2020/08/06_网络编程_第2课_基于UDP协议的服务器、客户端的搭建

笔记本: 网络编程
创建时间: 2020/8/6 星期四 10:06
作者: ileemi

前言

从分层的角度来说，能够操作的是传输层，传输层的常用协议 TCP，UDP。

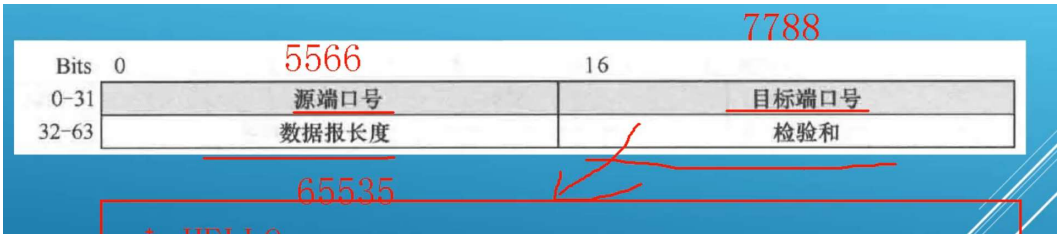
很多企业通用协议满足不了自己软件需求的时候，会基于TCP，UDP协议来进行自定义协议，在其基础上进行扩展来满足自己软件的需求。

连接广域网需要做端口转发

UDP

> 用户数据包协议（ user datagram protoco1，UDP）是一个轻量级的协议，封装数据并将其从一台主机的一个端口发送到另一台主机的一个端口。
> **UDP数据包包含一个8字节的报头，后面跟着数据。**

协议定义：



当信号被干扰的时候，进行传输的数据是又可能丢失的，着就需要校验和来验证发送的数据和接收的数据时候匹配（校验和：通过一个算法将传输的数据整体算出一个值来）。

其校验和的算法（简单校验，越简单越快越高效）可以是将传输数据的ASCII数值相加。校验和的算法不能太过于复杂发送数据接收数据的速度就会变慢。**校验和的有效性由操作系统去决定。**校验和不对的情况下，数据包会被丢掉，**该协议是不可靠的。**

源端口号（ source prot,16bit）标识数据发送方将UDP数据包发送出去的端口。当数据包接收方需要响应的时候，这个域非常有用。
目标端口号（ destination prot,16bit）是数据报的目标端口。UD模块将数据报发送给与这个端岩口绑定的进程。
数据长度（1 ength,16bit）是指包括头和数据部分在内的总字节数。
校验和（checksum,16bit）由UDP报头、数据部分和IP头的某些域计算得到。它是一个可选项，如果不做计算，取值为0，如果底层验证了数据这个域被忽略。

UDP的特点及其目的

> UDP不提供复杂的控制机制，利用IP提供面向无连接的通信服务。并且它将应用程序发来的数据在收到的那一刻，立即按照原来发送到网络上的种机制。

> 即使网络出现拥堵的情况下，UDP也无法进行流量控制等避免网络拥塞行为。此外传输途中即使出现丢包，UDP也不负责重发。甚至出现包的到达顺序乱掉时也没有纠正的功能

UDP的应用场合

> 包总量比较少的通信（DNS等）
> 视频、音频等多媒体通信
> 限定于LAN等特定网络中的应用程序
> 广播通信

不考虑信息完整性，一般可以用UDP

伯克利套接字

伯克利套接字将网络里面所有操作的协议封装了一个库，不在需要了解协议也可以编写代码，操作简单。现在已经标准化了，几乎所有的操作系统都支持。

伯克利套接字应用程序接口（ Berkeyley Sockets API)，最初是作为 BSD4.2 的部分发布的，提供了**进程与 TCP/IP 模型各个层之间通信的标准方法。**发布以来，这个 API 已经被移植到每一个主要的操作系统和流行的编程语言，所以它是网络编程中名副其实的标准。

Windows上的伯克利套接字为了系统的安全性做了很多限制。

SOCKET 初始化

说明要使用传输层的哪个协议（Windows上传输层，网络层）

> API -- socket
> 标准类型：
> int socket(int af, int type, int protocol)

现在一般的socket编程都是C/S模式。

C/S -- 客户端 服务器 -- Client(n个) / Server(1个)
B/S -- 浏览器 服务器

服务器的创建

目的：接收客户端发送的数据

> 步骤：
> 1. 初始化套接字库，说明要使用伯克利套接字的版本
> 2. 创建绑定到特定服务提供程序的套接字(说明要使用的协议)
> 3. 绑定端口（向系统申请一个没有使用的端口）
> 4. 收发数据
> 5. 关闭 `socket`

写跨平台网络程序，注意微软的标准。

注册 `socket` 前需要调用 `WSAStartup` 函数 初始化套接字库，说明使用 伯克利套接字库的第几个版本（包含头文件 `Winsock2.h`）。

微软的socket函数不是按照标准来做的，当要在Windows编写跨平台程序时，要注意这个函数，其返回值，错误码没有按照标准来做。

关闭 socket API：`closesocket`
绑定 socket API：`bind`

绑定的IP地址一般不是广域网，如果需要使用广域网的IP就需要做端口转发。推荐使用回环地址：127.0.0.1（将地址发送给网卡会返回），数据不会到达路由器，网卡会反弹。
软件发布的时候适合填局域网或者广域网的地址。
参数填 ADDR_ANY(0)，上述的三个地址都可以连。

inet_addr -- 高版本的VS中不支持IPV6，可以使用 inet_pton，InetPton。

TCP/IP 协议 端口，IP地址在内存中是大尾方式存储的，不同计算机网络通讯CPU是不一样（访问内存的方式不一样，有大尾，小尾）。所有需要使用 **htons htonl** 根据需求进行传唤。

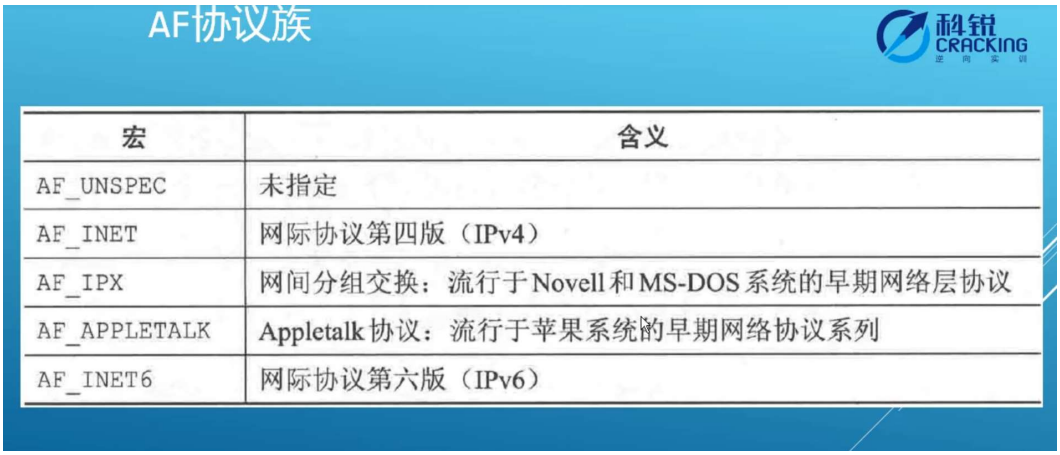
inet_pton -- 会自动将IP地址转换成大尾方式存储。

绑定指定的域名

填写域名的话，需要使用 API -- getaddrinfo 进行转换。

一个域名可以多个IP地址，绑定域名的好处就是防止IP地址变化。

AF协议族



宏	含义
AF_UNSPEC	未指定
AF_INET	网际协议第四版（IPv4）
AF_IPX	网间分组交换：流行于Novell和MS-DOS系统的早期网络层协议
AF_APPLETALK	Appletalk协议：流行于苹果系统的早期网络协议系列
AF_INET6	网际协议第六版（IPv6）

TYPE 类型

宏	含义
SOCK_STREAM	数据包代表有序的、可靠的数据流分段
SOCK_DGRAM	数据包代表离散的报文
SOCK_RAW	数据包头部可以由应用层自定义
SOCK_SEQPACKET	与SOCK_STREAM类似，但是数据包接收时需要整体读取

PROTOCOL 协议

宏	需要的类型	含义
IPPROTO_UDP	SOCK_DGRAM	数据包封装UDP数据报
IPPROTO_TCP	SOCK_STREAM	数据包封装TCP报文段
IPPROTO_IP/0	Any	为给定的类型使用默认协议

服务器收发数据

recvfrom -- 函数接收数据报并存储源地址
示例代码：

```
#include <iostream>
// 包含对应的头文件，库
#include <Winsock2.h>
#pragma comment(lib, "Ws2_32.lib")
```

```c
#include <ws2tcpip.h>

int main()
{
  // 1. 初始化套接字库，说明要使用伯克利套接字的版本
  WSADATA wsaData;
  if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
  {
    printf("WSAStartup Error: %08x\r\n", WSAGetLastError());
    return 0;
  }


  // 2. 创建绑定到特定服务提供程序的套接字(说明要使用的协议)
  /*
  参数1：网际协议第四版（IPV4）
  参数2：使用UDP协议
  参数3：使用传输层的UDP协议
  */
  SOCKET s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
  if (INVALID_SOCKET == s)
  {
    // 通过 WSAGetLastError 函数显示错误信息
    printf("socket init Error: %08x\r\n", WSAGetLastError());
    return 0;
  }
  // socket 初始化成功
  printf("socket init ok: %08x\r\n", s);

  // 3. 绑定端口（向系统申请一个没有使用的端口）
  //sockaddr addr;
  //addr.sa_family = AF_INET;
  //addr.sa_data = //端口（2个字节） IP地址（4个字节）
  // 申请一个端口号，使用sockaddr结构体比较麻烦
  //*(short*)addr.sa_data = 5566;
  sockaddr_in addr;                    // sockaddr_in 结构体将端口，IP地址，保留数据分开
  addr.sin_family = AF_INET;      // 使用IPV4协议
  addr.sin_port = htons(6666); // 向操作系统申请一个端口，同时使用 htons 转换大小尾
  /*
   union
   {
      struct { UCHAR s_b1,s_b2,s_b3,s_b4; } S_un_b;
      struct { USHORT s_w1,s_w2; } S_un_w;
      ULONG S_addr;
  } S_un;
```

```
    IP地址的格式是一个共用体，可以一个字节一个字节写
    也可以两个字节两个字节，还可以一块写
    IP地址 -- 广域网 - 115.52.xxx.254，局域网 -- 192.168.0.105(首选)
    推荐使用回环地址：127.0.0.1（将地址发送给网卡会返回）
    参数为 ADDR_ANY(0)：广域网，局域网，回环地址都可以连接这个软件
    ADDR_ANY -- 任意地址
   */
   //addr.sin_addr.S_un.S_addr = ADDR_ANY;
   //addr.sin_addr.S_un.S_addr = 0x7F000000;
   //转换表示Internet标准中的数字的字符串。""符号中为一个因特网地址
   //addr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
   //InetPton(AF_INET, L"127.0.0.1", &addr.sin_addr); // 不是标准
   // inet_addr -- 高版本的VS中不推荐使用，这个函数不支持IPV6，可以使用
 inet_pton

   /*
   参数1：IP地址要转换的格式：AF_INET -- IPV4 和 AF_INET6 -- IPV6
   参数2：要转换的IP地址（客户端连接服务器的IP地址）
   参数3：成员的地址
   inet_pton -- 会自动将IP地址转换成大尾方式存储
   */
   inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);


#if 0  // 域名转IP地址 API -- getaddrinfo
   struct addrinfo* result = NULL;
   struct addrinfo* ptr = NULL;
   struct addrinfo hints = { 0 };
   struct sockaddr_in* sockaddr_ipv4;
   hints.ai_family = AF_INET;  //获取IPV4的地址
   if (getaddrinfo("ileemi.top", NULL, &hints, &result))
   {
     printf("getaddrinfo Error\r\n");
     return 0;
   }
   // 循环遍历所有的IP地址
   for (ptr = result; ptr != NULL; ptr = ptr->ai_next)
   {
     printf("AF_INET (IPv4)\n");
     sockaddr_ipv4 = (struct sockaddr_in*)ptr->ai_addr;
     char szIP[MAXBYTE];
     inet_ntop(AF_INET, &sockaddr_ipv4->sin_addr, szIP, sizeof(szIP));
     printf("\tIPv4 address %s\n", szIP);
   }
#endif // 0  // 域名转IP地址 API -- getaddrinfo

   // 绑定bind(s, (sockaddr*)&addr, sizeof(addr));
```

```
    // 使用遍历域名得出的IP地址（使用第一个）result->ai_addr
    if (bind(s, (sockaddr*)&addr, sizeof(addr)) == SOCKET_ERROR)
    {
      // 显示错误信息
      printf("bind Error: %08x\r\n", WSAGetLastError());
      return 0;
    }
    printf("bind ok \r\n");

      //  4.  收发数据
      //  客户端的端口不需要固定，也就不需要绑定端口
      printf("recvfrom...\r\n");
      char  szBuff[260];       //  接收客户端的数据
      char  szIP[200];           //  存储客户端的IP地址
      sockaddr_in  caddr;     //  存储客户端IP地址和端口的地址
      int  nLen  =  sizeof(caddr);  //  存储客户端IP地址和端口的地址结构体的大小
    while (true)
    {
      /*
      参数1：标识绑定的套接字
      参数2：接收数据的缓冲区
      参数3：接收的数据长度
      参数4：标志（一般为0）
      参数5：传出参数 客户端IP地址和端口的地址
      参数6：存储客户端数据的大小
      返回值：收到数据的字节数
      */
      // 没有数据，线程阻塞
      int nRet = recvfrom(s, szBuff, sizeof(szBuff), 0, (sockaddr*)&caddr, &nLen);
      if (nRet == SOCKET_ERROR)
      {
        printf("recvfrom error:%08x\n", WSAGetLastError());
        break;
      }
      // 数据接收正常，显示接收到的数据
      szBuff[nRet] = '\0';
      inet_ntop(AF_INET, &caddr.sin_addr, szIP, sizeof(szIP));
      printf("IP: %s, Port: %d, Data: %s, Bytes: %d\r\n", szIP, htons(caddr.sin_port), szBuff, nRet);

      // 回复数据
      sendto(s, "ok", strlen("ok"), 0, (sockaddr*)&caddr, sizeof(caddr));
    }

    // 创建一个 socket，操作系统会为其分配很多的资源
```

```
    // 防止内存泄漏，最后要关闭申请的 socket
    closesocket(s);

    // 反初始化库，从底层Windows套接字 .dll 中注销
    WSACleanup();
    return 0;
}
```

客户端的创建

> 接收数据：recvfrom

服务器没开，客户端也可以发送数据，因为该协议是按照UDP来编写的，客户端的写和服务器的编写类似。

客户端的端口号一般由操作系统去分配，若绑定了端口号，客户端的端口就是固定的。

收发数据的缓冲取要小于等于服务区收发数据的缓冲区大小。或大于服务器收发数据的缓存区，其发送的数据会被服务器给丢掉。

> 发送数据：sendto
> sendto 函数不能判断发送的数据是否发送给了服务器，只能判断数据是否传递给了操作系统。

网络编程多使用多线程，防止接收信息是出现阻塞问题的发生。

两个客户端进行通信，就设计到C/S模式，就需要将一个客户端发送的数据通过服务器转发到另一台客户端。
进行转发的条件，发送方客户端需要告诉服务器目标客户端的IP以及端口。标识客户区的方法就是，服务器中多一个表，保存每个客户端的IP以及端口号，或者给每个客户端一个ID号（可以保护客户端的隐私数据）。

原理就是，数据发送给服务器，服务器进行转发。

> 关于发送的数据暂时是没有加密的，如果有需求，可以在发送前给定一个算法，将要发送的书进行加密，接收数据的机器进行对应的解密（发送数据加密，接收数据界面）。

> QQ, 微信的数据加密算法以及密钥都是随机生成的，当两台客户端连接后，会分配对应的加密算法（RSA加密），依靠服务器解密。拦截数据也无法解密。每天设备的加密算法都是不一样的。

代码示例：

```
#include <Winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "Ws2_32.lib")
```

```
HANDLE g_hThread = NULL;

// 接收服务器向客户端发送的数据
DWORD WINAPI RecvThread(LPVOID lpParameter)
{
  char szBuff[260];       // 接收客户端的数据
  sockaddr_in addr;
  int nLen = sizeof(addr);
  SOCKET s = (SOCKET)lpParameter;

  while (true)
  {
    // 接收服务器返回的数据并显示
    int nRet = recvfrom(s, szBuff, sizeof(szBuff), 0, (sockaddr*)&addr,
&nLen);
    if (nRet == SOCKET_ERROR)
    {
      printf("recvfrom error:%08x\n", WSAGetLastError());
      break;
    }
    szBuff[nRet] = '\0';
    char szIP[200];
    inet_ntop(AF_INET, &addr.sin_addr, szIP, sizeof(szIP));
    printf("IP: %s, Port: %d, Data: %s, Bytes: %d\r\n", szIP,
htons(addr.sin_port), szBuff, nRet);
  }
  printf("RecvThread Exit...\r\n");
}

int main()
{
  // 1. 初始化套接字库，说明要使用伯克利套接字的版本
  WSADATA wsaData;
  if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
  {
    printf("WSAStartup Error: %08x\r\n", WSAGetLastError());
    return 0;
  }

  // 2. 创建绑定到特定服务提供程序的套接字(说明要使用的协议)
  /*
  参数1：网际协议第四版（IPV4）
  参数2：使用UDP
  参数3：使用传输层的UDP协议
  */
  SOCKET s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
  if (INVALID_SOCKET == s)
```

```
    {
      // 通过 WSAGetLastError 函数显示错误信息
      printf("socket init Error: %08x\r\n", WSAGetLastError());
      return 0;
    }
    // socket 初始化成功
    printf("socket init ok: %08x\r\n", s);

    // 3. 收发数据 -- 多线程
    // 客户端的端口不需要固定，也就不需要绑定端口
    printf("recvfrom...\r\n");
    char szBuff[260];       // 接收客户端的数据
    sockaddr_in addr; // 存储服务器相关信息的结构体
    int nLen = sizeof(addr);
    addr.sin_family = AF_INET;
    addr.sin_port = htons(6666);
    inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
    while (true)
    {
      scanf_s("%s", szBuff, sizeof(szBuff));
      if (strcmp(szBuff, "exit") == 0)
      {
        break;
      }
      // 发送数据 返回值为 发送数据的字节数
      int nRet = sendto(s, szBuff, strlen(szBuff), 0, (sockaddr*)&addr,
sizeof(addr));

      // sendto 函数不能判断发送的数据是否发送给了服务器
      // 只能判断数据是否传递给了操作系统
      if (nRet == SOCKET_ERROR) {
        printf("sendto error:%08X\n", WSAGetLastError());
        break;
      }
      // 发送成功，显示发送的数据以及字节数
      printf("sendto data:%s bytes:%d\n", szBuff, nRet);

      // 防止服务器发送的数据客户端没有收到不能进行再次向服务器发送信息
      // 创建一个线程，用于接收服务器返回的信息
      if (g_hThread == NULL)
      {
        g_hThread = CreateThread(NULL, 0, RecvThread, (LPVOID)s, 0, NULL);
      }
    }

    // 防止内存泄漏，最后要关闭申请的 socket
    closesocket(s);
```

```
    // socket 关闭，等待线程结束
    WaitForSingleObject(g_hThread, -1);

    CloseHandle(g_hThread);

    // 反初始化库，从底层Windows套接字 .dll 中注销
    WSACleanup();
    return 0;
}
```