tcp/ip 五层模型：应用层，传输层，网络层，链路层，物理层

根据每一层提供的服务，以及协议，接口进行讲解

应用层：

负责应用程序之间的数据沟通

我们程序员自己写的程序

自定制协议：最常用的就是结构体

知名协议 http

数据的序列化 与 反序列化：数据进行持续化存储时，数据的组织就是序列化

持续化存储：内存是一个易失性介质，当我们将数据存储到内存中时，一旦断电，内存中存储的数据就会丢失，持续化存储就是将数据存储在某个地方，无论是否断电，都可以找到，数据不会丢失。其实就是存在硬盘上。

我们将一个数据向磁盘上进行存储的时候，如何去组织数据，就是序列化

http:

应用层的协议都是自定制协议，不过有些大佬们写的协议用的场景以及频率高了，使用的人多了，这个协议就成了知名协议。

url: 网址(统一资源定位符)

http://username:password@www.baidu.com:80/index.html?wd=c%2B%2B#ch

协议名://用户名:密码@服务器地址:端口号/资源路径?查询字符串#片段标识符

urlencode/urldecode

因为http的url中特殊字符一般都有特殊含义，因此我们日胶的查询字符中不能随意出现特殊字符，如果非要有特殊字符，就需要对特

殊字符进行转义(url编码 urlencode)

将一个字节的前四位/后四位转换为16进制数据，合到一起进行显示，使用%标识这是经过url编码的字符 + %2b

http是一个明文传输协议---传输层使用tcp协议

http的协议格式

request：

首行：GET https://www.baidu.com/index.html/1.1

METHOD(方法) URL() VERSION(协议版本)

请求方法：(GET POST HEAD) OPTIONS

首行与头部之间以换行来进行间隔(\r\n)

头部：key: val\r\nkey: val\r\n

以一个个的键值对组成，键值对：key：val \r\n

每个键值对之间以\r\n间隔

常见的请求头：

Host：服务器的主机地址

User-Agent：告诉服务器用户的系统版本、硬件版本以及浏览器版本。

Content-Len：正文长度(避免tcp协议的粘包问题)

Comtent-Type：正文格式

Accept-Encoding：告诉服务器能接收的压缩格式都有哪些

Accept-Language：能接收的语言

Referer：告诉服务器自己是从哪里跳转过来的，也就是上次所在的界面(用于流量统计)

Cookie：小饼干(为了让用户浏览网页更加舒服)

空行：为了间隔头部与正文的\r\n\r\n

正文：

GET请求没有正文

POST请求有正文

response：

首行：

VERSION：协议版本

STATU：状态码

STATU_DESC：状态描述

\r\n

头部：

一个个的键值对，以\r\n间隔，key: val

Location: 重定向的位置

Transfer-Encoding: 服务器在给客户端响应数据的时候，客户端向服务端请求一个资源，服务端就要给客户端回复一个资源，

方法一：Content-Length如果回复的之歌资源比较少的话，就可以直接获取一个长度，给客户端下发正文长度有多长，然后直接把资源全部发给客户端。

方法二：服务器觉得正文太长，或者服务器也不确定正文有具体多长的时候，就使用 chunked 方式进行发送。chunked响应指的是，正文在回复的时候，头部当中可以没有Content-Length，此时就代表对方接受数据的时候就不知道从那块开始接收了。chunked功能：回复数据的时候，每次给你发送一段数据之前，先告诉你我这次发送的之短数据有多长，先给你一个长度，但是这个长度是一个16进制数字，接下来才是正文信息，如果正文发送完了，最后再给你一个长度0，此时就代表正文发送完毕。

Set-Cookie: 当前用于添加(设置)一个Cookie信息，即它的路径等，要保存的信息都有哪些进行一个记录。在浏览器中有一个缓存文件(Cookie文件)

expires：过期时间、最大的生存时间

空行：

正文：

http服务器：

明文传输(以明文字符串形式组织协议)

http传输层使用tcp协议

http协议格式

sudo service iptables stop 关闭防火墙

GET/POST:

get没有正文：提交的数据在url中 1kb

post有正文：提交的表单数据

都是用于请求获取资源的

传输层：

负责端与端之间的数据传输。端口：负责标记一台主机上的进程。

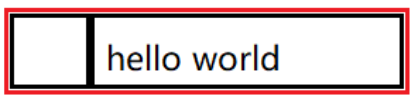
协议：UDP/TCP

udp协议：

udp特性：无连接，不可靠，面向数据报

优点：传输速度快，没有粘包问题

缺点：无法保证数据安全传输



协议中的端口号：负责端与端之间传输

协议中的长度：udp数据包长度，因为有这个长度，因此udp每条数据之间都有明显的边界，因此不会出现粘包问题

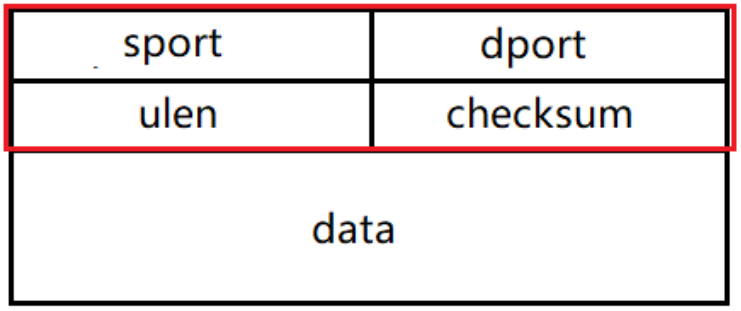
udp每个包的长度都是有限制的--不超过64k

为什么是64k？因为udp头部当中有一个ulen是一个uint16大小，他标记的udp数据包的长度，这个无符号两个字节最大的长度就是65535，所以一个数据包的最大的长度就是64k。

传输大数据的时候，需要在应用层进行切分

因为udp不可靠传输，所以无法保证安全，也无法保证包序，因此如果应用层进行分段，那么就要在应用层进行包序的管理。

校验和：校验数据包的完整性，正确性；计算方法：二进制反码求和(课后调研)

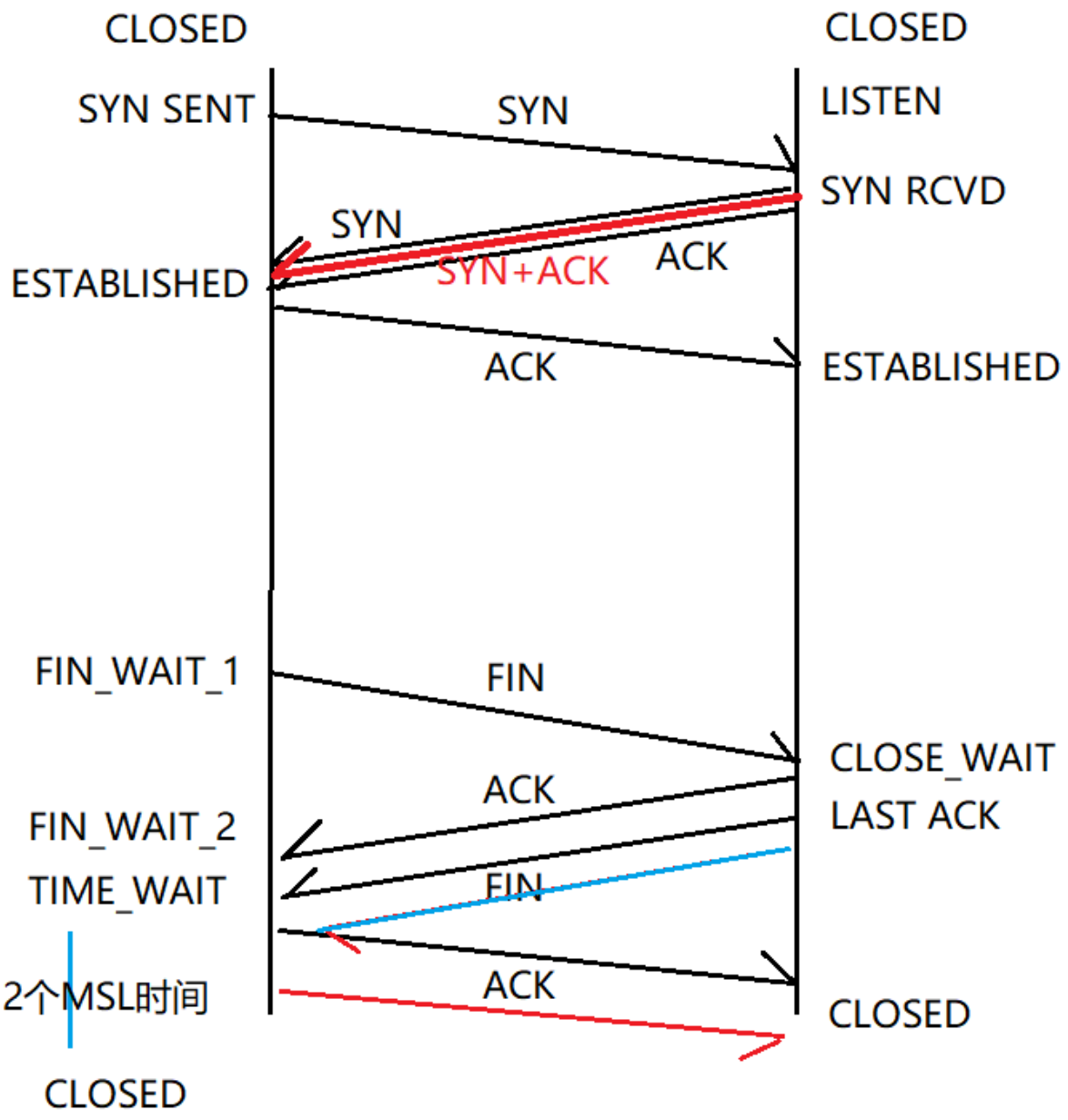


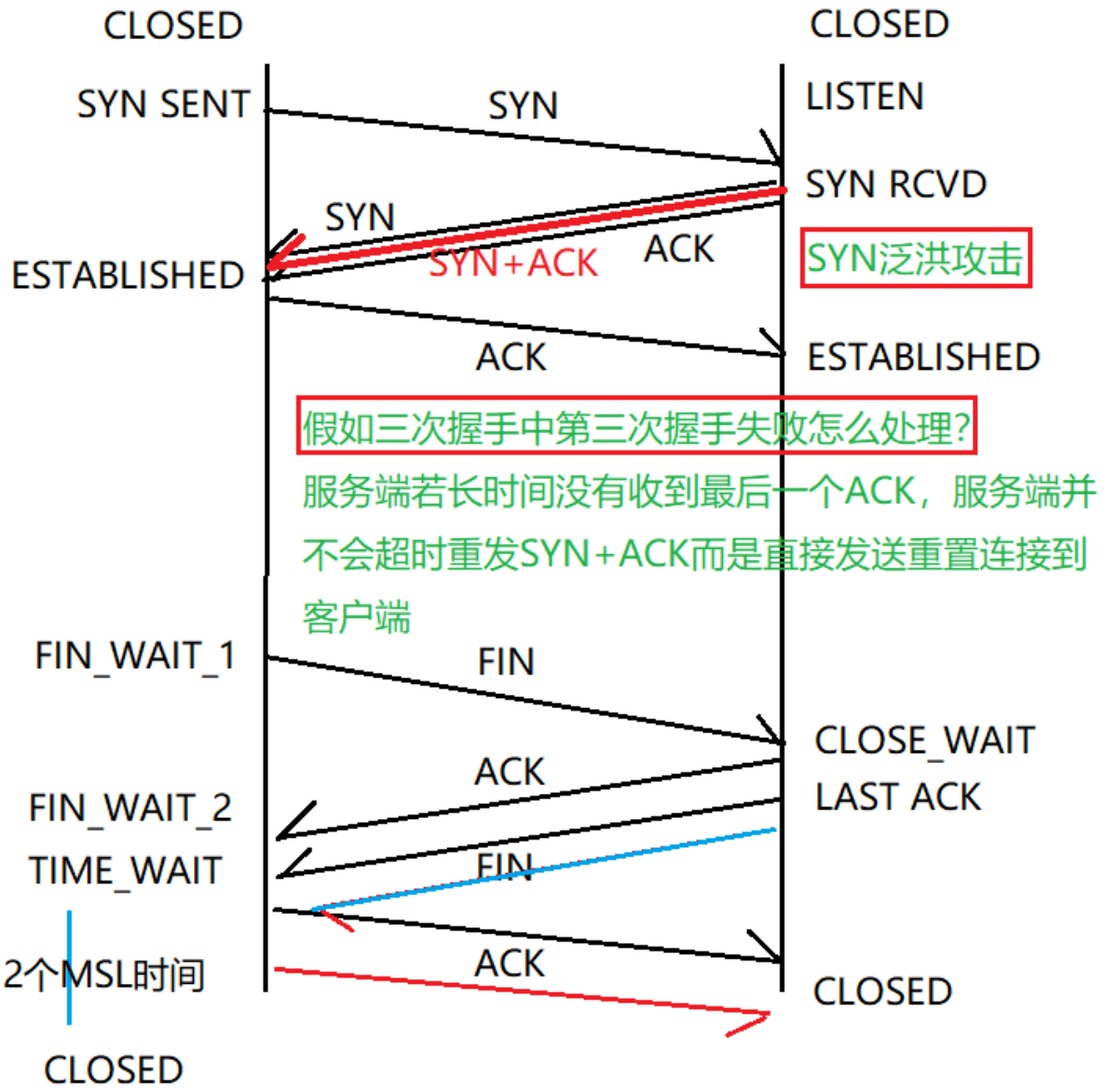
tcp协议：

有连接，可靠传输，面向字节流

1.连接管理：tcp三次握手与四次挥手

三次握手：因为两次不安全，四次没必要


CLOSED
CLOSED
SYN SENT
SYN
LISTEN
SYN RCVD
SYN
ESTABLISHED
SYN+ACK
ACK
ACK
ESTABLISHED
FIN_WAIT_1
FIN
CLOSE_WAIT
ACK
LAST ACK
FIN_WAIT_2
TIME_WAIT
FIN
ACK
2个MSL时间
CLOSED
CLOSED



MSL时间：报文最大生存周期，指的是一个报文在网络中可以生存的最大时间。Linux中默认为30秒，可设置。

主动关闭连接方在发送最后一个ACK之后会进入TIME_WAIT状态等待啷个MSL时间

TIME_WAIT作用：防止最后的ACK丢失导致对端重发的FIN包对新起的相同地址电口连接造成影响

因为最后一个ACK丢失(或者因为网络状态不好导致在报文最大周期之内)之后，对端没有收到ACK，一次重发FIN请求；

假如没有这个等待时间，而直接重新启动程序，就有可能会直接收到这个重发的FIN包对新连接造成影响

为什么是两个MSL时间：第一个MSL是主动关闭方第一个ACK的最大生存周期，第二个MSL是重发的FIN包在网络上的最大生存周期，如果主动关闭方在两个MSL时间后都没有收到重发的FIN请求，就可以关闭了，因为不管ACK有没有成功，重发的FIN都会消失在网络中

可靠传输：

1. 如何确定对方收到了数据

   确认应答机制

2. 假如对方没有收到数据怎么处理

   超时重传机制

3. 如何保证有序传输

   序列号/确认序列号

4. 校验和

因为tcp为了保证可靠传输，因此牺牲了传输性能，但是并不是无药可救，针对整体的性能下降还是可以挽救以下的

1. 滑动窗口机制

   快速重传(若接收到后续数据，但是前边的数据丢了，则连发三条重

   发请求)

   为什么不是一条就重传，而是接到三条重传请求才重传？

   考虑网络延迟问题

4. 拥塞窗口机制

   探测网络状况，防止一开始大量丢包

2. 延迟应答机制

   集中等待时间

   保持最大窗口(保持最大吞吐量)

3. 捎带应答机制

面向字节流：

优点：收发灵活

缺点：造成粘包问题

字节流发送数据会将字节流数据先放在缓冲区中，等待数据大小超过缓冲区，然后一次发送，对端接收的时候有可能就将多条数据当作一条数据接收完毕进行处理

因为字节流数据无明显边界，因此造成粘包问题；

解决方法：想办法在应用层加上边界

1.数据定常，

2.特殊字符间隔　http　　json

3.TLV数据格式

```
struct{
    uint8_t type;
    uint64_t len;
    char val[0];
}
```

tcp协议当中所涉及到的数据：

源端口，目的端口，序号，确认序号，校验和，窗口大小，头部大小，6位保留，6位标志位，紧急指针，选项数据

tcp连接方式：

tcp长连接：

tcp短连接：

网络断开/对端关闭连接，程序如何快速判断连接已经断开？

tcp内部有自己的连接检测机制(保活机制)，检测连接是否正常

发送端判断：操作系统在连续发送多次数据没有ACK之后，认为连接断开，此时向进程发送SIGPIPE信号，进程使用默认处理方式(推出进程)

接收方判断：recv返回0(对端关闭连接，连接断开)

网络层：

负责地址管理与路由选择

典型协议：IP协议

地址管理：

1.网段划分：

将IP地址划分了两个部分

网络号：相邻网络不能具备相同网络号

主机号

192.168.2.124/24　　后边的24表示前24位是网络号，即：192.168.2.

子网中主机号总共有64个，但是实际可分配主机号只有62个，因为每个局域网中都有两个特殊地址：

主机号全为0：局域网的网络号

主机号全为1：局域网的广播地址，套接字只有udp广播(网络层实现的)，tcp没有实现广播(需要用户应用层实现)

127.0.0.1：本地虚拟回环网卡，用于本地的网络回环测试

私网地址/公网地址：

组建私网(局域网)是不能随意使用网络号的，因为一旦随意使用，用可能造成分配的ip地址与公网地址冲突

RFC 1918规定能够用于组建私网的网络号只有那么几个：

172.16.*.* ~ 172.31.*.*

10.*.*.*

192.168.*.*

链路层：负责相邻设备之间的数据帧传输

arp协议:介于网络层与数据链路层的协议

功能：获取相邻设备的mac地址

00:0C:29:2C:9D:45　　FF:FF:FF:FF:FF

192.168.2.1

arp网络攻击：局域网的欺骗攻击

因为每次发送数据都获取mac地址的话，效率太低，因此，当获取到相邻设备的mac地址后会在计算机中做一个缓存；但是这个缓存保存的时间不会太长，因为arp协议通过ip地址获取mac地址的，但是现在的ip地址大多数都是dhcp自动分配的，如果断网重新连接后，可能ip地址就不一样了，原先的地址有可能会被其他人使用，导致mac匹配错误。

MTU：最大传输单元

MTU = IP头大小 + UDP/TCP头长度 + 数据长度

假如MTU = 1500　使用udp协议传输数据，数据最大长度？1500-20-8=1472

mtu对udp的影响：

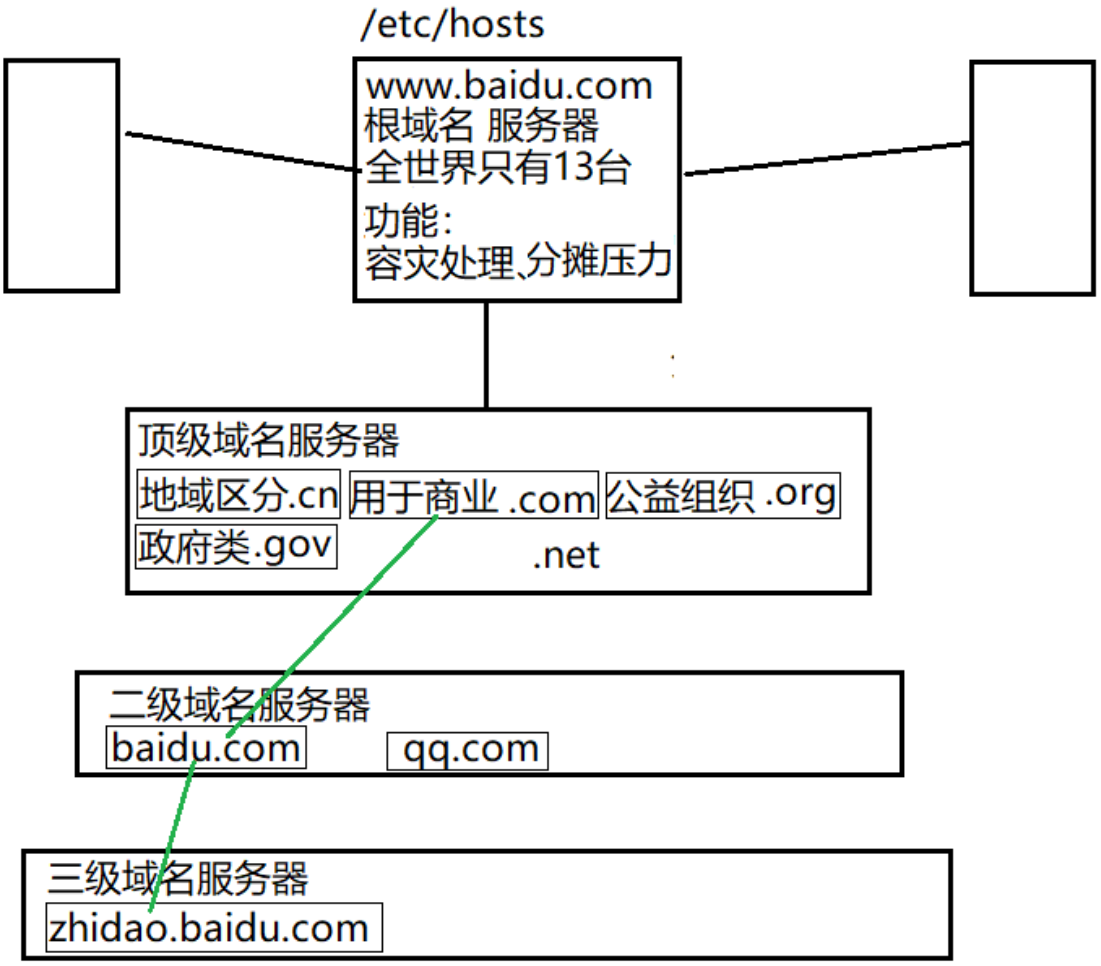
因为udp不会在传输层分包，因此有可能在网络层进行数据分片，但是如果分片后，进行传输的时候，有任意一个数据片出错，将会导致整个udp数据包被丢弃(udp不会重传-丢包)，因此一般情况下，我们在使用udp传输数据的时候，会将udp的数据包在应用层分为大小不超过最大传输段大小

mtu对tcp的影响：

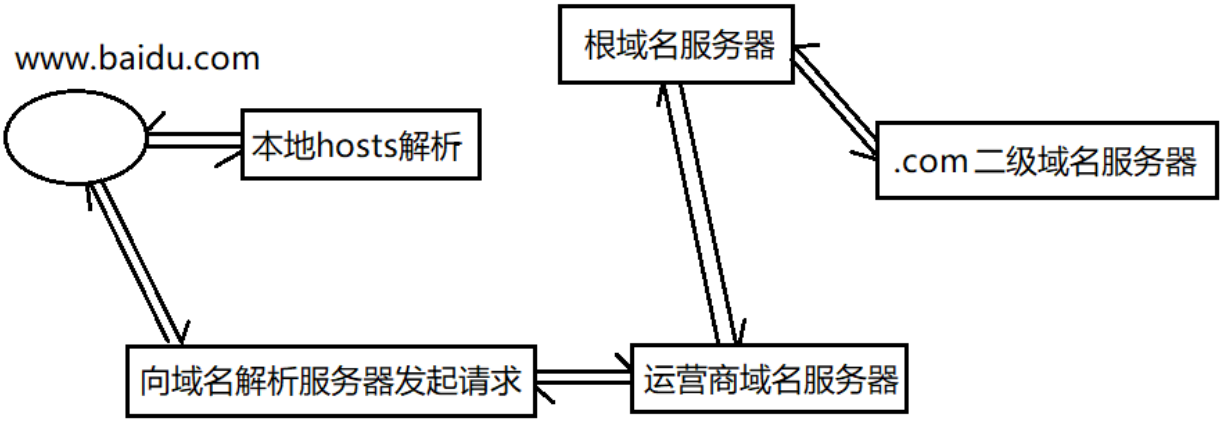
tcp在传输层建立连接的时候就会与对方进行协商，协商最大数据段大小(MSS = MTU - IP - TCP),取双方协商中最小的一个进行传输层的对数据分段；意味着TCP发送的数据在网络层基本上是不会出现数据分片。

其他重要协议及技术：

DNS：域名解析系统



面试题：浏览器中键入url，按下回车都发生了哪些事情



1. 域名解析--获取IP地址
2. 组织http请求(http协议格式)
3. 传输层使用tcp(tcp协议，特性)
4. 网络层IP数据报(IP协议，地址管理路由选择)
5. 链路层(以太头格式，arp，mtu)

ICMP协议：

一个网络层的协议

探测网络是否联通

使用icmp协议探测网络的命令ping　　eg: ping www.baidu.com

面试题：ssh使用22号端口　ftp使用21号端口，ping使用多少端口？

ping使用的是icmp协议实现，因为icmp是网络层协议，涉及不到端口，而端口是传输层的协议，所以ping命令不使用端口。

NAT/NAPT技术：

负责网络层的地址替换

代理服务器与NAT区别：

NAT是网络的是服务，进行网络层的地址替换，一般部署在网络层的出口位置(路由器)

代理是引用的程序，运行部署在服务器上，代理服务器可以部署在任何位置