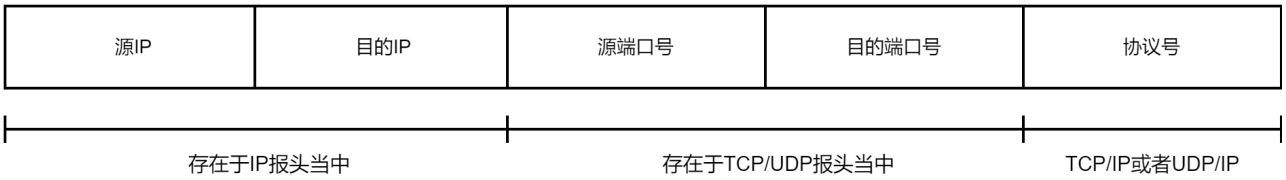
传输层

1.五元组标识通信



在TCP/IP协议中，用"源IP"，"目的IP"，"源端口号"，"目的端口号"，"协议号"，这五元组来标识一个通信。

- 端口号：标识另一台主机上的一个进程。
- 端口号如何找到进程？内核实现通过哈希算法，端口号对应进程PID。
- 源IP：发送数据的主机IP
- 目的IP：最终接收数据主机IP
- 源端口号：发送数据主机中发送数据的进程
- 目的端口号：最终接收数据主机中要接收数据的进程。

2.UDP协议

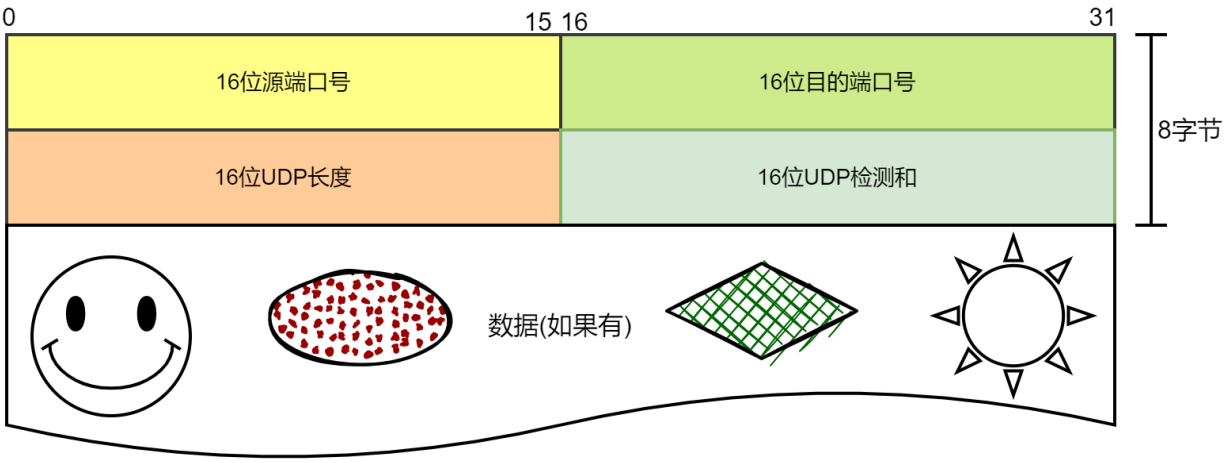
2.1UDP协议的特征

1. 无连接：知道对端的端口号和IP地址就直接进行传输，不需要建立连接
2. 不可靠(中性词)：他只管将数据传送给对端，数据是否被接收，是否被遗漏他都不关心，不会返回错误信息(没有确认机制，没有重传机制)
3. 面向数据报：对数据全部接收，照样发送，不能够灵活的控制读写数据得到次数和数量

面向数据报的解释：

用UDP传输100个字节的数据，如果发送端调用一次sendto发送100个字节的数据，那么接受端就要调用一次recvfrom接收100个字节的，不能够循环调用10次sendto来发送10字节的数据，不能够拆分也不会进行合并。

2.2UDP协议的报头



如果检测和出错，就会直接丢弃

对于网络当中的每一层协议，我们都要思考两个问题

1.如何做到数据和报头的有效分离?

UDP报头是定长的8字节，可以将报头分离。报头的16位UDP长度减去8字节，就是数据长度。

2.如何做到向上交付?

传输层向上交付，交付给应用层，实际是交付给应用层的某个进程，UDP当中的目的端口号就是确认向上交付给应用层的哪个进程。

2.3UDP缓冲区的理解

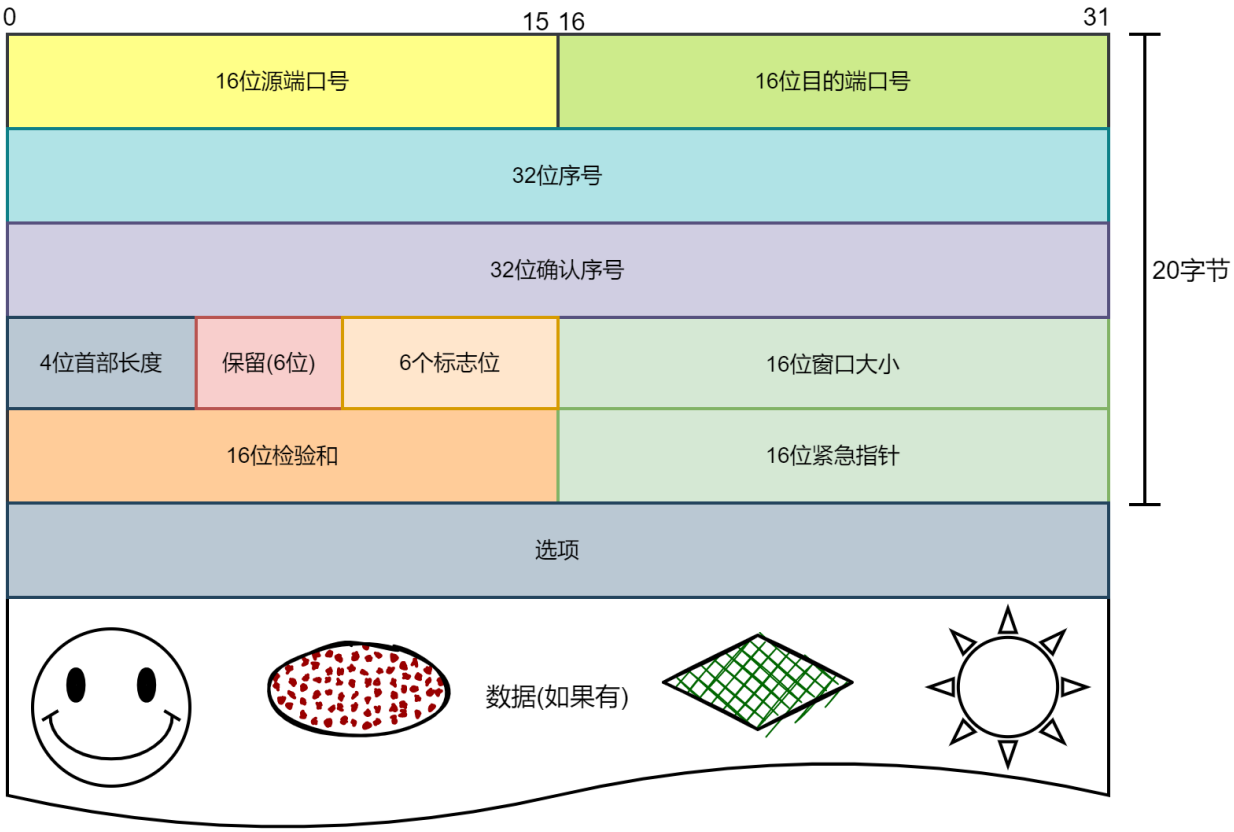
1. UDP没有真正意义上的发送缓冲区，调用sendto直接将数据交给内核，由内核将数据传给网络层。
2. UDP有接收缓冲区，但是这个缓冲区不能保证收到的UDP数据报顺序和发送UDP数据报的顺序一致，如果接收缓冲区满了，再到达UDP的数据会被直接丢弃，丢弃之后也不管了。
3. 顺序接收也是可靠性的一种。
4. 数据发送是一次性发送多条数据，发送数据的顺序和接收数据的顺序不一定一致，因为可能选择的网络路径不同。
5. UDP是一种无连接的协议，它可以在两台计算机之间进行数据传输。UDP 不像 TCP 那样提供可靠的数据传输机制，但是它具有更快的传输速度和更少的开销。
6. UDP 是全双工通信协议，这意味着它可以同时收发数据。它允许两台计算机在同一时间内进行双向通信。

3.TCP协议

3.1 TCP协议的特征

1. 有连接：在进行通信之前，会先在客户端和服务端之间构建连接，当链接成功之后才会进行通信
2. 可靠：他有确认机制，重传机制，当数据没有到达对端或者有遗失时，他会有补救措施
3. 面向字节流：对数据的发送和接收都是基于字节流的
4. 全双工：可以同时接收和发送数据

3.2 TCP报头



1．如何封装解包?

TCP包头里面包含有4位首部长度(单位是4字节)，代表由0000-1111(0-15),也就表明了报头的最大的长度是15×4=60字节，但是一般TCP的报头都是20个字节，也就是说包头里面的4位首部长度都是0101，知道了报头的大小，也就知道了如何进行封装和解包。

2．如何进行向上交付?

传输层向上交付，交付给应用层，实际是交付给应用层的某个进程，UDP当中的目的端口号就是确认向上交付给应用层的哪个进程。

报头里面各个指标的含义

1．16位源端口号：请求端发送请求的具体进程号码

2．16位目的端口号：接受请求的目的端的具体进程号码

3．32位序号：TCP传输数据是按字节流的，按字节为单位发送数据。在发送数据时，是一次性发送多条数据，接收端收到数据的顺序不一定和发送端发送数据的 顺序一致，因为可能走的网络路径不同，不一致的话，可能导致最终数据错误。TCP为了保证数据是有序的，会将发送的每一个字节的数据编号。接收方虽然在接收数据时不一定是有序的，但是在就收到数据后，会按照序号将数据排好序，放在接收缓冲区中。序列号保证了，数据的有序性。

4．32位确认序号：确认序列号是接收方收到数据后，向发送方确认应答时，在报文的确认序列号中填写的序号。确认序列号在原始序列号的基础上加1。有两个作用：

告诉发送方，在此确认序列号之间的数据，都收到了。

告诉发送方，下一次发送数据时的序列号。

5．4位首部长度：里面的长度是0000-1111(0-15)，代表报头的实际长度，另外它的单位是4字节

6．6个标志位：不同的标志位代表这不同的报头类型

- ACK：代表确认号是否有效
- SYN：请求建立连接，将SYN标识称作同步报文段
- RST：请求重新建立连接，将RST称作复位报文段
- PSH：提示接收端将接收缓冲区里面的数据读走
- URG：紧急指针，一般和16位紧急指针一起使用(该标志位一般很少使用)
- FIN：请求断开连接，将FIN也称作结束报文段

7. 16位窗口大小(流量控制)：表示的是接收缓冲区的剩余大小，通过该窗口来控制数据传输的速度
8. 16位紧急指针：一般和URG标志位一起使用，表示该报文数据需要紧急优先处理，不能继续排序，16位紧急指针指向的位置就是该数据的地址，但是一次只能移动一个字节的大小，该紧急指针一般很少会使用
9. 16位校验和：发送端进行填充，CRC校验，接收端校验不通过，则会认为数据不通过，就会认为数据有问题，此处的检验和不光包含TCP的首部，也会包含部分的TCP数据

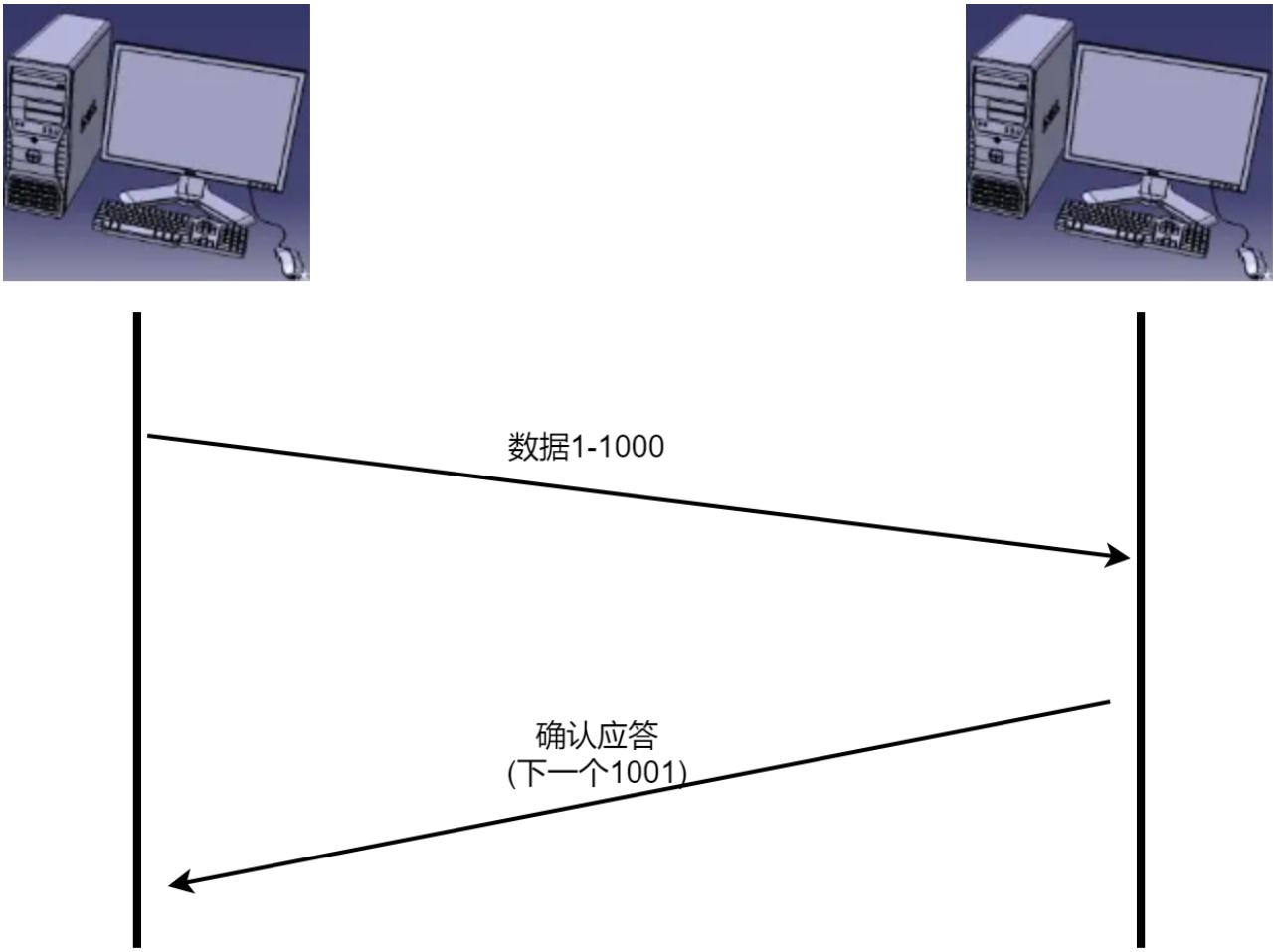
3.3 TCP可靠性机制

1. **检验和**

- tcp校验需要将ip伪首部、tcp报头、tcp数据分为16位的字，然后进行累加（如果总长度为奇数个字节，则在最后增添一个位都为0的字节 ），最后对累加的和进行按位取反即可。
- 这个16位的检验和是由发送端进行填写，然后由CRC进行校验，如果接收端的校验不通过，那就表明数据有问题。

2. **序列号**

TCP由于是面向字节流的，因此再进行传输的时候，它会将数据给按照字节流的形式进行传输，在这里每个发送请求的主机会给每个字节的数据分配一个序号，同时这里的序号不仅可以标识每一个数据，同时还让每一个数据进行有序的在字节流当中，便于主机AB的传输和收发。



3. **确认应答**

1. 而确认应答一般是反馈主机面对请求主机发送的一种序号，一般和ACK确认标志位一起使用，这里确认应答的序号一般是在请求端发来的序号基础之上 + 1，但是确认应答的深层含义是：反馈主机收到的确认序号，也就意味这在这个序号之前所有序号的数据已经确认收到，之后的序号目前还不能够确认
2. TCP的确认应答机制是对历史数据的确认，对于当前数据是不能够进行确认的
3. 由于TCP对于最后一条确认ACK不能够保证其是否遗失，因此TCP 协议也不是100%可靠的，他只是对于历史的数据100%可靠

4. **超时重传**

- 在进行数据传输的过程当中，主机A给主机B发送数据，这时可能会面临两个问题

1. 发送的数据在网络当中遗失
2. 主机B发送的确认标志位丢失

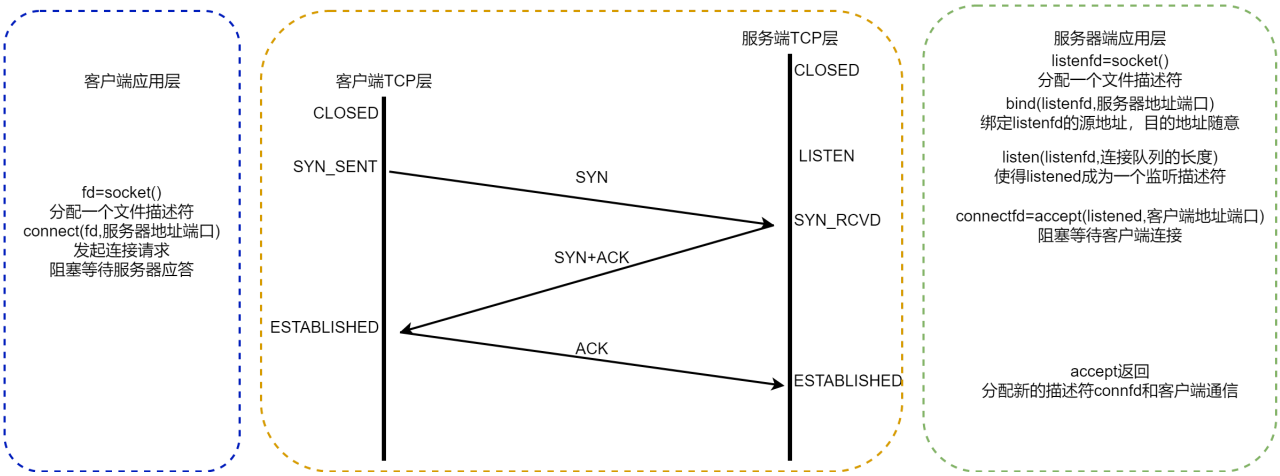
- 面对丢失的数据，主机A迟迟收不到来自主机B的确认ACK，这时TCP该怎么做？这时TCP就会采用超时重传机制

1. Linux中，超时以500ms为一个单位进行控制，每次判定重传时间都是500ms的整数倍。
2. 如果重发一次没有得到响应，会等待2×500ms进行重传。
3. 如果仍没有得到应答，等待4×500ms进行重传，依次类推，以指数形式增长。
4. 但是累计到一定次数的重传次数，TCP会认为对方主机或者网络出现异常，会强制关闭连接。

- 这里如果是主机B的确认ACK丢失的话，主机A会频繁的给主机B发送相同的数据，这时序列号的做种就会体现出来，当该数据的序号已经被主机B给确认过之后，主机A再发的数据就会被主机B给丢弃掉，达到一个去重的目的。

5. 连接管理

1. 三次握手



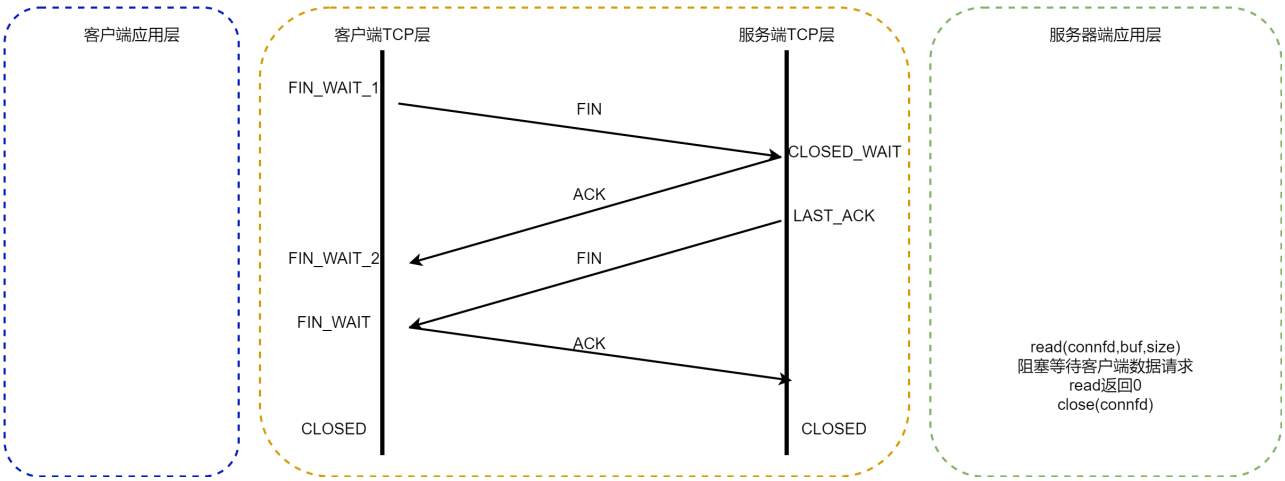
- 为什么是三次握手?

1. 握手的目的是两台主机之间进行通信，3次握手是双方之间进行连接的最少次数。
2. TCP协议是一种全双工的协议，因此两个主机之间必须要满足收发数据正常，第一第二次握手，可以保证主机A是满足条件的，但是第一第二次握手却只能够满足主机B是收正常的，但是不能够保证主机B是发正常的，只有当主机A发送第三次ACK被主机B收到之后，才能够说明主机B是发数据正常。

- 1，2，4，5....握手为什么不行

1. 1，2握手不满住主机A和主机B之间是否是正常的连接和全双工的通信协议，另外TCP协议的三次握手是最大概率的连接成功，是不能够保证100%连接成功的，因为第三次ACK可能会丢失。因此4，5次握手不是不可以，只是三次就能够成功多了话就会浪费两台主机之间的连接成本。
2. 这里主机A在进行完第二次握手之后，主机A就会认为连接就建立成功，但是有一定的时间间隔之后如果主机B收到主机A发送过来的ACK之后，主机B 才算建立连接是成功的

2. 四次挥手



1. 为什么是四次挥手?

因为断开连接是主机A和主机B双方的事情，一方想要断开连接需要给对方发送FIN，对方同意之后发送ACK，对方也需要给你发送FIN，当你同意之后也发送ACK，经过这四个步骤之后，通信双方才能够说明真正的断开了连接，因此需要至少四次挥手才行。

2. TCP协议规定，主动断开连接的一方处于TIME_WAIT状态，等待两个MSL(报文最大生存时间)，才能回到

CLOSED状态。为什么等待两个MSL?

- 保证两个传输方向上阻塞再网络里的报文都已经消散。
- 保证了最后一个ACK的到达，如果最后一个ACK丢失，服务器重发FIN，客户端等待两个MSL，可以收到重发的FIN。再丢失ACK的概率很小。

3. 四次挥手CLOSE_WAIT状态

- 连接由CLOSE_WAIT状态变为LAST_ACK状态，需要应用层调用close关闭套接字， 如果进程一直处于CLOSE_WAIT状态，说明在应用层的程序中有BUG，没有关闭关闭套接字。

6. **流量控制**

主机B的接收缓冲区处理数据是有限的，如果主机A发送缓冲区数据发得太快，接收缓冲区很快就被打满了，这个时候发送区再发送数据，数据会被丢弃，导致发送端需要重传数据，这样效率不高。

TCP会根据接收缓冲区的剩余缓冲区的大小，来决定发送数据的速度，这个机制叫做流量控制。

接收端一旦发现自己的接收缓冲区快满了，就会将窗口大小设置成一个更小的值通知发送方。发送方接收到窗口大小后，会减慢自己的发送速度。

如果接收缓冲区满了，就会将窗口设为0，这个时候发送方就不会发送数据，但是发送方怎么知道什么时候可以再发送数据呢?

两种方法：两种方法是同时运行的。这都是内核做的，与上层无关。

方法一：

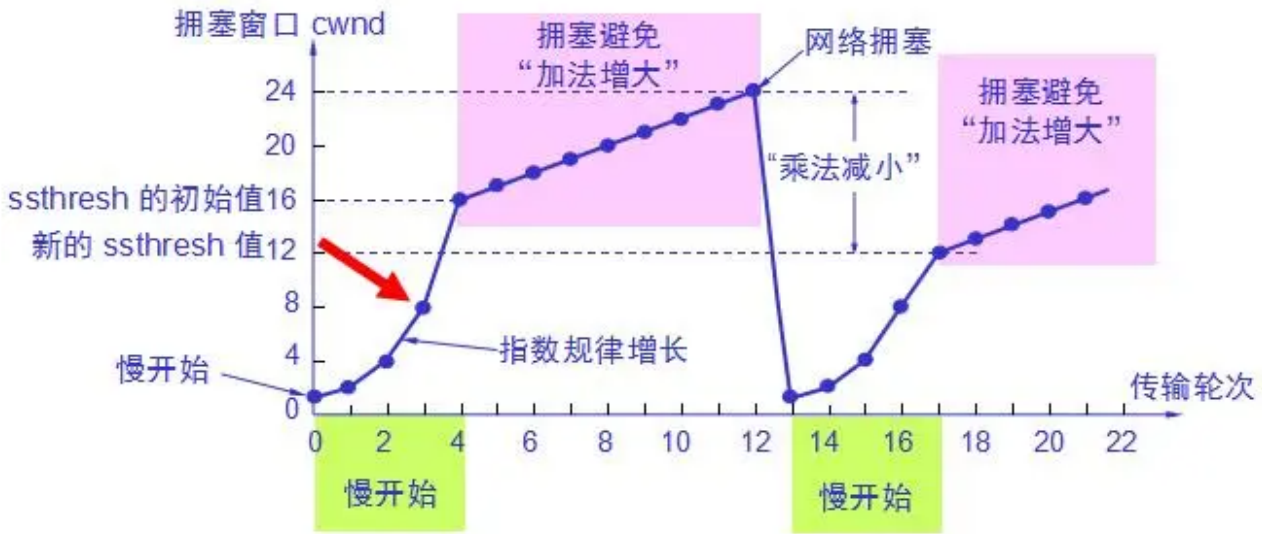
窗口探测，发送端隔一段时间，发送一个窗口探测数据段，让接收端响应回来字节的窗口大小。

方法二：

窗口更新通知：当接收方可以接收数据，会发送窗口更新通知给发送端。

如果窗口探测多次，接收缓冲区还是满的，发送端可以发送带有PSH标志位的报文给接收端，催进程取数据。如果还是满的，说明是应用层程序有BUG

7. **拥塞控制**



少量的数据报文丢失，采取超时重传可以补救，但是大量的数据丢失，就需要考虑阻塞控制了

前面讲述的都是发送端和接收端的控制情况，但是数据再发送端到接收端还需要经过网络的传输，如果某一时刻网络当中非常的拥挤，如果此时我们还继续往网络里面发送大量的数据，势必会造成网络

拥堵，更有可能将网络给冲垮。

据此我们引入阻塞窗口的概念，与前面的滑动窗口和接收缓冲区的窗口大小不同，阻塞窗口代表的是一个整数。

在发送端或者接收端发送一个ACK时，阻塞窗口就加上1

在最开始阶段，发送端会采取慢开始的节奏，此时的阻塞窗口就会处于基值1，随着传输轮次的增加，阻塞窗口会呈现指数的增加，当增加到某一个阈值的时候，阻塞窗口变成正相关增加(避免阻塞快速增加)，如果不幸增加阻塞的阈值，此时网络阻塞发送的数据会阻塞在网络当中，使得接收端无法及时发送ACK给发送端，发送端就会隔一段时间进行超时重传，此时阻塞的阈值变成原先的一般，阻塞窗口变成1，该过程持续这样下去。

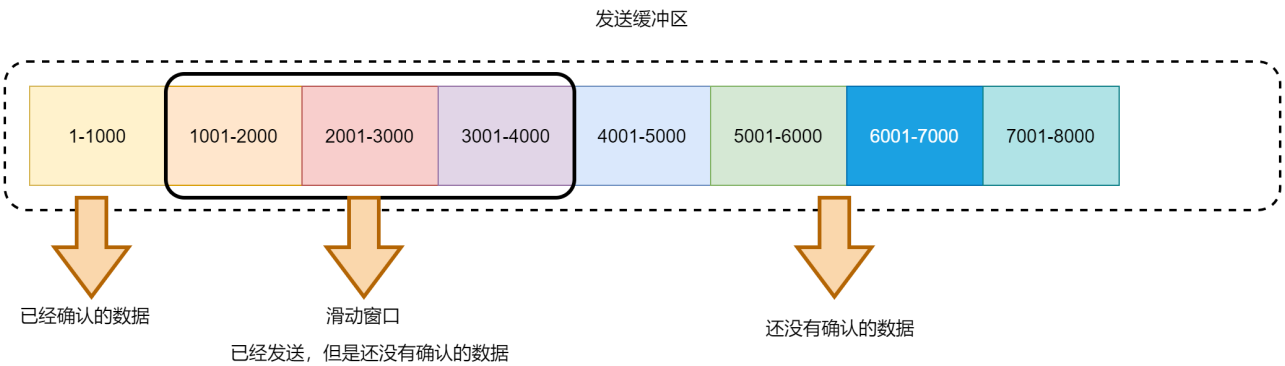
阻塞控制的目的是为了缓解某一时刻网络中数据量太多而带来的压力

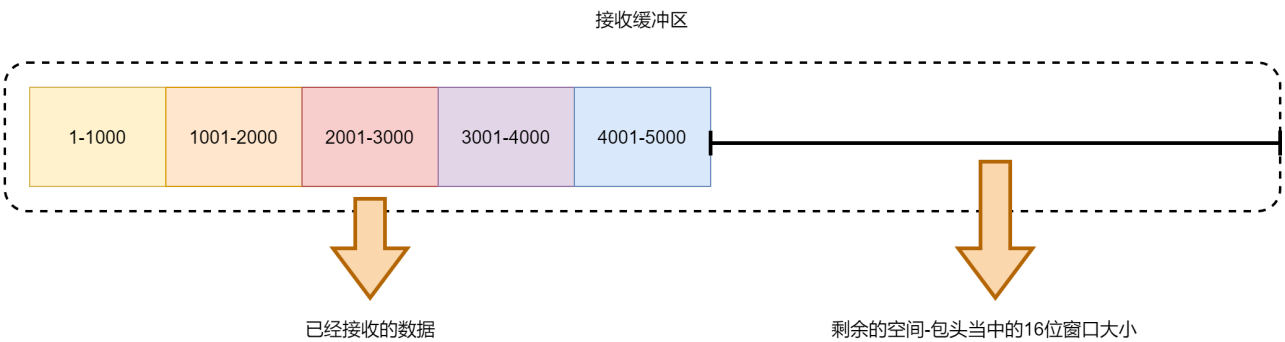
发送端发送数据的大小一般是阻塞窗口与缓冲区16位窗口大小中的最小值

在进行TCP 通信的时候，刚开始的时候网络的吞吐量是逐渐增加的，随着网络发生拥挤，吞吐量就会立刻下降

3.4 TCP提高性能机制

1. 滑动窗口





1. 滑动窗口的大小不是一个固定的值，而是一个动态变化的值
2. 滑动窗口的本质其实是两个指针，分别指向滑动窗口的前后两部分
3. 当滑动窗口左边的数据收到了来自接收端发来的ACK确认，那么滑动窗口的左边指针就会进行右移
4. 当接收端的16位窗口大小不足或者太小的时候，那么滑动窗口的右指针就会移动很慢，或者干脆直接不移
5. 当发送端收到来自接收端发来的某个序号的ACK时候，代表这个序号前面所有的数据都已经被接收端收到，哪怕目前还没有收前面数据的确认序号

6. 当滑动窗口里面的某一个序号的数据丢失，其余的序号数据没有丢失，那么丢失数据及其后面的所有确认序号都将收不到，只有发送端会经过超时重传之后，这个丢失数据的序号被接收端确认收到之后，这个数据及其后面的数据确认序号才会被发送出来

2. **快速重传**

连续收到3次相同确认序号的应答，直接重发数据的机制是高速重发机制，也叫快重发。

与超时重传机制的区别：

- 超时重发是在没有收到响应后的一段时间才会重发数据。(基础，常常使用)
- 而快重传是在短时间内快速的发送报文，以达到某种反响的结果(偶尔使用)
- 当发送报文数小于3时，快重发不起效。

3. **延迟应答**

延迟应答主要使用在接收端的缓冲区当中，如果接收端的缓冲区不足(发送端发送数据的速率将会大大降低)，这时如果接收端缓冲区的发送ACK应答较快，此时多次发送的16位窗口大小中值就会比较的小，这就会影响发送端的发送速率，但是如果接收端缓冲区延迟发送ACK，这时如果应用层将接收端缓冲区中的内容拿走，那么接收端缓冲区的可用空间变大，那么发送端的发送数据的速率将会变快，有利于提高TCP传输数据的效率

4. **捎带应答**

捎带应答，顾名思义就是在发送某种报文的时候，顺带也发一下其他的信息，最经典的栗子就是在三次握手的时候，第二次握手发送的数据当中接收端除了发送ACK确认报文同时，还捎带发送了SYN连接报文。

3.5 TCP的两个问题

1. **面向字节流**

TCP当中，他在应用层将数据拷贝到发送缓冲区之后，应用层就不管了，而在发送缓冲区和接收缓冲区，里面的数据都是以字节的形式存在于缓冲区当中

2. **粘包问题**

1. 上面说到，TCP当中的数据是以字节的形式放在缓冲区当中，当有许多的数据都放在缓冲区当中时，就会出现不知道每一个TCP数据的边界，当上层取数据的时候，少取或者多取其他的数据到这个TCP当中，这就是出现了粘包问题。
2. 这个问题对于TCP，他是没有能力来进行避免或者说处理的，于是这个问题就只能交给应用层来进行处理，在http协议的报头当中的Content_length就是来获取有效载荷的大小，就是来避免粘包问题
3. 但是在UDP当中，由于它的报头是固定的20字节，有效载荷在报头里面有，因此UDP不存在说粘包问题
4. 避免粘包问题的措施，定制定长的报头和有效载荷，明确两个包的边界值

3.6 TCP异常

3.7 理解Listen的第二个参数