2020/08/11_网络编程_第5课_HTTP、ICMP

笔记本： 网络编程
创建时间： 2020/8/11 星期二 10:00
作者： ileemi

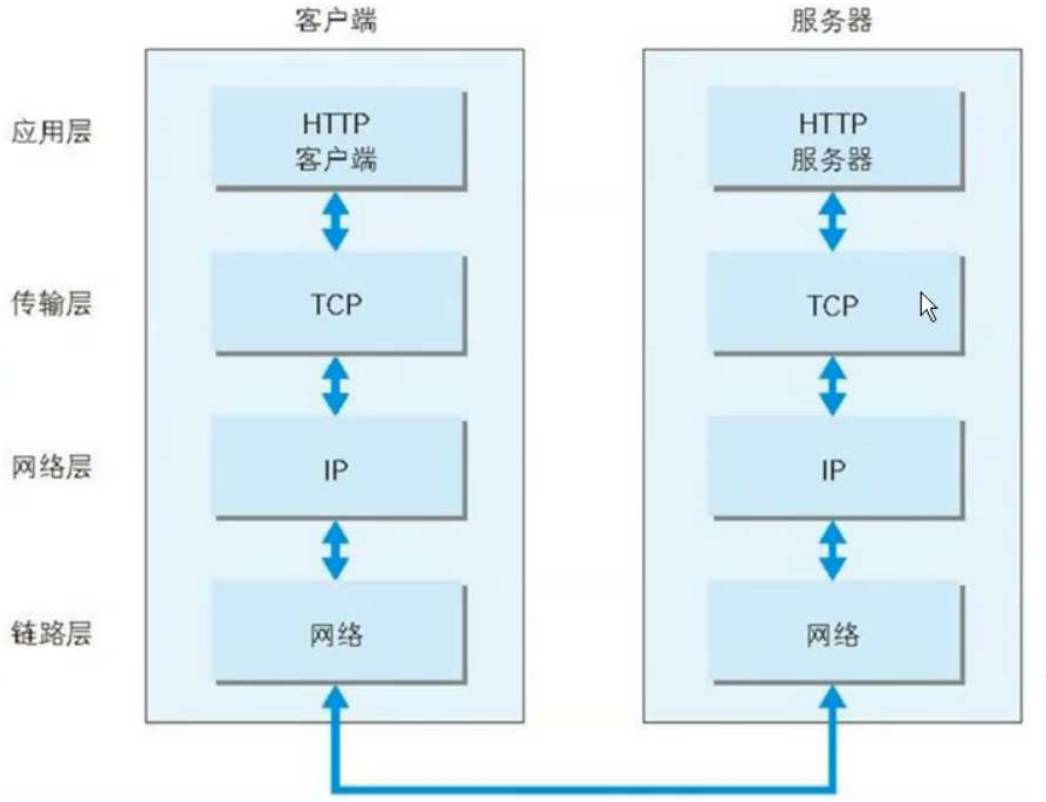
标准化协议

http、ftp、pop3、icmp

HTTP协议

应用层

HTTP：超文本传输协议（不同设备上共享数据）
URL：统一资源定位符



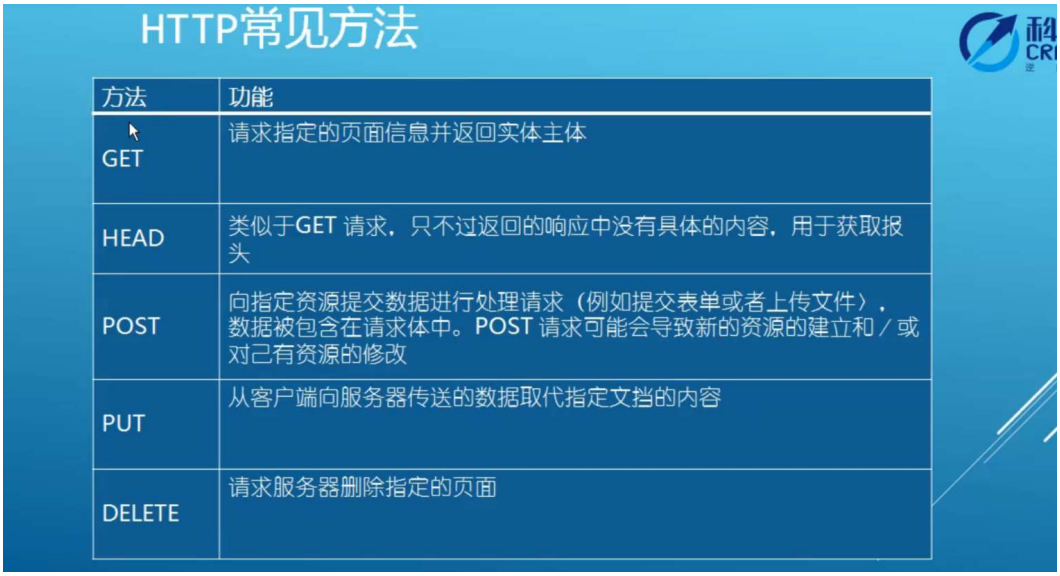
Web

Web（World wide Web）即全球广域网，也称为万维网，它是一种基于超文本和HTTP 的、全球性的、动态交互的、跨平台的分布式图形信息系统。是建立在Internet 上的一种网络服务，为浏览者在 Internet 上查找和浏览信息提供了图形化的、易于访问的直观界面，其中的文档及超级链接将 Internet 上的信息节点组织成一个互为关联的网状结构。

URL

URL（Universal Resource Locator）：**统一资源定位符，一种定位资源的主要访问机制的字符串，一个标准的URL必须包括：协议、地址、资源。**

HTTP常见方法



方法 功能
GET： 请求指定的页面信息并返回实体主体
HEAD：类似于GET请求，只不过返回的响应中没有具体的内容，用于获取报头
POST：向指定资源提交数据进行处理请求（例如提交表单或者上传文件）数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或对已有资源的修改
PUT：从客户端向服务器传送的数据取代指定文挡的内容
DELETE：请求服务器删除指定的页面

> 转域名 API：`gethostbyname`

发送数据
添加报文
请求头部

格式：
Get /文件路径名 协议版本
Host:域名
Accept:文件类型
Comnection:Keep-Alive -- 保持连接
Accept-encoding: -- 压缩格式
Accept-Language: -- 语言

请求头部 -- q（数据传输的优先级）

```
71  '3.发送数据  协议
72  ar packet[] = {
73   "GET /portal/home/index.html HTTP/1.1\r\n"
74   "Host:www.jszg.edu.cn\r\n"
75   "Accept:text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, image/apng, */*; q=0.8
76                   I
77  };
78  t ret = send(g_Socket, (char*)packet, sizeof(packet), 0);
79  (ret <= 0) {
```

HTTPS

支持数据 报文二进制传递，可以之明传输文本的加密算法
443端口

// openssl 库

添加了一层加密，数据传输前先加密，加密后进行传输。

HTTPS 就是加过密的 HTTP。使用HTTPS后，浏览器客户端和 Web 服务器传输的数据是加密的，只有浏览器和服务器端知道内容。

HTTPS=HTTP+TLS 或者 SSL。采用HTTPS的网站需要去数字证书认证机构 C Certificate Authority,CA）申请证书。

通过这个证书，浏览器在请求数据前与 Web 服务器有几次握手验证，以证明相互的身份，然后对 HTTP 请求和响应进行加密。HTTPS 采用 443 端口。

代码示例：

```
// Client.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//
#include <stdio.h>
#include <Winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "Ws2_32.lib")
SOCKET g_Socket;
class CSocket {
public:
    CSocket() {
        //初始化套接字库
        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            printf("
[server] WSAStartup error:%08X\n", WSAGetLastError());
        }
    }
    ~CSocket() {
        //反初始化库
        WSACleanup();
    }
}g_init;
void show_error_msg(const char* pre) {
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        WSAGetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language

        (LPTSTR)& lpMsgBuf,
        0,
        NULL
```

```
    );
    printf("%s:%s", pre, (LPCTSTR)lpMsgBuf);
    // Free the buffer.
    LocalFree(lpMsgBuf);
}
int main()
{
    // 1.初始化套接字(说明使用的协议)
    g_Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //TCP协议
    if (INVALID_SOCKET == g_Socket) {
        show_error_msg("[client] socket init error");
        return 0;
    }
    printf("[client] socket init ok s=%08X\n", g_Socket);
    // 2.连接服务器(三次握手)
    sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(80);
    hostent *host = gethostbyname("www.jszg.edu.cn");
    int** p = (int**)host->h_addr_list;
    int *p2 = p[0];
    addr.sin_addr.S_un.S_addr = *p2;
    if (connect(g_Socket, (sockaddr*)& addr, sizeof(addr)) == SOCKET_

        show_error_msg("[client] connect server error");
        return 0;
    }
    printf("[client] connect server ok\n");
    // 3.发送数据 协议
    /*
    HTTPS (库) 端口: 443
    https://github.com/openssl/openssl
    SSL_library_init();
    SSL_CTX* sslContext = SSL_CTX_new(SSLv23_client_method());
    SSL* sslHandle = SSL_new(sslContext);
    SSL_set_fd(sslHandle, ServerSocket)
    SSL_connect(sslHandle)
    SSL_write();   //send
    SSL_read();    //recv
    SSL_shutdown(sslHandle);
    SSL_free(sslHandle);
    SSL_CTX_free(sslContext);
    */
    char packet[] = {
        "GET /portal/home/index.html HTTP/1.1\r\n"
        "Host:www.jszg.edu.cn\r\n"
        //"Accept:text/html, application/xhtml+xml, application/xml;q=0.9,
```

```
文件类型
        //"Connection:Keep-Alive\r\n"      //常连接
        //"Accept-encoding:gzip,  deflate,  br\r\n"      //压缩算法
        //"Accept-Language:en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7\r\n"  //
语言
        //"Cache-control:no-cache\r\n"
        "User-
agent:Mozilla/5.0  (Macintosh;  Intel  Mac  OS  X  10_13_2)  AppleWebKit/537
浏览器信息
        "\r\n"
        };
    int  nRet  =  send(g_Socket,  (char*)packet,  sizeof(packet),  0);
    if  (nRet  <=  0)  {
        show_error_msg("[client]  send  server  error");
    }
    //  4.接收数据
    char  szBuff[0x10000];
    nRet  =  recv(g_Socket,  (char*)szBuff,  sizeof(szBuff),  0);
    if  (nRet  <=  0)  {
        show_error_msg("[client]  send  server  error");
    }
    szBuff[nRet]  =  '\0';
    printf("bytes:%d  data:\n%s\n",  nRet,  szBuff);
    //  将获取指定的网页数据写入到文件
    FILE  *fp  =  fopen("test.html",  "w+");
    fwrite(szBuff,  1,  nRet,  fp);
    fclose(fp);
    //关闭socket
    closesocket(g_Socket);
    return  0;
}
```

ICMP

邮箱协议：POP3（接收邮件）、STMP（发送邮件）

网络层协议，ICMP 基于 UDP 协议，主要用于在主机与路由器之间传递控制信息，包括报告错误、交换受限控制和状态信息等。当遇到 IP 数据无法访问目标、IP 路由器无法按当前的传输速率转发数据包等情况时，会自动发送 ICMP 消息。

> ICMP是（ Internet Control Message Protocol) Internet 控制报文协议。它是 TCP/IP 协议族的一个子协议，用于在 IP 主机、路由器之间传递控制消息。

控制消息是指网络通不通、主机是否可达、路由是否可用等网络本身的消息。这些控制消息虽然并不传输用户数据，但是对于用户数据的传递起着重要的作用。

步骤：

1. 初始化套接字（说明使用的协议）
2. 使用 gethostbyname 转换域名（发送给路由器的，不需要填写端口号）
3. 发送数据：sendto
4. 接收数据：recvfrom

ICMP 没有端口，可用来查看IP地址是否可用。

select（设置超时） -- 函数确定一个或多个套接字的状态，如有必要，等待执行同步 I/O。
GetTickCount -- 返回（retrieve）从操作系统启动所经过（elapsed）的毫秒数。

代码示例：

```
// ICMP.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#include <iostream>
// POP3 STMP 邮箱协议
// ICMP 测试网络状态
#include <stdio.h>
#include <Winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "Ws2_32.lib")

SOCKET g_Socket;
class CSocket {
public:
    CSocket() {
        // 初始化套接字库
        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            printf("
[server] WSAStartup error:%08X\n", WSAGetLastError());
        }
    }
    ~CSocket() {
        // 反初始化库
        WSACleanup();
    }
}g_init;

void show_error_msg(const char* pre) {
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
```

```
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        WSAGetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language

        (LPTSTR)& lpMsgBuf,
        0,
        NULL
    );
    printf("%s:%s", pre, (LPCTSTR)lpMsgBuf);
    // Free the buffer.
    LocalFree(lpMsgBuf);
}

// IP 头
struct ip_hdr
{
    unsigned char h_len : 4;            // length of header
    unsigned char version : 4;          // Version of IP
    unsigned char tos;                  // Type of service
    unsigned short total_len;           // total length of the pack

    unsigned short ident;               // unique identifier
    unsigned short frag_and_flags; // flags
    unsigned char ttl;                  // ttl
    unsigned char proto;                // protocol(TCP ,UDP etc

    unsigned short checksum;            // IP checksum
    unsigned int sourceIP;
    unsigned int destIP;
};

#define ECHO_REQUEST    8
#define ECHO_REPLY      0

// ICMP 头
struct icmp_hdr
{
    unsigned char    icmp_type;     // 类型
    unsigned char    icmp_code;     // 代码
    unsigned short icmp_cksum;    // 效验和
    unsigned short icmp_id;         // n
    unsigned short icmp_seq;        // n
    unsigned int     icmp_data;     // GetTickout()
};
```

```
// UDP 头
struct udp_header {
    unsigned short src_port; // 源端口号16bit
    unsigned short dst_prot; // 目的端口号16bit
    unsigned short len;          // 数据包长度16bit
    unsigned short check_sum;// 校验和16bit
};

//网际校验和是被校验数据16位值的反码和(ones-complement sum)
WORD CalcCheckSum(IN unsigned short* addr, IN int len)
{
    int    nleft = len;
    int    sum = 0;
    unsigned short* w = addr;
    unsigned short answer = 0;
    while (nleft > 1) {
        sum += *w++;
        nleft -= 2;
    }
    if (nleft == 1) {
        *(unsigned char*)(&answer) = *(unsigned char*)w;
        sum += answer;
    }
    sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    answer = ~sum;
    return (answer);
}

int main()
{
    // 1. 初始化套接字(说明使用的协议)
    g_Socket = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP); // 原始套
接字，网络层
    if (INVALID_SOCKET == g_Socket) {
        show_error_msg("[client] socket init error");
        return 0;
    }
    printf("[client] socket init ok s=%08X\n", g_Socket);
    sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = 0;
    hostent* host = gethostbyname("ileemi.top");
    int** p = (int**)host->h_addr_list;
    int* p2 = p[0];
    addr.sin_addr.S_un.S_addr = *p2;
```

```
    // 2. 发送数据 协议
    char szBuff[32] = { 0 };
    for (int i = 0; i < 4; i++) {
        icmp_hdr* icmp = (icmp_hdr*)szBuff;
        icmp->icmp_type = ECHO_REQUEST;
        icmp->icmp_code = 0;
        icmp->icmp_cksum = 0;    // 校验和
        icmp->icmp_id = i;
        icmp->icmp_seq = i;
        icmp->icmp_data = GetTickCount();
        icmp-
>icmp_cksum = CalcCheckSum((unsigned short*)szBuff, sizeof(szBuff));
        int send_bytes = sendto(g_Socket, (char*)szBuff, sizeof(szBuff)

        if (send_bytes <= 0) {
            show_error_msg("[client] sendto server error");
        }

        // 3. 接收数据
        char recv_buf[0x1000];
        sockaddr_in server_addr;    // 用来存储服务器的相关信息
        int nServerInfoLen = sizeof(server_addr);
        // 设置超时
        fd_set readfds;
        //readfds.fd_count = 1;
        //readfds.fd_array[0] = g_Socket;
        FD_ZERO(&readfds);
        FD_SET(g_Socket , &readfds);
        timeval tv = {1, 0};
        if (select(0, &readfds, NULL, NULL, &tv) == 0) {
            puts("请求超时");
            continue;
        }

        int ret = recvfrom(g_Socket, (char*)recv_buf,
            sizeof(recv_buf), 0, (sockaddr*)& server_addr, &nServerInfo

        if (ret <= 0) {
            show_error_msg("[client] recvfrom server error");
        }
        ip_hdr* ip = (ip_hdr *)recv_buf;
        icmp_hdr* icmp2 = (icmp_hdr*)(ip + 1);
        udp_header* udp = (udp_header * )(icmp2 + 1);

        void* data = udp + 1;
        if (icmp2->icmp_type == ECHO_REPLY && icmp2-
>icmp_id == i) {
```

```
            printf("来自 %s 的回复: 字节=%d 时间
=%dms TTL=%d\n",    //64
                inet_ntoa(*(in_addr*)& ip->sourceIP),
                send_bytes,
                GetTickCount() - icmp2->icmp_data,
                ip->ttl);
        }
        Sleep(1000);
    }
    //关闭socket
    closesocket(g_Socket);
    return 0;
}
```