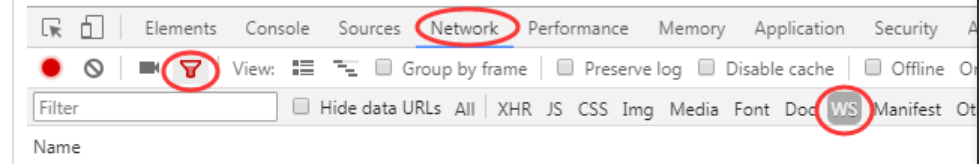


原来你是这样的Websocket--抓包分析

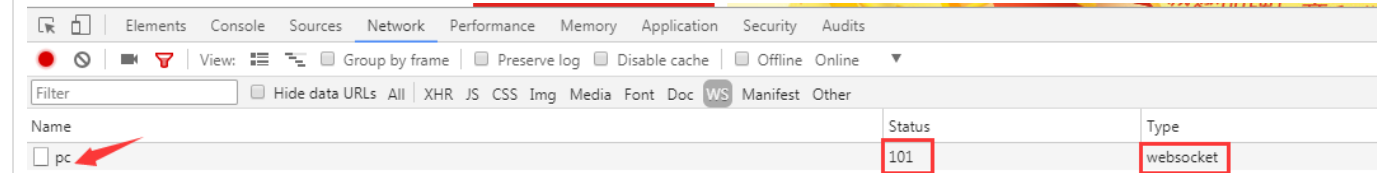
之前自己一个人负责完成了公司的消息推送服务，和移动端配合完成了扫码登录、订单消息。自己对Websocket协议的理解，自己通过进行抓包的方式学习了一番。现在分享出来，希望对

Chrome控制台

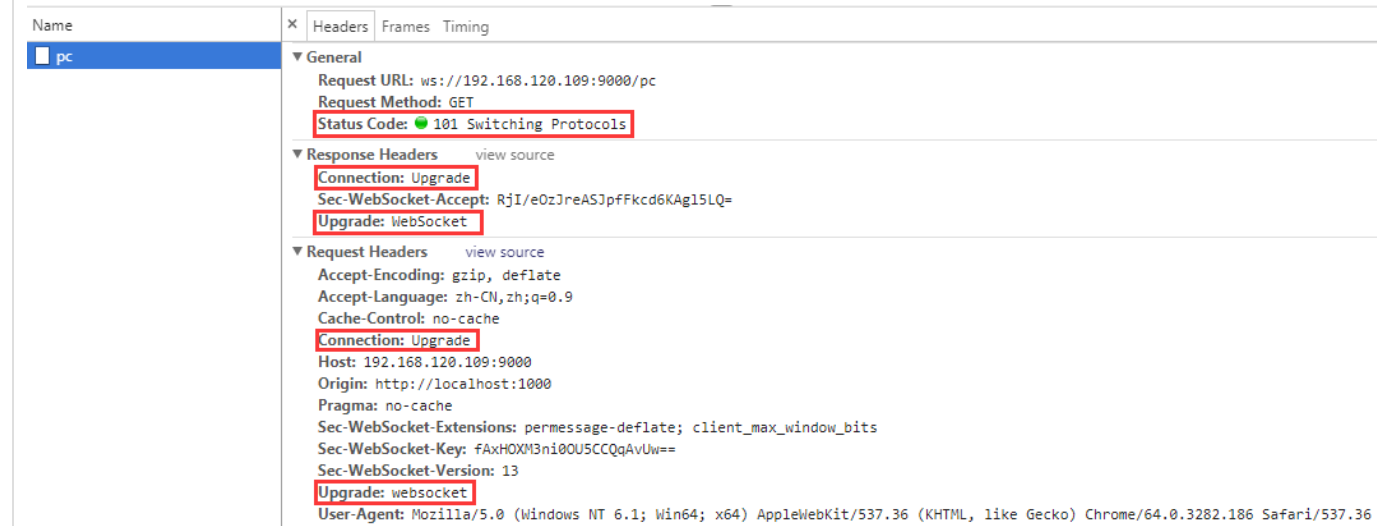
(1)F12进入控制台，点击Network，选中ws栏，注意选中Filter。



(2)刷新页面会得到一个ws链接。

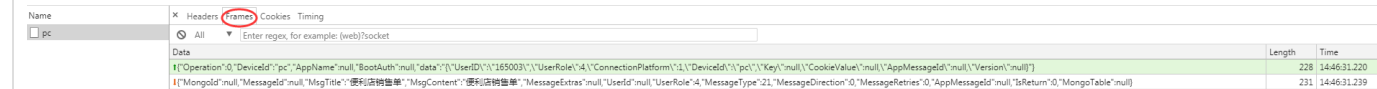


(3)点击链接可以查看链接详情



注意红框标出的信息，后面会详细说明。

(4)当然也可以切换到Frames查看发出和接收的消息,但是非常的简陋，只能看到消息内容，数据长度和时间



Fiddler:抓包调试利器

已保存 原来你是这样的Webso... 位于 我的第一个笔记本

Evernote 中的视图

删除剪裁

相关笔记

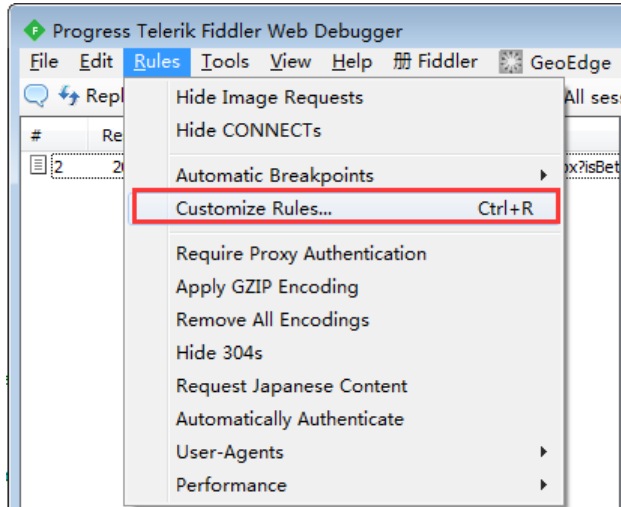
WebSocket和Stomp协议

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

(1)打开Fiddler，点开菜单栏的Rules，选择Customize Rules...



(2)这时会打开CustomRules.js文件，在class Handlers中加入以下代码

```
static function OnWebSocketMessage(oMsg: WebSocketMessage) {  
    // Log Message to the LOG tab  
    FiddlerApplication.Log.LogString(oMsg.ToString());  
}  
  
class Handlers  
{  
    static function OnWebSocketMessage(oMsg: WebSocketMessage) {  
        // Log Message to the LOG tab  
        FiddlerApplication.Log.LogString(oMsg.ToString());  
    }  
}
```

(3)保存后就可以在Fiddler右边栏的Log标签里，看到WebSocket的数据包

下列图中红框标出的Client.1代表客户端发出的第一条消息；对应的Server.1代表服务端发出的第一条消息。MessageType:Text代表正常的通话消息；Close代表会话关闭。

客户端发出的消息：

```
17:09:59:3490 WSSession565260.WebSocket'WebSocket #565260'  
MessageID: Client.1  
MessageType: Text  
PayloadString: {"Operation":0,"DeviceId":"pc","AppName":null,"BootAuth":null,"data":{"Userid":"1234567890","UserRole":4,"ConnectionPlatform":1,"DeviceId":"pc","Key":null,"CookieValue":null}}  
Masking: B2-9F-FF-85
```

服务端发出的消息：

```
17:09:59:4100 WSSession565260.WebSocket'WebSocket #565260'  
MessageID: Server.1  
MessageType: Text  
PayloadString: {"MongoId":"59a3911fc3b7321740dd9718","MessageId":"764fb037-3332-4324-b47c-1543577079742182812","MsgTitle":"ceshibianliandian","MsgContent":"土豆子","MessageExtras":{"我这是测试","UserId":"1234567890","UserRole":4,"M  
Masking: <none>
```

然后我们会发现每次会话关闭都是由客户端发起的：

```
17:11:58:8120 WSSession565260.WebSocket'WebSocket #565260'  
MessageID: Client.12  
MessageType: Close  
PayloadString:  
Masking: EB-A2-F1-64  
  
17:11:58:8130 WSSession565260.WebSocket'WebSocket #565260'  
MessageID: Server.13  
MessageType: Close  
PayloadString: 03-E8  
Masking: <none>
```

相对于Chrome控制台来说Fiddler抓包更加详细一些，能知道会话消息是由客户端还是服务端发出并且能知道消息类型。但是这仍然满足不了深入理解学习WebSocket协议的目的。如果是处理HTTP、HTTPS，还是用Fiddler。其他协议比如TCP,UDP 就用WireShark。TCP/IP协议是传输层协议，主要解决数据如何在网络中传输，而HTTP、WebSocket是应用层协议，主要解决如何包装数据。因为应用层是在传输层的基础上包装数据，所以我们还是从底层开始了解WebSocket到底是个啥?是如何工作的?

WireShark

WireShark（前称Ethereal）是一个网络封包分析软件。网络封包分析软件的功能是撷取网络封包资料。WireShark抓包是根据TCP/IP五层协议来的，也就是物理层、数据链路层、网络层和应用层。

TCP三次握手

我们都知道，TCP建立连接时，会有三次握手过程。下图是WireShark截获到的三次握手的手包是没有数据的）。

No.	Time	Source	Destination	Protocol	Length	Info
1616	42.543328	192.168.120.68	192.168.120.217	TCP	66	51000→9030 [SYN] Seq=0 Win=8192 Len=0 MSS=1460
1619	42.543797	192.168.120.217	192.168.120.68	TCP	66	9030→51000 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
1620	42.544226	192.168.120.68	192.168.120.217	TCP	60	51000→9030 [ACK] Seq=1 Ack=1 Win=65700 Len=0
1621	42.544758	192.168.120.68	192.168.120.217	HTTP	559	GET /pc HTTP/1.1
1622	42.545099	192.168.120.217	192.168.120.68	TCP	54	9030→51000 [ACK] Seq=1 Ack=506 Win=2048512 Len=0
1623	42.545690	192.168.120.217	192.168.120.68	HTTP	183	HTTP/1.1 101 Switching Protocols
1637	42.737998	192.168.120.68	192.168.120.217	TCP	60	51000→9030 [ACK] Seq=506 Ack=130 Win=65568 Len=0
2082	52.738475	192.168.120.217	192.168.120.68	TCP	55	[TCP Keep-Alive] 9030→51000 [ACK] Seq=129 Ack=506
2083	52.738756	192.168.120.68	192.168.120.217	TCP	66	[TCP Keep-Alive ACK] 51000→9030 [ACK] Seq=506
2120	53.383538	192.168.120.68	192.168.120.217	WebSock	255	WebSocket Text [FIN] [MASKED]
2121	53.388346	192.168.120.217	192.168.120.68	WebSock	335	WebSocket Text [FIN]
2132	53.589690	192.168.120.68	192.168.120.217	TCP	60	51000→9030 [ACK] Seq=707 Ack=411 Win=65288 Len=0
2431	63.589101	192.168.120.217	192.168.120.68	TCP	55	[TCP Keep-Alive] 9030→51000 [ACK] Seq=410 Ack=506
2432	63.589406	192.168.120.68	192.168.120.217	TCP	66	[TCP Keep-Alive ACK] 51000→9030 [ACK] Seq=707
2651	69.537057	192.168.120.217	192.168.120.68	WebSock	335	WebSocket Text [FIN]
2665	69.789161	192.168.120.68	192.168.120.217	TCP	60	51000→9030 [ACK] Seq=707 Ack=692 Win=65008 Len=0
2988	79.786558	192.168.120.217	192.168.120.68	TCP	55	[TCP Keep-Alive] 9030→51000 [ACK] Seq=691 Ack=506
2989	79.786966	192.168.120.68	192.168.120.217	TCP	66	[TCP Keep-Alive ACK] 51000→9030 [ACK] Seq=707
3139	85.259025	192.168.120.217	192.168.120.68	WebSock	335	WebSocket Text [FIN]
3151	85.464686	192.168.120.68	192.168.120.217	TCP	60	51000→9030 [ACK] Seq=707 Ack=973 Win=64728 Len=0
3383	95.467025	192.168.120.217	192.168.120.68	TCP	55	[TCP Keep-Alive] 9030→51000 [ACK] Seq=972 Ack=707 Win=2048256 Len=1
3384	95.467348	192.168.120.68	192.168.120.217	TCP	66	[TCP Keep-Alive ACK] 51000→9030 [ACK] Seq=707 Ack=973 Win=64728 Len=0 SLE=972 SRE=973
3691	101.48798	192.168.120.217	192.168.120.68	WebSock	335	WebSocket Text [FIN]
3698	101.68413	192.168.120.68	192.168.120.217	TCP	60	51000→9030 [ACK] Seq=707 Ack=1254 Win=64444 Len=0
3984	111.68150	192.168.120.217	192.168.120.68	TCP	55	[TCP Keep-Alive] 9030→51000 [ACK] Seq=1253 Ack=707 Win=2048256 Len=1
3985	111.68192	192.168.120.68	192.168.120.217	TCP	66	[TCP Keep-Alive ACK] 51000→9030 [ACK] Seq=707 Ack=1254 Win=64444 Len=0 SLE=1253 SRE=1254

已保存 原来你是这样的Webso... 位于 我的第一个笔记本

Evernote 中的视图

删除剪藏

相关笔记

WebSocket和Stomp协议

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

点击上图中的数据包就可以查看每个数据包的详情，这里我们需要明确几个概念才能看懂每个数据包代表啥意义：

SYN:同步比特，建立连接。

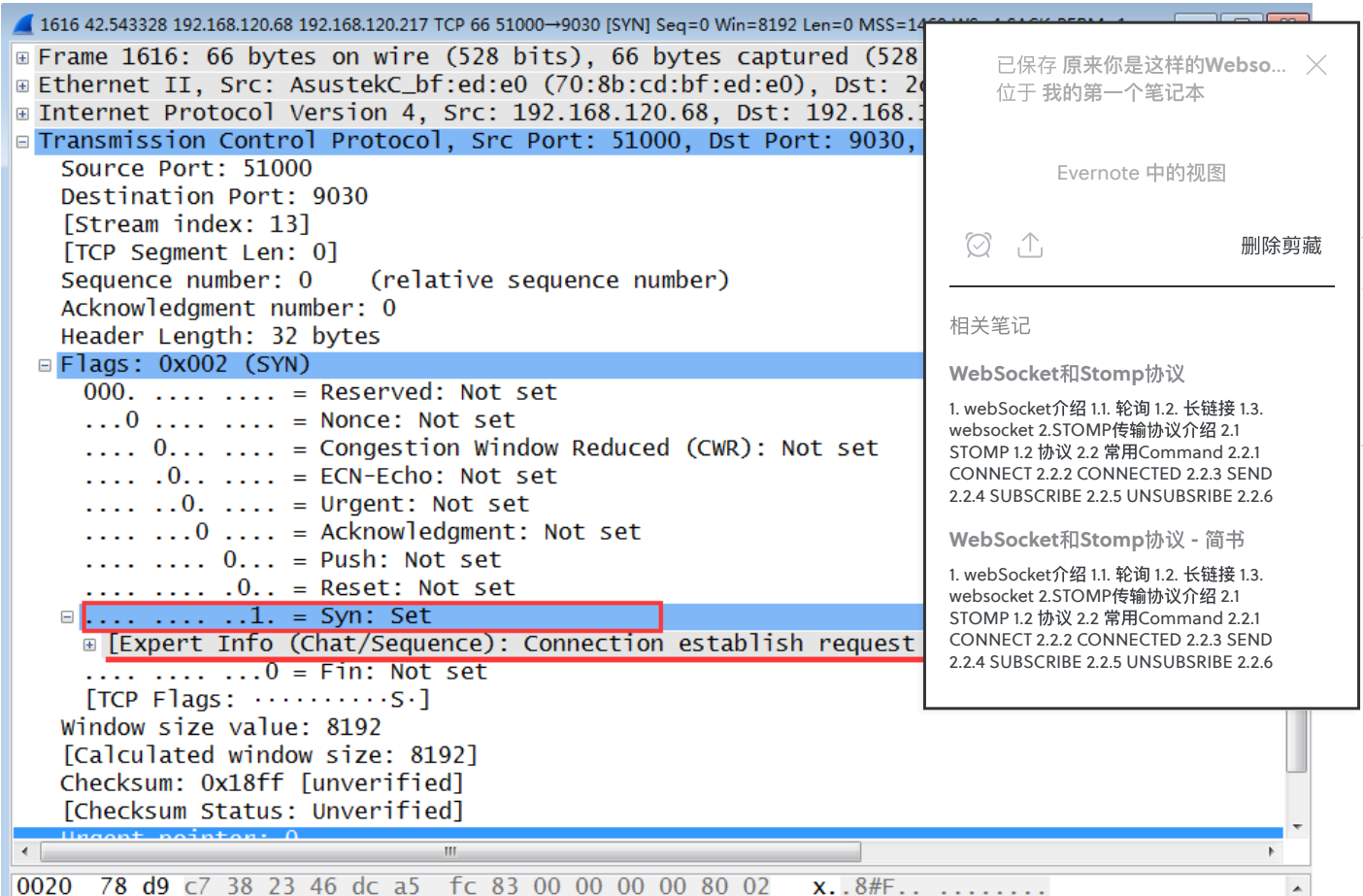
ACK:确认比特，置1表示这是一个确认的TCP包，0则不是。

PSH:推送比特，当发送端PSH=1时，接收端应尽快交付给应用进程。

• 第一次握手

可以看到我们打开的Transmission Control Protocol即为传输层（Tcp）

SYN置为1，客户端向服务端发送连接请求包。



已保存 原来你是这样的Webso...
位于 我的第一个笔记本

Evernote 中的视图



删除剪裁

相关笔记

WebSocket和Stomp协议

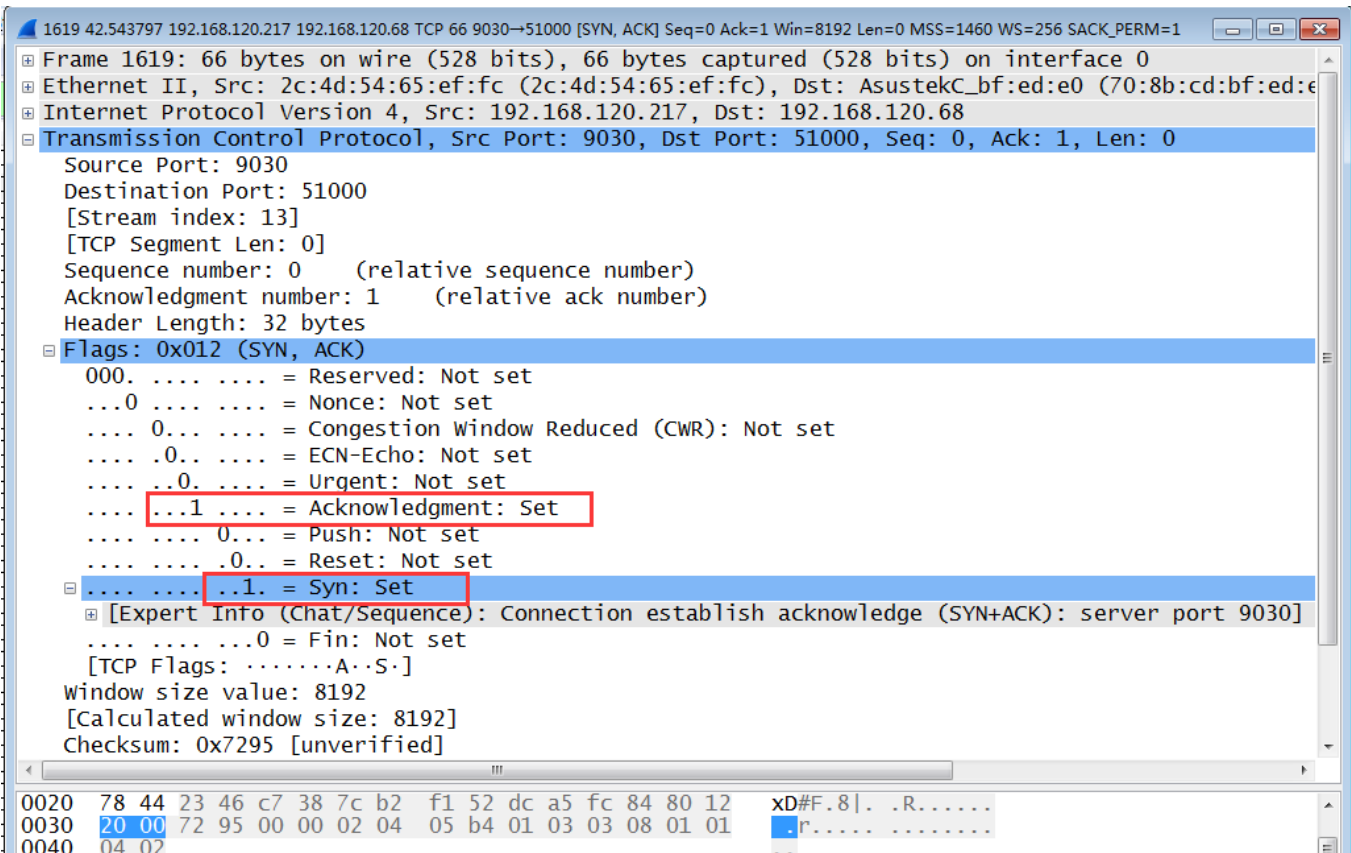
1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2. STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2. STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

• 第二次握手

服务器收到客户端发过来的TCP报文，由SYN=1知道客户端要求建立联机，向客户端发送一个SYN=1，ACK=1的TCP报文，将确认序号设置为客户端的序列号加1。



• 第三次握手

客户端接收到服务器发过来的包后检查确认序列号是否正确，即第一次发送的序号+1，以及标志位ACK是否为1。若正确则再次发送确认包，ACK标志为1。链接建立成功，可以发送数据了。

1620 42.544226 192.168.120.68 192.168.120.217 TCP 60 51000→9030 [ACK] Seq=1 Ack=1 Win=65700 L

Frame 1620: 60 bytes on wire (480 bits), 60 bytes captured (C)

Ethernet II, Src: AsustekC_bf:ed:e0 (70:8b:cd:bf:ed:e0), Dst

Internet Protocol Version 4, Src: 192.168.120.68, Dst: 192.1

Transmission Control Protocol, Src Port: 51000, Dst Port: 90

Source Port: 51000
Destination Port: 9030
[Stream index: 13]
[TCP Segment Len: 0]
Sequence number: 1 (relative sequence number)
Acknowledgment number: 1 (relative ack number)
Header Length: 20 bytes

Flags: 0x010 (ACK)

000. = Reserved: Not set
...0 = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
....0... = Push: Not set
....0.. = Reset: Not set
....0. = Syn: Not set
....0 = Fin: Not set
[TCP Flags:A....]
Window size value: 16425
[Calculated window size: 65700]
[window size scaling factor: 4]
Checksum: 0xcb8c [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
[SEQ/ACK analysis]

0000 2c 4d 54 65 ef fc 70 8b cd bf ed e0 08 00 45 00 ,MTe..p.
0010 00 28 20 2c 40 00 40 06 a8 35 c0 a8 78 44 c0 a8 .(,@.@. .5..xD
0020 78 d9 c7 38 23 46 d6 a5 fc 84 7c b2 f1 53 50 10 y 8#5 l S

已保存 原来你是这样的Webso... X
位于 我的第一个笔记本

Evernote 中的视图

删除剪裁

相关笔记

WebSocket和Stomp协议

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

一次特殊的HTTP请求

紧接着是一次Http请求（第四个包），说明Http的确是使用Tcp建立连接的。

先来看传输层（Tcp）：PSH（推送比特）置1，ACK置1，PSH置1说明开始发送数据，同时发送数据ACK要置1，因为需要接收到这个数据包的端给予确认。PSH为1的情况，一般只出现在 DATA内容不为0的包中，也就是说PSH为1表示的是有真正的TCP数据包内容被

传递。

```
Frame 1621: 559 bytes on wire (4472 bits), 559 bytes captured
Ethernet II, Src: AsustekC_bf:ed:e0 (70:8b:cd:bf:ed:e0), Dst:
Internet Protocol Version 4, Src: 192.168.120.68, Dst: 192.168.120.68
Transmission Control Protocol, Src Port: 51000, Dst Port: 9030
  Source Port: 51000
  Destination Port: 9030
  [Stream index: 13]
  [TCP Segment Len: 505]
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 506 (relative sequence number)]
  Acknowledgment number: 1 (relative ack number)
  Header Length: 20 bytes
  Flags: 0x018 (PSH, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    .... 0... = Congestion Window Reduced (CWR): Not set
    .... .0.. = ECN-Echo: Not set
    .... ..0. = Urgent: Not set
    .... ...1... = Acknowledgment: Set
    .... .... 1... = Push: Set
    .... .....0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
  [TCP Flags: .....AP...]
  Window size value: 16425
  [Calculated window size: 65700]
  [Window size scaling factor: 4]
  Checksum: 0x2489 [unverified]
```

已保存 原来你是这样的Webso... X
位于 我的第一个笔记本

Evernote 中的视图



删除剪裁

相关笔记

WebSocket和Stomp协议

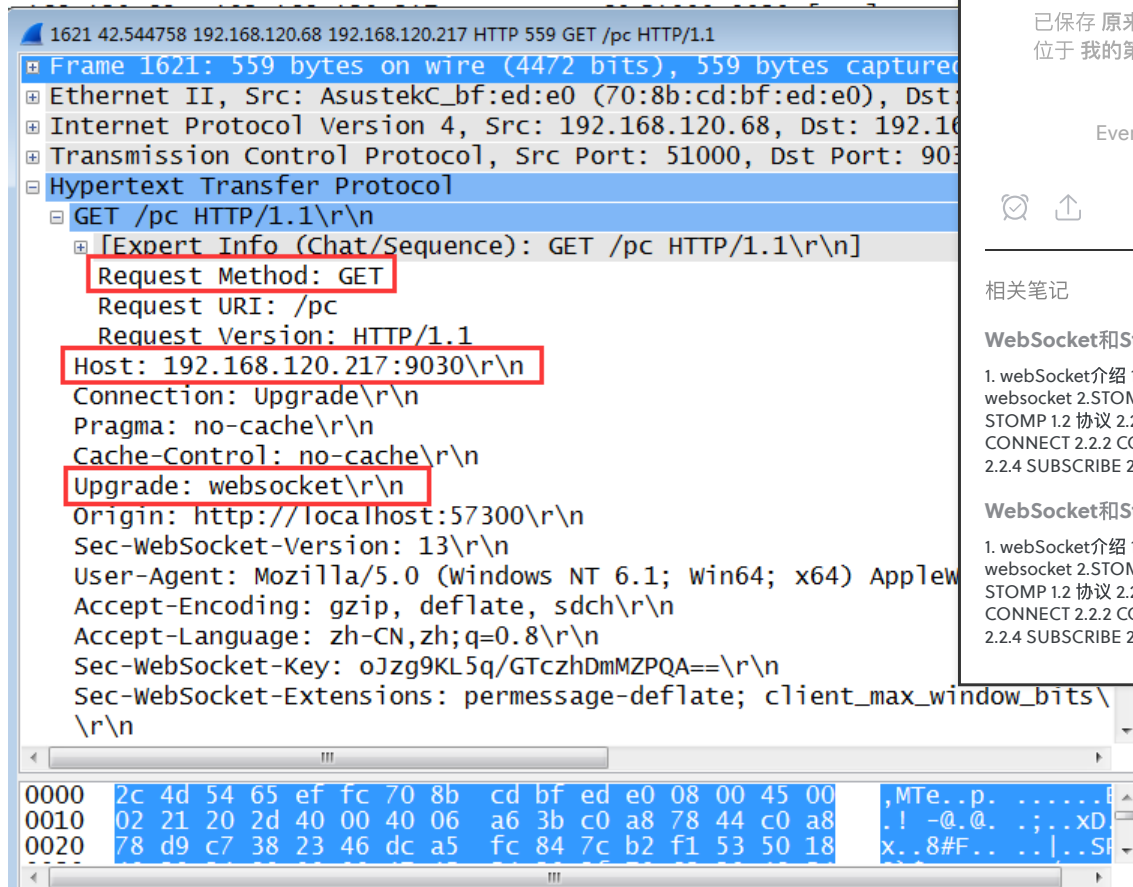
1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

再来看应用层 (Http)：这是一次特殊的Http请求，为什么是一次特殊的Http请求呢？Http请求头中Connection:Upgrade Upgrade:websocket,Upgrade代表升级到较新的Http协议或者切换到不同的协议。很明显WebSocket使用此机制以兼容的方式与HTTP服务器建立连接。WebSocket协议有两个部分：握手建立升级后的连接，然后进行实际的数据传输。首先，客户端通过使用 Upgrade: WebSocket和Connection: Upgrade头部以及一些特定于协议的头来请求WebSocket连接，以建立正在使用的版本并设置握手。服务器，如果它支持协议，回复与相同Upgrade: WebSocket和Connection: Upgrade标题，并完成握手。握手完成后，数据传输开始。这些信息在前面的Chrome控制台中也可以看到。

请求:



已保存 原来你是这样的Webso... X
位于 我的第一个笔记本

Evernote 中的视图



删除剪裁

相关笔记

WebSocket和Stomp协议

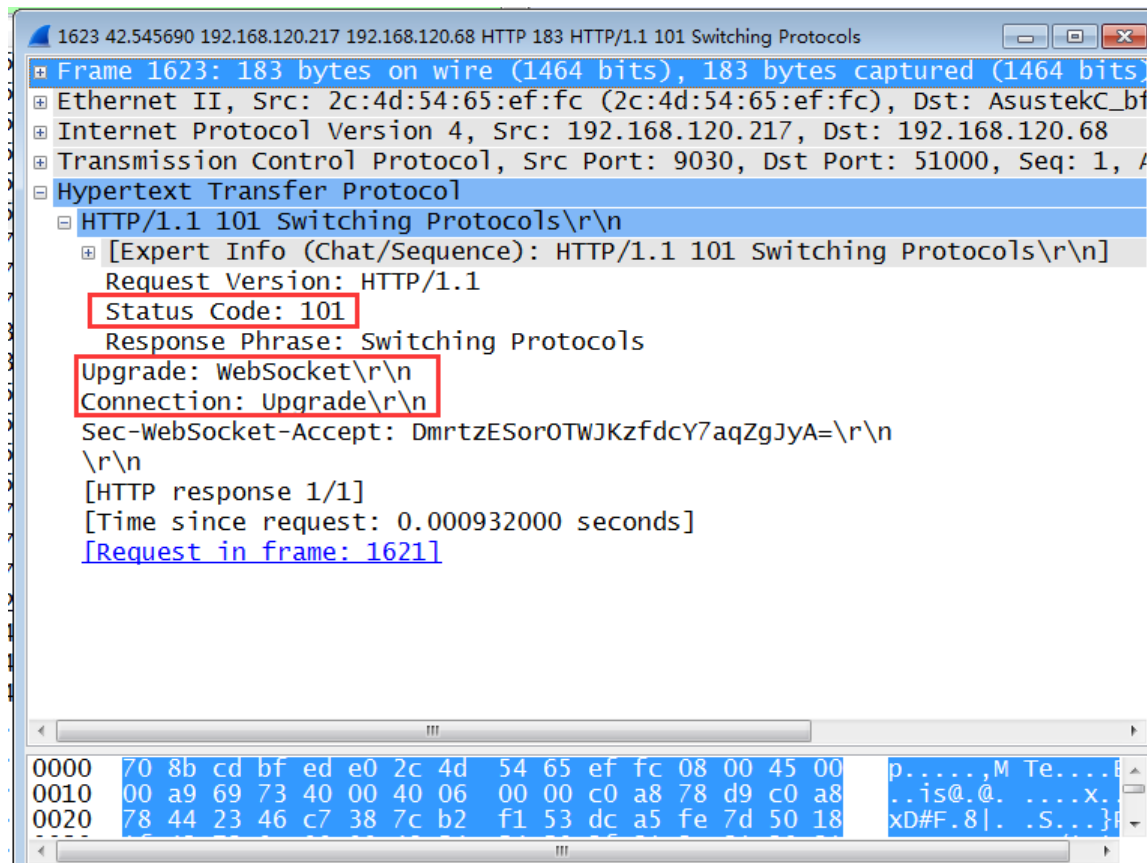
1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

响应:

响应状态码 101 表示服务器已经理解了客户端的请求, 在发送完这个响应后, 服务器将会切换到在Upgrade请求头中定义的那些协议。



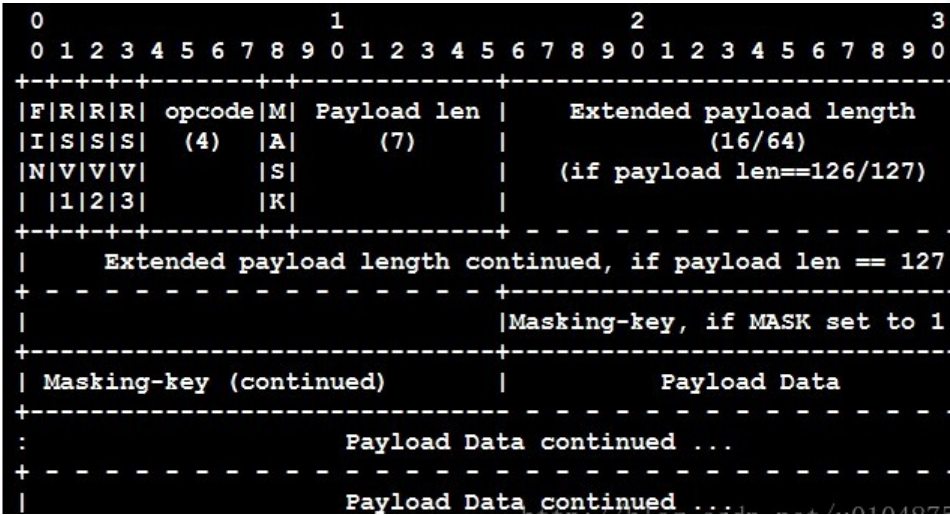
由此我们可以总结出:

Websocket协议本质上是一个基于TCP的协议。建立连接需要握手, 客户端 (浏览器) 首先向服务器 (web server) 发起一条特殊的

http请求，web server解析后生成应答到浏览器，这样子一个websocket连接就建立了，直到其中一方关闭连接。

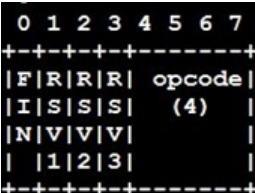
Websocket的世界

通信协议格式是WebSocket格式，服务器端采用Tcp Socket方式接收数据，进行解析，协议



首先我们需要知道数据在物理层，数据链路层是以二进制进行传递的，而在应用层是以16进制字节流进行传输的。

第一个字节：

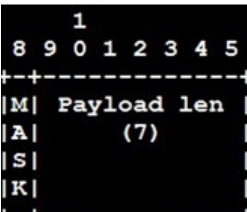


- FIN:1位，用于描述消息是否结束，如果为1则该消息为消息尾部,如果为零则还有后续数据包;
- RSV1,RSV2,RSV3：各1位，用于扩展定义的,如果没有扩展约定的情况则必须为0
- OPCODE:4位，用于表示消息接收类型，如果接收到未知的opcode，接收端必须关闭连接。

Webdocket数据帧中OPCODE定义：

- 0x0表示附加数据帧
- 0x1表示文本数据帧
- 0x2表示二进制数据帧
- 0x3-7暂时无定义，为以后的非控制帧保留
- 0x8表示连接关闭
- 0x9表示ping
- 0xA表示pong
- 0xB-F暂时无定义，为以后的控制帧保留

第二个字节：



- MASK:1位，用于标识PayloadData是否经过掩码处理，客户端发出的数据帧需要进行掩码处理，所以此位是1。数据需要解码。
- PayloadData的长度：7位，7+16位，7+64位
- 如果其值在0-125，则是payload的真实长度。
- 如果值是126，则后面2个字节形成的16位无符号整型数的值是payload的真实长度。
- 如果值是127，则后面8个字节形成的64位无符号整型数的值是payload的真实长度。

已保存 原来你是这样的Webso... X
位于 我的第一个笔记本

Evernote 中的视图

 删除剪藏

相关笔记

WebSocket和Stomp协议

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket

1... .. = Fin: True

.000 = Reserved: 0x0

.... 0001 = Opcode: Text (1)

1... .. = Mask: True

.111 1110 = Payload length: 126 Extended Payload Length (

Extended Payload length (16 bits): 193

Masking-Key: ab9accc1

Masked payload

Payload

Line-based text data

0000 2c 4d 54 65 ef fc 70 8b cd bf ed e0 08 00 45 00 ,MT

0010 00 f1 24 93 40 00 40 06 a3 05 c0 a8 78 44 c0 a8 ..\$

0020 78 d9 c7 38 23 46 dc a5 fe 7d 7c b2 f1 d4 50 18 x..

0030 40 08 2b 07 00 00 81 fe 00 c1 ab 9a cc c1 d0 b8 @.+

0040 83 b1 ce e8 ad b5 c2 f5 a2 e3 91 aa e0 e3 ef ff ...

0050 ba a8 c8 ff 85 a5 89 a0 ee b1 c8 b8 e0 e3 ea ea ...

0060 bc 8f ca f7 a9 e3 91 f4 b9 ad c7 b6 ee 83 c4 f5 ...

0070 b8 80 de ee a4 e3 91 f4 b9 ad c7 b6 ee a5 ca ee ...

0080 ad e3 91 b8 b7 9d 89 cf bf a4 d9 d3 88 9d 89 a0 ...

0090 90 e3 9a a8 ff f5 9e ac fb f9 92 aa 90 e3 87 c6

00a0 ee 94 d8 ff be 93 c4 f6 a9 9d 89 a0 f8 ed f7 b8

00b0 8f ae c5 f4 a9 a2 df f3 a3 af fb f6 ad b5 cd f5

00c0 be ac f7 b8 f6 f0 87 c6 ee 85 ce ec a5 a2 ce d3

00d0 a8 9d 89 a0 90 e3 db f9 90 e3 87 c6 ee 8a ce e3

00e0 90 e3 91 f4 b9 ad c7 b6 90 e3 e8 f5 a3 aa c2 ff

00f0 9a a0 c7 ef a9 9d 89 a0 a2 b4 c7 f6 b1 e3 d6

已保存 原来你是这样的Webso... X

位于 我的第一个笔记本

Evernote 中的视图

删除剪裁

相关笔记

WebSocket和Stomp协议

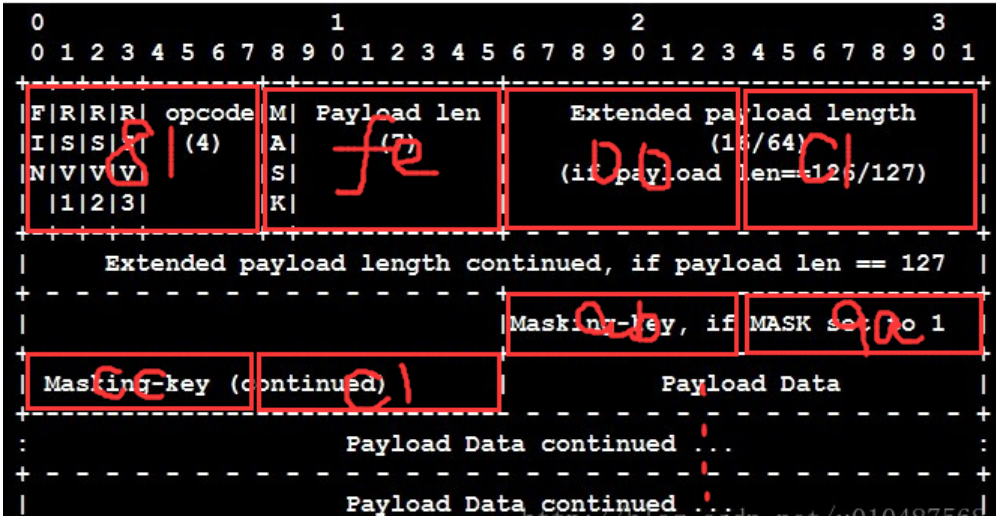
1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

上图是客户端发送给服务端的数据包，其中PayloadData的长度为二进制：01111110-->十进制：126；如果值是126，则后面2个字节形成的16位无符号整型数的值是payload的真实长度。也就是圈红的十六进制：00C1-->十进制：193 byte。所以PayloadData的真实数据长度是193 bytes；

根据我们的分析，客户端到服务端数据包 websocket 帧图应该为：



我们再来抓包分析一下服务器到客户端的数据包：

WebSocket

1... = Fin: True
 .000 = Reserved: 0x0
 0001 = Opcode: Text (1)
0... = Mask: False
 .111 1110 = Payload length: 126 Extended Payload Length (16 bits): 277
 Extended Payload length (16 bits): 277
 Payload

Line-based text data

已保存 原来你是这样的Webso...
 位于 我的第一个笔记本

Evernote 中的视图

删除剪裁

相关笔记

WebSocket和Stomp协议

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

0030	1f 41 73 a2 00 00 81 7e 01 15 7b 22 4d 6f 6e 67
0040	6f 49 64 22 3a 22 35 39 39 66 65 61 63 65 63 33
0050	62 37 33 32 31 38 38 34 39 35 61 34 35 34 22 2c
0060	22 4d 65 73 73 61 67 65 49 64 22 3a 22 35 34 66
0070	34 64 33 33 30 2d 38 39 39 66 2d 34 33 62 63 2d
0080	61 66 39 33 2d 62 36 39 30 34 38 32 30 33 32 30
0090	62 31 34 30 39 38 35 37 32 22 2c 22 4d 73 67 54
00a0	69 74 6c 65 22 3a 22 63 65 73 68 69 62 69 61 6e
00b0	6c 69 64 69 61 6e 22 2c 22 4d 73 67 43 6f 6e 74
00c0	65 6e 74 22 3a 22 e5 9c 9f e8 b1 86 e5 ad 90 22
00d0	2c 22 4d 65 73 73 61 67 65 45 78 74 72 61 73 22
00e0	3a 22 e6 88 91 e8 bf 99 e6 98 af e6 b5 8b e8 af
00f0	95 22 2c 22 55 73 65 72 49 64 22 3a 22 31 32 33
0100	34 35 36 37 38 39 30 22 2c 22 55 73 65 72 52 6f
0110	6c 65 22 3a 34 2c 22 4d 65 73 73 61 67 65 54 79
0120	70 65 22 3a 32 31 2c 22 4d 65 73 73 61 67 65 44
0130	69 72 65 63 74 69 6f 6e 22 3a 31 2c 22 4d 65 73
0140	73 61 67 65 52 65 74 72 69 65 73 22 3a 30 7d

可以发现服务器发送给客户端的数据包中第二个字节中MASK位为0，这说明服务器发送的数据帧未经过掩码处理，这个我们从客户端和服务端的数据包截图中也可以发现，客户端的数据被加密处理，而服务端的数据则没有。（如果服务器收到客户端发送的未经掩码处理的数据包，则会自动断开连接；反之，如果客户端收到了服务端发送的经过掩码处理的数据包，也会自动断开连接）。

掩码处理:

```
,MTe..p. ....E.  
..$.@.@. ....xD.  
x..8#F.. }|...P.  
@.+... ..  
  
.....  
.....  
.....  
.....  
.....  
.....  
.....
```

未掩码处理：

```
.As...~ ..{"Mong
oId":"59 9feacec3
b7321884 95a454",
"Message Id":"54f
4d330-89 9f-43bc-
af93-b69 04820320
b1409857 2","MsgT
itle":"c eshibian
lidian", "MsgCont
ent":".. ....."
,"Messag eExtras"
:".....
.", "User Id":"123
4567890" ,"UserRo
le":4,"M essageTy
pe":21," MessageD
irection ":1,"Mes
sageRetr ies":0}
```

已保存 原来你是这样的Webso... ✕
位于 我的第一个笔记本

Evernote 中的视图

删除剪藏

相关笔记

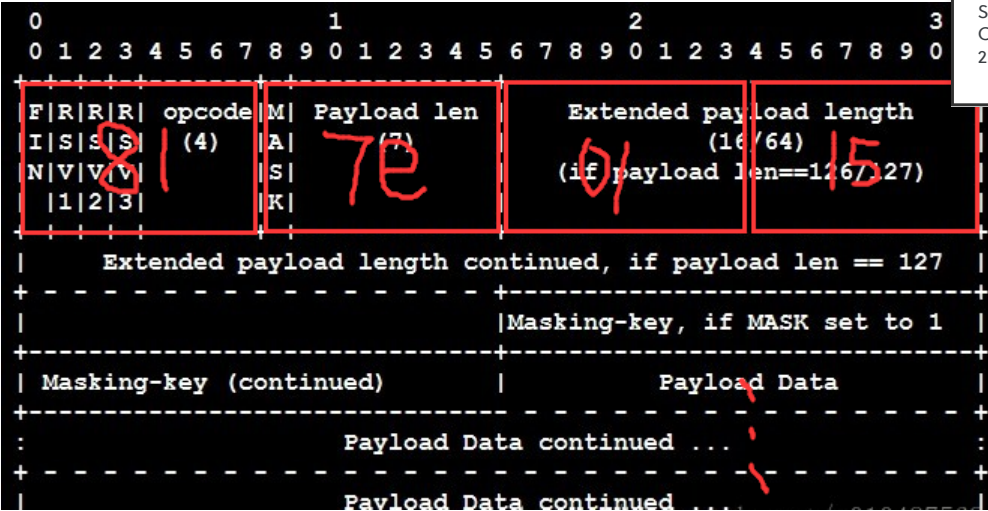
WebSocket和Stomp协议

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

根据我们的分析，服务端到客户端数据包的websocket帧图应该为：



TCP KeepAlive

...	111.68...	192.168.120.217	192.168.120.68	TCP	55 [TCP Keep-Alive] 9030→51000 [ACK] Seq=
...	111.68...	192.168.120.68	192.168.120.217	TCP	66 [TCP Keep-Alive ACK] 51000→9030 [ACK]
...	121.68...	192.168.120.217	192.168.120.68	TCP	55 [TCP Keep-Alive] 9030→51000 [ACK] Seq=
...	121.68...	192.168.120.68	192.168.120.217	TCP	66 [TCP Keep-Alive ACK] 51000→9030 [ACK]
...	131.68...	192.168.120.217	192.168.120.68	TCP	55 [TCP Keep-Alive] 9030→51000 [ACK] Seq=
...	131.68...	192.168.120.68	192.168.120.217	TCP	66 [TCP Keep-Alive ACK] 51000→9030 [ACK]
...	141.68...	192.168.120.217	192.168.120.68	TCP	55 [TCP Keep-Alive] 9030→51000 [ACK] Seq=
...	141.68...	192.168.120.68	192.168.120.217	TCP	66 [TCP Keep-Alive ACK] 51000→9030 [ACK]
...	146.48...	192.168.120.68	192.168.120.217	TCP	60 [TCP Keep-Alive] 51000→9030 [ACK] Seq=
...	146.48...	192.168.120.217	192.168.120.68	TCP	66 [TCP Keep-Alive ACK] 9030→51000 [ACK]

如上图所示，TCP保活报文总是成对出现，包括TCP保活探测报文和TCP保活探测确认报文。
TCP保活探测报文是之前TCP报文的确认序列号减1，并设置1个字节，内容为“00”的应用层数据，如下图所示：

Wireshark · 分组 3984 · websocket

Frame 3984: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0

Ethernet II, Src: 2c:4d:54:65:ef:fc (2c:4d:54:65:ef:fc), Dst: Asustek

Internet Protocol Version 4, Src: 192.168.120.217, Dst: 192.168.120.68

Transmission Control Protocol, Src Port: 9030, Dst Port: 51000, Seq: 1253

Source Port: 9030

Destination Port: 51000

[Stream index: 13]

[TCP Segment Len: 1]

Sequence number: 1253 (relative sequence number)

[Next sequence number: 1254 (relative sequence number)]

Acknowledgment number: 707 (relative ack number)

Header Length: 20 bytes

Flags: 0x010 (ACK)

Window size value: 8001

[Calculated window size: 2048256]

[Window size scaling factor: 256]

Checksum: 0x728a [unverified]

[Checksum Status: Unverified]

Urgent pointer: 0

[SEQ/ACK analysis]

Data (1 byte)

Data: 00

[Length: 1]

0000 70 8b cd bf ed e0 2c 4d 54 65 ef fc 08 00 45 00 p.....,M Te....E.

0010 00 29 6a bb 40 00 40 06 00 00 c0 a8 78 d9 c0 a8 .)j.@.@.x...

0020 78 44 23 46 c7 38 7c b2 f6 37 dc a5 ff 46 50 10 xD#F.8|. .7...FP.

0030 1f 41 72 8a 00 00 00 00 .Ar....

No.: 3984 · Time: 111.661501 · Source: 192.168.120.217 · Destination: 192.168.120.68 · Protocol: TCP · Length: 55 · Info: [TCP Keep-Alive] 9030→51000 [ACK] Seq=1253 Ack=707 Win=2048256 Len=1

TCP保活探测确认报文就是对保活探测报文的确认，其报文格式如下：

Wireshark · 分组 3985 · websocket

Frame 3985: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

Ethernet II, Src: AsustekC_bf:ed:e0 (70:8b:cd:bf:ed:e0), Dst: 2c:4d:54:65:ef:fc (2c:4d:54:65:ef:fc)

Internet Protocol Version 4, Src: 192.168.120.68, Dst: 192.168.120.217

Transmission Control Protocol, Src Port: 51000, Dst Port: 9030, Seq: 707, Ack: 1254, Len: 0

Source Port: 51000

Destination Port: 9030

[Stream index: 13]

[TCP Segment Len: 0]

Sequence number: 707 (relative sequence number)

Acknowledgment number: 1254 (relative ack number)

Header Length: 32 bytes

Flags: 0x010 (ACK)

Window size value: 16111

[Calculated window size: 64444]

[Window size scaling factor: 4]

Checksum: 0xa932 [unverified]

[Checksum Status: Unverified]

Urgent pointer: 0

Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), SACK

[SEQ/ACK analysis]

0000 2c 4d 54 65 ef fc 70 8b cd bf ed e0 08 00 45 00 ,MTe..p.E.

0010 00 34 39 54 40 00 40 06 8f 01 c0 a8 78 44 c0 a8 .49T@.@.xD..

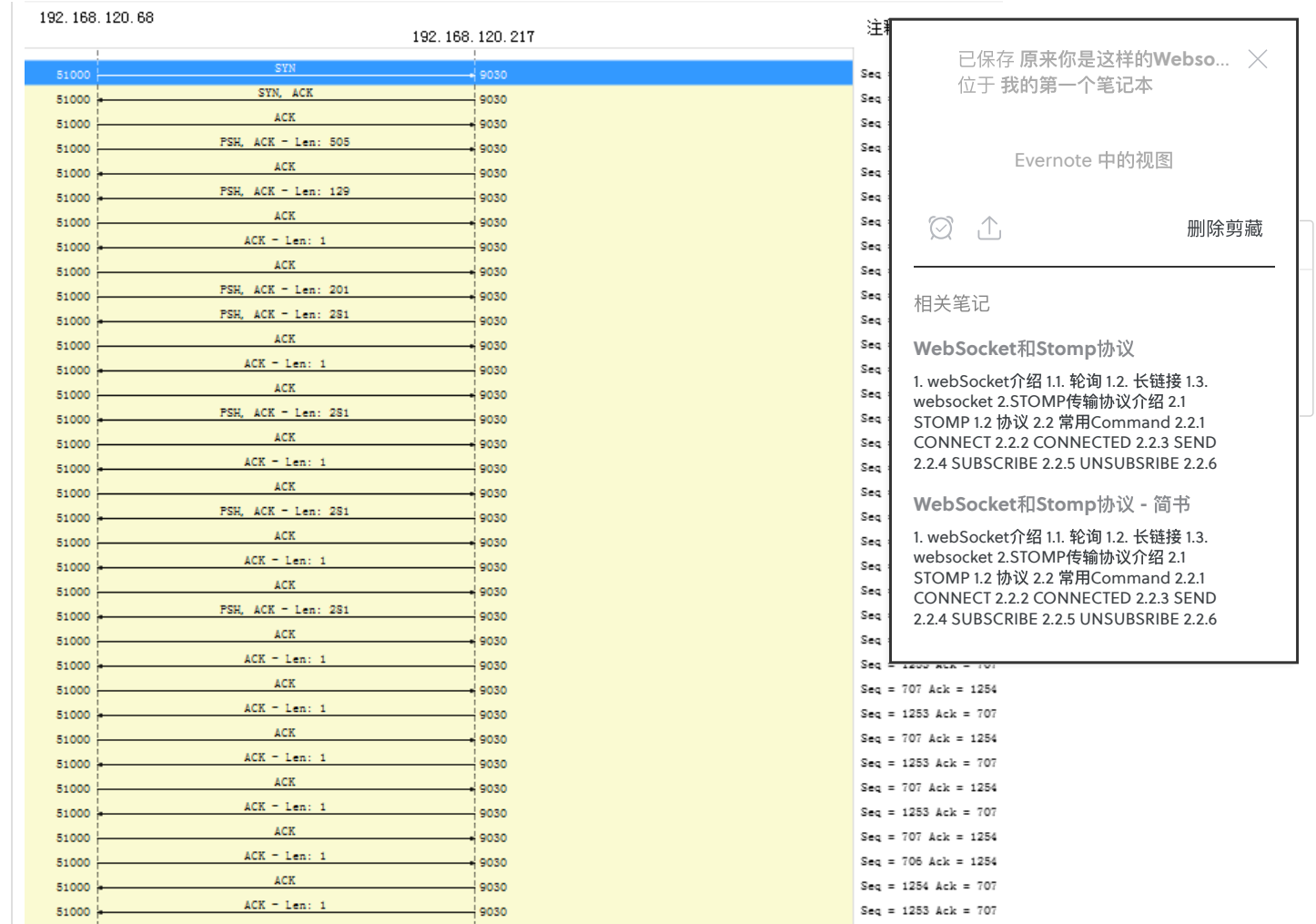
0020 78 d9 c7 38 23 46 dc a5 ff 46 7c b2 f6 38 80 10 x..8#F.. .F|.8..

0030 3e ef a9 32 00 00 01 01 05 0a 7c b2 f6 37 7c b2 >..2.... ..|.7|.

0040 f6 38 .8

No.: 3985 · Time: 111.661920 · Source: 192.168.120.68 · Destination: 192.168.120.217 · Protocol: TCP · Length: 66 · Info: [TCP Keep-Alive ACK] 51000→9030 [ACK] Seq=707 Ack=1254 Win=64444 Len=0 SLE=1253 SRE=1254

因为Websocket通过Tcp Socket方式工作，现在考虑一个问题，在一次长连接中，服务器怎么知道消息的顺序呢？这就涉及到tcp的序列号（Sequence Number）和确认号（Acknowledgment Number）。我的理解是序列号是发送的数据长度；确认号是接收的数据长度。这样讲比较抽象，我们从TCP三次握手开始（结合下图）详细分析一下。



包1:
TCP会话的每一端的序列号都从0开始，同样的，确认号也从0开始，因为此时通话还未开始，没有通话的另一端需要确认

包2:
服务端响应客户端的请求，响应中附带序列号0（由于这是服务端在该次TCP会话中发送的第一个包，所以序列号为0）和相对确认号1（表明服务端收到了客户端发送的包1中的SYN）。需要注意的是，尽管客户端没有发送任何有效数据，确认号还是被加1，这是因为接收的包中包含SYN或FIN标志位。

包3:
和包2中一样，客户端使用确认号1响应服务端的序列号0，同时响应中也包含了客户端自己的序列号（由于服务端发送的包中确认收到了客户端发送的SYN，故客户端的序列号由0变为1）此时，通信的两端的序列号都为1。

包4: 客户端-->服务器
这是流中第一个携带有效数据的包（确切的说，是客户端发送的HTTP请求），序列号依然为1，因为到上个包为止，还没有发送任何数据，确认号也保持1不变，因为客户端没有从服务端接收到任何数据。需要注意的是，包中有效数据的长度为505字节

包5: 服务器-->客户端
当上层处理HTTP请求时，服务端发送该包来确认客户端在包4中发来的数据，需要注意的是，确认号的值增加了505（505是包4中有效数据长度），变为506，简单来说，服务端以此来告知客户端端，目前为止，我总共收到了506字节的数据，服务端的序列号保持为1不变。

包6: 服务器-->客户端
这个包标志着服务端返回HTTP响应的开始，序列号依然为1，因为服务端在该包之前返回的包中都不带有有效数据，该包带有129字节的有效数据。

包7:
由于上个数据包的发送，TCP客户端的确认序列号增长至130，从服务端接收了129字节的数据，客户端的确认号由1增长至130
理解了序列号和确认序列号是怎么工作的之后，我们也就知道“TCP保活探测报文是将之前TCP报文的确认序列号减1，并设置1个字节”为什么要这么搞了。减一再加一，是为了保证一次连接中keep alive不影响序列号和确认序列号。Keep alive 中的1byte 00的数据并不是真正要传递的数据，而是tcp keep alive约定俗称的规则。

总结：

WebSocket 是一个独立的基于 TCP 的协议，它与 HTTP 之间的唯一关系就是它的握手（handshake，即 http request）经由 HTTP 服务器解释。再严谨一点:WebSocket是一个网络通讯协议,只要理解基于websokect的即时通讯。

分类: [WebSocket](#)

推荐 22 反对 0

« 上一篇: [我看依赖注入](#)
» 下一篇: [【眼见为实】数据库并发问题 封锁协议 隔离级别](#)

posted @ 2018-03-18 23:01 范海涛

评论列表

1楼 2018-03-18 23:01 范海涛



大赞，
请问一下，如果是https请求，wireshark还能够解密吗？像fiddler一样

支持(0) 反对(0)

2楼 [楼主] 2018-03-19 08:29 CoderFocus



@ 尼玛范爷
谢谢你的赞。
可以的，Wireshark中支持SSL解析器，具体的操作可以参照一下网上的教程。

支持(0) 反对(0)

3楼 2018-03-19 10:14 范海涛



@ 喜欢天黑却怕鬼
好的，谢谢。上次搞了一阵子没弄明白

支持(0) 反对(0)

4楼 2018-03-26 13:34 捕头的爱



李团长到此一游！

支持(0) 反对(0)

5楼 2018-12-01 11:05 SuriFuture



请教下，我使用带参数的url发起websocket请求，建立连接后，之后的通信往来还会包含这些url参数吗？还是直接和服务器的ip地址通讯？

支持(0) 反对(0)

6楼 [楼主] 2018-12-01 14:16 CoderFocus



@ SuriFuture
不会 你的参数只是作用于发起链接 此时还是使用http协议 链接建立后 后续的通信都是建立在这次链接上的

支持(0) 反对(0)

7楼 2019-01-16 17:06 沙漠寒雪

阿里云全站加速DCDN全面支持WebSocket协议
<https://yq.aliyun.com/articles/686839?spm=a2c4e.11153959.0.0.d2924403Ln7z49>

支持(0) 反对(0)

已保存 原来你是这样的Webso... ✕
位于 我的第一个笔记本

Evernote 中的视图


删除剪藏


相关笔记

WebSocket和Stomp协议
1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书
1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

注册用户登录后才能发表评论，请[登录](#)或[注册](#)，[访问](#)网站首页。

Copyright © 2019 CoderFocus
Powered by .NET Core 3.0.0 on Linux
Powered By [Cnblogs](#) | [Silence](#) Theme By [Esofar](#)


已保存 原来你是这样的Webso... 
位于 我的第一个笔记本

Evernote 中的视图



删除剪裁

相关笔记

WebSocket和Stomp协议

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6

WebSocket和Stomp协议 - 简书

1. websocket介绍 1.1. 轮询 1.2. 长链接 1.3. websocket 2.STOMP传输协议介绍 2.1 STOMP 1.2 协议 2.2 常用Command 2.2.1 CONNECT 2.2.2 CONNECTED 2.2.3 SEND 2.2.4 SUBSCRIBE 2.2.5 UNSUBSCRIBE 2.2.6