38 | WebSocket：沙盒里的TCP

2019-08-23 Chrono

透视HTTP协议

进入课程 >

讲述：Chrono

时长 11:36　大小 13.28M

在之前讲 TCP/IP 协议栈的时候，我说过有“TCP Socket”，它实际上是一种功能接口，通过这些接口就可以使用 TCP/IP 协议栈在传输层收发数据。

那么，你知道还有一种东西叫“WebSocket”吗？

单从名字上看，“Web”指的是 HTTP，“Socket”是套接字调用，那么这两个连起来又是什么意思呢？

所谓“望文生义”，大概你也能猜出来，“WebSocket”就是运行在“Web”，也就是 HTTP 上的 Socket 通信规范，提供与“TCP Socket”类似的功能，使用它就可以像“TCP Socket”一样调用下层协议栈，任意地收发数据。

更准确地说，“WebSocket”是一种基于 TCP 的轻量级网络通信协议，在地位上是与 HTTP“平级”的。

为什么要有 WebSocket

不过，已经有了被广泛应用的 HTTP 协议，为什么要再出一个 WebSocket 呢？它有哪些好处呢？

其实 WebSocket 与 HTTP/2 一样，都是为了解决 HTTP 某方面的缺陷而诞生的。HTTP/2 针对的是“队头阻塞”，而 WebSocket 针对的是“请求 - 应答”通信模式。

那么，“请求 - 应答”有什么不好的地方呢？

“请求 - 应答”是一种“**半双工**”的通信模式，虽然可以双向收发数据，但同一时刻只能一个方向上有动作，传输效率低。更关键的一点，它是一种“被动”通信模式，服务器只能“被动”响应客户端的请求，无法主动向客户端发送数据。

虽然后来的 HTTP/2、HTTP/3 新增了 Stream、Server Push 等特性，但“请求 - 应答”依然是主要的工作方式。这就导致 HTTP 难以应用在动态页面、即时消息、网络游戏等要求“**实时通信**”的领域。

在 WebSocket 出现之前，在浏览器环境里用 JavaScript 开发实时 Web 应用很麻烦。因为浏览器是一个“受限的沙盒”，不能用 TCP，只有 HTTP 协议可用，所以就出现了很多“变通”的技术，“**轮询**”（polling）就是比较常用的的一种。

简单地说，轮询就是不停地向服务器发送 HTTP 请求，问有没有数据，有数据的话服务器就用响应报文回应。如果轮询的频率比较高，那么就可以近似地实现“实时通信”的效果。

但轮询的缺点也很明显，反复发送无效查询请求耗费了大量的带宽和 CPU 资源，非常不经济。

所以，为了克服 HTTP“请求 - 应答”模式的缺点，WebSocket 就“应运而生”了。它原来是 HTML5 的一部分，后来“自立门户”，形成了一个单独的标准，RFC 文档编号是 6455。

WebSocket 的特点

WebSocket 是一个真正“**全双工**”的通信协议，与 TCP 一样，客户端和服务器都可以随时向对方发送数据，而不用像 HTTP“你拍一，我拍一”那么“客套”。于是，服务器就可以变得更加“主动”了。一旦后台有新的数据，就可以立即“推送”给客户端，不需要客户端轮询，“实时通信”的效率也就提高了。

WebSocket 采用了二进制帧结构，语法、语义与 HTTP 完全不兼容，但因为它的主要运行环境是浏览器，为了便于推广和应用，就不得不“搭便车”，在使用习惯上尽量向 HTTP 靠拢，这就是它名字里“Web”的含义。

服务发现方面，WebSocket 没有使用 TCP 的“IP 地址 + 端口号”，而是延用了 HTTP 的 URI 格式，但开头的协议名不是“http”，引入的是两个新的名字：“**ws**”和“**wss**”，分别表示明文和加密的 WebSocket 协议。

WebSocket 的默认端口也选择了 80 和 443，因为现在互联网上的防火墙屏蔽了绝大多数的端口，只对 HTTP 的 80、443 端口“放行”，所以 WebSocket 就可以“伪装”成 HTTP 协议，比较容易地“穿透”防火墙，与服务器建立连接。具体是怎么“伪装”的，我稍后再讲。

下面我举几个 WebSocket 服务的例子，你看看，是不是和 HTTP 几乎一模一样：

复制代码

```
ws://www.chrono.com
ws://www.chrono.com:8080/srv
wss://www.chrono.com:445/im?user_id=xxx
```

要注意的一点是，WebSocket 的名字容易让人产生误解，虽然大多数情况下我们会在浏览器里调用 API 来使用 WebSocket，但它不是一个“调用接口的集合”，而是一个通信协议，所以我觉得把它理解成“**TCP over Web**”会更恰当一些。

WebSocket 的帧结构

刚才说了，WebSocket 用的也是二进制帧，有之前 HTTP/2、HTTP/3 的经验，相信你这次也能很快掌握 WebSocket 的报文结构。

不过 WebSocket 和 HTTP/2 的关注点不同，WebSocket 更**侧重于“实时通信”**，而 HTTP/2 更侧重于提高传输效率，所以两者的帧结构也有很大的区别。

WebSocket 虽然有“帧”，但却没有像 HTTP/2 那样定义“流”，也就不存在“多路复用”“优先级”等复杂的特性，而它自身就是“全双工”的，也就不需要“服务器推送”。所以综合起来，WebSocket 的帧学习起来会简单一些。

下图就是 WebSocket 的帧结构定义，长度不固定，最少 2 个字节，最多 14 字节，看着好像很复杂，实际非常简单。

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-------+-+-------------+-------------------------------+
|F|R|R|R| opcode|M| Payload len |    Extended payload length    |
|I|S|S|S|  (4)  |A|     (7)     |             (16/64)           |
|N|V|V|V|       |S|             |   (if payload len==126/127)   |
| |1|2|3|       |K|             |                               |
+-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +
|     Extended payload length continued, if payload len == 127  |
+ - - - - - - - - - - - - - - - +-------------------------------+
|                               |Masking-key, if MASK set to 1  |
+-------------------------------+-------------------------------+
| Masking-key (continued)       |          Payload Data         |
+-------------------------------- - - - - - - - - - - - - - - - +
:                     Payload Data continued ...                :
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
|                     Payload Data continued ...                |
+---------------------------------------------------------------+
```

开头的两个字节是必须的，也是最关键的。

第一个字节的第一位“**FIN**”是消息结束的标志位，相当于 HTTP/2 里的“END_STREAM”，表示数据发送完毕。一个消息可以拆成多个帧，接收方看到“FIN”后，就可以把前面的帧拼起来，组成完整的消息。

“FIN”后面的三个位是保留位，目前没有任何意义，但必须是 0。

第一个字节的后 4 位很重要，叫**“Opcode**”，操作码，其实就是帧类型，比如 1 表示帧内容是纯文本，2 表示帧内容是二进制数据，8 是关闭连接，9 和 10 分别是连接保活的 PING 和 PONG。

第二个字节第一位是掩码标志位“**MASK**”，表示帧内容是否使用异或操作（xor）做简单的加密。目前的 WebSocket 标准规定，客户端发送数据必须使用掩码，而服务器发送则必须不使用掩码。

第二个字节后 7 位是“**Payload len**”，表示帧内容的长度。它是另一种变长编码，最少 7 位，最多是 7+64 位，也就是额外增加 8 个字节，所以一个 WebSocket 帧最大是 2^64。

长度字段后面是“**Masking-key**”，掩码密钥，它是由上面的标志位“MASK”决定的，如果使用掩码就是 4 个字节的随机数，否则就不存在。

这么分析下来，其实 WebSocket 的帧头就四个部分：“**结束标志位 + 操作码 + 帧长度 + 掩码**”，只是使用了变长编码的“小花招”，不像 HTTP/2 定长报文头那么简单明了。

我们的实验环境利用 OpenResty 的“lua-resty-websocket”库，实现了一个简单的 WebSocket 通信，你可以访问 URI“/38-1”，它会连接后端的 WebSocket 服务“ws://127.0.0.1/38-0”，用 Wireshark 抓包就可以看到 WebSocket 的整个通信过程。

下面的截图是其中的一个文本帧，因为它是客户端发出的，所以需要掩码，报文头就在两个字节之外多了四个字节的“Masking-key”，总共是 6 个字节。

```
WebSocket
    1... .... = Fin: True
    .000 .... = Reserved: 0x0
    .... 0001 = Opcode: Text (1)
    1... .... = Mask: True
    .000 1111 = Payload length: 15
    Masking-Key: 5aa62869
    Masked payload
    Payload

0000  02 00 00 00 45 00 00 3d  6e 27 40 00 80 06 00 00   ····E··= n'@·····
0010  7f 00 00 01 7f 00 00 01  e7 30 00 50 fc 73 d0 e5   ····· ·0·P·s··
0020  72 a5 ef 7a 50 18 08 04  75 19 00 00 81 8f 5a a6   r··zP··· u·····Z·
0030  28 69 32 c3 44 05 35 86  5f 0c 38 d5 47 0a 31 c3   (i2·D·5· _·8·G·1·
0040  5c                                                 \
```

而报文内容经过掩码，不是直接可见的明文，但掩码的安全强度几乎是零，用“Masking-key”简单地异或一下就可以转换出明文。

WebSocket 的握手

和 TCP、TLS 一样，WebSocket 也要有一个握手过程，然后才能正式收发数据。

这里它还是搭上了 HTTP 的“便车”，利用了 HTTP 本身的“协议升级”特性，“伪装”成 HTTP，这样就能绕过浏览器沙盒、网络防火墙等等限制，这也是 WebSocket 与 HTTP 的另一个重要关联点。

WebSocket 的握手是一个标准的 HTTP GET 请求，但要带上两个协议升级的专用头字段：

“Connection: Upgrade”，表示要求协议“升级”；

“Upgrade: websocket”，表示要“升级”成 WebSocket 协议。

另外，为了防止普通的 HTTP 消息被“意外”识别成 WebSocket，握手消息还增加了两个额外的认证用头字段（所谓的“挑战”，Challenge）：

Sec-WebSocket-Key：一个 Base64 编码的 16 字节随机数，作为简单的认证密钥；

Sec-WebSocket-Version：协议的版本号，当前必须是 13。

```
Hypertext Transfer Protocol
  GET /38-0 HTTP/1.1\r\n
  Upgrade: websocket\r\n
  Host: 127.0.0.1:80\r\n
  Sec-WebSocket-Key: y7KXwBSpVrxtkR0O+bQt+Q==\r\n
  Sec-WebSocket-Version: 13\r\n
  Connection: Upgrade\r\n
  \r\n
```

服务器收到 HTTP 请求报文，看到上面的四个字段，就知道这不是一个普通的 GET 请求，而是 WebSocket 的升级请求，于是就不走普通的 HTTP 处理流程，而是构造一个特殊的“101 Switching Protocols”响应报文，通知客户端，接下来就不用 HTTP 了，全改用 WebSocket 协议通信。（有点像 TLS 的“Change Cipher Spec”）

WebSocket 的握手响应报文也是有特殊格式的，要用字段“Sec-WebSocket-Accept”验证客户端请求报文，同样也是为了防止误连接。

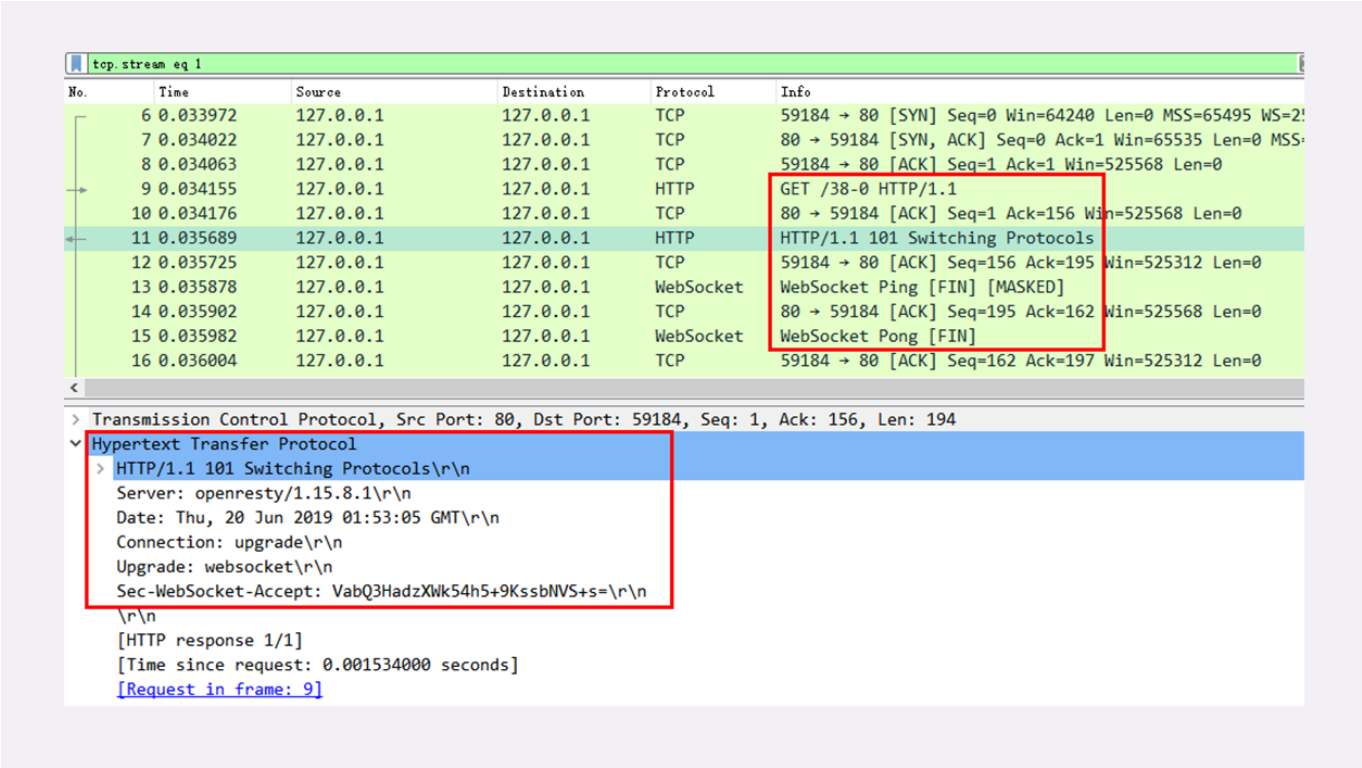
具体的做法是把请求头里“Sec-WebSocket-Key”的值，加上一个专用的 UUID “258EAFA5-E914-47DA-95CA-C5AB0DC85B11”，再计算 SHA-1 摘要。

复制代码

```
encode_base64(
  sha1(
    Sec-WebSocket-Key + '258EAFA5-E914-47DA-95CA-C5AB0DC85B11' ))
```

客户端收到响应报文，就可以用同样的算法，比对值是否相等，如果相等，就说明返回的报文确实是刚才握手时连接的服务器，认证成功。

握手完成，后续传输的数据就不再是 HTTP 报文，而是 WebSocket 格式的二进制帧了。



小结

浏览器是一个“沙盒”环境，有很多的限制，不允许建立 TCP 连接收发数据，而有了 WebSocket，我们就可以在浏览器里与服务器直接建立“TCP 连接”，获得更多的自由。

不过自由也是有代价的，WebSocket 虽然是在应用层，但使用方式却与“TCP Socket”差不多，过于“原始”，用户必须自己管理连接、缓存、状态，开发上比 HTTP 复杂的多，所以是否要在项目中引入 WebSocket 必须慎重考虑。

1. HTTP 的“请求 - 应答”模式不适合开发“实时通信”应用，效率低，难以实现动态页面，所以出现了 WebSocket；
2. WebSocket 是一个“全双工”的通信协议，相当于对 TCP 做了一层“薄薄的包装”，让它运行在浏览器环境里；
3. WebSocket 使用兼容 HTTP 的 URI 来发现服务，但定义了新的协议名“ws”和“wss”，端口号也沿用了 80 和 443；
4. WebSocket 使用二进制帧，结构比较简单，特殊的地方是有个“掩码”操作，客户端发数据必须掩码，服务器则不用；
5. WebSocket 利用 HTTP 协议实现连接握手，发送 GET 请求要求“协议升级”，握手过程中有个非常简单的认证机制，目的是防止误连接。

课下作业

1. WebSocket 与 HTTP/2 有很多相似点，比如都可以从 HTTP/1 升级，都采用二进制帧结构，你能比较一下这两个协议吗？
2. 试着自己解释一下 WebSocket 里的”Web“和”Socket“的含义。
3. 结合自己的实际工作，你觉得 WebSocket 适合用在哪些场景里？

欢迎你把自己的学习体会写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。

课外小贴士

01 WebSocket 标准诞生于 2011 年，比 HTTP/2 要大上四岁。

02 最早 WebSocket 只能从 HTTP/1.1 升级，因为 HTTP/2 取消了“Connection”头字段和协议升级机制，不能跑在 HTTP/2 上，所以就有草案提议扩展 HTTP/2 支持 WebSocket，后来形成了 RFC8441。

03 虽然 WebSocket 完全借用了 HTTP 的 URI 形式，但也有一点小小的不兼容：不支持 URI 后面的“#”片段标识，“#”必须被编码为“%23”。

04 WebSocket 强制要求客户端发送数据必须使用掩码，这是为了提供最基本的安全防护，让每次发送的消息都是随机、不可预测的，抵御“缓存中毒”攻击。但如果运行在 SSL/TLS 上，采用加密通信，那么掩码就没有必要了。

05 WebSocket 协议里的 PING、PONG 帧对于保持长连接很重要，可以让链路上总有数据在传输，防止被服务器、路由、网关认为是“无效连接”而意外关闭。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　37 | CDN：加速我们的网络服务

下一篇　39 | HTTP性能优化面面观（上）

精选留言 (7)

写留言

Leon
2019-08-23

工作场景遇到过用户订阅股票的股价，股价波动时实时推送给海量订阅的用户，面试场景

被问到两次，一 千万粉丝的明星发布动态如何推送给粉丝 二 海量用户的主播直播如何推送弹幕 当时回答消息队列，其实web socket才是比较好的方案

展开

作者回复: WebSocket适合实时通信交互的场景，和消息队列其实是两个领域，不冲突，可以互相结合使用。

3 3

许童童
2019-08-23

思考题：
1.WebSocket 和 HTTP/2 都是用来弥补HTTP协议的一些缺陷和不足，WebSocket 主要解决双向通信、全双工问题，HTTP/2 主要解决传输效率的问题，两者在二进制帧的格式上也不太一样，HTTP/2 有多路复用、优先级和流的概念。
...

展开

作者回复: great。

2

Cris
2019-08-23

老师，我想问下，uri里的端口号，有什么用？为什么它是和协议对应的（http默认80，https默认443），却又写在域名的后面？

展开

作者回复: 可以参考一下第6讲，端口号是跟tcp协议相关的概念。

因为域名实际上是ip地址的等价替换，所以端口号就可以跟在域名后面。

1

徐海浪
2019-08-23

1. WebSocket 与 HTTP/2 有很多相似点，比如都可以从 HTTP/1 升级，都采用二进制帧结构，你能比较一下这两个协议吗？
差别：HTTP/2是请求与响应的模式，而WebSocket是双向的，服务器也可以主动向客户

端发起请求。
2. 试着自己解释一下 WebSocket 里的”Web“和”Socket“的含义。...
展开

作者回复:
1.在WebSocket里没有请求响应的概念，收发的都是数据帧，通信的双方可以自己解释帧的含义。

2.应该是基于web，也就是http协议。

3.对。

1 1

-W.LI-
2019-08-23

老师好!websocket单机服务器能支持多少链接啊?之前没用过websocket。看帖子好像是通过key-value形式存储所有链接。需要用得时候通过key拿到链接往外写数据。希望老师科普下web socket的简单应用和实现，性能分析。
需要服务器主动推的感觉都可以用websocket做。
聊天工具:用户A，用户B，...
展开

作者回复:
WebSocket其实就是给tcp加了一层简单的包装，所以它的并发能力取决于服务器，并不是kv的形式，你应该把它理解成运行在http上的tcp，用tcp的思路去考虑它。

1

安排
2019-08-24

是js的实时web应用会调用websocket api吗？没接触过js，不太懂这套运行流程。

作者回复: WebSocket API是HTML5的一部分，所以JS就可以使用WebSocket实现实时通信。

安排
2019-08-24

虽然大多数情况下我们会在浏览器里调用 API 来使用 WebSocket。这句话不太理解，websocket不是只有浏览器自己会调用吗？用户还能在浏览器之上调用websocket api，可以举个例子吗？

作者回复: WebSocket是一个通用的协议，只是大多数情况下是在http、浏览器里使用。

完全可以自己写一个应用程序，使用WebSocket来通信。