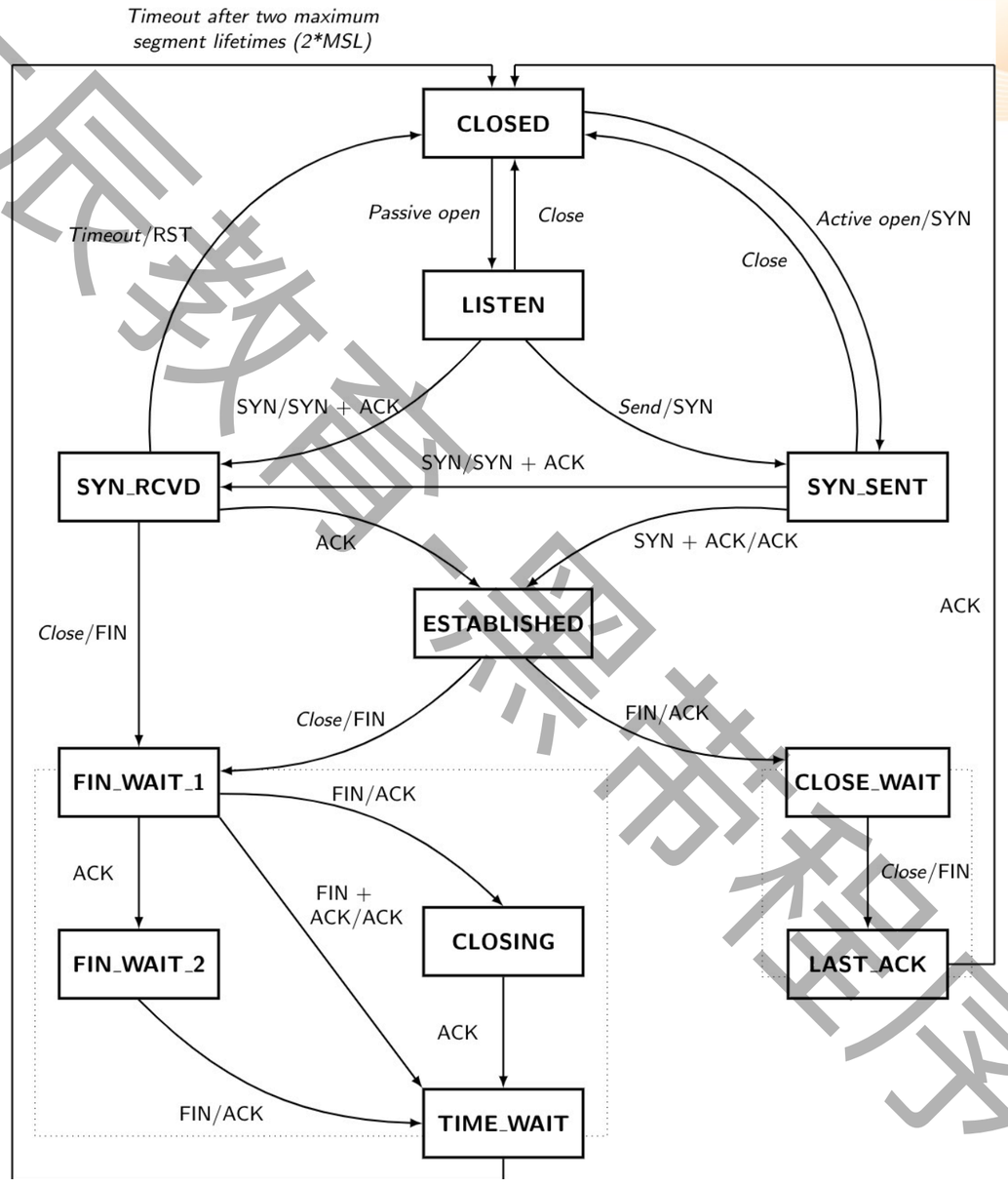
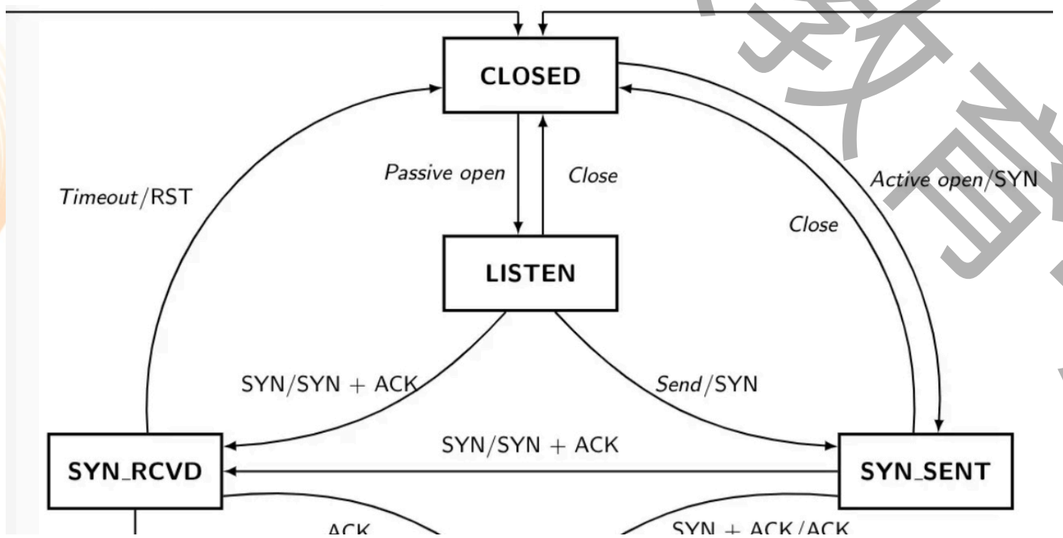
TCP 11 种状态变迁和模拟


Timeout after two maximum segment lifetimes (2*MSL)
CLOSED
Passive open
Close
Active open/SYN
Timeout/RST
Close
LISTEN
SYN/SYN + ACK
Send/SYN
SYN_RCVD
SYN/SYN + ACK
SYN_SENT
ACK
SYN + ACK/ACK
ACK
ESTABLISHED
Close/FIN
Close/FIN
FIN/ACK
FIN_WAIT_1
CLOSE_WAIT
FIN/ACK
ACK
FIN + ACK/ACK
Close/FIN
FIN_WAIT_2
CLOSING
LAST_ACK
ACK
FIN/ACK
TIME_WAIT

1、CLOSED

这个状态是一个「假想」的状态，是 TCP 连接还未开始建立连接或者连接已经彻底释放的状态。因此CLOSED状态也无法通过 netstat 或者 ss 等工具看到。

CLOSED 状态切换



从 CLOSE 状态转换为其它状态有两种可能：主动打开（Active Open）和被动打开（Passive Open）

- 被动打开：一般来说，服务端会监听一个特定的端口，等待客户端的新连接，同时会进入LISTEN状态，这种被称为「被动打开」
- 主动打开：客户端主动发送一个SYN包准备三次握手，被称为「主动打开（Active Open）」

2、LISTEN

一端（通常是服务端）调用 bind、listen 系统调用监听特定端口时进入到LISTEN状态，等待客户端发送 SYN 报文三次握手建立连接。

在 Java 中只用一行代码就可以构造一个 listen 状态的 socket。

```
ServerSocket serverSocket = new ServerSocket(9999);
```

ServerSocket 的构造器函数最终调用了 bind、listen，接下来就可以调用 accept 接收客户端连接请求了。

C/C++ 版本

```
int main() {
    short port = 9090;
    int fd = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in serv_addr = {0};

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(port);

    int optval = 1;
    setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval));

    if (::bind(fd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))) {
        exit(1);
    }
    if (listen(fd, 4096) < 0) {
        exit(1);
    }
    cout << "fd: " << fd << endl;
    std::cin.get();
}
```

3、SYN-SENT

客户端发送 SYN 报文等待 ACK 的过程进入 SYN-SENT状态。同时会开启一个定时器，如果超时还没有收到 ACK 会重发 SYN。

SYN-SENT 模拟

使用 packetdrill 可以非常快速的构造一个处于SYN-SENT状态的连接

```
// 新建一个 socket
+0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3

// 客户端 connect
+0 connect(3, ..., ...) = -1

+0 `sleep 1000000`
```

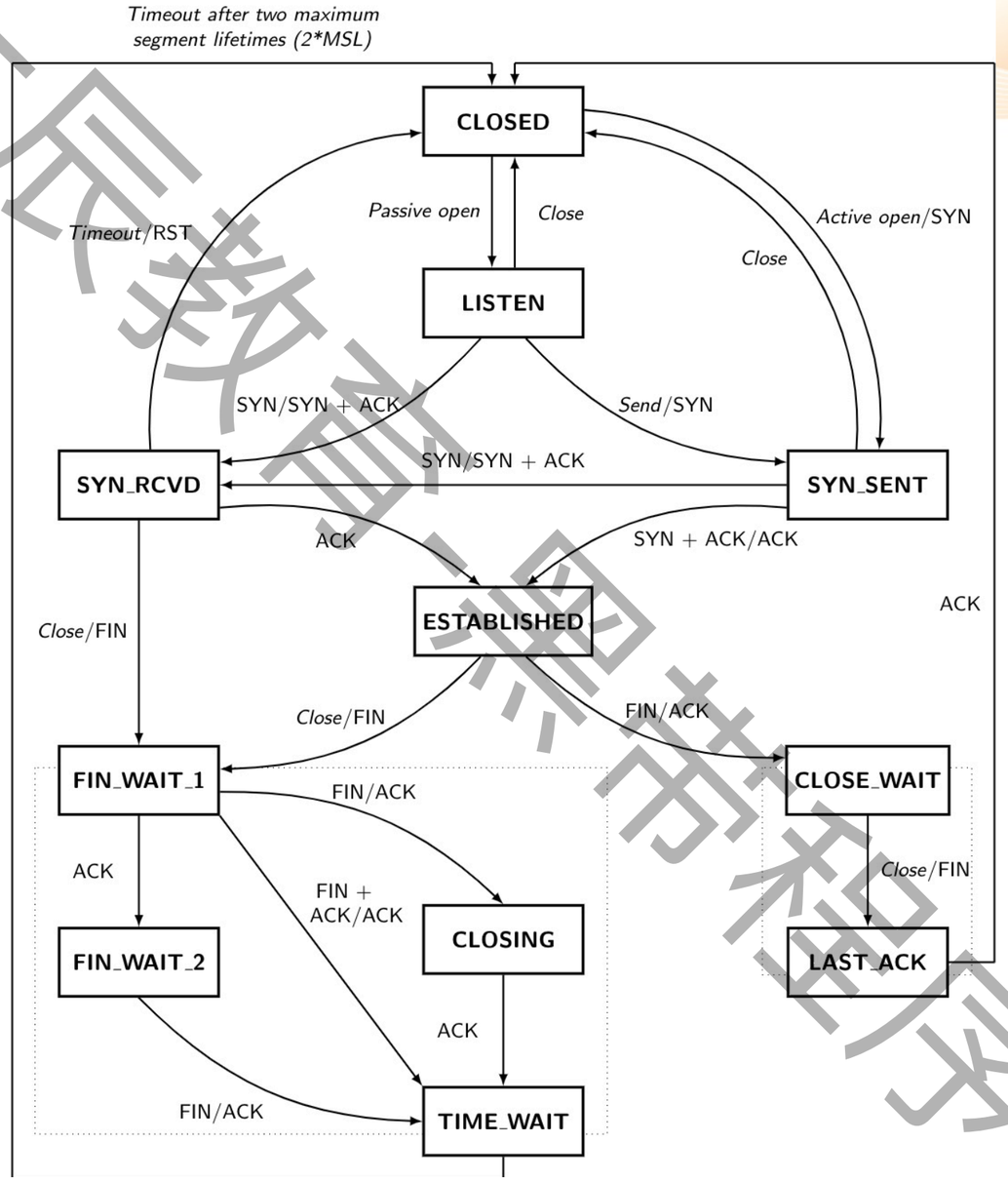
4、SYN-RCVD

服务端收到SYN报文以后会回复 SYN+ACK，然后等待对端 ACK 的时候进入SYN-RCVD

```
--tolerance_usecs=1000000
0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
+0 setsockopt(3, SOL_TCP, TCP_NODELAY, [1], 4) = 0
+0 setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
+0 bind(3, ..., ...) = 0
+0 listen(3, 1) = 0

+0  < S 0:0(0) win 65535  <mss 100>
+0  > S. 0:0(0) ack 1 <...>
// 故意注释掉下面这一行
// +.1 < . 1:1(0) ack 1 win 65535

+0 `sleep 1000000
```

5、ESTABLISHED

SYN-SENT或者SYN-RCVD状态的连接收到对端确认ACK以后进入ESTABLISHED状态，连接建立成功。


Timeout after two maximum segment lifetimes (2*MSL)
CLOSED
Passive open
Close
Active open/SYN
Timeout/RST
Close
LISTEN
SYN/SYN + ACK
Send/SYN
SYN_RCVD
SYN/SYN + ACK
SYN_SENT
ACK
SYN + ACK/ACK
ACK
ESTABLISHED
Close/FIN
Close/FIN
FIN/ACK
FIN_WAIT_1
CLOSE_WAIT
FIN/ACK
ACK
FIN + ACK/ACK
Close/FIN
FIN_WAIT_2
CLOSING
LAST_ACK
ACK
FIN/ACK
TIME_WAIT

ESTABLISHED 状态变迁

ESTABLISHED状态的连接有两种可能的状态转换方式:

- 调用 close 等系统调用主动关闭连接，这个时候会发送 FIN 包给对端，同时自己进入FIN-WAIT-1状态
- 收到对端的 FIN 包，执行被动关闭，收到 FIN 包以后会回复 ACK，同时自己进入CLOSE_WAIT状态

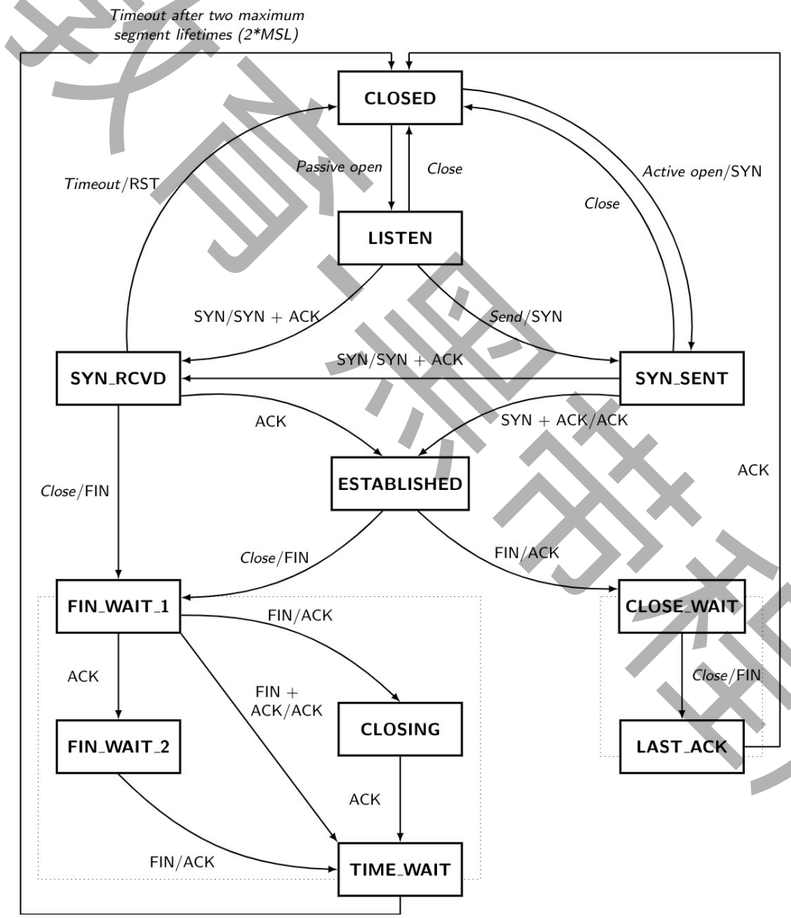
6、FIN-WAIT-1

主动关闭的一方发送了 FIN 包，等待对端回复 ACK 时进入FIN-WAIT-1状态。

FIN-WAIT-1 模拟重现

```
--tolerance_usecs=1000000
0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
+0 bind(3, ..., ...) = 0
+0 listen(3, 1) = 0

+0  < S 0:0(0) win 65535  <mss 100>
+0  > S. 0:0(0) ack 1 <...>
.1 < . 1:1(0) ack 1 win 65535

+.1 accept(3, ..., ...) = 4

+.1 close(4) = 0  // 服务端主动断开连接

+0 `sleep 1000000`
```

FIN_WAIT1状态的切换:



FIN_WAIT1状态的切换:

- 当收到 ACK 以后，FIN-WAIT-1状态会转换到FIN-WAIT-2状态
- 当收到 FIN 以后，会回复对端 ACK，FIN-WAIT-1状态会转换到CLOSING状态
- 当收到 FIN+ACK 以后，会回复对端 ACK，FIN-WAIT-1状态会转换到TIME_WAIT状态，跳过了FIN-WAIT-2状态

7、FIN-WAIT-2

处于 FIN-WAIT-1状态的连接收到 ACK 确认包以后进入FIN-WAIT-2状态，这个时候主动关闭方的 FIN 包已经被对方确认，等待被动关闭方发送 FIN 包。

FIN-WAIT-2 模拟重现

```
--tolerance_usecs=1000000
0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
+0 bind(3, ..., ...) = 0
+0 listen(3, 1) = 0

+0  < S 0:0(0) win 65535  <mss 100>
+0  > S. 0:0(0) ack 1 <...>
// 故意注释掉下面这一行
.1 < . 1:1(0) ack 1 win 65535

+.1  accept(3, ..., ...) = 4

// 服务端主动断开连接
+.1 close(4) = 0

// 向协议栈注入 ACK 包，模拟客户端发送了 ACK
+.1 < . 1:1(0) ack 2 win 257

+0 `sleep 1000000`
```

8、CLOSE-WAIT(被动关闭方)

当有一方想关闭连接的时候，调用 close 等系统调用关闭 TCP 连接会发送 FIN 包给对端，这个被动关闭方，收到 FIN 包以后进入CLOSE-WAIT状态。

CLOSE-WAIT 模拟

```
--tolerance_usecs=1000000
0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
+0 bind(3, ..., ...) = 0
+0 listen(3, 1) = 0

+0  < S 0:0(0) win 65535  <mss 100>
+0  > S. 0:0(0) ack 1 <...>
// 故意注释掉下面这一行
.1 < . 1:1(0) ack 1 win 65535

+.1  accept(3, ..., ...) = 4

// 向协议栈注入 FIN 包，模拟客户端发送了 FIN，主动关闭连接
+.1 < F. 1:1(0) win 65535  <mss 100>
// 预期协议栈会发出 ACK
+0 > . 1:1(0) ack 2 <...>
+0 `sleep 1000000`
```

9、TIME-WAIT（主动关闭方）

TIME-WAIT可能是所有状态中面试问的最频繁的一种状态了。这个状态是收到了被动关闭方的 FIN 包，发送确认 ACK 给对端，开启 2MSL 定时器，定时器到期时进入 CLOSED 状态，连接释放。

TIME-WAIT 模拟

```
0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
+0 bind(3, ..., ...) = 0
+0 listen(3, 1) = 0

+0  < S 0:0(0) win 65535  <mss 100>
+0  > S. 0:0(0) ack 1 <...>
.1 < . 1:1(0) ack 1 win 65535

+.1  accept(3, ..., ...) = 4

// 服务端主动断开连接
+.1 close(4) = 0
+0 > F. 1:1(0) ack 1 <...>

// 向协议栈注入 ACK 包，模拟客户端发送了 ACK
+.1 < . 1:1(0) ack 2 win 257

// 向协议栈注入 FIN，模拟服务端收到了 FIN
+.1 < F. 1:1(0) win 65535  <mss 100>

+0 `sleep 1000000`
```

10、LAST-ACK（被动关闭方）

LAST-ACK 顾名思义等待最后的 ACK。是被动关闭的一方，发送 FIN 包给对端等待 ACK 确认时的状态。

LAST-ACK 模拟重现

```
0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
+0 bind(3, ..., ...) = 0
+0 listen(3, 1) = 0

+0  < S 0:0(0) win 65535  <mss 100>
+0  > S. 0:0(0) ack 1 <...>
.1 < . 1:1(0) ack 1 win 65535

+.1  accept(3, ..., ...) = 4

// 向协议栈注入 FIN 包，模拟客户端发送了 FIN，主动关闭连接
+.1 < F. 1:1(0) win 65535  <mss 100>
// 预期协议栈会发出 ACK
+0 > . 1:1(0) ack 2 <...>

+.1 close(4) = 0
// 预期服务端会发出 FIN
+0 > F. 1:1(0) ack 2 <...>

+0 `sleep 1000000`
```

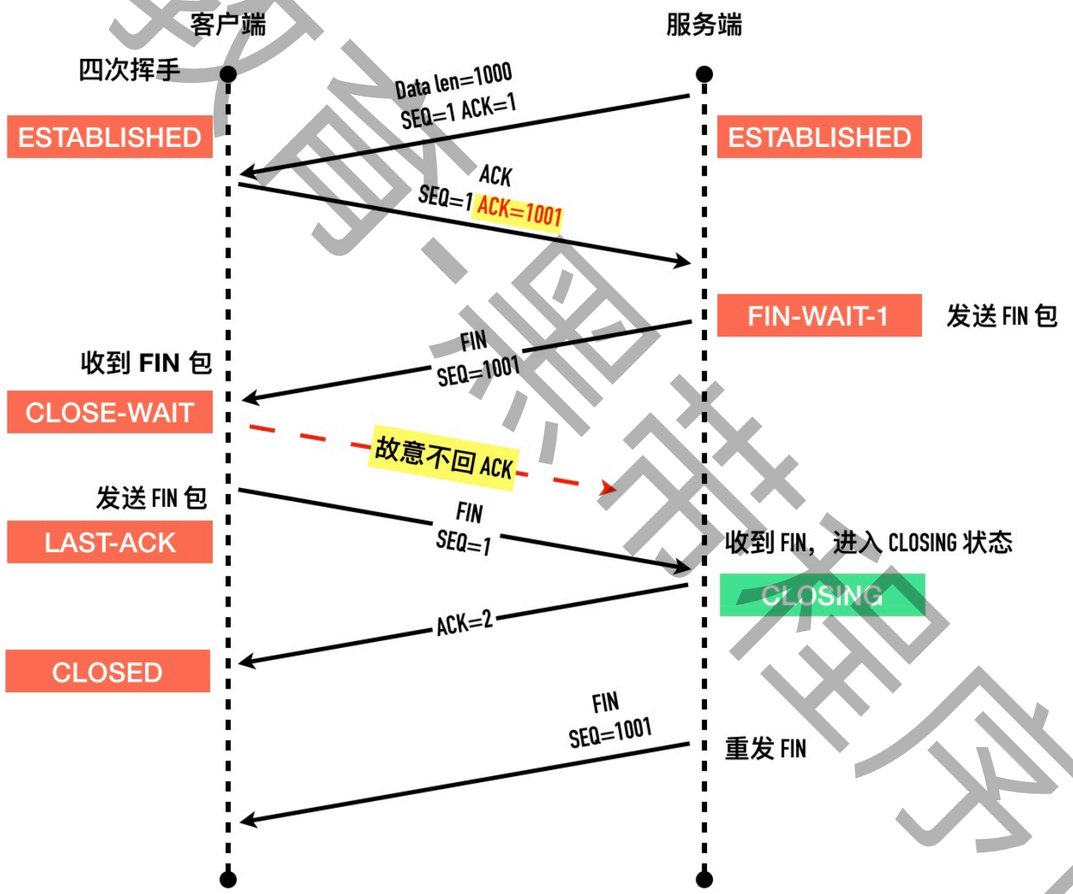
11、CLOSING

CLOSING状态比较多在「同时关闭」的情况下出现。这里的同时关闭中的「同时」其实并不是时间意义上的同时，而是指的是在发送 FIN 包还未收到确认之前，收到了对端的 FIN 的情况。

CLOSING 模拟

```
// 服务端随便传输一点数据给客户端
+0.100 write(4, ..., 1000) = 1000
// 断言服务端会发出 1000 字节的数据
+0 > P. 1:1001(1000) ack 1 <...>

// 确认 1000 字节数据
+0.01 < . 1:1(0) ack 1001 win 257

// 服务端主动断开，会发送 FIN 给客户端
+.1 close(4) = 0
// 断言协议栈会发出 ACK 确认（服务端->客户端）
+0 > F. 1001:1001(0) ack 1 <...>

// 客户端在未对服务端的 FIN 做确认时，也发出 FIN 要求断开连接
+.1 < F. 1:1(0) ack 1001 win 257

// 断言协议栈会发出 ACK 确认（服务端->客户端）
+0 > . 1002:1002(0) ack 2 <...>

// 故意不回 ACK，让连接处于 CLOSING 状态
// +.100 < . 2:2(0) ack 1002 win 25
```

交互过程



1、下列TCP连接建立过程描述正确的是：

- A、服务端收到客户端的 SYN 包后等待 2*MSL 时间后就会进入 SYN_SENT 状态
- B、服务端收到客户端的 ACK 包后会进入 SYN_RCVD 状态
- C、当客户端处于 ESTABLISHED 状态时，服务端可能仍然处于 SYN_RCVD 状态
- D、服务端未收到客户端确认包，等待 2*MSL 时间后会直接关闭连接

2、TCP连接关闭，可能有经历哪几种状态：

- A、LISTEN
- B、TIME-WAIT
- C、LAST-ACK
- D、SYN-RCVD