嵌入式系统工程师

Socket编程

大纲

- 概述
- Socket编程编程(TCP)
  - 热身
  - 创建套接字
  - 服务器
  - 客户端
  - 数据传输
  - 关闭连接

大纲

- 概述
- Socket编程编程(TCP)
  - 热身
  - 创建套接字
  - 服务器
  - 客户端
  - 数据传输
  - 关闭连接

概述

- ARPANet是我们熟知的Internet的前身
- 在ARPANet发展的同时,UNIX也在迅速发展,加州大学Berkeley分校开发出具有自己风格的UNIX,我们通常称之为BSD
- 20世纪80年代初，在系统内实现了TCP/IP协议的4.1BSD和4.2BSD被相继开发出来

概述

- 加州大学Berkeley分校在BSD内实现了针对于TCP/IP协议应用程序编程接口，即Socket
  - 网络间通信要解决的是不同主机进程间的通信
    - 需要解决的首要问题是网络间进程标识问题
    - 以及多重协议的识别问题

- 随着UNIX操作系统的广泛应用，Socket成为最流行的网络通信应用程序的开发接口

大纲

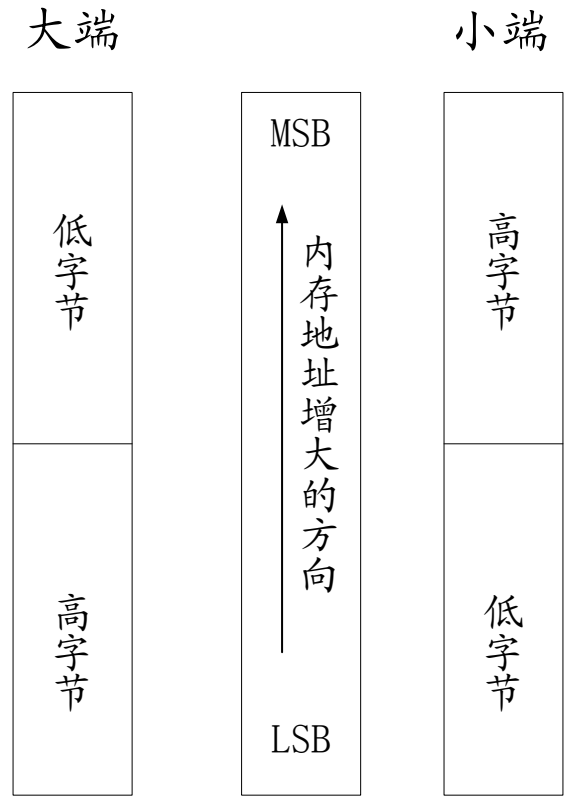
- 概述
- Socket编程编程(TCP)
  - 热身
  - 创建套接字
  - 服务器
  - 客户端
  - 数据传输
  - 关闭连接

热身——字节序

- Socket是BSD提供的网络应用编程接口，现在它已经是网络编程中的标准
- Socket是一种特殊的进程间通信方式，不同机器上的进程都可以使用这种方式进行通信
  - 网络中的数据传输是一种I/O操作
  - read、write、close操作可应用于Socket描述符
  - Socket是一种文件描述符，代表了一个通信管道的一个端点
  - 在Socket类型的文件描述符上，可以完成建立连接，数据传输等操作
  - 常用的Socket类型有两种：
    - 1. 流式Socket：SOCK_STREAM，提供面向连接的Socket
    - 2. 数据报式Socket：SOCK_DGRAM，提供面向无连接的Socket

热身——字节序

- 字节序
  - 概念：是指多字节数据的存储顺序
  - 分类：
    - 小端格式：将低位字节数据存储在低地址
    - 大端格式：将高位字节数据存储在低地址



思考：0x0102，大端如何存储，小端又如何存储

热身——字节序

- 字节序
  - 特点：
    - 网络协议指定了通讯字节序--大端
    - 只有在多字节数据处理时才需要考虑字节序
    - 运行在同一台计算机上的进程相互通信时，一般不用考虑字节序
    - 异构计算机之间通讯，需要转换自己的字节序为网络字节序

热身——字节序

- 确定主机字节序程序

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
    union{
        short s;
        char c[sizeof(short)];
    }un;

    un.s = 0x0102;

    if( (un.c[0] == 1) && (un.c[1] == 2))
    {
        printf("big-endian\n");
    }
    else if((un.c[0] == 2) && (un.c[1] == 1))
    {
        printf("little-endian\n");
    }

    return 0;
}
```

热身——字节序

- 字节序转换函数
- 主机字节序数据转换成网络字节序数据
  - uint32_t htonl(uint32_t hostint32);
  - uint16_t htons(uint16_t hostint16);
    - 以上返回网络字节序数据
- 网络字节序数据转换成主机字节序数据
  - uint32_t ntohl(uint32_t netint32);
  - uint16_t ntohs(uint16_t netint16);
    - 以上返回主机字节序数据

热身——字节序

- 字节序转换函数
  - uint32_t htonl(uint32_t hostint32);
  - 功能:
    - 将32位主机字节序数据转换成网络字节序数据
  - 参数:
    - uint32_t: unsigned int
    - hostint32: 待转换的32位主机字节序数据
  - 返回值:
    - 成功: 返回网络字节序的值
  - 头文件: #include <arpa/inet.h>

热身——字节序

- 字节序转换函数
  - uint16_t htons(uint16_t hostint16);
  - 功能:
    - 将16位主机字节序数据转换成网络字节序数据
  - 参数:
    - uint16_t: unsigned short int
    - hostint16: 待转换的16位主机字节序数据
  - 返回值:
    - 成功: 返回网络字节序的值
  - 头文件: #include <arpa/inet.h>

热身——字节序

- 字节序转换函数
  - uint32_t ntohl(uint32_t netint32);
  - 功能:
    - 将32位网络字节序数据转换成主机字节序数据
  - 参数:
    - uint32_t:　　unsigned int
    - netint32:　　待转换的32位网络字节序数据
  - 返回值:
    - 成功: 返回主机字节序的值
  - 头文件: #include <arpa/inet.h>

热身——字节序

- 字节序转换函数
  - uint16_t ntohs(uint16_t netint16);
  - 功能:
    - 将16位网络字节序数据转换成主机字节序数据
  - 参数:
    - uint16_t:　　unsigned short int
    - netint16:　　待转换的16位网络字节序数据
  - 返回值:
    - 成功: 返回主机字节序的值
  - 头文件: #include <arpa/inet.h>

热身——字节序

- 示例

```c
#include <stdio.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    int a = 0x01020304;
    short int b = 0x0102;

    printf("htonl(%08x) = %08x\n", a, htonl(a));
    printf("htons(%04x) = %04x\n", b, htons(b));

    return 0;
}
```

- 结果

```
edu@edu-T:~/share/test$ ./test
htonl(01020304) = 04030201
htons(0102) = 0201
edu@edu-T:~/share/test$
```

热身——套接字地址结构

- sockaddr_in套接字地址结构:
  - 在IPv4因特网域(AF_INET)中,套接字地址结构用sockaddr_in命名

```
struct in_addr{
        in_addr_t    s_addr;              //4字节
    };
struct sockaddr_in{
        sa_family_t      sin_family;     //2字节
        in_port_t        sin_port;       //2字节
        struct in_addr   sin_addr;       //4字节
        unsigned char    sin_zero[8];    //8字节
    };
```

  - 头文件: #include <netinet/in.h>

热身——通用套接字地址结构

- sockaddr通用套接字地址结构
  - 地址标识了特定通信域中的套接字端点，地址格式与特定通信域相关，为了使不同格式地址能被传入套接字函数，地址被强制转换成通用套接字地址结构

```
struct sockaddr{
    sa_family_t sa_family;      //2字节
    char        sa_data[14];    //14字节
 };
```

  - 头文件：#include <netinet/in.h>

热身——地址转换函数

- `int inet_pton(int family, const char *strptr, void *addrptr);`
- 功能:
  - 将点分十进制数串转换成32位无符号整数
- 参数:
  - family　　协议族
  - strptr　　点分十进制数串
  - addrptr　　32位无符号整数的地址
- 返回值:
  - 成功: 1
  - 失败: 其它
- 头文件: `#include <arpa/inet.h>`

热身——地址转换函数

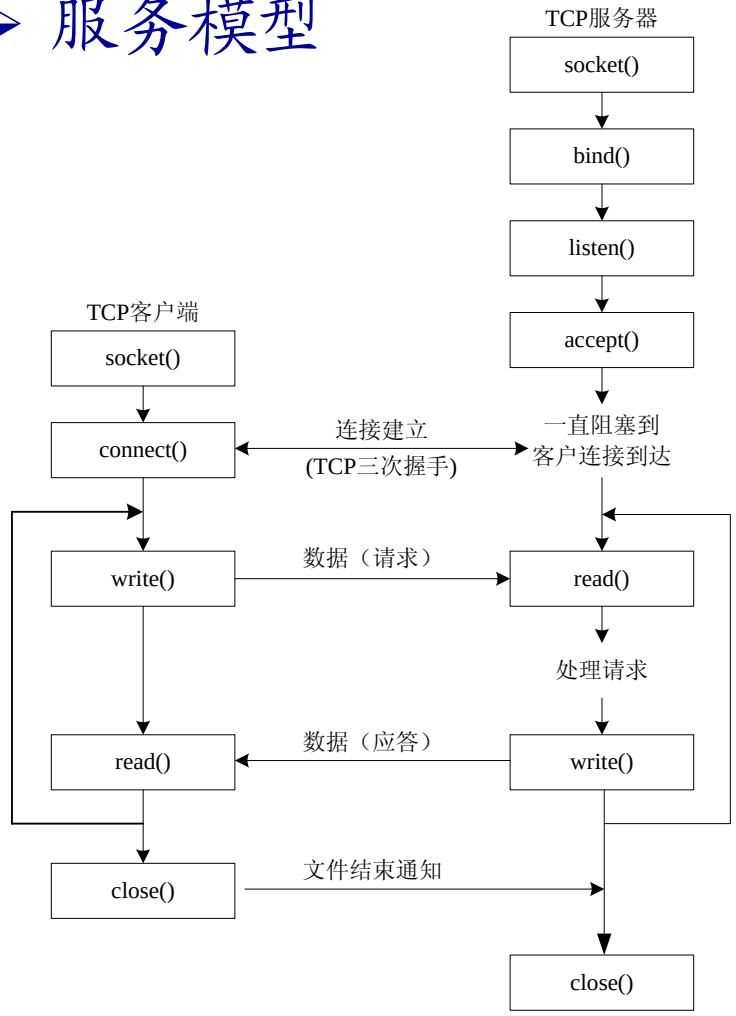
- const char *inet_ntop(int family, const void *addrptr, char *strptr, size_t len);
- 功能:
  - 将32位无符号整数转换成点分十进制数串
- 参数:
  - family　　协议族
  - addrptr　　32位无符号整数
  - strptr　　点分十进制数串
  - len　　strptr缓存区长度
- len的宏定义
  - #define INET_ADDRSTRLEN　16
  - #define INET6_ADDRSTRLEN　46　//for ipv6

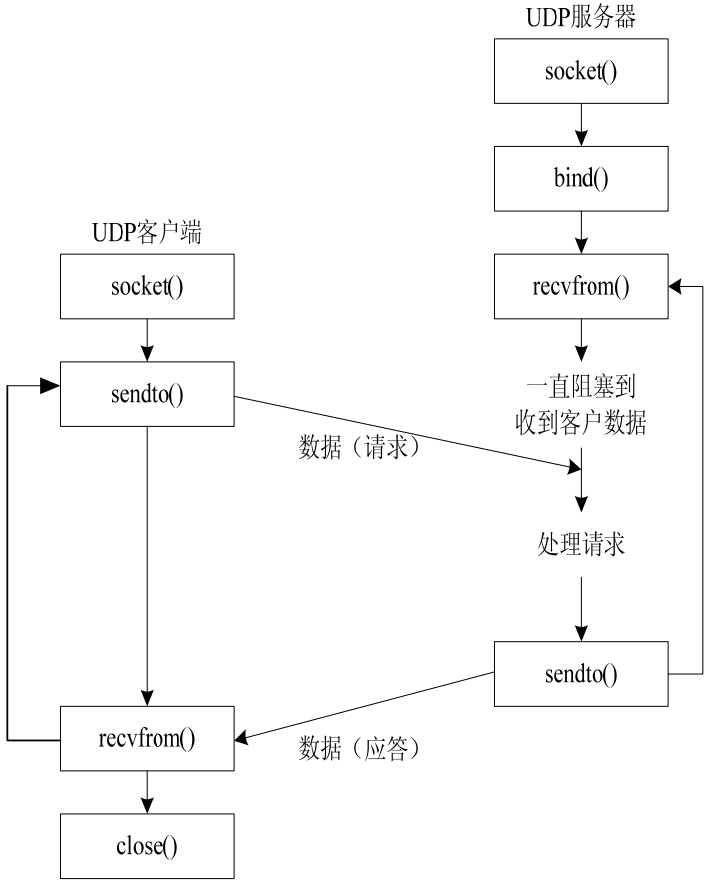
热身——地址转换函数

- 返回值:
  - 成功:则返回字符串的首地址
  - 失败:返回NULL
- 头文件: #include <arpa/inet.h>

热身——服务模型

➢ 服务模型


TCP服务器
socket()
bind()
listen()
accept()
一直阻塞到
客户连接到达
TCP客户端
socket()
connect()
连接建立
(TCP三次握手)
write()
数据（请求）
read()
处理请求
read()
数据（应答）
write()
close()
文件结束通知
close()


UDP服务器
socket()
bind()
recvfrom()
一直阻塞到
收到客户数据
处理请求
sendto()
UDP客户端
socket()
sendto()
数据（请求）
recvfrom()
数据（应答）
close()

大纲

- 概述
- Socket编程编程(TCP)
  - 热身
  - 创建套接字
  - 服务器
  - 客户端
  - 数据传输
  - 关闭连接

创建套接字

- 创建套接字是进行任何网络通信时必须做的第一步
- int socket(int family, int type, int protocol);
- 功能:
  - 创建一个用于网络通信的I/O描述符（套接字）
- 参数:
  - family: 协议族
    - AF_INET, AF_INET6, AF_LOCAL, AF_ROUTE, AF_KEY
  - type: 套接字类型
    - SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, SOCK_SEQPACKET
  - protocol: 协议类别
    - 0, IPPROTO_TCP, IPPROTO_UDP, IPPROTO_SCTP

创建套接字

- 返回值：套接字
- socket创建的套接字特点
  - 使用socket创建套接字时，系统不会分配端口
  - 使用socket创建的是主动套接字，但作为服务器，需要被动等待别人的连接
- 头文件：#include <sys/socket.h>

创建套接字

- Socket示例:
  - 创建TCP套接字

```
//创建TCP套接字
int sockfd = 0;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if(sockfd < 0)
{
    perror("socket");
    exit(-1);
}
```

大纲

- 概述
- Socket编程编程(TCP)
  - 热身
  - 创建套接字
  - 服务器
  - 客户端
  - 数据传输
  - 关闭连接

服务器

- 做为服务器需要具备的条件
  - 具备一个可以确知的地址，以便让别人找到我
  - 让操作系统知道你是一个服务器，而不是一个客户端
  - 等待连接的到来
- 对于面向连接的TCP协议来说，连接的建立才真正意味着数据通信的开始

服务器

- int bind(int sockfd,const struct sockaddr *myaddr, socklen_t addrlen);
- 功能:
  - 将本地协议地址与sockfd绑定
- 参数:
  - sockfd: socket套接字
  - myaddr: 指向特定于协议的地址结构指针
  - addrlen: 该地址结构的长度
- 返回值:
  - 成功: 返回0
  - 失败: 其他
- 头文件: #include <sys/socket.h>

- bind示例:

```
int err_log = 0;
struct sockaddr_in my_addr;
unsigned short port = 8000;

bzero(&my_addr, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(port);
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);

err_log = bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
if(err_log != 0)
{
    perror("binding");
    close(sockfd);
    exit(-1);
}
```

- 注意: INADDR_ANY 通配地址,值为0

服务器

- int listen(int sockfd, int backlog);
- 功能:
  - 将套接字由主动修改为被动
  - 使操作系统为该套接字设置一个连接队列，用来记录所有连接到该套接字的连接
- 参数:
  - sockfd: socket监听套接字
  - backlog: 连接队列的长度
- 返回值:
  - 成功: 返回0
  - 失败: 其他
- 头文件: #include <sys/socket.h>

- listen示例:

```
err_log = listen(sockfd, 10);
if(err_log != 0)
{
    perror("listen");
    close(sockfd);
    exit(-1);
}
```

服务器

- `int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);`
- 功能:
  - 从已连接队列中取出一个已经建立的连接，如果没有任何连接可用，则进入睡眠等待
- 参数:
  - sockfd: socket监听套接字
  - cliaddr: 用于存放客户端套接字地址结构
  - addrlen: 套接字地址结构体长度
- 返回值: 已连接套接字
- 注意: accept函数返回的是一个已连接套接字，这个套接字代表当前这个连接
- 头文件: `#include <sys/socket.h>`

- accept示例:

```
struct sockaddr_in client_addr;
char recv_buf[512] = "";
char cli_ip[INET_ADDRSTRLEN] = "";
socklen_t client_addr_len = sizeof(client_addr);

int connfd = accept(sockfd, (struct sockaddr*)&client_addr, &client_addr_len);
if(connfd < 0)
{
    perror("accept");
    close(sockfd);
    exit(-1);
}

inet_ntop(AF_INET, &client_addr.sin_addr, cli_ip, INET_ADDRSTRLEN);
printf("cli_ip:%s, port = %d\n", cli_ip, ntohs(client_addr.sin_port));

read(connfd, recv_buf, sizeof(recv_buf));
```

演示echo服务器

大纲

- 概述
- Socket编程编程(TCP)
  - 热身
  - 创建套接字
  - 服务器
  - 客户端
  - 数据传输
  - 关闭连接

客户端

- 作为客户端需要具备的条件
  - 知道服务器的IP地址以及端口号

客户端

- int connect(int sockfd,const struct sockaddr *addr,socklen_t len);
- 功能:
  - 主动跟服务器建立链接
  - 连接建立成功后才可以开始传输数据（对于TCP协议）
- 参数:
  - sockfd: socket套接字
  - addr: 需连接的服务器地址结构
  - addrlen: 地址结构体长度

客户端

- 返回值:
  - 成功：0
  - 失败：其他
- 注意：connect函数建立连接之后不会产生新的套接字
- 头文件：#include <sys/socket.h>

- connect示例:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char *argv[])
{
    //创建套接字
    int sockfd = 0;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
    }
```

客户端

- connect示例:

```
int err_log = 0;
unsigned short port = 8000;                 // 服务器的端口号
char *server_ip = "192.168.222.183";        // 服务器ip地址
struct sockaddr_in server_addr;

bzero(&server_addr,sizeof(server_addr));// 初始化服务器地址
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port);
inet_pton(AF_INET, server_ip, &server_addr.sin_addr);

// 主动连接服务器
err_log = connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
if(err_log != 0)
{
    perror("connect");
    close(sockfd);
    exit(-1);
}
```

练习echo客户端

大纲

- 概述
- Socket编程编程(TCP)
  - 热身
  - 创建套接字
  - 服务器
  - 客户端
  - 数据传输
  - 关闭连接

数据传输

- 当连接建立后，通信的两端便具备两个套接字
  - 套接字也是一种文件描述符，所以read、write函数可以用于从这个连接中取出或向其写入数据

```
    //通信
    while(1)
    {
        char send_buf[512] = "";
        char recv_buf[512] = "";

        fgets(send_buf, sizeof(send_buf), stdin);
        send_buf[strlen(send_buf)-1] = '\0';

        write(sockfd, send_buf, strlen(send_buf));
        read(sockfd, recv_buf, sizeof(recv_buf));
        printf("%s\n", recv_buf);
    }

    close(sockfd);
    return 0;
}
```

数据传输

- ssize_t send(int sockfd, const void* buf, size_t nbytes, int flags);
- 功能:
  - 用于发送数据
  - 注意: 不能用TCP协议发送0长度的数据包
- 参数:
  - sockfd: socket套接字
  - buf: 待发送数据缓存区的地址
  - nbytes: 发送缓存区大小(以字节为单位)
  - flags: 套接字标志(常为0)
- 返回值: 成功发送的字节数
- 头文件: #include <sys/socket.h>

数据传输

- ssize_t recv(int sockfd, void *buf,size_t nbytes, int flags);
- 功能:
  - 用于接收网络数据
- 参数:
  - sockfd: 套接字
  - buf: 指向接收网络数据的缓冲区
  - nbytes: 接收缓冲区的大小(以字节为单位)
  - flags: 套接字标志(常为0)
- 返回值: 成功接收到字节数
- 头文件: #include <sys/socket.h>

大纲

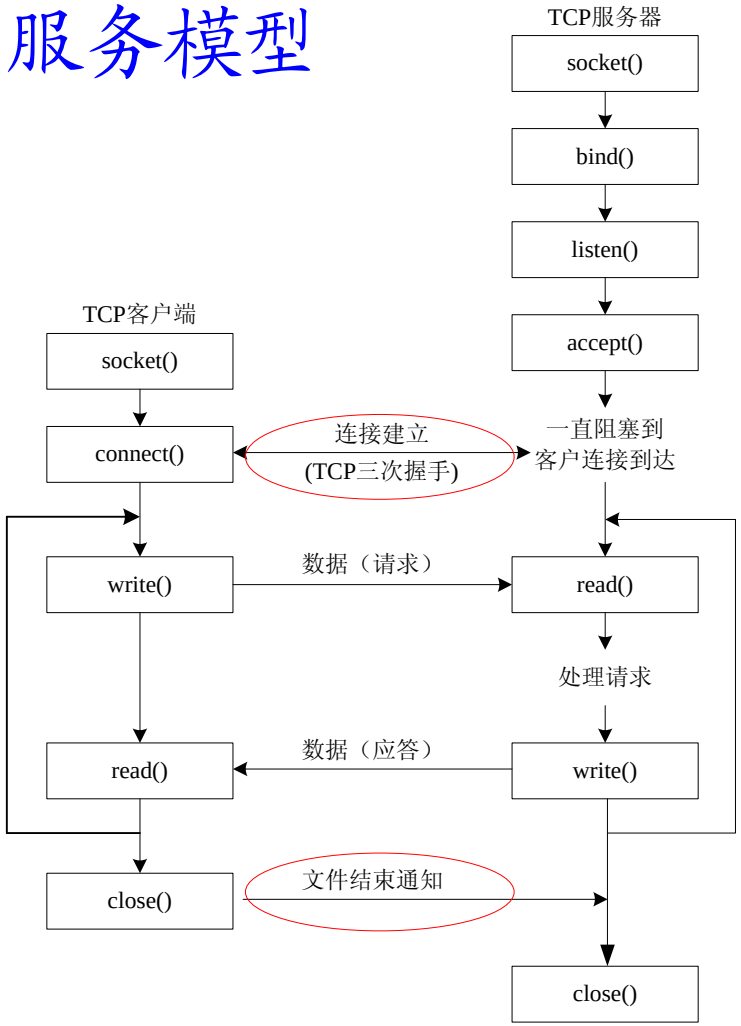
- 概述
- Socket编程编程(TCP)
  - 热身
  - 创建套接字
  - 服务器
  - 客户端
  - 数据传输
  - 关闭连接

关闭连接

- 使用close函数即可关闭套接字
  - 关闭一个代表已连接套接字将导致另一端接收到一个0长度的数据包
  - 做服务器时
    - 关闭socket创建的监听套接字将导致服务器无法继续接受新的连接，但不会影响已经建立的连接
    - 关闭accept返回的已连接套接字将导致它所代表的连接被关闭，但不会影响服务器的监听
  - 做客户端时
    - 关闭连接就是关闭连接，不意味着其他

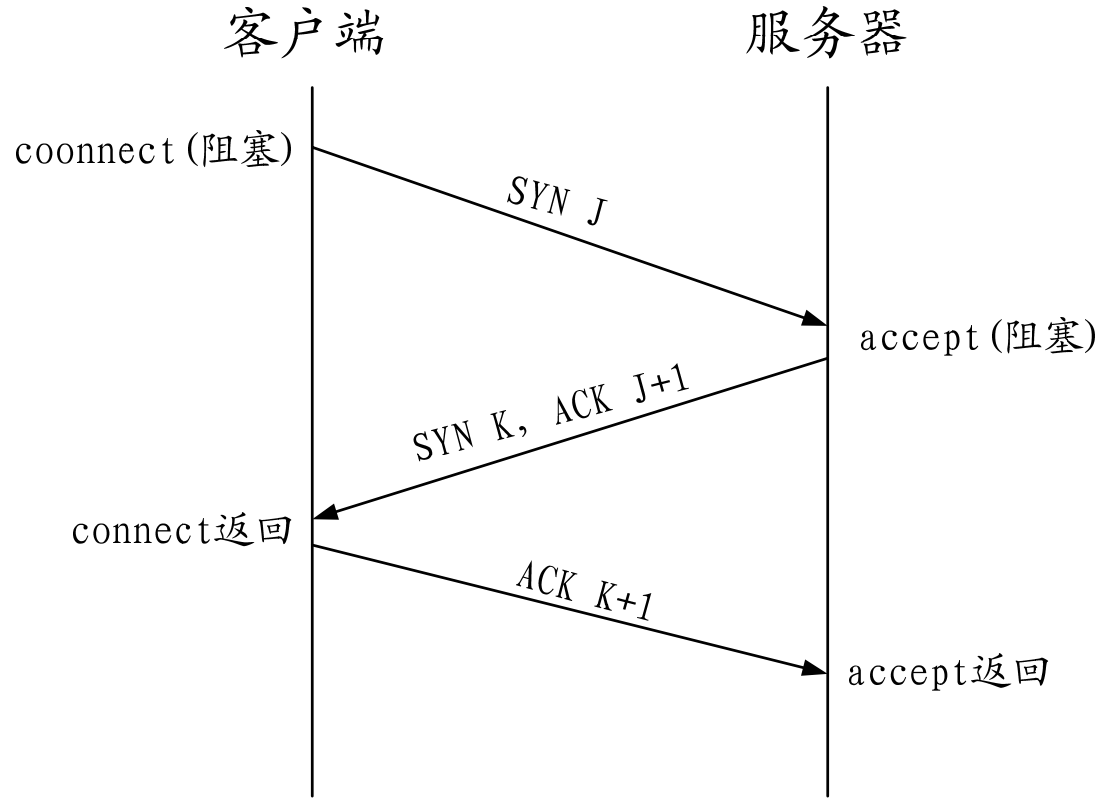
Socket编程——总结

- 面向连接的服务模型



Socket编程——总结

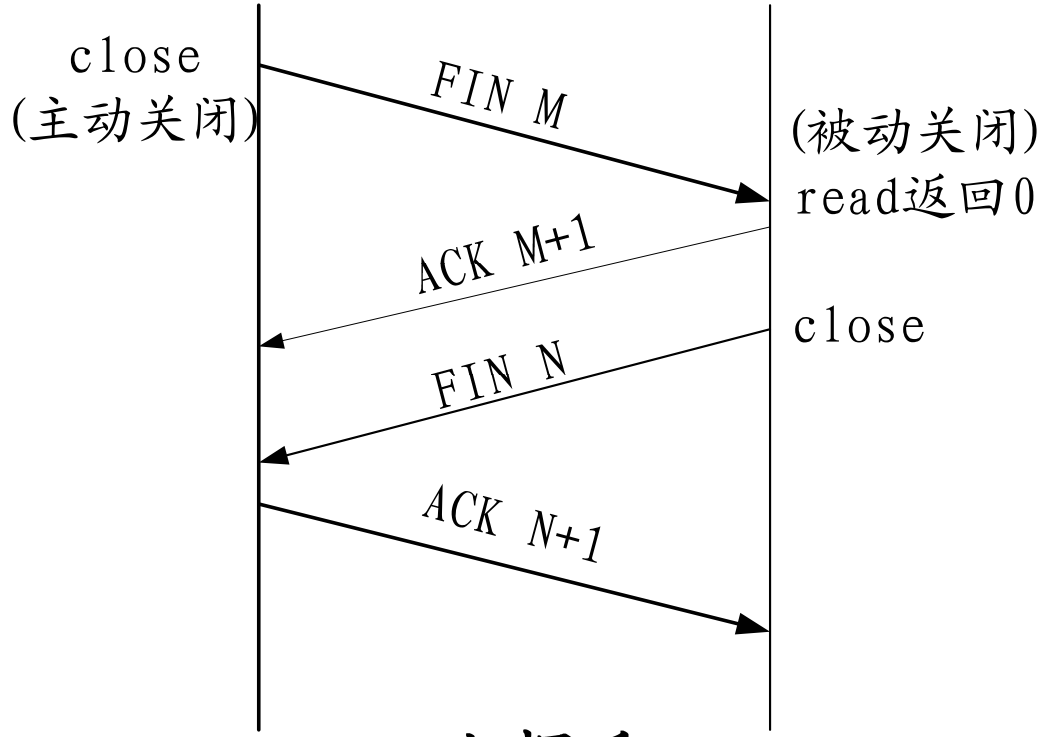
- TCP面向连接的三次握手过程



三次握手

Socket编程——总结

- TCP面向连接的四次挥手过程



四次挥手

演示TCP聊天程序