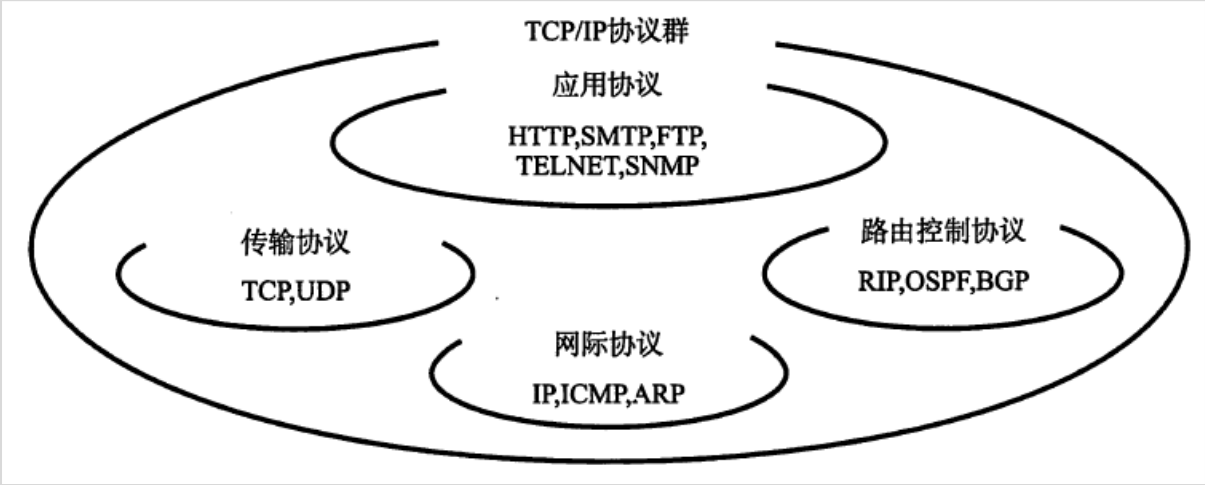
四、TCP 三次握手

TCP/IP协议

TCP/IP传输协议，传输控制/网络协议，也叫作网络通讯协议，它是在网络的使用中的最基本的通信协议

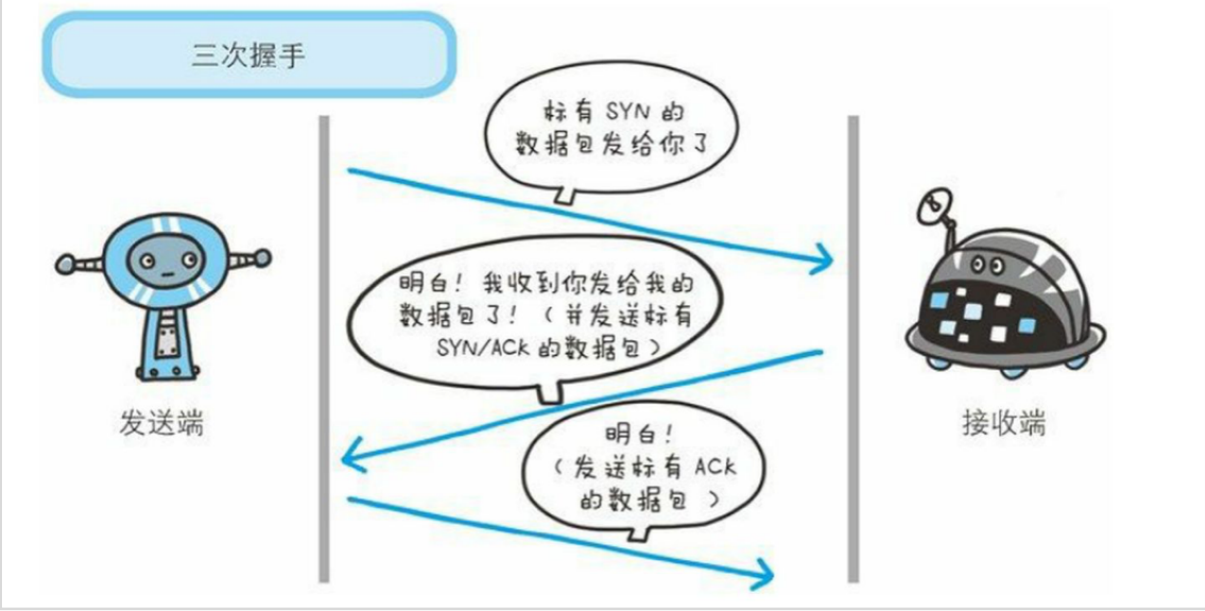
TCP/IP协议群

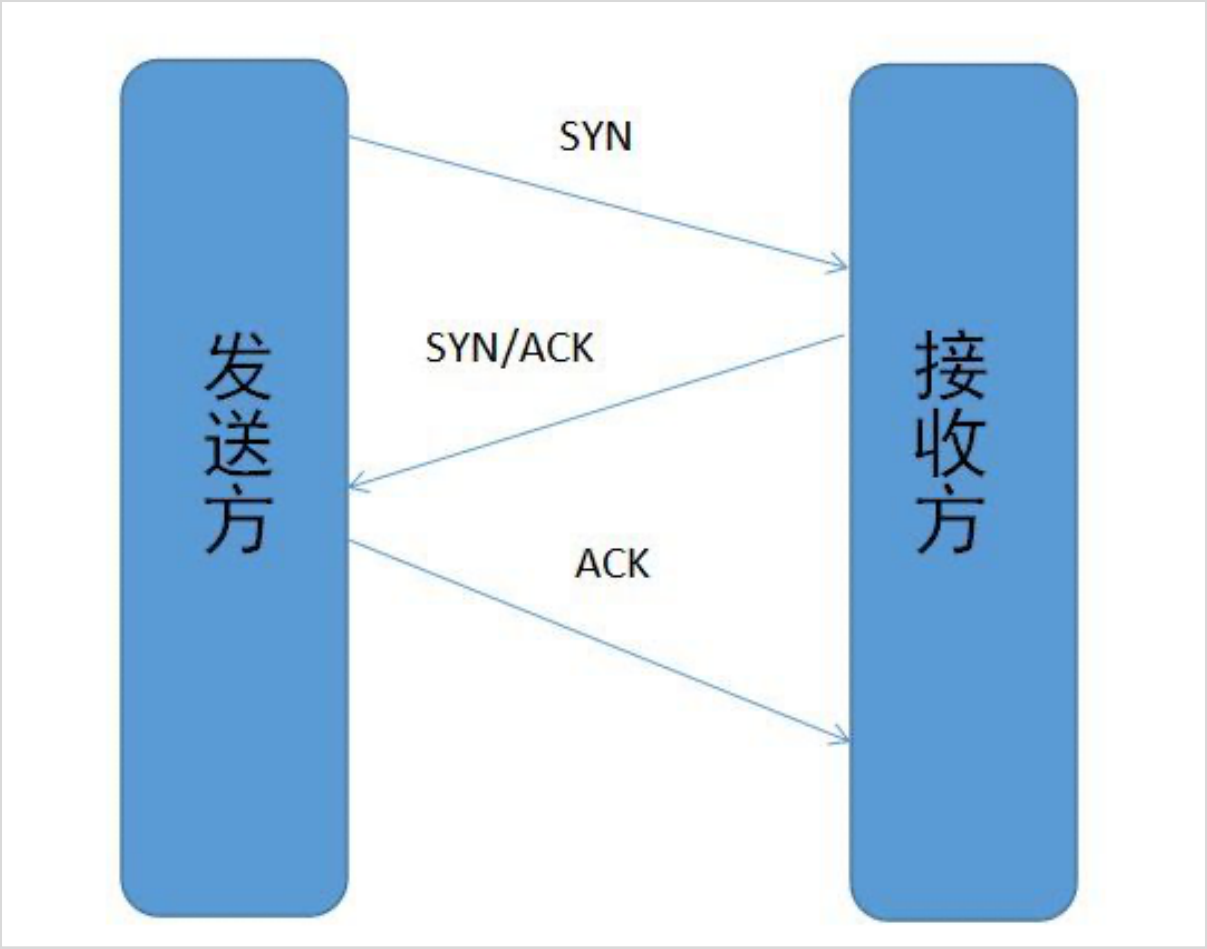
TCP/IP是基于TCP和IP这两个最初的协议之上的不同的通信协议的大集合



三次握手

为了准确无误地把数据送达目标处，TCP协议采用了三次握手策略





客户端-发送带有 SYN 标志的数据包-**一次握手**-服务端
服务端-发送带有 SYN/ACK 标志的数据包-**二次握手**-客户端
客户端-发送带有带有 ACK 标志的数据包-**三次握手**-服务端

为什么要三次握手

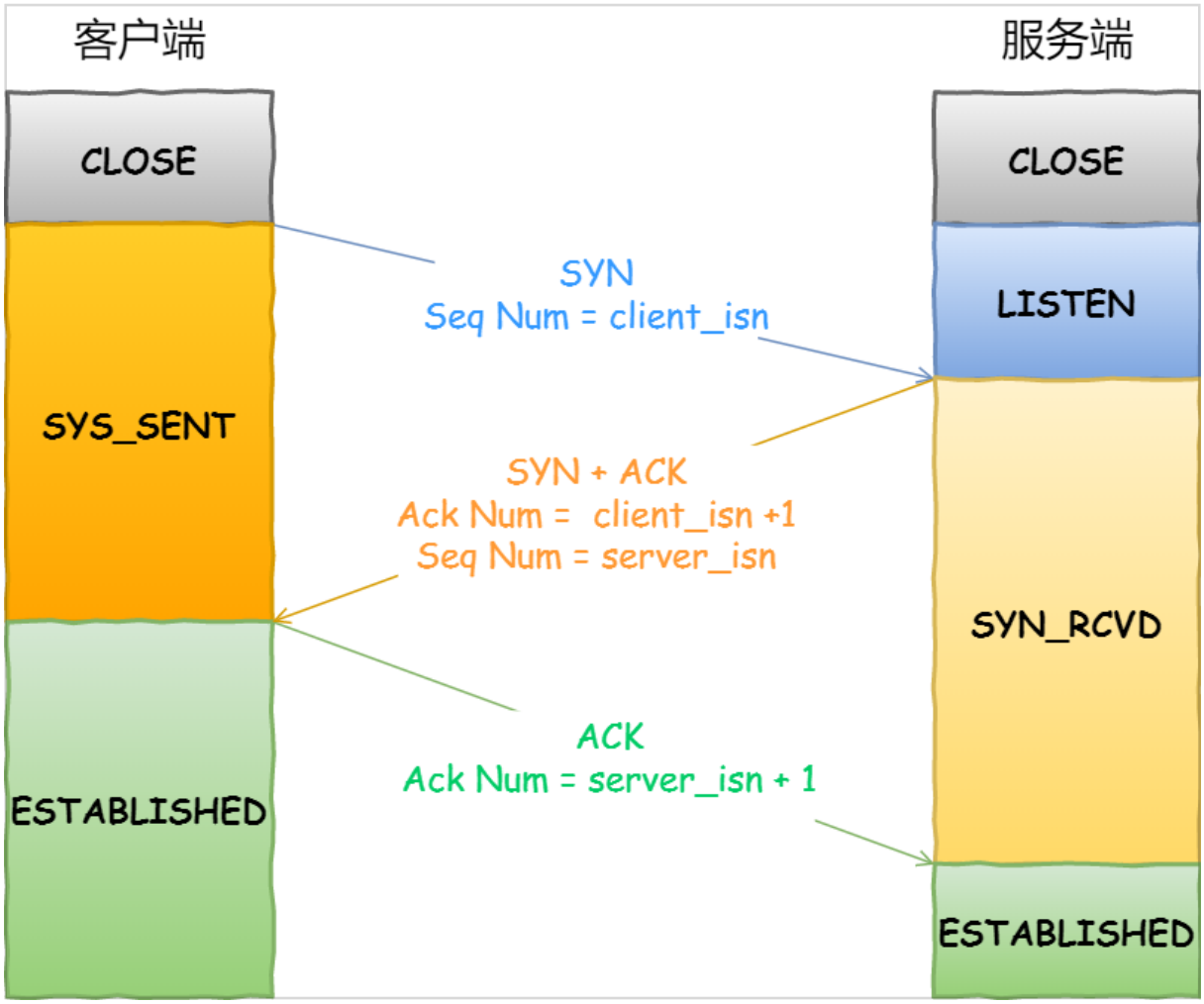
三次握手的目的是建立可靠的通信信道，说到通讯，简单来说就是数据的发送与接收，而三次握手最主要的目的就是双方确认自己与对方的发送与接收是正常的

- 第一次握手：Client 什么都不能确认；Server 确认了对方发送正常
- 第二次握手：
  - Client 确认了：自己发送、接收正常，对方发送、接收正常；
  - Server 确认了：自己接收正常，对方发送正常（**注意这里还不能确认自己发送政策**）
- 第三次握手：
  - Client 确认了：自己发送、接收正常，对方发送、接收正常；
  - Server 确认了：自己发送、接收正常，对方发送、接收正常

所以三次握手就能确认双方收发功能都正常，缺一不可

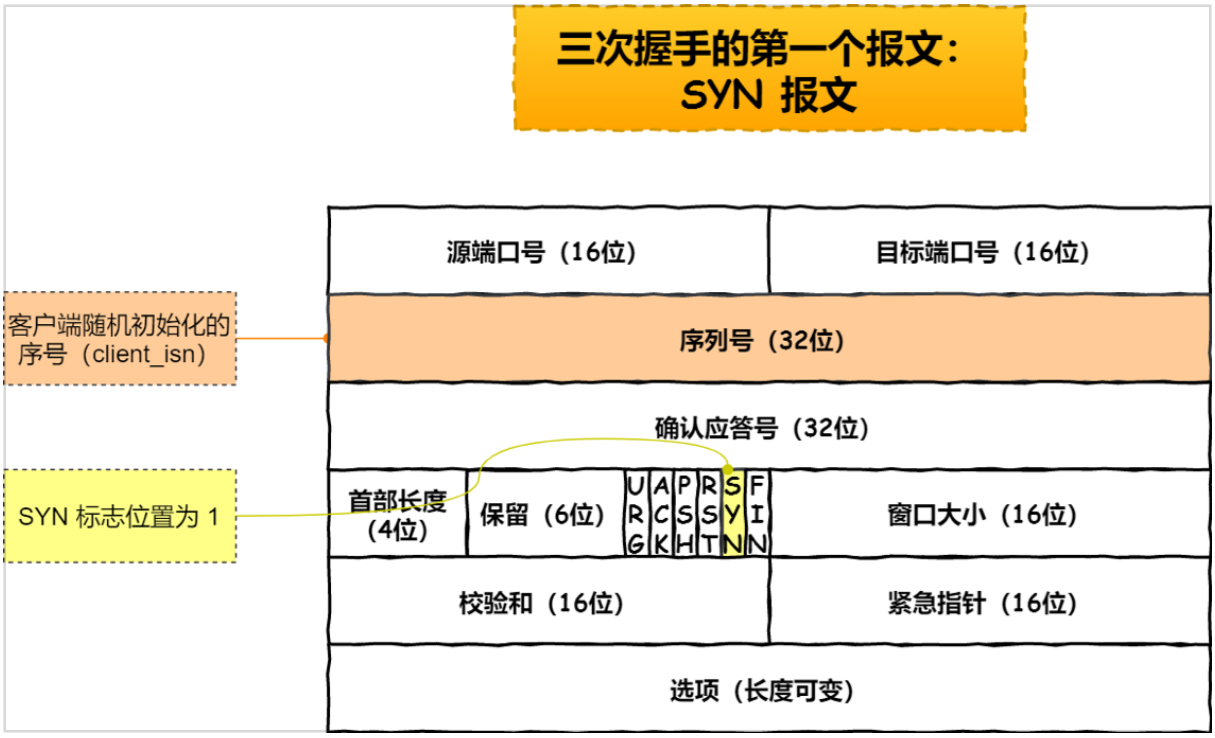
三次握手的三个报文详解

> TCP 是面向连接的协议，所以使用 TCP 前必须先建立连接，而**建立连接是通过三次握手来进行的**



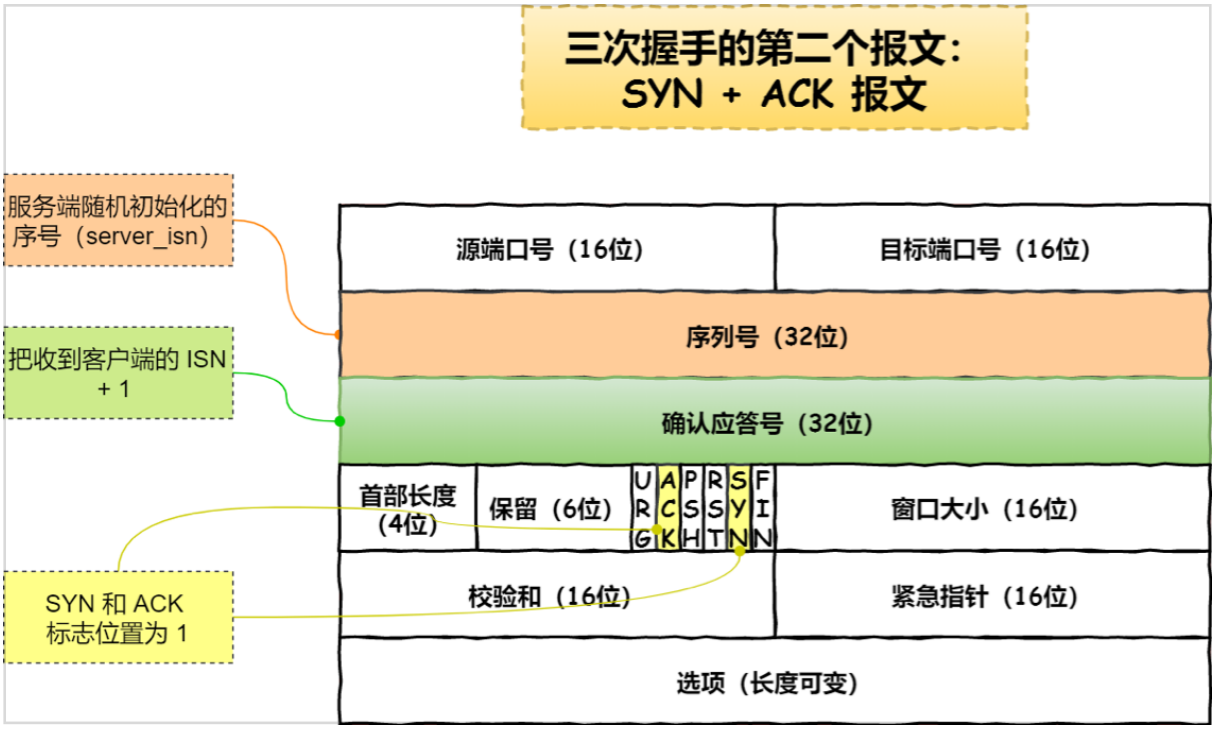
- 一开始，客户端和服务端都处于`CLOSED`状态。先是服务端主动监听某个端口，处于`LISTEN`状态

1. 三次握手的第一个报文：**SYN报文**



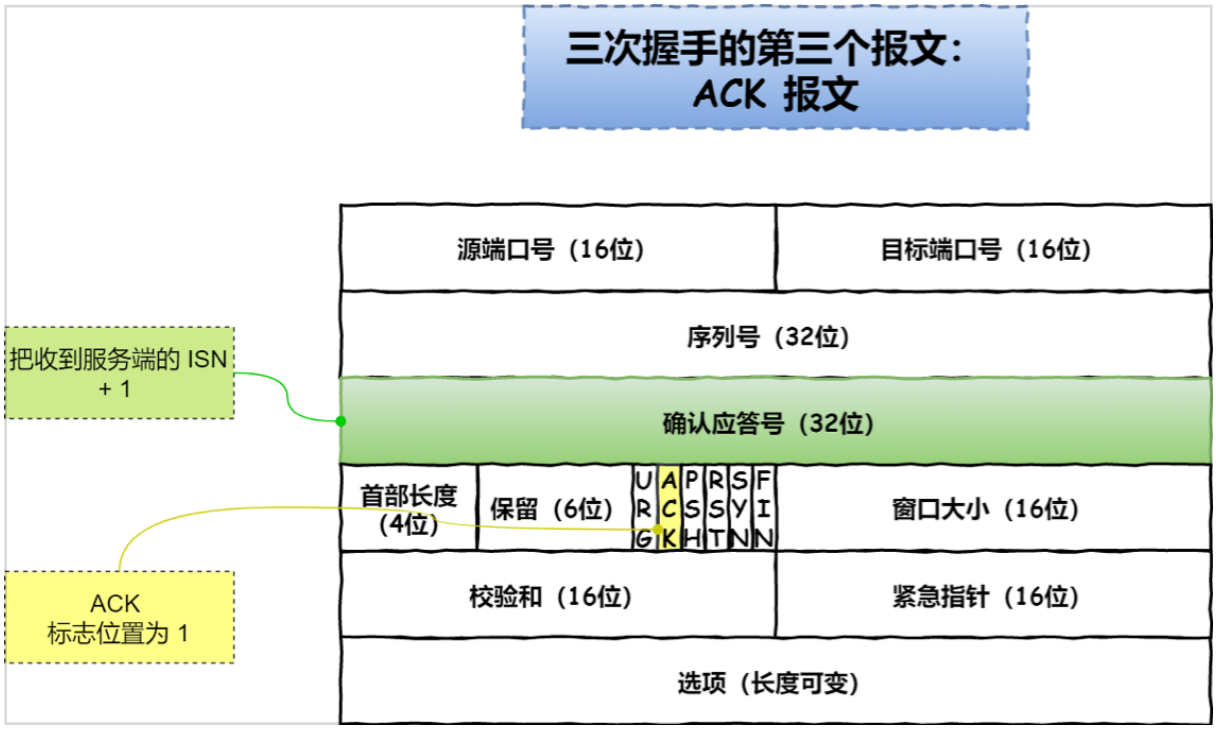
- 客户端会**随机初始化序号**(client_isn)，将此序号置于 TCP 首部的「序号」字段中，同时把 SYN 标志位置为 1 ，表示 SYN 报文。接着把第一个 SYN 报文发送给服务端，表示向服务端发起连接， 该报文不包含应用层数据 ，之后客户端处于 `SYN-SENT` 状态

2. 三次握手的第二个报文：**SYN+ACK报文**



- 服务端收到客户端的 `SYN` 报文后，**首先服务端也随机初始化自己的序号(server_isn)**，将此序号填入 TCP 首部的「序号」字段中，**其次把 TCP 首部的「确认应答号」字段填入** `ack = client_isn + 1 = 客户端的syn+1`，接着把 `SYN` 和 `ACK` 标志位置为 1 。最后把该报文发给客户端， 该报文也不包含应用层数据 ，之后服务端处于 `SYN-RCVD` 状态

3. 三次握手的第三个报文：**ACK报文**



- 客户端收到服务端报文后，还要向服务端回应最后一个应答报文，首先该应答报文 TCP 首部 `ACK` 标志位置为 1 ，其次「确认应答号」字段填入 `server_isn + 1 -> ack = 服务端的seq +1` ，最后把报文发送给服务端，这次报文可以携带客户到服务器的数据，之后客户端处于 `ESTABLISHED` 状态。

- 服务器收到客户端的应答报文后，也进入 `ESTABLISHED` 状态。

> 从上面的过程可以发现**第三次握手是可以携带数据的，前两次握手是不可以携带数据的**，这也是面试常问的题

> 一旦完成三次握手，双方都处于 ESTABLISHED 状态，此时连接就已建立完成，客户端和服务端就可 以相互发送数据了

关于三次握手的问题：

1. 如何在 Linux 系统中查看 TCP 状态?

   TCP 的连接状态查看，在 Linux 可以通过 netstat -napt 命令查看

2. 为什么要传回SYN

   接收端传回发送端所发送的SYN，是为了告诉发送端，我接收到的信息确实就是你所发送的信号了

3. 传了SYN，为啥还要传ACK

   双方通信无误必须是两者互相发送信息都无误。传了 SYN，证明发送方到接收方的通道没有问题，但是接收方到发送方的通道还需要 ACK 信号来进行验证

> **序列号**：在建立连接时由计算机生成的随机数作为其初始值，通过 SYN 包传给接收端主机，每发送一次数据，就「累加」一次该「数据字节数」的大小。**用来解决网络包乱序问题。**
>
> **确认应答号**：指下一次「期望」收到的数据的序列号，发送端收到这个确认应答以后可以认为在这个序号以前的数据都已经被正常接收。**用来解决不丢包的问题。**

4. 为什么是三次握手，不是两次、四次?

> 相信大家比较常回答的是：“因为三次握手才能保证双方具有接收和发送的能力。”
>
> 这回答是没问题，但这回答是片面的，并没有说出主要的原因。
>
> 在前面我们知道了什么是 **TCP 连接**：
>
> - 用于保证可靠性和流量控制维护的某些状态信息，这些信息的组合，包括**Socket、序列号和窗口大小**称为连接。
>
> 所以，重要的是**为什么三次握手才可以初始化Socket、序列号和窗口大小并建立 TCP 连接。**

   接下来以三个方面分析三次握手的原因:

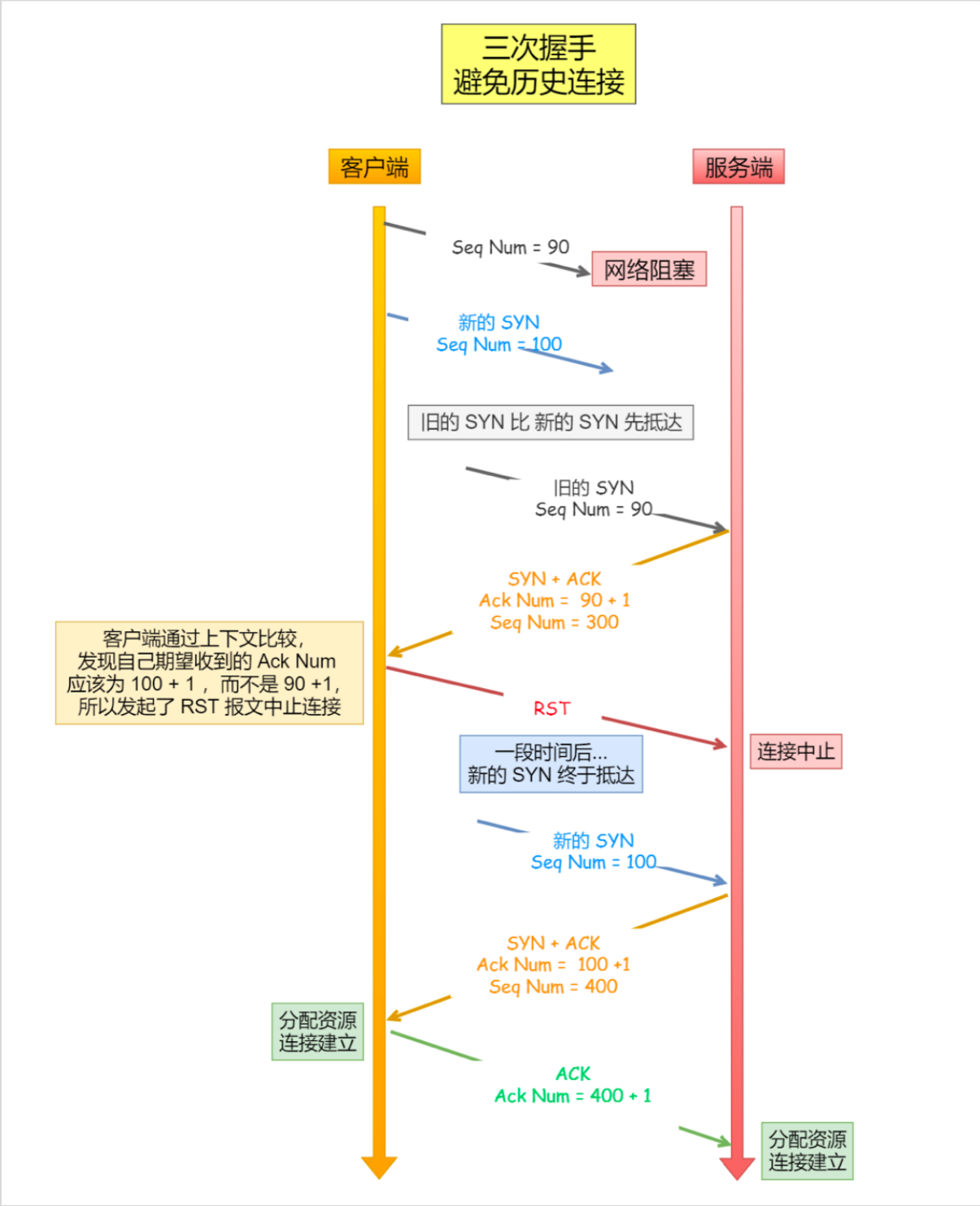
   - 三次握手才可以**阻止重复历史连接的初始化**(主要原因)
   - 三次握手才可以**同步双方的初始序列号**
   - 三次握手才可以**避免资源浪费**

- 原因1：**避免历史连接的建立（序列号过期或超时）**（主要原因）

  简单来说，三次握手的**首要原因是为了防止旧的重复连接初始化造成混乱**（防止旧的数据包先

到达目标主机而造成混乱）

> 网络环境是错综复杂的，往往并不是如我们期望的一样，先发送的数据包，就先到达目标主机，反而它 很骚，可能会由于网络拥堵等乱七八糟的原因，会使得旧的数据包，先到达目标主机，那么这种情况下 TCP 三次握手是如何避免的呢？



客户端连续发送多次 SYN 建立连接的报文，在**网络拥堵**情况下:

- 一个「旧 SYN 报文」比「最新的 SYN 」 报文早到达了服务端; 那么此时服务端就会回一个 SYN + ACK 报文给客户端;

- 客户端收到后可以根据自身的上下文，判断这是一个历史连接(序列号过期或超时)，那么客户端就会发送 RST 报文给服务端，表示中止这一次连接。

如果是两次握手，那么就不能判断是否是历史连接

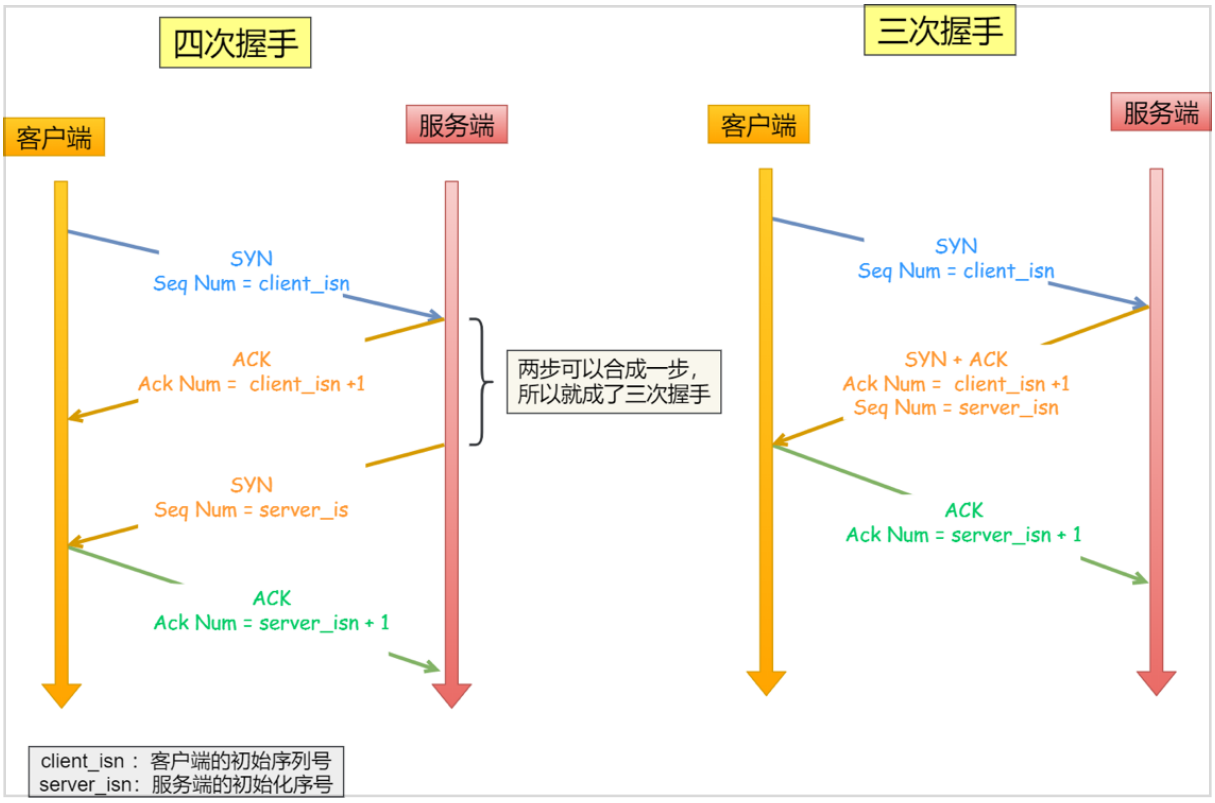
- 如果是历史连接(序列号过期或超时)，则第三次握手发送的报文是 RST 报文，以此中止历史连接;
- 如果不是历史连接，则第三次发送的报文是 ACK 报文，通信双方就会成功建立连接

- 原因2：同步**双方的初始序列号**

  TCP 协议的通信双方， 都必须维护一个「序列号」， 序列号是可靠传输的一个关键因素，它的作用:

  - 接收方可以去除重复的数据;
  - 接收方可以根据数据包的序列号按序接收;
  - 可以标识发送出去的数据包中， 哪些是已经被对方收到的;

可见，序列号在 TCP 连接中占据着非常重要的作用，所以当客户端发送携带「初始序列号」的 SYN 报文的时候，需要服务端回一个 ACK 应答报文，表示客户端的 SYN 报文已被服务端成功接收，那当服务端发送「初始序列号」给客户端的时候，依然也要得到客户端的应答回应，**这样一来一回，才能确保双方的初始序列号能被可靠的同步。**



四次握手其实也能够可靠的同步双方的初始化序号，但由于**第二步和第三步可以优化成一步**，所以就成 了「三次握手」

而两次握手只保证了一方的初始序列号能被对方成功接收，没办法保证双方的初始序列号都能被确认接收

- 原因3：**避免资源浪费**

  如果只有「两次握手」，当客户端的 SYN 请求连接在网络中阻塞，客户端没有接收到 ACK 报文， 就会重新发送 SYN ，由于没有第三次握手，服务器不清楚客户端是否收到了自己发送的建立连接的 ACK 确认信号，所以每收到一个 SYN 就只能先主动建立一个连接，这会造成什么情况呢?

  如果客户端的 SYN 阻塞了，重复发送多次 SYN 报文，那么服务器在收到请求后就会**建立多个冗余的无效链接，造成不必要的资源浪费。**

- 总结

  TCP 建立连接时，通过三次握手**能防止历史连接的建立，能减少双方不必要的资源开销，能帮助双方同步初始化序列号**。序列号能够保证数据包不重复、不丢弃和按序传输

  - 不使用「两次握手」和「四次握手」的原因:
    - 「两次握手」：无法防止历史连接的建立，会造成双方资源的浪费，也无法可靠的同步双方序列号;
    - 「四次握手」：三次握手就已经理论上最少可靠连接建立，所以不需要使用更多的通信次数。

5. 为什么客户端和服务端的初始序列号ISN是不相同的?

   1. **如果相同，则无法判断是否是历史报文**

   如果一个已经失效的连接被重用了，但是该旧连接的历史报文还残留在网络中，如果序列号相同，那么就无法分辨出该报文是不是历史报文，如果历史报文被新的连接接收了，则会产生数据错乱

   所以，**每次建立连接前重新初始化一个序列号主要是为了通信双方能够根据序号将不属于本连接的报文段丢弃**

   2. 为了安全性：

   防止黑客伪造的相同序列号的 TCP 报文被对方接收

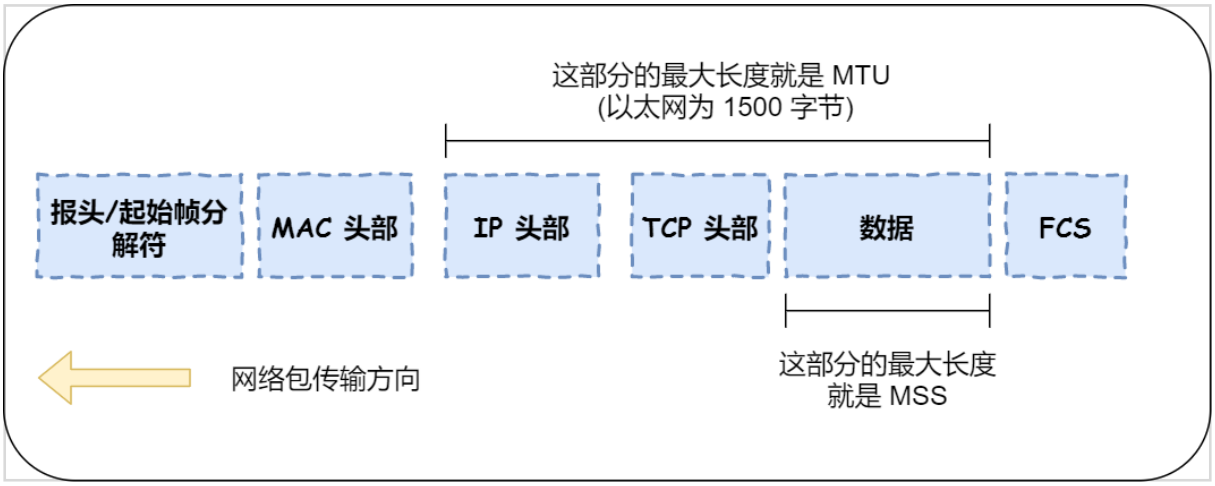
6. 初始化序列号ISN是如何随机产生的?
   - 起始 `ISN` 是基于时钟的，每 4 毫秒 + 1，转一圈要 4.55 个小时
   - 一个算法是*ISN = M + F (localhost, localport, remotehost, remoteport)*
     - `M` 是一个计时器，这个计时器每隔 4 毫秒加 1。
     - `F` 是一个 Hash 算法，根据源 IP、目的 IP、源端口、目的端口生成一个随机数值。要保证 Hash 算法不能被外部轻易推算得出，用 MD5 算法是一个比较好的选择

7. 既然IP层会分片，为什么TCP层还需要MSS呢?

先复习 MTU 和 MSS

MTU：最大传输单元（Maximum Transmission Unit）

MSS：最大报文长度（Maximum Segment Size）



`MTU`：一个网络包的最大长度，以太网中一般为 1500 字节

`MSS`：除去 IP 和 TCP 头部之后，一个网络包所能容纳的 *TCP* 数据 的最大长度

应该在它方便的时候以报文段的形式发送数据”。TCP 可从缓存中取出并放入报文段中的数据数量受限于**最大报文段长度**（Maximum Segment Size，MSS）。MSS 通常根据最初确定的由本地发送主机发送的最大链路层帧长度（即所谓的**最大传输单元**（Maximum Transmission Unit，MTU））来设置。设置该 MSS 要保证一个 TCP 报文段（当封装在一个 IP 数据报中）加上 TCP/IP 首部长度（通常 40 字节）将适合单个链路层帧。以太网和 PPP 链路层协议都具有 1500 字节的 MTU，因此 MSS 的典型值为 1460 字节。已经提出了多种发现路径 MTU 的方法，并基于路径 MTU 值设置 MSS（路径 MTU 是指能在从源到目的地的所有链路上发送的最大链路层帧［RFC 1191］）。注意到 MSS 是指在报文段里应用层数据的最大长度，而不是指包括 TCP 首部的 TCP 报文段的最大长度。（该术语很容易混淆，但是我们不得不采用它，因为它已经根深蒂固了。）

既然IP层会分片，为什么TCP层还需要MSS呢?

【因为IP层没有超时重传，超时重传是依赖传输层的TCP的，所以当MTU中的TCP报文的某一片数据丢失后，接收方不会响应ack给对方，只能等发送方的TCP再超时后，再重发整个TCP报文，效率很低】

如果在 TCP 的整个报文(头部 + 数据)交给 IP 层进行分片，会有什么异常呢?

当 IP 层有一个超过 MTU 大小的数据(TCP 头部 + TCP 数据)要发送，那么 IP 层就要进行分片，把 数据分片成若干片，保证每一个分片都小于 MTU。把一份 IP 数据报进行分片以后，由目标主机的 IP 层来进行重新组装后，再交给上一层 TCP 传输层。

这看起来井然有序，但这存在隐患的，**那么当如果一个IP分片丢失，整个IP报文的所有分片都得重传**

因为 IP 层本身没有超时重传机制，它由传输层的 TCP 来负责超时和重传
当接收方发现 TCP 报文(头部 + 数据)的某一片丢失后，则不会响应 ACK 给对方，那么发送方的 TCP 在超时后，就会重发「整个 TCP 报文(头部 + 数据)」

因此，可以得知由 IP 层进行分片传输，是非常没有效率的

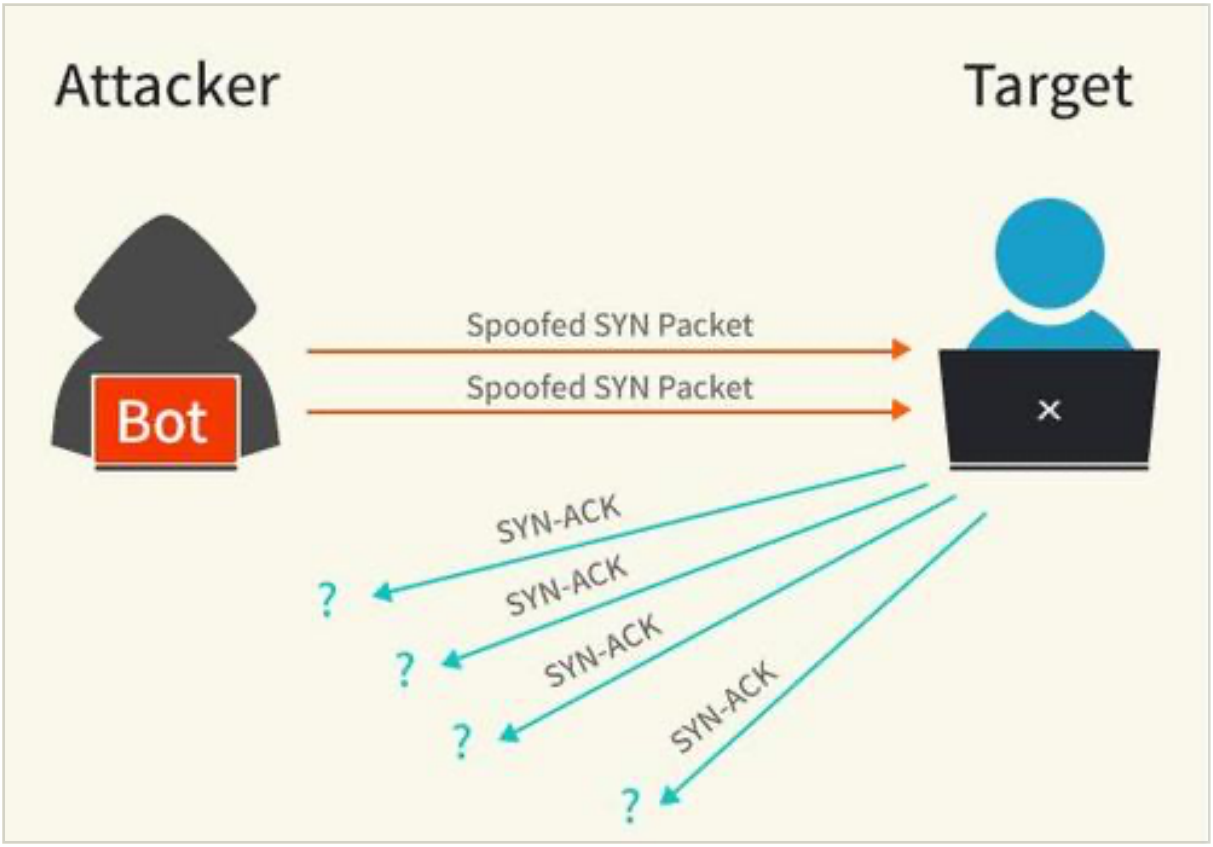
所以，为了达到最佳的传输效能，TCP 协议在**建立连接的时候通常要协商双方的MSS值**，当 *TCP* 层发现数据超过 *MSS* 时，则就先会进行分片，当然由它形成的 *IP* 包的长度也就不会大于 *MTU*，自然也就不用 *IP* 分片了。

```
[SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
[SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM=1 WS=16384
[ACK] Seq=1 Ack=1 Win=66304 Len=0
```

经过 TCP 层分片后，如果一个 TCP 分片丢失后，**进行重发时也是以MSS为单位**，而不用重传所有的分片，大大增加了重传的效率

8. 什么是 SYN 攻击? + 如何避免 SYN 攻击?

我们都知道 TCP 连接建立是需要三次握手，假设攻击者短时间伪造不同 IP 地址的 SYN 报文，服务 端每接收到一个 SYN 报文，就进入 SYN_RCVD 状态，但服务端发送出去的 ACK + SYN 报文，无 法得到未知 IP 主机的 ACK 应答，久而久之就会**占满服务端的SYN接收队列(未连接队列)**，使得服务器不能为正常用户服务



- 避免SYN攻击方式一：

  通过修改 Linux 内核参数，控制队列大小和当队列满时应做什么处理

  - 当网卡接收数据包的速度大于内核处理的速度时，会有一个队列保存这些数据包。控制该队列的最大值如下参数：

  ```
  net.core.netdev_max_backlog
  ```

  - SYN_RCVD 状态连接的最大个数：

  ```
  net.ipv4.tcp_max_syn_backlog
  ```

  - 超出处理能时，对新的 SYN 直接回报 RST，丢弃连接：

  ```
  net.ipv4.tcp_abort_on_overflow
  ```

- 避免SYN攻击方式二：

  暂未整理（小林图解网络page129）