**[SuperSocket 1.6.4 通过FixedHeaderReceiveFilter解析自定义协议](http://www.cnblogs.com/xiepeixing/p/4228214.html)**

SuperSocket 提供了一些通用的协议解析工具, 你可以用他们简单而且快速的实现你自己的通信协议:

* **TerminatorReceiveFilter** (SuperSocket.SocketBase.Protocol.TerminatorReceiveFilter, SuperSocket.SocketBase)
* **CountSpliterReceiveFilter** (SuperSocket.Facility.Protocol.CountSpliterReceiveFilter, SuperSocket.Facility)
* **FixedSizeReceiveFilter** (SuperSocket.Facility.Protocol.FixedSizeReceiveFilter, SuperSocket.Facility)
* **BeginEndMarkReceiveFilter** (SuperSocket.Facility.Protocol.BeginEndMarkReceiveFilter, SuperSocket.Facility)
* **FixedHeaderReceiveFilter** (SuperSocket.Facility.Protocol.FixedHeaderReceiveFilter, SuperSocket.Facility)

由于本次项目涉及的通信协议是**头部格式固定并且包含内容长度的协议**这里主要讲解使用**FixedHeaderReceiveFilter**来拆解.

通信协议格式如下:

|  |  |  |
| --- | --- | --- |
|  | | |
| 代码 | 字节数 | 说明 |
| 68H | 1 | 帧起始码 |
| DLC | 4 | 设备逻辑地址 |
| SEQ | 2 | 主站地址与命令序号 |
| 68H | 1 | 帧起始码 |
| C | 1 | 控制码 |
| L | 2 | 数据长度(DATA长度) |
| DATA | 变长 | 数据内容 |
| CS | 1 | 校验码 |
| 16H | 1 | 结束码 |

在FixedHeaderReceiveFilter,头部指数据内容之前的数据(即数据长度L之前的部分),以上协议可以知道,头部包含11个字节.

首先,根据协议的需要来定义自己的请求类型,先实现一个客户端请求的实体类RequestInfo,改RequestInfo类必须实现接口 IRequestInfo，该接口只有一个名为"Key"的字符串类型的属性.SuperSocket设计了两个RequestInfo类:StringRequestInfo 和BinaryRequestInfo,这里我们自定义一个来GDProtocolRequestInfo实现:

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using SuperSocket.SocketBase.Protocol;

namespace GDServer

{

public class GDProtocolRequestInfo : IRequestInfo

{

/// <summary>

/// [不使用]

/// </summary>

public string Key { get; set; }

/// <summary>

/// 设备逻辑地址

/// </summary>

public string DeviceLogicalCode { get; set; }

/// <summary>

/// 命令序列号

/// </summary>

public string Seq { get; set; }

/// <summary>

/// 控制码

/// </summary>

public string ControlCode { get; set; }

/// <summary>

/// 数据长度

/// </summary>

public string Length { get; set; }

/// <summary>

/// 数据域

/// </summary>

public string Data { get; set; }

/// <summary>

/// CS校验

/// </summary>

public string Cs { get; set; }

/// <summary>

/// 当前完整帧

/// </summary>

//public string EntireFrame { get; set; }

}

}

[复制代码](javascript:void(0);)

然后设计基于类FixedHeaderReceiveFilter实现自己的接收过滤器GDProtocolReceiveFilterV2,主要实现GetBodyLengthFromHeader和ResolveRequestInfo方法,实现如下:

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using SuperSocket.SocketBase.Protocol;

using SuperSocket.Facility.Protocol;//

using SuperSocket.Common;//

namespace GDServer

{

/// <summary>

/// 广东规约过滤器V2,(帧格式为GDProtocolRequestInfo)

/// </summary>

public class GDProtocolReceiveFilterV2 : FixedHeaderReceiveFilter<GDProtocolRequestInfo>

{

public GDProtocolReceiveFilterV2()

: base(11)

{

}

/// <summary>

/// 获取数据域和结尾字节长度

/// </summary>

/// <param name="header"></param>

/// <param name="offset"></param>

/// <param name="length"></param>

/// <returns></returns>

protected override int GetBodyLengthFromHeader(byte[] header, int offset, int length)

{

//length为头部(包含两字节的length)长度

//获取高位

byte high = header[offset + length - 1];

//获取低位

byte low = header[offset + length - 2];

int len = (int)high \* 256 + low;

return len + 2;//结尾有2个字节

}

/// <summary>

/// 实现帧内容解析

/// </summary>

/// <param name="header"></param>

/// <param name="bodyBuffer"></param>

/// <param name="offset"></param>

/// <param name="length"></param>

/// <returns></returns>

protected override GDProtocolRequestInfo ResolveRequestInfo(ArraySegment<byte> header, byte[] bodyBuffer, int offset, int length)

{

GDProtocolRequestInfo res = new GDProtocolRequestInfo();

string entireFrame = BytesToHexStr(header.Array) + BytesToHexStr(bodyBuffer.CloneRange(offset, length));

//res.EntireFrame = entireFrame;

res.DeviceLogicalCode = entireFrame.Substring(2, 8);

res.Seq = entireFrame.Substring(10, 4);

res.ControlCode = entireFrame.Substring(16, 2);

res.Length = entireFrame.Substring(18, 4);

int dataLen = int.Parse(HEXtoDEC(ReverseHexString(res.Length)));

res.Data = entireFrame.Substring(22, dataLen \* 2);

res.Cs = entireFrame.Substring(22 + dataLen \* 2, 2);

return res;

}

/// <summary>

/// 高低对调

/// </summary>

/// <param name="str"></param>

/// <returns></returns>

string ReverseHexString(string str)

{

char[] buff = new char[str.Length];

for (int i = 0; i < str.Length; i += 2)

{

buff[i] = str[str.Length - i - 2];

buff[i + 1] = str[str.Length - 1 - i];

}

string s = new string(buff);

return s;

}

/// <summary>

/// 16进制转10进制

/// </summary>

/// <param name="HEX"></param>

/// <returns></returns>

string HEXtoDEC(string HEX)

{

return Convert.ToInt64(HEX, 16).ToString();

}

/// <summary>

/// 转化bytes成16进制的字符

/// </summary>

/// <param name="bytes"></param>

/// <returns></returns>

string BytesToHexStr(byte[] bytes)

{

string returnStr = "";

if (bytes != null)

{

for (int i = 0; i < bytes.Length; i++)

{

returnStr += bytes[i].ToString("X2");

}

}

return returnStr;

}

}

}

[复制代码](javascript:void(0);)

先创建新的AppSession,GDProtocolSessionV2,新的AppServer将使用GDProtocolSessionV2.GDProtocolSessionV2代码如下:

[复制代码](javascript:void(0);)

using SuperSocket.SocketBase;

using SuperSocket.SocketBase.Protocol;

using System;

namespace GDServer

{

public class GDProtocolSessionV2 : AppSession<GDProtocolSessionV2, GDProtocolRequestInfo>

{

protected override void HandleException(Exception e)

{

}

}

}

[复制代码](javascript:void(0);)

使用该协议的方法是使用接收或者自己定义的接收过滤器工厂来在 SuperSocket 中启用该协议

[复制代码](javascript:void(0);)

using SuperSocket.SocketBase;

using SuperSocket.SocketBase.Protocol;

namespace GDServer

{

public class GDProtocolServerV2 : AppServer<GDProtocolSessionV2, GDProtocolRequestInfo>

{

public GDProtocolServerV2()

: base(new DefaultReceiveFilterFactory<GDProtocolReceiveFilterV2, GDProtocolRequestInfo>()) //使用默认的接受过滤器工厂 (DefaultReceiveFilterFactory)

{

}

}

}

[复制代码](javascript:void(0);)

这样,GDProtocolServerV2就完成了,下面是测试代码:

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using GDServer;

namespace Test

{

class Program

{

static void Main(string[] args)

{

Console.ForegroundColor = ConsoleColor.Red;

var gdServer = new GDProtocolServerV2();

gdServer.Setup(2015);

gdServer.NewSessionConnected += gdServer\_NewSessionConnected;

gdServer.NewRequestReceived += gdServer\_NewRequestReceived;

gdServer.SessionClosed += gdServer\_SessionClosed;

gdServer.Start();

Console.WriteLine("server is:" + gdServer.State.ToString());

while (true)

{

if (Console.ReadKey().KeyChar == 'q')

{

gdServer.Stop();

gdServer.Dispose();

return;

}

}

}

static void gdServer\_SessionClosed(GDProtocolSessionV2 session, SuperSocket.SocketBase.CloseReason value)

{

Console.WriteLine(session.RemoteEndPoint.ToString() + " closed. reason:" + value);

}

static void gdServer\_NewRequestReceived(GDProtocolSessionV2 session, GDProtocolRequestInfo requestInfo)

{

var info = requestInfo;

Console.WriteLine("receive from: " + session.RemoteEndPoint.ToString());

Console.WriteLine("DeviceLogicalCode:" + info.DeviceLogicalCode);

Console.WriteLine("Seq:" + info.Seq);

Console.WriteLine("ControlCode:" + info.ControlCode);

Console.WriteLine("Length:" + info.Length);

Console.WriteLine("Data:" + info.Data);

Console.WriteLine("Cs:" + info.Cs);

Console.WriteLine("-------------------------------------------------------------");

}

static void gdServer\_NewSessionConnected(GDProtocolSessionV2 session)

{

Console.WriteLine(session.RemoteEndPoint.ToString() + " connected.");

}

}

}

[复制代码](javascript:void(0);)

分别发送符合该协议格式的帧(用TCP调试助手使用hex方式发送)

68 77 77 12 34 00 01 68 A1 03 00 11 11 11 DC 16

68 77 77 12 34 41 01 68 01 0C 00 01 00 00 00 00 00 00 00 30 80 10 80 94 16

68 77 77 12 34 41 01 68 88 08 00 00 00 30 80 00 10 80 00 16 16

68 77 77 12 34 41 01 68 95 23 00 00 0B 00 00 10 00 00 00 00 00 FF FF FF FF FF FF FF FF 00 00 5B 00 00 00 00 00 00 00 00 00 00 00 00 00 32 9E 16

打印结果如下:

server is:Running  
127.0.0.1:34360 connected.  
receive from: 127.0.0.1:34360  
DeviceLogicalCode:77771234  
Seq:0001  
ControlCode:A1  
Length:0300  
Data:111111  
Cs:DC  
-------------------------------------------------------------  
receive from: 127.0.0.1:34360  
DeviceLogicalCode:77771234  
Seq:4101  
ControlCode:01  
Length:0C00  
Data:010000000000000030801080  
Cs:94  
-------------------------------------------------------------  
receive from: 127.0.0.1:34360  
DeviceLogicalCode:77771234  
Seq:4101  
ControlCode:88  
Length:0800  
Data:0000308000108000  
Cs:16  
-------------------------------------------------------------  
receive from: 127.0.0.1:34360  
DeviceLogicalCode:77771234  
Seq:4101  
ControlCode:95  
Length:2300  
Data:000B0000100000000000FFFFFFFFFFFFFFFF00005B0000000000000000000000000032  
Cs:9E  
-------------------------------------------------------------

以上代码请自行引入SuperSocket的dll和System.configuration.dll