没设计到连接的概念，这时候再引入“三次握手”，因为“三次握手”里面要用到这个确认的过程。

##### TCP的连接

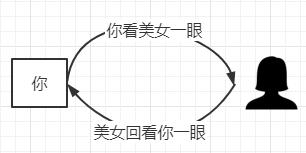
先不看七层模型中的其他层，只看传输控制层，这个传输控制，只讲TCP，如何制造出一个所谓的连接？这时候他有一个概念 三次握手 ，四次分手，以及加上他的确认机制。



###### TCP的三次握手：

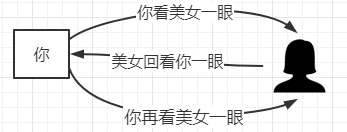
为什么需要三次握手才能建立一个连接？

比如说上层应用层 调用传输控制层，要你跟百度建立一连接，传输控制层创建一个数据包 sync，就是想握手，也就是你看了美女一眼，看完之后，美女喯看了你一眼。按理说这时候双方已经确认了，可以建立通信所谓的连接了。



###### 为什么是三次的？

你需要再回看美女一眼。



**站在方向层面：**

站在客户端一边，我给你发送想握手，你给我回SYNC+ACK确认，这个回送是一个确认，服务端确认你发的东西了。两次握手时，站在客户端，其实已经确认我的output和input都是OK的。



站在服务端，只能确定的是，接收到了客户端来握手的请求，也给客户端回了一个确认，但是回的这个确认，是不知道客户端是否接收到了的。所以需要客户端再回一个确认。

**站在IO层面：**

三趟数据包走完之后，双方才会在内存开辟线程，开辟对象，开辟所有的描述符，建立相应的资源。如果三次没完成，这些资源是不会开辟的。三次握手会有资源的消耗。

|  |
| --- |
| 在BIO阻塞IO里，Server这一端要准备一个SocketServer，SocketServer准备好后，是listen状态，然后while循环，当有客户端跟你建立连接的时候，你需要抛出一个线程，这个线程专门解决这个连接，又来一个线程后，再开辟一个线程来处理。 |

###### 疑问：怎么确认第三次送达了？

这个事别较真，第一次握手你发出去包，接收到对方回应，说明你的input，output没问题了。所以只要你发了，就会默认认为对方一定会受到，所以一定会开辟资源。如果非得死较真的话这个事没玩了。

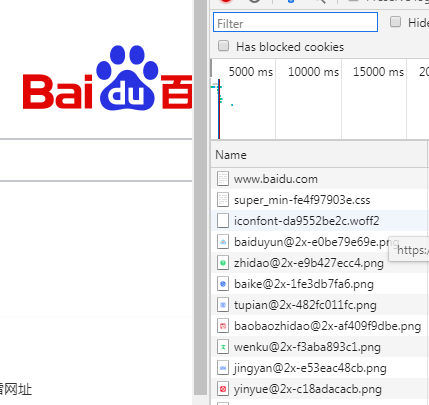
###### 注意：

三次握手只是在建立连接，中间的包的来回发送，是没有应用层数据的。只有三次握手建立好之后，网络传输层反馈给应用层，应用层开辟资源、准备线程、准备对象之后，两边的tomcat（或者JVM）从连接中拿到IO了，应用层才会把htpp请求的字符串，通过IO往外写。这时候通过了三次握手之后，才是发送数据。且TCP协议是发送一个就会给一个确认，通过确认机制保证传输的可靠。

###### 四次分手

传输完了后最终一定会有个四次分手，三次握手之后为什么要有一个分手？？这就要说道Socket套接字，套接字里面有一个最基本的资源：端口号，一台操作系统的端口号数量是有限的（65535个）。这是一个不大的数字，随便打开一个网页，后边连接数多了去了，建立了很多连接。客户端在操作系统上随机申请了一个端口号，与服务器端建立连接。申请的端口号，用完之后还给操作系统，只借不还，很快65535就会消耗殆尽。新的请求就建立不起来了，所以开辟的资源，用完后要及时关闭，回收资源。

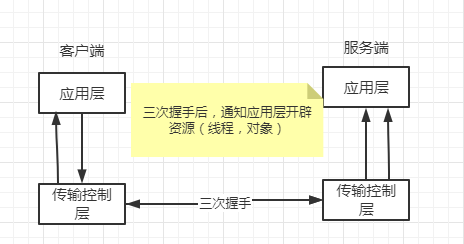
所以一定要有四次分手的过程，不管是你手工的，还是操作系统硬性的回收，最终分手的事一定会发生。



|  |
| --- |
| **概念：**  套接字（socket）是一个抽象层，应用程序可以通过它发送或接收数据，可对其进行像对文件一样的打开、读写和关闭等操作。套接字允许应用程序将I/O插入到网络中，并与网络中的其他应用程序进行通信。网络套接字是IP地址与端口的组合。  **简述：**  [传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82/4329536" \t "https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%AD%97/_blank)实现[端到端](https://baike.baidu.com/item/%E7%AB%AF%E5%88%B0%E7%AB%AF/8851783" \t "https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%AD%97/_blank)的通信，因此，每一个传输层连接有两个端点。那么，传输层连接的端点是什么呢？不是[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA/455151" \t "https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%AD%97/_blank)，不是主机的[IP地址](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80/150859" \t "https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%AD%97/_blank)，不是应用进程，也不是传输层的协议端口。传输层连接的端点叫做套接字（socket）。根据RFC793的定义：[端口号](https://baike.baidu.com/item/%E7%AB%AF%E5%8F%A3%E5%8F%B7/10883658" \t "https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%AD%97/_blank)拼接到IP地址就构成了套接字。所谓套接字，实际上是一个通信端点，每个套接字都有一个套接字序号，包括主机的IP地址与一个16位的主机端口号，即形如（主机IP地址：端口号）。例如，如果IP地址是210.37.145.1，而端口号是23，那么得到套接字就是（210.37.145.1：23）。  总之，套接字Socket=（IP地址：端口号），套接字的表示方法是点分十进制的IP地址后面写上端口号，中间用冒号或逗号隔开。每一个传输层连接唯一地被通信两端的两个端点（即两个套接字）所确定。  套接字可以看成是两个网络应用程序进行通信时，各自通信连接中的一个端点。通信时，其中的一个网络应用程序将要传输的一段信息写入它所在主机的Socket中，该Socket通过网络接口卡的传输介质将这段信息发送给另一台主机的Socket中，使这段信息能传送到其他程序中。因此，两个应用程序之间的数据传输要通过套接字来完成。  在网络应用程序设计时，由于TCP/IP的核心内容被封装在操作系统中，如果应用程序要使用TCP/IP，可以通过系统提供的TCP/IP的编程接口来实现。在Windows环境下，网络应用程序编程接口称作Windows Socket。为了支持用户开发面向应用的通信程序，大部分系统都提供了一组基于TCP或者UDP的应用程序编程接口（API），该接口通常以一组函数的形式出现，也称为套接字（Socket）。 |

###### 分手为什么是四次？

三次握手的时候，一定会通知应用层开辟资源，对象、线程抛出去了。



此时如果客户端说，我请求百度主页请求完了，我要和你断开连接

此时客户端不会立即销毁线程，销毁所有的数据

1. 客户端会发送一个“我想断开连接”的数据包发送到服务端。
2. 服务端回复：确认了。

（由于TCP是可靠的传输机制，必须确认）

服务端有两种可能：

1. 要继续和客户端传输数据，你不能断开，你要继续接收数据，回收确认
2. 服务端也想断开连接。

3 ，此时服务端也发送“我也想断开连接”

4，客户端回送，我知道了

只有两端都发现了自己，以及对方都想断开连接这个行为之后，再随机等一个时间片，两者才会把内存资源销毁，谁也不会辜负谁。

总之，4次分手就是因为：TCP是面向连接的可靠的传输协议。

|  |
| --- |
| 结论：  **三次握手 ---------> 数据传输 --------> 四次分手**  **是最小粒度，不可被分割。**  **例： 负载均衡一个client两个server，三次握手不会说前两次在Server1，第三次发给了Server2** |

#### UDP协议

 参考模型中一种无连接的[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82" \t "https://baike.baidu.com/item/_blank)协议，提供面向事务的简单不可靠信息传送服务

#### 命令查看Socket

每一层都有协议，每一层都有表格

命令：netstat -natp

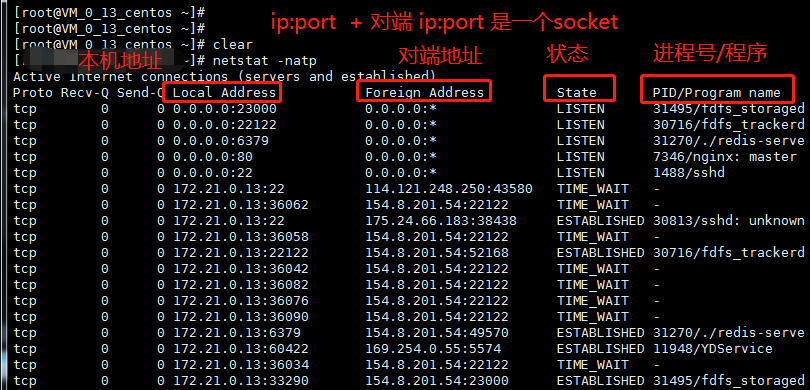
-natp

n:显示ip地址，不要显示逻辑名称

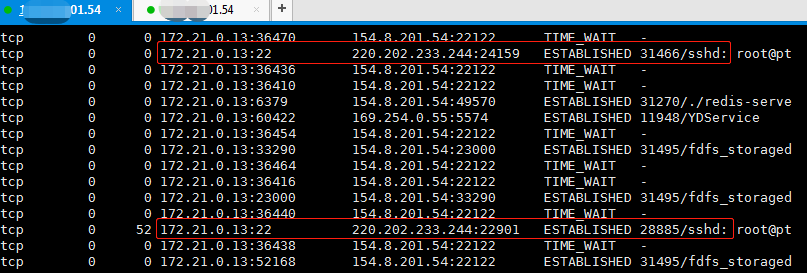
a: 所有

t: tcp

p:pid 显示通信是哪个进程号



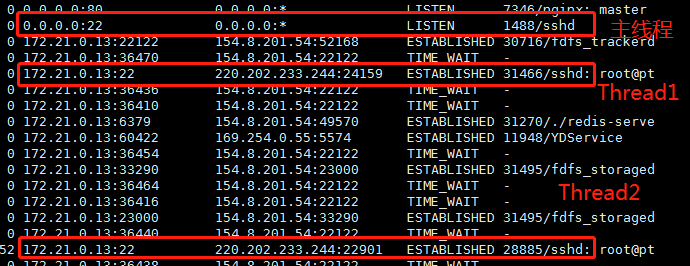
举例

下图中圈的两条数据意思是，Linux本机 172.21.0.13:22 端口，被远程Windows ip是220.202.233.244:24159 、220.202.233.244:22901两个随机端口连接着，协议是ssh协议，sshd表示后台进程。两条是xshell开了两个窗口。

Pid意思是主线程，

0.0.0.0.22 父进程，是listen状态，类似于java的主线程，主线程跑着的时候，有人建立连接的时候，会抛出一个线程去处理那个请求。

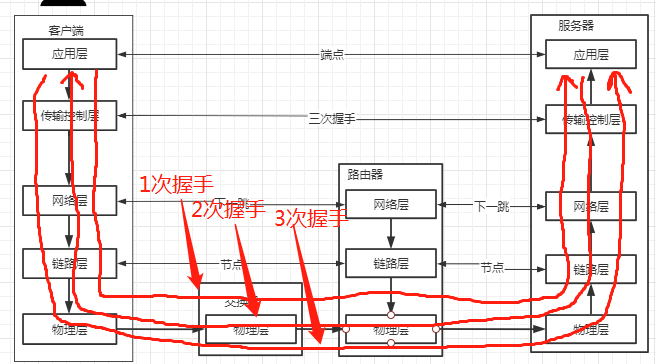
客户端IP：端口 + 服务端IP:端口，会确定一个socket套接字，一个独立的连接，他们各自传输的时候他们的IO是不互相影响的。



传输控制层在创建握手包，确认包，创建所谓的包，但是它依然得经过物理层链路层，才能把数据发出去。每个层只干他自己的事情，传输控制层只创建包。

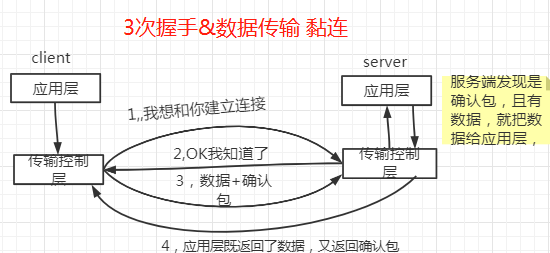
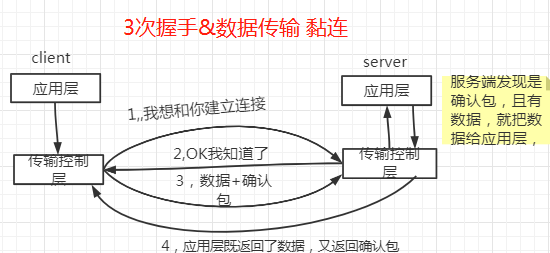
说的通俗点：

1. 【浏览器】给传输控制层说，我想和百度建立连接。
2. 【传输控制层】准备了握手包，就在这阻塞着，调用网路层。
3. 【网络层】把网络梳理清楚了，调用【链路层】知道“下一跳”去哪
4. 【链路层】调用【物理层】网卡，做二进制编码，最终握手的包发出去。再回来，再发出去，再回来。



#### 三次握手&数据传输 偷工减料

如果应用层有数据，三次握手和数据传输黏连，省包了。



除非应用层没数据，只是建立连接，就真是三次握手。

上边讲的是传输控制层，讲的是包的定义，包怎么才能发出去，并不是传输控制层来搞定，由网络层专门负责路由的事情。

### 6 网络层

#### 查看网络配置

cat /etc/sysconfig/network-scripts/ifcfg-eth0

ifcfg-eth0 : if interface 、 cfg config 、 eth 以太网，eth0第一块网卡



一般说到网络配置的时候有四个维度，

1. IP地址

IP地址又叫点分字节，.隔开的是一个字节，一个字节有个二进制8位，每个字节表示的数范围是0~255 ，IP地址理论地址是255.255.255.255

1. 掩码

一般是 255.255.255.0，IP地址与掩码会发生二进制的按位与，得出网络号。IP地址有两个信息，一个是国家省市街道，一个是门牌号。

如下图的

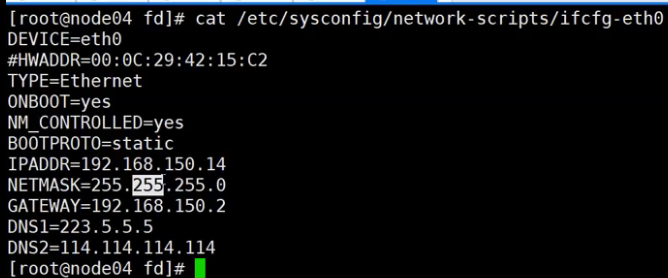
192.168.150.14

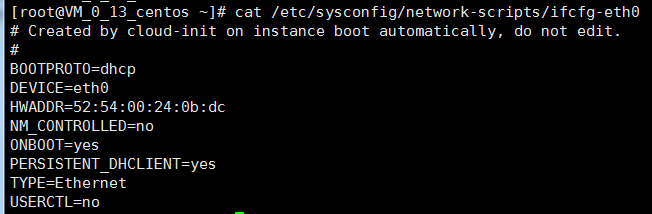
& 255.255.255.0

192.168.150.0

得出网络号192.168.150.0，192.168.150.14 是这个网络里的第14号。互联网就是很多个这样的小网络组合起来的，所以叫做互联网。

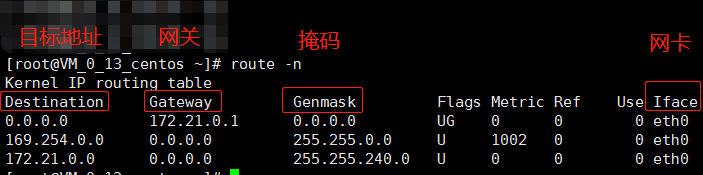
1. 网关
2. DNS域名解析





#### 网络层的表---路由表

路由表，动态生成的



#### TCP/IP协议IP层怎么定义通信的过程？

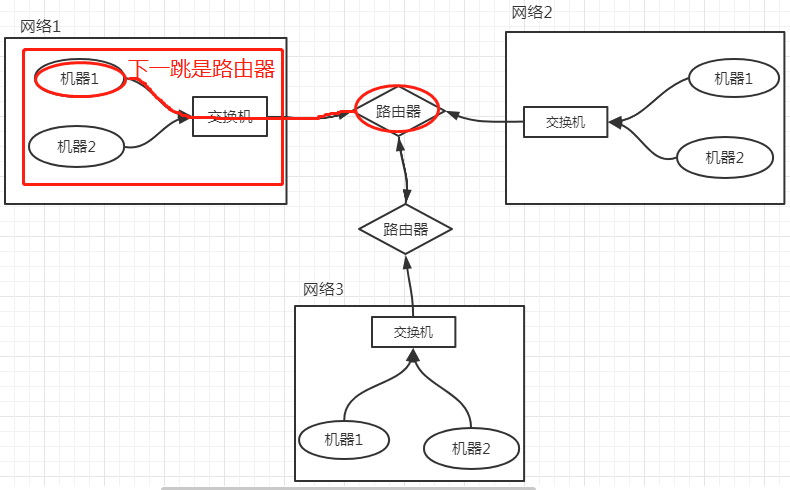
##### 方式一：保存所有设备的拓扑图

在互联网中的每一个设备的内存里，都存它和除了他自己的之外的，所有设备的连接方式以及拓普图，每一个设备中都有这么一份路径图，自己与其他设备怎么访问的 以及他们之间的链路。每个设备中都存着这么一个全局的拓普，如果他想访问其中的某一个点，只需要经过一个图计算，寻找一个最短路径即可。

弊端：互联网中的设备太多了，这么多路径进行最短路径计算速度慢。

##### 方式二：下一跳机制

采用下一跳的机制，不是让每个节点都存储完整的链路。



网络1里的机器1 想访问其他网络的其他机器

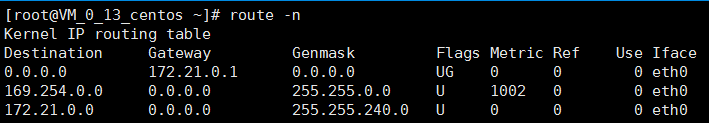
机器1 的只存本局域网的机器地址，机器1 就知道它的出路，他的网关，这个网络的边缘在哪里路由器1。而不知道除了这个网络之外的其他网络。

路由器如果是网络1的默认网关，机器1 就把数据包扔给路由器，

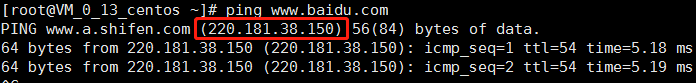
路由器也是存了自己最近的路由器的地址，然后把数据包扔给下一跳的设备。

###### 选择下一跳

路由表



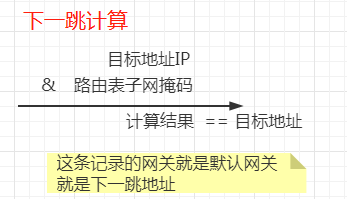
ping [www.baidu.com。](http://www.baidu.com。)



机器是怎么把数据包发给百度的呢？

可以看到百度的地址是 **220.181.8.150** 。拿目前的数据来看，谁也看不出来百度在哪，

但是，拿着百度的地址220.181.8.150 与你机器路由表的每一条记录的子网掩码做按位与运算，和路由表的目标地址进行匹配，如果得出的地址和gateway网关一样，这个网关就是默认网关，就是下一跳的地址。数据就会从这块【网卡】扔出去



**路由表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | Destination（目标地址） | Gateway（网关） | 掩码（Genmask） | Iface（网卡） |
| row1 | 0.0.0.0 | 172.21.0.1 | 0.0.0.0 | eth0 |
| row2 | 169.254.0.0 | 0.0.0.0 | 255.255.0.0 | eth0 |
| row3 | 172.21.0.0 | 0.0.0.0 | 255.255.240.0 | eth0 |

目的地址百度：

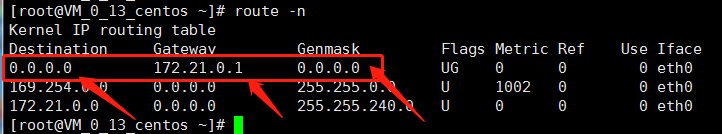
BaiduIP = **220.181.8.150 (二进制11011100.10110101.1000.10010110)**

row1：**220.181.8.150 & 0.0.0.0** == 目标地址0.0.0.0 【网关172.21.0.1下一跳】

Row2：**220.181.8.150 & 255.255.0.0** == 35.74.0.0 ≠ 目标地址169.254.0.0

Row3：**220.181.8.150** & 255.255.240.0 == 35.74.255.0 ≠ 目标地址172.21.0.0

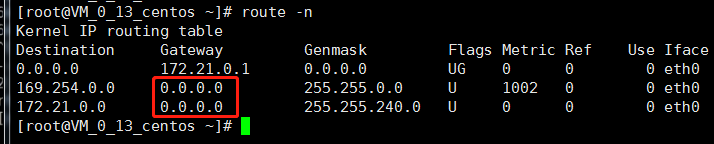
会发现，任何地址，只要和路由表里掩码 0.0.0.0做与运算，结果一定是0.0.0.0，一定和目标地址匹配上，则路由表里这一行的网关就是默认网关。就会把数据包扔给默认网关。



家里的默认网关不是0.1就是1.1（路由器），家里的任何数据包都扔到了路由器上。由路由器（如果设备具有转发能力）再到他的内部的路由表去计算，（路由器内部也有路由表，互联网中的任何设备都有路由表），最终找到合适的下一跳。

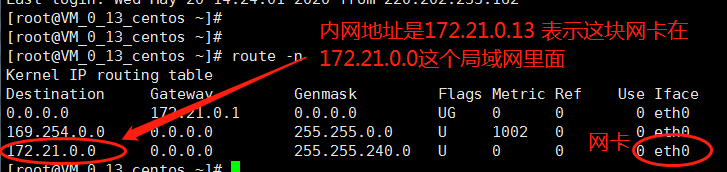
|  |
| --- |
| 网络工程师规划网络：骨干网或者城域网的路由器，都设置了很多带有下一跳的条目，就会判定什么数据发到这个网络，什么样的数据发到那个网络。 |

网关可以有多个，默认网关只有一个，只有网与网之间才需要下一跳转发一下，才需要路由器，一个局域网之间可以直接通信，通过交换机可以直接转发数据包，不需要走路由。（下图的网关是0.0.0.0就表示是一个局域网的机器，可以直接通信）



\*\*\*\*\*上图用的是腾讯云，内网地址是72.21.0.13\*\*\*\*

172.21.0.0 就表示，



gateway中看到4个0就表示不需要走路由，我这个设备（网卡）在destination这个局域网中具体的ip。

gateway中看到不是4个0，表示destination这个网我一定不是直连的。

有多个下一跳的时候，走的那个下一跳，是运营商干的事，网络工程师干的事、在路由器里加的有下一跳的条目，路由器之间会互相学习（开放路由协议），简单画出拓扑图。

思考：

如果你的数据包想从这台机器发出去，目标地址是写百度的220.181.8.150，还是写默认网关的172.21.0.1 ？

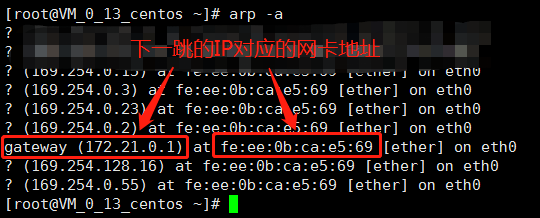
肯定是百度220.181.8.150，如果是写默认网关，则到了网关，网关一看，目标地址是自己，就不给百度转发了。但是目标地址是百度，那么数据包怎么能发给网关呢？？这不是网络层关注的问题，网络层就是找到下一跳。怎么解决？？外面再套一层，就有了链路层

### 7 链路层

网络层找到了下一跳地址172.21.0.1，但是自己要访问的目标地址是百度220.181.8.150，怎么才能找到下一跳172.21.0.1呢？，IP地址外面再套一层，写链路层的通信地址---网卡地址mac地址。

链路层也有一张表 用命令 arp -a 查看。

Arp是一个协议，DNS会解析域名和ip地址的映射，arp会解释ip地址和网卡硬件地址的映射。DNS是全网的逻辑域名---IP地址，arp是同一局域网内的，它会解析：同一局域网有哪些IP，这个IP在网卡硬件mac地址是什么，

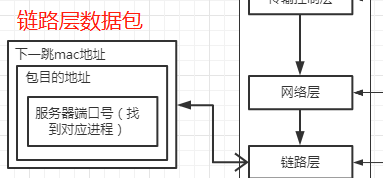


最终，链路层会封一个数据包，里面有三个地址：

1. 下一跳的mac地址
2. 数据包最终的目的地址
3. 服务器端的端口号，找到对应的进程

最终确定了从这个主机出去之后，数据包给了哪台设备，通过IP找到最终的端点，通过端口号找到给目标机器的哪个进程。

在经历下一跳到下一跳的传输过程中，I数据包的IP地址、端口号不会改变，下一跳的mac地址会变成下一个下一跳的mac地址，这样数据包在网络之间传输，最终找到目标地址，找到进程，把数据包发给他。



|  |
| --- |
| 结论：  TCP/IP协议是基于下一跳的机制，IP是端点间的，mac地址是节点间的。 |

#### 链路层数据传输

##### 环境：

###### 两个网络：

网络一192.168.1.0 ，网络二 192.168.3.0。

###### 路由器：

两个网口连接两个网络。连接网络一的：192.168.1.1。连接网络二的：192.168.3.1

###### 计算器：

计算机1的IP：192.168.1.4

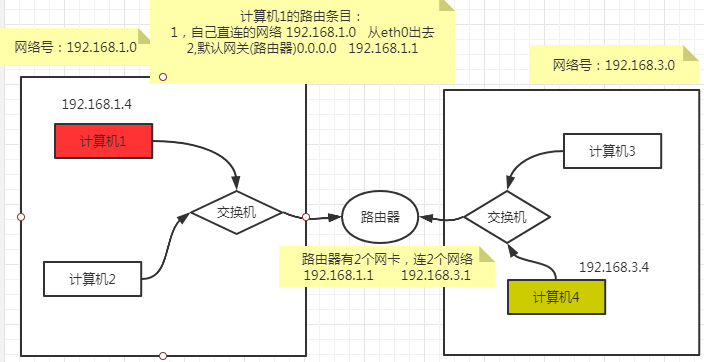
计算机1的路由表里路由条目：

1. 192.168.1.0 自己直连的网络，从eth0 出去
2. 192.168.1.1 0.0.0.0 默认网关，下一跳

计算机4的IP：192.168.3.4

###### 问题：

网络一里计算机1想要访问网络二计算机4

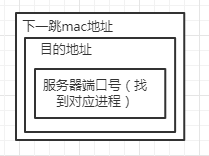


###### 步骤一 ARP协议

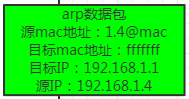
通过ARP协议，计算机1就知道了本网络内网关的mac地址。

下一跳地址是192.168.1.1 ，目标地址是192.168.2.4

数据包从计算机1发出，到了默认网关路由器



刚开机的时候 ，电脑内存里是没有链路层的arp表的（局域网的mac地址和IP关系），此时会先发一个arp的数据包：arp数据包比较特殊，目标mac地址全是ffffff，



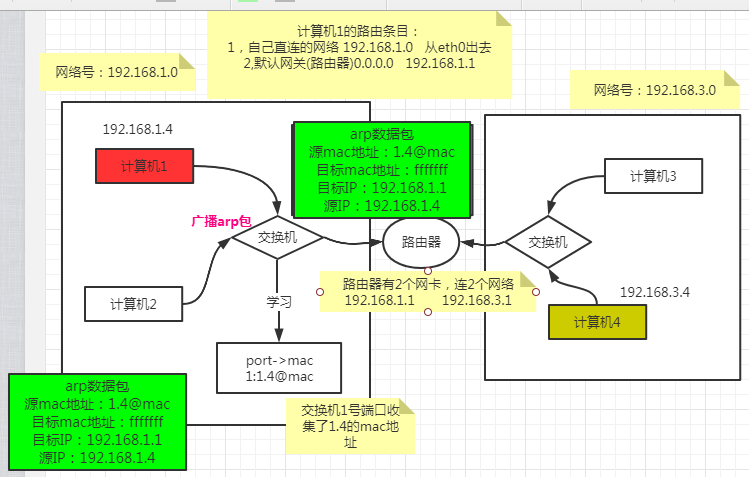
这个数据包到了交换机，交换机有个特殊的约束，看见目标mac地址全fffff的数据包，这个包会被广播，广播给除计算机1之外的其他机器，这里就会广播个路由器和计算机2。

同时，交换机会学习，交换机的1号端口网卡进了一个数据包，源mac地址是1.4的。

如果一台计算机收到一个数据包目标地址不是自己，会把这个数据包丢弃。

计算机2丢弃arp数据包

路由器响应数据包



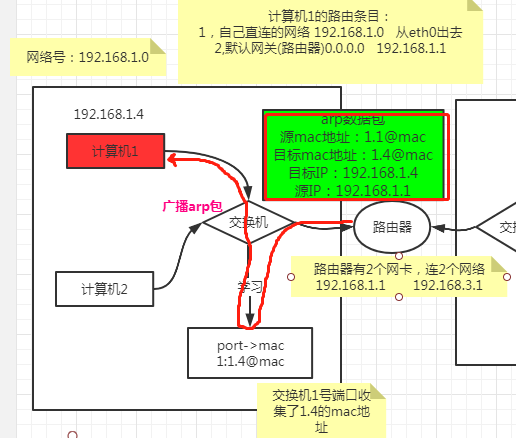
计算机约定，来的数据包是ARP，会把自己的mac地址的包给他发回去。

所以：

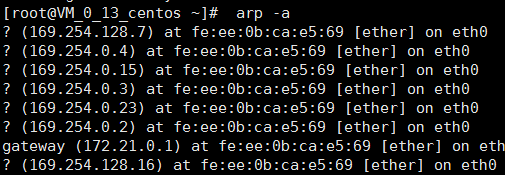
路由器再封一个数据包，目标地址是计算机1的mac地址。

数据包到了交换机，由于交换机之前已经学习过，交换机网卡1号端口进过1.4的数据包。有了这个记忆，交换机不会广播数据包，直接把返回的arp数据包通过1号端口转发给计算机1。

此时，计算机1就学了一条IP-----》mac地址 的映射记录。



链路层 arp缓存表



刚开机的时候，网卡一激活，网卡只要一通电，网卡就会往外通告自己是谁。你没有请求也会收到别人的通告。以减少通信时候ARP的过程。

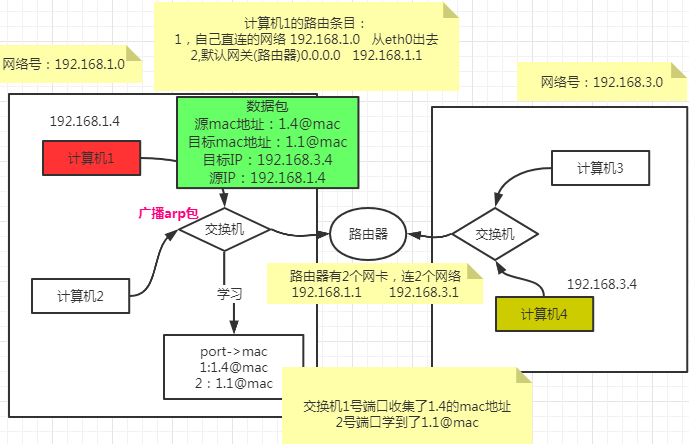
###### 步骤二 计算机1封真正数据包

源mac地址：计算机1的mac地址

目标mac地址：网关1.1的mac地址

目标IP：计算机4的IP地址192.168.3.4

源IP：计算机1的IP地址192.168.1.4



假设从路由器返回ARP数据包的时候，是从2号端口进的。交换机学到了 2:1.1@mac地址。

此时真正的数据包经过交换机，会定点从2号端口转发给路由器。

###### 步骤三 路由器的路由表

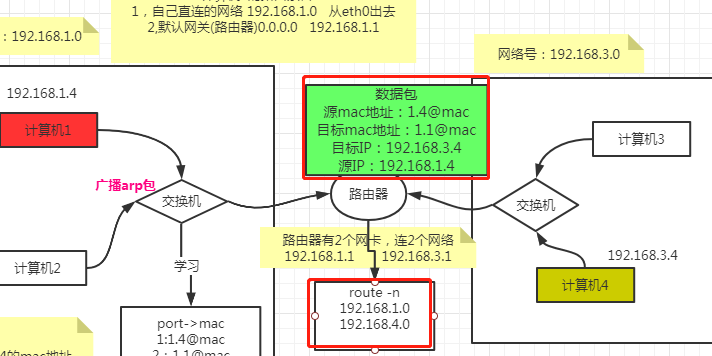
路由器上也有一张路由表。

route -n

192.168.1.0

192.168.3.0

路由器拿到数据包，找到目标地址192.169.3.4

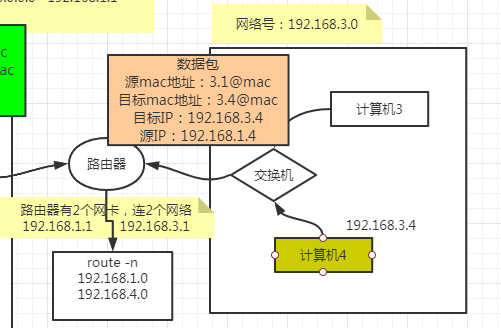


经过计算，数据包会转发到右边网络的交换机这，又回到了ARP协议。

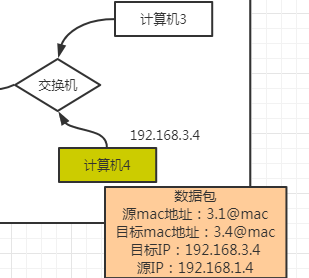
路由器也有链路层，如果这个路由器之前学习过ARP协议了，它应该知道3.4计算网卡的端口号是多少，mac地址是多少。如果没有就要走一遍ARP学习的过程。

###### 从路由器到3.0网络的交换机：

Mac地址转换，路由器将源mac地址改为自己的mac地址，目标mac地址改为3.4mac地址，IP地址不变



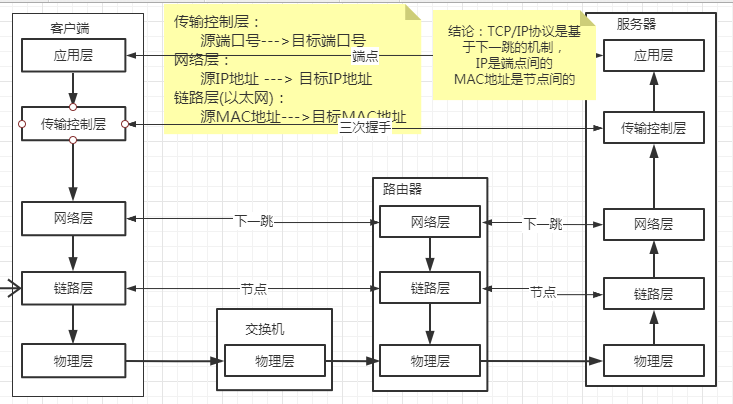
经过交换机后，会找到计算机4。



3.4计算机发现mac地址是自己的，撕掉。

发现IP地址是自己的，收进来，然后根据里面某个端口号找到某个进程。

### 8 总结



MAC地址：解决的是点与点之间，一跳一跳换来换去，从哪跳到哪去

IP地址：跳的时候找到合适的路，最终能跳到目标节点上。

端口号：跳到目标服务的时候，应该找到哪台服务器。

一些问题：

路由器有网关吗？

路由器是三层的有网络，交换机是2层的，没有路由表。

交换机是接同一网络的，路由器是衔接不同网络的。

某一个设备身上，某一个设备都会有某一个网络网关的概念。

# 高并发负载均衡：LVS的DR,TUN,NAT模型推导

# 高并发负载均衡：LVS的DR模型试验搭建

# 高并发负载均衡：基于keepalived的LVS高可用搭建