Winsock2 服务提供者接口(SPI):

一、简述:

1、一般用于提供给**操作系统开发商**、**传输堆栈商**在基础协议的基础上,开发更高级的服务.
2、因为**Winsock服务体系**符合**Windows开放服务体系**.所以,它支持**第三方服务提供者**插入到其中.
3、只要上层和下层的**边缘**支持Winsock2 SPI,即可向他们中间安装**第三方提供者程序**.
4、**普通开发者**一般都是开发SPI的**LSP(分层服务提供者)**,即第三方提供者,可用于监控Winsock API执行,HOOK Winsock API,甚至利用LSP技术注入DLL.
5、基础协议(TCP、UDP、原始)的提供者其实就是DLL,**编写分层协议提供者就是在编写DLL**,然后安装在Winsock目录上,让系统上的所有使用基础协议的网络程序调用.

- **【重点】网络程序是如何调用Winsock2 服务提供者进行网络通讯:**

1、当网络程序使用**WSAStartup**加载库时,系统并不做什么.
2、而是当程序真正创建套接字时,会先调用**WSCEnumProtocols**函数,遍历系统内安装的所有提供者(分层、基础、协议链),当先找到一个与要求使用的协议符合的,那么导出此提供者的DLL,才开始调用提供者的**WSPStartup**初始化函数,才能使用**send**,**recv**(TCP协议提供者的DLL)
或**sendto**，**recvfrom**(UDP协议提供者的DLL)等函数的功能.

二、SPI(服务提供者接口)由两个部分组成:

一、传输服务提供者:

> 提供建立连接、传输数据、流控制、出错控制。**共两种类型:**

- 基础服务提供者:

实现传输协议的细节,导出**Winsock**接口(此接口直接实现协议). //TCP、UDP、原始一般都有与之关联的内核模式协议驱动,TCP、UDP由系统内的Tcpip.sys驱动。

- 分层服务提供者(LSP):

将自己安装到**Winsock目录**(Winsock目录的概念在下面)中**基础提供者(TCP/UDP)**的上一层,也可能安装在**其他提供者之间,**可截获程序的**Winsock API**。依靠基础服务提供者作为通信基础,实现更高

层的**通信函数**。

二、命名空间服务提供者：

1、与传输服务提供者相似,可截获名称**解析API(gethostbyname、WSALookupServiceBegin)**的调用.
2、此类提供者需在命名空间目录安装自己.

三、SPI(服务提供者)函数集合类型：

- **头文件：ws2spi.h**
  SPI函数类型总数：4种类型,每一种类型都有自己所属的开头,例如WSC、WSP

类型	解释
WSC	安装、移除、修改**分层服务提供者**和**命名空间提供者**程序
WSP	分层服务提供者的**API**
WPU	分层服务提供者使用的**支持函数**
NSP	**命名空间服务提供者**的API

四、Winsock协议目录的概念：

一、SPI提供三种协议：

1、**分层协议**：处在基础协议的上一层,依靠基础协议作为通信基础。
2、**基础协议**：能够独立、安全、远程端点实现数据通信的协议。
3、**协议链**：将一系列基础协议和分层协议按特定顺序连接在一起。
注意：只有管理员用户组能够安装、移除Winsock目录入口!

二、WSAPROTOCOL_INFO结构体：

说明：描述某个协议(分层协议、基础协议)的完整信息,一个WSAPROTOCOL_INFO结构体称为一个Winsock目录入口。

```
typedef struct _WSAPROTOCOL_INFOW {
    DWORD dwServiceFlags1;        //描述[协议]提供的服务的位掩码
    DWORD dwServiceFlags2;        //保留
    DWORD dwServiceFlags3;        //保留
```

```
    DWORD dwServiceFlags4;        //保留
    DWORD dwProviderFlags;        //此[协议]在[Winsock目录]中的[表示方式]
    GUID ProviderId;                    //由[服务提供商]安排的GUID唯一标示符
    DWORD dwCatalogEntryId;       //WS2_32.DLL为每一个WSAPROTOCOL_INFOW结构安
排的唯一标示符(目录入口ID)


    WSAPROTOCOLCHAIN ProtocolChain; /*1)与[此协议]相关联的WSAPROTOCOLCHAIN结
构.
                                                                    2)说
明了[此协议]在[分层协议]中所处的位置.*/

    int iVersion;                       //[协议]版本标示符
    int iAddressFamily;             //传递给socket/WSASocket函数的[地址加载参
数]
    int iMaxSockAddr;               //地址的最大长度(以字节为单位)
    int iMinSockAddr;               //地址的最小长度(以字节为单位)
    int iSocketType;                //传递给socket函数的[套接字类型参数]
    int iProtocol;                      //传递给socket函数的[协议参数]
    int iProtocolMaxOffset;         //添加到iProtocol的最大值
    int iNetworkByteOrder;        //顺序类型:大尾顺序(BIGENDIAN),小尾顺序(LITT
LEENDIAN)
    int iSecurityScheme;            //安全方案

    DWORD dwMessageSize;          /*[此协议]支持的最大消息长度(以字节为单位)
                                    1)0为基于流协议(如TCP),没有最大长度的概念.
                                    2)1为发送消息的最大长度依赖于下层网络的MTU(最
大传输单元),在套接字绑定后,应使用SO_MAX_MSG_SIZE套接字选项.
                                       获取发送消息的最大长度.
                                    3)-1为此协议是基于消息的,但是对发送的消息没有
最大长度的限制.
                                  */

    DWORD dwProviderReserved;                                   //保留给服务提供者
使用.
    WCHAR  szProtocol[WSAPROTOCOL_LEN+1];  //随意编辑的,此协议的可读字符串.一般
用于说明是什么协议
} WSAPROTOCOL_INFOW, FAR * LPWSAPROTOCOL_INFOW;
```

五、遍历系统所有已安装的协议：

一、使用的API函数：

```
int WSAEnumProtocols(
```

```
        LPINT                           lpiProtocols,       //一个数组，NULL为函数
将返回所有协议，否则只检索数组中列出的那些协议。
        LPWSAPROTOCOL_INFO              lpProtocolBuffer,   //取信息的缓冲区
        LPDWORD                         lpdwBufferLength    //参数2缓冲区的长度
        //如果参数2为NULL,参数3为0,执行后,WSAENOBUFS错误,参数3包含了所需的缓冲区长
度.
);//返回值：系统中安装的协议数量,失败为SOCKET_ERROR.
```

注意：此函数仅能够遍历基础协议、协议链，但是不能遍历分层协议.

二、支持遍历分层协议的函数,功能与上面相同:

```
intWSCEnumProtocols(
        LPINT                lpiProtocols,
        LPWSAPROTOCOL_INFOW lpProtocolBuffer,
        LPDWORD              lpdwBufferLength,
        LPINT                lpErrno //相当于WSAGetLastError()执行的结果
);
```

注意：因为SPI是用于开发系统组件的函数,所以他只使用Unicode字符串,与Windows系统相对应。

遍历系统内安装的所有协议例子：

```
Hello.h
#pragma once
#include <iostream>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <mstcpip.h>
#include <string.h>
#include <tchar.h>
using namespace std;
#pragma warning(disable:4996)
#pragma comment(lib, "Ws2_32.lib")
//系统安装协议遍历实验
class ProtocolTraversestheExperiment
{
public:
    ProtocolTraversestheExperiment()
    {
        WSADATA wsa;
        WSAStartup(MAKEWORD(2, 2), &wsa);
    }
    ~ProtocolTraversestheExperiment()
    {
        WSACleanup();
    }
```

```
    LPWSAPROTOCOL_INFO GetProvider(LPINT lpnTotalProtocols)
    {
        DWORD dwSize = 0;
        LPWSAPROTOCOL_INFO pProtoInfo = NULL;
        if (WSAEnumProtocols(NULL, pProtoInfo, &dwSize) == SOCKET_ERROR)
        {
            if (WSAGetLastError() != WSAENOBUFS)
                return NULL;
        }
        pProtoInfo = (LPWSAPROTOCOL_INFO)new WSAPROTOCOL_INFO[dwSize / si
zeof(WSAPROTOCOL_INFO)];
        if (!pProtoInfo)
            return NULL;
        ZeroMemory(pProtoInfo, dwSize);
        *lpnTotalProtocols = WSAEnumProtocols(NULL, pProtoInfo, &dwSize);
        return pProtoInfo;
    }
    void FreeProvider(LPWSAPROTOCOL_INFO pProtoInfo,int i)
    {
        if(i == 1)
            delete pProtoInfo;
        else
            delete[] pProtoInfo;
    }
};
```

```
Hello.cpp
#include "Hello.h"
int main(int argc,char** argv)
{
    system("color 4e");
    ProtocolTraversestheExperiment s;
    int ProtocolsCount = 0;
    LPWSAPROTOCOL_INFO info = s.GetProvider(&ProtocolsCount);
    if (ProtocolsCount != 0)
    {
        for (int i = 0; i < ProtocolsCount; i++)
        {
            wprintf(_T("Protocol:%s \r\n"), info[i].szProtocol);
            wprintf(_T("CatalogEntryId:%d         ChainLen:%d \n\n"), inf
o[i].dwCatalogEntryId, info[i].ProtocolChain.ChainLen);
        }
        s.FreeProvider(info, ProtocolsCount);
    }
    getchar();
    return 0;
}
```