

个人资料



stoneson

访问：307660次

积分：5353

等级：BLOG > 5

排名：第3764名

原创：169篇 转载：78篇

译文：42篇 评论：37条

文章搜索

文章分类

Android (112)

C#.Net (41)

FLEX (28)

ICE (17)

J2ME (7)

Javascript (11)

linux (1)

移动开发 (36)

算法 (10)

系统架构 (25)

PHP (10)

iPhone (28)

net (5)

websocket (6)

html5 (7)

iOS (5)

mssql (4)

接口 (2)

mysql (3)

Silverlight (3)

文章存档

2016年04月 (1)

微信小程序实战项目——点餐系统

程序员11月书讯，评论得书啦

Get IT技能知识库，50个领域一键直达

websocket规范 RFC6455 中文版

标签：websocket 服务器 终端 http服务器 扩展 tcp

2012-10-12 11:188289人阅读评论(2)收藏举报

分类：

websocket (5)

翻译自：<http://tools.ietf.org/rfc/rfc6455.txt>

InternetEngineering Task Force (IETF)

I. Fette

Request forComments: 6455

Google,Inc.

Category: StandardsTrack

A.Melnikov

ISSN: 2070-1721Isode Ltd.

December2011

WebSocket协议

摘要

WebSocket协议使在控制环境下运行不受信任代码的客户端和能够选择与那些代码通信的远程主机之间能够双向通信。用于这个的安全模型是以origin为基础的安全模型，一般被浏览器使用。协议包含打开握手，其次是基本消息框架，在TCP之上。这项技术的目的是为基于浏览器的、需要与服务器双向通信的应用程序提供一种不依赖于打开多个HTTP连接的机制（例如，使用XMLHttpRequest或<iframe> 和长轮询）。

本备忘录的状态

这是一个Internet标准跟踪文件。

这个文档是因特网工程师任务组（IETF）的一个产品。它代表了IETF社区的共识。它已接受公众审查，因特网工程指导组（IESG）证明可出版。关于互联网标准的进一步信息在RFC5741的第2章节。

关于本文档当前状态的信息、勘误表和如何提供反馈，可以在<http://www.rfc-editor.org/info/rfc6455>找到。

版权声明

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. 介绍

|   |  |
|---|--|
| <div>2014年12月 (1)</div> <div>2014年10月 (4)</div> <div>2014年06月 (1)</div> <div>2014年05月 (9)</div> <div>展开</div>   | <div>1.1. 背景</div> <div>这部分是不规范的。</div> <div>历史上，创建需要在客户端和服务端间双向通信的网络应用程序（如即时消息和游戏程序）要求滥用HTTP来轮询服务器来获得更新，通过不同HTTP请求来发送上行通知。</div> <div>这导致各种问题：<ul style="list-style-type: none"><li>服务器被迫为每个客户端使用一些不同的底层TCP连接：一个用来向客户端发送消息，为每个到来的消息使用一个新的。</li><li>通信（wire）协议具有很高的开销，因为每个客户端到服务器的消息有HTTP头。</li><li>客户端侧的脚本被迫维护输出连接到输入连接的映射来追踪响应。</li></ul></div> <div>一个简单的解决方法是为双向传输使用单一的TCP连接。这是WebSocket协议提供的。结合WebSocket API（WSAPI），它为web页面到远程服务器的双向通信提供了HTTP轮询的替代方案。</div> <div>同样的技术也可用于各种web应用程序：游戏，股票行情，多用户协同编辑的应用程序，用户界面实时展示服务器侧服务等。</div> <div>WebSocket协议设计用来取代使用HTTP作为传输层的双向通信技术，并从现有的基础设施（代理、过滤、认证）受益。这些技术作为效率与可靠性的平衡而实现，因为HTTP最初并不是用于双向通信的（见RFC6202有多更讨论）。WebSocket尝试解决在现有HTTP基础设施的环境下现有HTTP双向通信技术的目标：像这样，它设计来工作于HTTP 80、443端口上，并支持HTTP代理和中间设施，即使这意味着增加现有环境的一些复杂性。然后，设计并没有将WebSocket局限于HTTP，未来的实现可以在特定的端口上使用更简单的握手，而不需要重新发明整个协议。最后一点是重要的，因为交互式消息的传输模式并不紧密符合标准的HTTP传输，会在一些部件上引起异常的负载。</div> <div>1.2. 协议概览</div> <div>这部分是不规范的。</div> <div>WebSocket 协议有两部分：握手和数据传输。</div> <div>来自客户端的握手开起来像下面：<pre>GET /chat HTTP/1.1 Host: server.example.com Upgrade: websocket Connection: Upgrade Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== Origin: http://example.com Sec-WebSocket-Protocol: chat, superchat Sec-WebSocket-Version: 13</pre></div> <div>来自服务器的握手开起来像下面：<pre>HTTP/1.1 101 Switching Protocols Upgrade: websocket Connection: Upgrade Sec-WebSocket-Accept: s3pLMbITxg9KYgzZtRbK+XOo= Sec-WebSocket-Protocol: chat</pre></div> <div>来自客户端的引导行遵从Request-Line格式，来自服务器的引导行遵从Status-Line格式。Request-Line和Status-Line 在[RFC2616]定义。</div> <div>在两种情况下，引导行后面跟着一组未排序的头域。这些头域的意义在本文档的第4章指定。额外的头域也可能出现，如cookie [RFC6265]。头的格式和解析在[RFC2616]定义。</div> |
| <div>阅读排行</div> <div>websocket规范 RFC645 (8278)</div> <div>WebSocket 规范 + WebS (8175)</div> <div>Android应用程序的国际 (7345)</div> <div>Xcode 4.1/4.2/4.3 免证书 (7107)</div> <div>memcached（windows） (7014)</div> <div>互联网骨干网 (6470)</div> <div>保持iOS设备屏幕常亮的 (4529)</div> <div>C#控制IE浏览器自动点击 (3444)</div> <div>[资料整理]四种代理服务 (3140)</div> <div>使用Directory.GetFiles, (2696)</div>   |  |
| <div>评论排行</div> <div>jasperreports+Flex报表 (7)</div> <div>Xcode 4.1/4.2/4.3 免证书 (4)</div> <div>as3与.net进行socket通信 (3)</div> <div>Flex中多线程的实现 (3)</div> <div>获取iPhone屏幕的大小 (2)</div> <div>flex DateField 实现日期 (2)</div> <div>websocket规范 RFC645 (2)</div> <div>Android应用程序的国际 (1)</div> <div>Android上基于JSON的数 (1)</div> <div>android通过webservice (1)</div>  |  |
| <div>推荐文章</div> <div>* 程序员10月书讯，评论得书</div> <div>* Android中Xposed框架篇---修改系统位置信息实现自身隐藏功能</div> <div>* Chromium插件（Plugin）模块（Module）加载过程分析</div> <div>* Android TV开发总结--构建一个TV app的直播节目实例</div> <div>* 架构设计：系统存储--MySQL简单主从方案及暴露的问题</div>   |  |
| <div>最新评论</div> <div>websocket规范 RFC6455 中文版 qifali4184: public</div> <div>websocket规范 RFC6455 中文版 jclplusAres: 赞</div> <div>C#压力测试（CC/DDOS攻击） newpowersky: 这个作用如何？？？</div> <div>C#实现DDOS攻击代码介绍 u网IE陪我去肉体: 还行吧 就那样</div> <div>flex DateField 实现日期和年份可 wangqiang256: 年与月重叠了</div> <div>flex DateField 实现日期和年份可 wangqiang256: 年与月重叠，出现横线怎么处理的？</div> <div>memcached（windows）安装与专心走路: 这个在windows上挺好用</div> <div>jasperreports+Flex报表方案新思 fx_blog: 不实用！</div> <div>[资料整理]四种代理服务器的原理</div> |  |

qq271351631: 能看懂一点意思, 我在研究一下吧, 实在不行, 只能QQ联系了!!

Xcode 4.1/4.2/4.3 免证书(IDP)开数据搬运工: xcode 4.6测试不成功。。。

Javascript

格式化 Javascript

JavaScript 压缩

CSS 代码格式化工具

在线工具 - 脚本之家

五笔输入法编码在线查询系统

资源

免费代理服务器列表,最新代理列表

SOCKS

在线PS

二维工坊

w3school 在线教程

  
web前端开工

  
web前端开发

  
过滤器

  
小米公司

一旦客户端和服务端都发送了他们的握手，如果握手成功，传输数据部分开始。这是一个双向传输通道，每个端都能独立、随意发送数据。

在成功握手后，客户端和服务端来回传输数据是以消息message为概念单位的。在传输介质上（on the wire），一个消息由一个或多个帧frame组成。WebSocket消息不需要对应到特定网络层的帧，因为分帧后的消息可能被中间设施合并或拆分。

一帧都有一个关联的类型。属于同一个消息的帧拥有相同的数据类型，广义地说，有文本数据（解释为UTF-8[RFC3629]文本）、二进制数据（它的解释留给了应用程序）和控制帧（不打算携带应用数据，携带的是协议层的信号，如连接关闭信号）类型。这个版本的协议定义了6种帧类型，并保留了10种为以后使用。

### 1.3. 打开握手

这部分是不规范的。

打开握手为了兼容基于HTTP的服务器端软件和中间设施，使同一个端口能够接受HTTP客户端和WebSocket客户端，为了这个目的，WebSocket客户端的握手是HTTP请求的升级。

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

为了兼容[RFC2616]，客户端握手里的头域可能以任意的顺序发送，因此不同头域接收到的顺序是不重要的。

GET方法[RFC2616]的Request-URI用于识别WebSocket连接的终端，允许一个IP地址服务多个域domain，和允许单个服务器提供多个WebSocket终端。

客户端在握手的Host头域里包含主机名，这样，客户端和服务端能够验证他们同意使用哪个主机。额外的头域用于选择WebSocket协议的选项。此版本中典型的可用选项有子协议选择器Sec-WebSocket-Protocol，Sec-WebSocket-Protocol列出客户端支持的扩展，Origin头域等等。Sec-WebSocket-Protocol请求头域可用来表明客户端可接受的子协议（WebSocket协议之上的应用程序层协议）。服务器选择一个或零个可接受的协议，输出到它的握手，来指明它选择了那个协议。

Sec-WebSocket-Protocol: chat

Origin 头域（RFC6454）用于保护WebSocket服务器不被未授权的运行在浏览器的脚本跨源使用WebSocket API。如果服务器不想接受来自这个源的连接，它可以拒绝连接，并发送一个合适的HTTP错误码。这个头域有浏览器客户端发送；对于非浏览器客户端，这个头域可能发送，如果它在客户端上下文环境中有意义。

最后，服务器得向客户端证明它接收到了客户端的WebSocket握手，为使服务器不接受非WebSocket连接。这防止攻击者通过XMLHttpRequest发送或表单提交精心构造的包来欺骗WebSocket服务器。

为了证明握手被接收，服务器把两块信息合并来形成响应。第一块信息来自客户端握手头域Sec-WebSocket-Key，如

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

对于这个头域，服务器取头的值（由于出现在头域，例如，base64编码[RFC4648]后的版本，消除任何前面后面的空白符），以字符串的形式拼接全局唯一的（GUID，[RFC4122]）标识：258EAF5E-E914-47DA-95CA-C5AB0DC85B11，此值不大可能被不明白WebSocket协议的网络终端使用。然后进行SHA-1 hash（160位）编码，再进行base64编码，将结果作为服务器的握手返回。

具体如下：

请求头：Sec-WebSocket-Key:dGhIIHNhbXBsZSBub25jZQ==

取值，字符串拼接后得到："dGhIIHNhbXBsZSBub25jZQ==258EAF5-E914-47DA-95CA-C5AB0DC85B11";

SHA-1后得到： 0xb3 0x7a 0x4f 0x2c 0xc0 0x62 0x4f 0x16 0x90 0xf6 0x46 0x06 0xcf 0x38 0x59 0x45  
0xb20xbe 0xc4 0xea

Base64后得到： s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

最后的结果值作为响应头Sec-WebSocket-Accept 的值。

来自服务器的握手比客户端的简单很多。首先是HTTP 状态行，状态码是101：

HTTP/1.1 101 Switching Protocols

任何非101的状态码表示WebSocket握手还没有完成，HTTP语义仍然生效。

Connection和Upgrade头域完成HTTP升级。Sec-WebSocket-Accept头表明服务器是否愿意接受连接。如果有，值必须是前面提到的算法得到的值，否则不能解释为服务器接受连接。

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept:s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

这些字段由WebSocket客户端为脚本页面检测，如果Sec-WebSocket-Accept的值不符合预期的值，如果头缺失，或HTTP状态码不是101，连接不会建立，WebSocket帧不会发送。

还可以包含一些可选字段。在协议的这个版本里，主要的可选字段是Sec-WebSocket-Protocol，表示服务器选择的子协议。WebSocket客户端验证服务器选择了一个客户端握手中指定的值。支持多个子协议的服务器必须确保它选择了一个基于客户端握手并在它自己的握手里指定了它。

服务器也可以设置cookie有关的可选头。

#### 1.4. 关闭握手

这部分是不规范的。

关闭握手远比打开握手简单。

任何一端都可以发送带有指定控制序号的数据的帧来开始关闭握手（详细在5.5.1节）。当接收到这样的帧时，如果还没有发送，另一端发送一个关闭帧（B）作为响应。当接收到那个（B）帧时，第一个端关闭连接，因为知道没有更多的数据需要传输。

发送表明关闭连接的控制帧后，端不应该再发数据；接收到表示应该关闭连接的控制帧后，端丢弃后面接收的所有数据。

两端都可以安全地同时开始关闭握手。

这个关闭握手想补充TCP的关闭握手（FIN/ACK），依据是，TCP关闭握手不总是端到端可靠的，特别是出现拦截代理和其他的中间设施。

通过发送一个关闭帧并等待返回关闭帧响应，可以避免一些数据丢失的特殊情况。例如，在一些平台上，如果一个套接字关闭时有数据在接收队列，RST包发送后，将导致recv()失败，因为接收到RST，即使仍有数据等待读取。

#### 1.5. 设计哲学

这部分是不规范的。

**WebSocket**协议的设计原则就是最小化框架（唯一的框架就是协议是基于帧的，而不是基于流的，并且支持区分Unicode文本和二进制帧的）。它希望元数据放在**WebSocket**上面的应用程序层。

概念上，**WebSocket**确实只是TCP上面的一层，做下面的工作：

- | 为浏览器添加web 的origin-based的安全模型。
- | 添加定位和协议命名机制来支持在同一个端口上提供多个服务和同一个IP上有多个主机名。
- | 在TCP上实现帧机制，来回到IP包机制，而没有长度限制。
- | 在带内包含额外的关闭握手，是为了能在有代理和其他中间设施的地方工作。

除了这些，**WebSocket**没有再添加任何东西。基本上，它想尽可能暴露原始的TCP给脚本，又给web约束。它还设计以这样的方式，它的服务器能和HTTP服务器共享同一个端口，通过使它的请求是一个合法的HTTP升级请求。在概念上可以使用其他的协议来建立客户端-服务器消息机制，但**WebSocket**的目的是提供一个相当简单的协议，能够与HTTP和已部署的HTTP基础设施共存（如代理），尽可能接近TCP，通过给定安全模型在现有基础设施上安全使用，伴随简化使用和使简单事情保持简单的额外目标。

协议考虑到可扩展，未来的版本很可能引入额外的概念，如multiplexing.

#### 1.6. 安全模型

这部分是不规范的。

**WebSocket**协议使用起源模型（origin model），浏览器用来限制哪些网页能够接触**WebSocket**服务器。当专用的客户端直接使用**WebSocket**协议时，起源模型没用了，因为客户端能够提供任意的origin字符串。

这个协议目的是不和现有协议（如SMTP、HTTP）的服务器建立连接，如果要求，允许HTTP服务器可选地支持此协议。这个通过一个严格的、精心制作的握手和在握手完成前限制数据进入连接来做到的。

类似地，当其他协议的数据发送到**WebSocket**时，使连接失败。这主要是通过要求服务器证明它读取了握手，在只有当握手含有适当的部分时，这些部分只能由**WebSocket**客户端发送。特别是，在写这个规范时，以Sec-开头的头不能被来自浏览器的攻击者使用HTML和JavaScriptAPI如XMLHttpRequest发送。

#### 1.7. 与TCP和HTTP的关系

这部分是不规范的。

**WebSocket**协议是独立的、基于TCP的协议。与HTTP的唯一关系是它的握手可以被HTTP服务器解释为一个升级请求。

默认地，**WebSocket**协议为常规连接使用80端口，为基于传输层安全（TLS）的连接使用443端口。

#### 1.8. 建立连接

这部分是不规范的。

当一个连接连接到被HTTP服务器共享的端口（对于端口80和443是很可能发生的场景），连接将展示给HTTP服务器的将是一个伴有Upgrade的、常规的GET请求。在只有一个IP地址和单个服务器对应所有传输到单个主机的相对简单的设置，这将允许特殊的方法来部署基于系统的**WebSocket**协议。在更复杂的设置下，一组用于**WebSocket**连接的主机从HTTP服务器分离是更容易管理的。在写这篇规范的时候，应当指出，在端口80和443上连接有显著的成功率，

#### 1.9. 使用**WebSocket**协议的子协议

这部分是不规范的。

客户端可以通过在它的握手包含Sec-WebSocket-Protocol头来要求服务器使用指定的子协议。如果指定了，为建立连接，服务器需要包含一个同样的头和一个选择了的子协议在它的响应。

这些子协议的命名要按照11.5节注册。为了避免潜在的冲突，推荐使用带有子协议发起人的域名的ASCII版本号的名字。例如，Excample集团准备创建一个聊天子协议，由web上的一些服务器来实现，它可以命名为“chat.example.com”。如果Example组织命名他们的竞争子协议为“chat.example.org”，那么两种子协议都能背服务器同时实现，由服务器根据客户端发送的数据动态选择使用子协议。

子协议的版本可以是向后不兼容的，通过改变子协议的名字，如从“bookings.example.NET”到“v2.bookings.examples.Net”。这些子协议可以认为完全独立于WebSocket客户端的。向后兼容的版本可通过复用同样的协议字符串来实现，但小心设计协议来支持扩展。

## 2. 一致性要求

All diagrams, examples, and notes in this specification are nonnormative, as are all sections explicitly marked non-normative. Everything else in this specification is normative.

The key words "MUST", "MUSTNOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("MUST", "SHOULD", "MAY", etc.) used in introducing the algorithm.

### 2.1. Terminology and Other Conventions

\_ASCII\_ shall mean the character-encoding scheme defined in [ANSI.X3-4.1986].

This document makes reference to UTF-8 values and uses UTF-8 notational formats as defined in STD 63 [RFC3629].

Key terms such as named algorithms or definitions are indicated like \_this\_.

Names of header fields or variables are indicated like [this].

Variable values are indicated like /this/.

This document references the procedure to \_Fail the WebSocket Connection\_. This procedure is defined in Section 7.1.7.

\_Converting a string to ASCII lowercase\_ means replacing all characters in the range U+0041 to U+005A (i.e., LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z) with the corresponding characters in the range U+0061 to U+007A (i.e., LATIN SMALL LETTER A to LATIN SMALL LETTER Z).

Comparing two strings in an \_ASCII case-insensitive\_ manner means comparing them exactly, code point for code point, except that the characters in the range U+0041 to U+005A (i.e., LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z) and the corresponding characters in the range U+0061 to U+007A (i.e., LATIN SMALL LETTER A to LATIN SMALL LETTER Z) are considered to also match.

The term "URI" is used in this document as defined in [RFC3986].

When an implementation is required to \_send\_ data as part of the WebSocket Protocol, the implementation MAY delay the actual transmission arbitrarily, e.g., buffering data so as to send fewer IP packets.

Note that this document uses both [RFC5234] and [RFC2616] variants of ABNF in different sections.

### 3. WebSocket URI

本规范定义了两种URI方案，使用在RFC5234定义的ABNF语法、术语和在RFC3986定义的URI规范的ABNF成果。

```
ws-URI = "ws:" "/" host [ ":" port ] path [ "?" query ]
wss-URI = "wss:" "/" host [ ":" port ] path [ "?" query ]
host = <host, defined in [RFC3986], Section 3.2.2>
port = <port, defined in [RFC3986], Section 3.2.3>
path = <path-abempty, defined in [RFC3986], Section 3.3>
query = <query, defined in [RFC3986], Section 3.4>
```

port部分是可选的；ws默认的端口是80，wss默认是443。

如果scheme部分匹配大小写不敏感的“wss”，那么URI就被称作是“安全的”。

“resource-name”可以由下面部分的拼接组成：

- | "/" 如果path部分是空的
- | path部分
- | ? 如果query部分不为空
- | 空的query部分

片段（fragment）标识符在WebSocket URI环境是没意义的，不允许在这些URI中使用。

与任何的URI方案一样，当字符“#”不表示片段的开始时，必须转义为%23。

### 4. 打开握手

#### 4.1. 客户端要求

为建立WebSocket连接，客户端打开一个（TCP）连接并发送一个在这个章节里定义的握手。一个连接最初定义为CONNECTING状态。客户端需要提供WebSocket URI的部件：host、port、resource name、和secure标志，这些都是第3章里讨论的WebSocket URI的组件，伴随使用一些协议和扩展。另外，如果客户端是个web浏览器，它提供origin。

客户端运行在受限环境，如连接到特定关卡的移动手持设备上的浏览器，可能把连接的管理卸载给另一个网络代理。在这种情况下，用于本规范目的的客户端包括手持设备软件和任意的这类代理。

当客户端准备用建立WebSocket连接，给定的一组（host, port, resource name, secure标记），连同使用一序列协议和扩展，和在浏览器情况下有个origin，它必须打开一个（TCP）连接，发送一个打开握手，从响应里读取服务器的握手。（TCP）连接应该如何打开的准确要求，在打开握手应该发送什么，服务器响应应该如何解释，将在下面的章节里。在下面的文本里，我们将使用使用第3章里的名字，如“/host/”和“/secure/ flag”。

1、传递给这个算法（/host/, /port/, /resource name., and /secure/ flag）的WebSocket URI组件必须是合法的，依照第3章说明的WebSocket URI规范。如果有任何的组件是非法的，客户端必须使WebSocket 连接失败，并中止这些步骤。

2、如果客户端已经有WebSocket 连接到用 /host/ 和 /port/ 对 鉴定的远程主机，即使远程主机有另外的名字，客户端必须等待，直到那个连接已经建立或失败。必须保证不能有超过1个的连接处于CONNECTING状态。如果有多个到同一个IP地址的连接同时尝试，客户端必须串行化他们，使在没有超过1个的连接在同时执行下面的步骤。

如果客户端不能确定远程主机的IP地址（例如，所有的通信都是通过一个代理服务器来完成的，代理服务器自己执行DNS查询），为了此步骤的目的，那么客户端必须假设每个主机名指向不同的远程主机，而且，客户端应该限制同时pending的连接数量到一个合理的低的数量（例如，客户端可能允许同时pending的连接到a.example.com和 b.example.com，但如果有30个连接同时要求连接到单个主机，可能是不允许的）。例如，



在一个web浏览器环境里，客户端需要考虑用户打开的网页选项卡的数量，来设置同时pending连接的限制数量。

注意：这使脚本仅仅通过打开大量到远程主机的WebSocket的连接来实施DNS估计更难了。当服务器受到攻击时，能通过在关闭之前挂起连接来减少负载，因为那将减少客户端重连的速率。

注意：客户端到单个远程主机的已建立WebSocket连接的数量是没有限制的。当承受高的负载时，服务器可以拒绝接收来自自己已经有过多数量连接的hosts/IP地址的连接，或断开耗资源的连接。

3、使用代理：如果客户端配置为使用代理，当使用WebSocket协议连接到/host/和/port/的主机，客户端应当连接到那个代理，告诉它打开一个TCP连接到给定的/host/和/port/的主机。

举例：例如，如果客户端对所有传输使用HTTP代理，如果它尝试连接到example.com服务器的端口80，它可能发送下面的行到代理服务器：

```
CONNECT example.com:80 HTTP/1.1
```

```
Host: example.com
```

如果有密码，连接看起来是这样的：

```
CONNECT example.com:80 HTTP/1.1
```

```
Host: example.com
```

```
Proxy-authorization: BasicZWRuYW1vZGU6bm9jYXBicyE=
```

如果客户端没有配置使用代理，一个直接TCP连接应该打开到/host/ 和/port/指定的主机。

注意：Implementations that do not expose explicit UI for selecting a proxy for WebSocket connections separate from other proxies are encouraged to use a SOCKS5 [RFC1928] proxy for WebSocket connections, if available, or failing that, to prefer the proxy configured for HTTPS connections over the proxy configured for HTTP connections.

For the purpose of proxy autoconfiguration scripts, the URI to pass the function MUST be constructed from /host/, /port/, /resource name/, and the /secure/ flag using the definition of a WebSocket URI as given in Section 3.

注意：The WebSocket Protocol can be identified in proxy autoconfiguration scripts from the scheme ("ws" for unencrypted connections and "wss" for encrypted connections).

4、如果（TCP）连接不能打开，因为直接连接失败还是使用的任何代理返回错误，客户端必须使用WebSocket连接失败，并中止连接尝试。

5、如果/secure/是true，客户端必须在（TCP）连接上执行TLS握手，在打开连接后，发送握手数据

【RFC2818】之前。如果这个失败了，客户端必须使WebSocket连接失败并中止连接。否则，所有在这条通道上的后续的通信必须使用加密通道【RFC5246】。

Clients MUST use the Server Name Indication extension in the TLS handshake [RFC6066].

一旦到服务器的（TCP）连接建立（包括通过代理或TLS加密的通道），客户端必须发送一个打开握手到服务器。握手包含一个HTTP Upgrade请求，连同一些必须的和可选的头域。打开握手的要求如下：

1. 握手必须是一个合法的【RFC2616】规定的HTTP请求。

2. 请求的方法必须是GET，HTTP版本必须至少是1.1。

例如，如果WebSocket URI是"ws://example.com/chat"，发送的第一行应该是"GET /chat HTTP/1.1"。

3. 请求的Request-URI部分必须符合第3章定义的/resource name/，或者是绝对的http/https URI，当解析后，有个/resource name/， /host/, /port/，符合对应的ws/wss URI。

4. 请求必须含有Host头域，它的值包含/host/加上可选的":后跟/port/（当没使用默认的端口时）。

5. 请求必须包含Upgrade头域，其值必须含有"websocket"关键字。

6. 请求必须含有Connection头域，其值必须含有"Upgrade"记号。

7. 请求必须包含名为Sec-WebSocket-Key的头域，其值必须是nonce组成的随机选择的16字节的被base64



编码后的值。必须为每个连接随机选择nonce。

注意：例如，如果随机选取的一序列字节值是0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a0x0b 0x0c 0x0d 0x0e0x0f 0x10，头域的值应该是"AQIDBAUGBwgJCgsMDQ4PEC=="。

8. 如果请求来自浏览器客户端，必须包含名为Origin的头域。如果连接来自非浏览器客户端，请求可能包括这个头域，如果那个客户端的语义符合此处浏览器客户端描述的用例。这个头域的值是建立连接的代码运行环境的origin的ASCII序列。在【RFC6454】有描述此头域值是如何构建的。

例如，下载子www.example.com的代码尝试建立连接到ww2.example.com，此头域的值应该是"http://www.example.com"。

9. 请求必须包含名为Sec-WebSocket-Version的头域，其值必须是13。

注意：Although draft versions of this document (-09, -10, -11, and -12) were posted (they were mostly comprised of editorial changes and clarifications and not changes to the wire protocol), values 9, 10, 11, and 12 were not used as valid values for Sec-WebSocket-Version. These values were reserved in the IANA registry but were not and will not be used.

10. 请求可能包含名为Sec-WebSocket-Protocol的头域。如果有，此值指示客户端希望使用的一个或多个逗号分隔的子协议，按优选顺序。组成此值的元素必须是非空字符串，字符范围在U+0021到U+007E，不包括在【RFC2616】定义的分隔符，而且必须是唯一的字符串。此头域值的ABNF是1#token，构建和规则在【RFC2616】定义。

11. 请求可能包含名为Sec-WebSocket-Extensions的头域。如果有，其值指示了客户端希望使用的协议级别的扩展。此头域的解释和格式在9.1节描述。

12. 请求可能包含任意的其他头域。例如cookies和/或认证相关的头域，如Authorization头域。

一旦客户端打开握手发送出去，在发送任何数据之前，客户端必须等待服务器的响应。客户端必须按如下步骤验证响应：

1、如果从服务器接收到的状态码不是101，按HTTP【RFC2616】程序处理响应。在特殊情况下，如果客户端接收到401状态码，可能执行认证；服务器可能用3xx状态码重定向客户端（但不要求客户端遵循他们），等等。否则按下面处理。

2、如果响应缺失Upgrade头域或Upgrade头域的值没有包含大小写不敏感的ASCII值"websocket"，客户端必须使WebSocket连接失败。

3、如果响应缺失Connection头域或其值不包含大小写不敏感的ASCII值"Upgrade"，客户端必须使WebSocket连接失败。

4、如果响应缺失Sec-WebSocket-Accept头域或其值不包含|Sec-WebSocket-Key|（作为字符串，非base64解码的）+ "258EAF55-E914-47DA-95CA-C5AB0DC85B11"的base64编码 SHA-1值，客户端必须使WebSocket连接失败。

5、如果响应包含Sec-WebSocket-Extensions头域，且其值指示使用的扩展不出现在客户端发送的握手（服务器指示的扩展不是客户端要求的），客户端必须使WebSocket连接失败。（解析此头域来决定哪个扩展是要求的在第9.1节描述。）

6、如果响应包含Sec-WebSocket-Protocol头域，且这个头域指示使用的子协议不包含在客户端的握手（服务器指示的子协议不是客户端要求的），客户端必须使WebSocket连接失败。

如果服务器响应不遵从4.2.2节和本节定义服务器握手的要求，客户端必须使WebSocket连接失败。

请注意，依据RFC2616，所有HTTP请求和HTTP响应的头域都是大小写不敏感的。

如果服务器的响应符合上面的验证要求，说明WebSocket连接建立，WebSocket连接处于OPEN状态。The `_Extensions In Use_` is defined to be a (possibly empty) string, the value of which is equal to the value of the `[Sec-WebSocket-Extensions]` header field supplied by the server's handshake or the null value if that header field was not present in the server's handshake. The `_Subprotocol In Use_` is defined to be the value of the `[Sec-WebSocket-Protocol]` header field in the server's handshake or the null value if that header field was not present in the server's handshake. Additionally, if any header fields in the server's handshake indicate that cookies should be set (as defined by [RFC6265]), these cookies are referred to as `_Cookies Set During the Server's Opening Handshake_`.

#### 4.2. 服务器端要求

服务器可能把连接的管理卸载给其他的网络代理，如负载均衡器和反向代理。在这种情况下，用于此规范目的的服务器被认为包含了服务器端基础设施的所有部分，从第一个终结TCP连接的设备一路直到处理请求并发送响应的服务器。

例如：一个数据中心可能有一台服务器用合适的握手来响应WebSocket请求，然后把连接传递给另一台服务器来实际处理数据帧。在本规范的目的中，“服务器”是两台电脑的结合。

##### 4.2.1. 读取客户端打开握手

当一个客户端开始WebSocket连接，它发送自己角色的打开握手。服务器必须解析至少部分的握手来获取需要的信息，来生成服务器角色的握手。

客户端握手包含下面的部分。如果服务器在读取握手时，发现客户端发送的握手不符合下面的描述（注意，安装RFC2616，头域的顺序是不重要的），包括但不限于任何违反ABNF语法指定的握手组件，服务器必须停止处理客户端握手，并返回恰当的HTTP响应，如 `400 badrequest`。验证步骤：

1、一个HTTP/1.1或更高的GET请求，包括应该解释为 `/resource name/` 的 "Request-URI"（或者是一个绝对的HTTP/HTTPS URI包含 `/resource name/`）。

2、一个Host头域包含服务器的职权（`authority`）。

3、一个Upgrade头域包含一个ASCII大小写不敏感的值 `"websocket"`。

4、一个Connection 头域包含一个ASCII大小写不敏感的令牌 `"Upgrade"`。

5、一个Sec-WebSocket-Key 头域有一个base64编码的值，解码后是一个16字节长度。

6、一个Sec-WebSocket-Version 头域且值为13。

7、可选的一个Origin头域。此头域由所有的浏览器客户端发送。缺少这个头域的连接不应该解释为来自浏览器客户端。

8、可选的一个Sec-WebSocket-Protocol 头域，有一序列的值，表示客户端希望用来交流的协议，按优选排序。

9、可选的一个Sec-WebSocket-Extensions头域。有一序列的值，表示客户端希望用来交流的扩展，按优选排序。此头域的解释在9.1节讨论。

10、可选的其他头域，用于发送cookie或请求认证到服务器。按RFC2616，未知的头域将被忽略。

##### 4.2.2. 发送服务器打开握手

当客户端建立一个到服务器的WebSocket连接，服务器必须完成下面的步骤来接受连接，并发送服务器打开握手。

1、如果连接在HTTPS（HTTP-over-TLS）端口上打开，在连接上执行TLS握手。如果失败了，关闭连接；否则，连接的所有后续的通信（包括服务器握手）必须在加密的通道上进行。

2、服务器可以执行额外的客户端验证，例如，通过返回401状态码和相应的|WWW-Authenticate|头域。

3、服务器可能用3xx状态码来重定向客户端。注意，这个步骤可以和上面描述的认证步骤一起、之前或之后发生。

4、建立下面的信息：

/origin/

客户端握手的|Origin|头域指示了建立连接的脚本的起源。起源被序列化为ASCII，并转换为小写。服务器可能使用这个信息来作为决定是否接收连接的部分因素。如果服务器不能验证起源，它将接收来自任何地方的连接。如果服务器不想接收这个连接，它必须返回合适的HTTP错误码（如403Forbidden），并中止WebSocket握手，按本节的描述。更多细节，见第10章。

/key/

客户端握手的|Sec-WebSocket-Key|包含一个base64编码的值，如果解码，将是16字节长度。此值（编码后的）用于创建服务器握手，来指示接收连接。服务器没有必要base64解码此值。

/version/

客户端握手的|Sec-WebSocket-Version|包含WebSocket协议的版本，客户端用这个来尝试通信。如果这个版本不符合服务器能理解的版本，服务器必须中止WebSocket握手，发送一个合适HTTP错误码（如426 Upgrade Required），且有一个|Sec-WebSocket-Version|头域来指示服务器能够理解的版本。

/resource name/

一个用于服务器提供的服务的标识符。如果服务器提供了多个服务，值应该能从resource name推断得到，从给定客户端握手的GET方法的"Request-URI"。如果请求的服务不可得，服务器必须发送合适的HTTP错误码（如404 Not Found），并中止WebSocket握手。

/subprotocol/

或者是空，或者是单个值表示服务器准备使用的子协议。值的选择必须从客户端握手推断得到，具体是从|Sec-WebSocket-Protocol|选择一个值，服务器将用于这个连接的。如果客户端握手不包含这样1个头域或者服务器不同意使用客户端请求的子协议，唯一可接受的值是null。没有这个域等价于null值（意味着如果服务器不打算使用任何推荐的子协议，它必须不发送回一个|Sec-WebSocket-Protocol|头域在它的响应里）。在这个目的上，Empty字符串不同于null值，且不是这个域的合法值。用于此头域值的ABNF是（token），构建和规则在RFC2616定义。

/extensions/

一个（可能空的）列表表示服务器准备使用的协议级扩展。如果服务器支持多个扩展，那么值必须从客户端握手推断得到，具体是从|Sec-WebSocket-Extensions|域选择一个或多个值。此域的缺失等价于null值。在这个目的上，Empty字符串不同于null值。没有列在客户端的扩展必须不被登记。选择和解释这些值的方法在9.1节讨论。

5、如果服务器选择接受连接，必须返回一个合法的HTTP响应，指示下面：

1. 一个状态行，有101响应码，按照RFC2616。这样一个响应看起来是这样：

"HTTP/1.1 101 Switching Protocols"

2. 一个Upgrade 头域，值为"websocket"，按照RFC2616。

3. 一个Connection头域，值为"Upgrade"。

4. 一个Sec-WebSocket-Accept 头域。The value of this header field is constructed by concatenating /key/, defined above instep 4 in Section 4.2.2,with the string "258EAF5-E914-47DA-95CA-C5AB0DC85B11",taking the SHA-1 hash of this concatenated value to obtain a 20-bytevalue and base64-encoding (see Section 4 of [RFC4648]) this 20-byte hash.

The ABNF [RFC2616] of this header field is defined as follows:

```
Sec-WebSocket-Accept= base64-value-non-empty
base64-value-non-empty= (1*base64-data [ base64-padding ]) |
base64-padding
base64-data= 4base64-character
base64-padding= (2base64-character "==") |
(3base64-character"=")
base64-character= ALPHA | DIGIT | "+" | "/"
```

NOTE: As an example, if the value of the|Sec-WebSocket-Key| header field in the client's handshake were "dGhlIHhnbXBsZSBub25jZQ==", the server would append the string "258EAF5-E914-47DA-95CA-C5AB0DC85B11" to form the string "dGhlIHhnbXBsZSBub25jZQ==258EAF5-E914-47DA-95CAC5AB0DC85B11". The server would then take the SHA-1 hash of this string, giving the value 0xb3 0x7a 0x4f 0x2c 0xc0 0x62 0x4f 0x16 0x90 0xf6 0x46 0x06 0xcf 0x38 0x59 0x45 0xb2 0xbe 0xc4 0xea. This value is then base64-encoded, to give the value "s3pPLMBiTxaQ9kYGzzhZRbK+xOo=", which would be returned in the |Sec-WebSocket-Accept| header field.

5. 可选的一个Sec-WebSocket-Protocol 头域，其值/subprotocol/在4.2.2节步骤4定义。

6. 可选的一个Sec-WebSocket-Extensions 头域，其值/extensions/在4.2.2节步骤4定义。如果使用多个extension，可以在罗列单个Sec-WebSocket-Extensions 头域，也可以分割在多个Sec-WebSocket-Extensions 。

这完成了服务器握手。如果服务器完成这些步骤，且没有终止WebSocket握手，服务器认为WebSocket连接已建立，那么WebSocket连接处于OPEN状态。在此时，服务器可能开始发送（和接收）数据。

#### 4.3. Collected ABNF for New HeaderFields Used in Handshake

This section is using ABNF syntax/rules from Section 2.1 of [RFC2616], including the "implied \*LWS rule".

Note that the following ABNF conventions are used in this section.

Some names of the rules correspond to names of the corresponding header fields. Such rules express values of the corresponding header fields, for example, the Sec-WebSocket-Key ABNF rule describes syntax of the|Sec-WebSocket-Key| header field value. ABNF rules with the "-Client" suffix in the name are only used in requests sent by the client to the server; ABNF rules with the "-Server" suffix in the name are only used in responses sent by the server to the client. For example, the ABNF rule Sec-WebSocket-Protocol-Client describes syntax of the|Sec-WebSocket-Protocol| header field value sent by the client to the server.

The following new header fields can be sent during the handshake from the client to the server:

```
Sec-WebSocket-Key= base64-value-non-empty
Sec-WebSocket-Extensions= extension-list
Sec-WebSocket-Protocol-Client= 1#token
Sec-WebSocket-Version-Client= version
base64-value-non-empty= (1*base64-data [ base64-padding ]) |
base64-padding
base64-data= 4base64-character
base64-padding= (2base64-character "==") |
(3base64-character"=")
```

```

base64-character= ALPHA | DIGIT | "+" | "/"
extension-list= 1#extension
extension= extension-token *( ";" extension-param )
extension-token= registered-token
registered-token =token
extension-param= token [ "=" (token | quoted-string) ]
;When using the quoted-string syntax variant, the value
;after quoted-string unescaping MUST conform to the
;'token' ABNF.
NZDIGIT= "1" | "2" | "3" | "4" | "5" | "6" |
"7" | "8" | "9"
version= DIGIT | (NZDIGIT DIGIT) |
("1" DIGIT DIGIT) | ("2" DIGIT DIGIT)
;Limited to 0-255 range, with no leading zeros

```

The following newheader fields can be sent during the handshake from the server to the client:

```

Sec-WebSocket-Extensions= extension-list
Sec-WebSocket-Accept= base64-value-non-empty
Sec-WebSocket-Protocol-Server= token
Sec-WebSocket-Version-Server= 1#version

```

#### 4.4. 支持多版本的WebSocket协议

This section provides some guidance on supporting multiple versions of the WebSocket Protocol in clients and servers.

Using the WebSocketversion advertisement capability (the [Sec-WebSocket-Version]header field), a client can initially request the version of theWebSocket Protocol that it prefers (which doesn't necessarily have to be the latest supported by the client). If the server supports therequested version and the handshake message is otherwise valid,the server will accept that version. If the server doesn't support therequested version, it MUST respond with a [Sec-WebSocket-Version]header field (or multiple [Sec-WebSocket-Version]header fields) containing all versions it is willing to use. Atthis point, if the client supports one of the advertisedversions, it can repeat the WebSocket handshake using a new version value.

The followingexample demonstrates version negotiation described above:

```

GET/chat HTTP/1.1
Host:server.example.com
Upgrade:websocket
Connection:Upgrade
...
Sec-WebSocket-Version:25

```

The response from the server might look as follows:

```

HTTP/1.1400 Bad Request
...
Sec-WebSocket-Version:13, 8, 7

```

Note that the last response from the server might also look like:

HTTP/1.1 400 Bad Request

...

Sec-WebSocket-Version: 13

Sec-WebSocket-Version: 8, 7

The client now repeats the handshake that conforms to version 13:

GET/chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

...

Sec-WebSocket-Version: 13

5. 数据成帧

5.1. 概述

在WebSocket协议，数据是用一序列帧来传输。为了避免使网络中间设施（如拦截代理）混乱和安全原因（更多讨论在10.3节），客户端必须标记所有发往服务器的帧（更多细节在5.3节）。（注意，进行标记是不管WebSocket协议是否使用了TLS。）当服务器接收到一个没有标记的帧时必须关闭连接。在这种情况下，服务器可能发送一个有状态码1002（协议错误）的关闭帧，如7.4.1节定义。服务器必须不标记它发给客户端的任何帧。客户端必须关闭连接，如果它检测到标记了的帧。在这种情况下，它可能使用1002状态码（协议错误）。（这些规则在未来的规范中可能放松。）

基本的成帧协议定义了帧类型有操作码、有效载荷的长度，指定位置的"Extension data"和"Application data"，统称为"Payload data"。保留了一些特殊位和操作码供后期扩展。

在打开握手完成后，终端发送一个关闭帧之前的任何时间里，数据帧可能由客户端或服务器的任何一方发送（见5.5.1节）。

5.2. 基本的帧协议

用于数据传输部分的有线格式用ABNF描述，细节在本节。（注意，不像本文档中的其他章节，本节的ABNF是作用于一组比特位上。每组比特位的长度在注释里指示。当在有线上编码后，最重要的比特的最左边的ABNF）。帧的一个高层概览见下图。当下图与ABNF有冲突时，图是权威的。

**FIN:** 1bit

表示此帧是否是消息的最后帧。第一帧也可能是最后帧。

**RSV1, RSV2, RSV3:** 各1bit

必须是0，除非协商了扩展定义了非0的意义。如果接收到非0，且没有协商扩展定义 此值的意义，接收端必须使WebSocket连接失败。

**Opcode:** 4bit

定义了"Payload data"的解释。如果接收到未知的操作码，接收端必须使WebSocket 连接失败。下面的值是定义了的。

%x0 表示一个后续帧

%x1 表示一个文本帧

%x2 表示一个二进制帧

%x3-7 为以后的非控制帧保留

%x8 表示一个连接关闭

%x9 表示一个ping

%xA 表示一个pong

%xB-F 为以后的控制帧保留

**Mask:** 1bit

定义了"Payload data"是否标记了。如果设为1，必须有标记键出现在masking-key，用 来unmask "payload data"，见5.3节。所有从客户端发往服务器的帧必须把此位设为1。

**Payload length:** 7bit, 7 + 16bit, 7 + 64bit

"Payloaddata"的长度，字节单位。如果值是0-125，则是有效载荷长度。如果是126，接下来的2字节解释为16位无符号整数，作为有效载荷长度。如果127，接下来的8字节解释为64位无符号整数（最高位必须是0），作为有效载荷长度。多字节长度数值以网络字节序表示。注意，在任何情况下，必须用最小数量的字节来编码长度，例如，124字节长的字符串不能编码为序列126, 0, 124。有效载荷长度是"Extension data"的长度加上"Application data"的长度。"Extension data"的长度可能是0，在这种情况下，有效载荷长度是"Applicationdata"的长度。

#### Masking-key: 0或4字节

所有从客户端发往服务器的帧必须用32位值标记，此值在帧里。如果mask位设为1，此字段（32位值）出现，否则缺失。更多的信息在5.3节，客户端到服务器标记。

#### Payload data: (x + y)字节

"Payloaddata" 定义为"extensiondata" 后接"application data"。

#### Extension data: x 字节

"Extensiondata"是0字节，除非协商了扩张。所有扩张必须指定"extensiondata"的长度，或者如何计算长度，如何使用扩展必须在打开握手时进行协商。如果有，"Extension data"包括在有效载荷长度。

#### Application data: y字节

任意"Applicationdata"占据了帧的剩余部分，在"Extensiondata"之后。"Applicationdata"的长度等于有效载荷长度减去"Extensiondata"的长度。

The base framing protocol is formally defined by the following ABNF [RFC5234]. It is important to note that the representation of this data is binary, not ASCII characters. As such, a field with a length of 1 bit that takes values %x0 / %x1 is represented as a single bit whose value is 0 or 1, not a full byte (octet) that stands for the characters "0" or "1" in the ASCII encoding. A field with a length of 4 bits with values between %x0-F again is represented by 4 bits, again NOT by an ASCII character or full byte (octet) with these values. [RFC5234] does not specify a character encoding: "Rules resolve into a string of terminal values, sometimes called characters. In ABNF, a character is merely a non-negative integer. In certain contexts, a specific mapping (encoding) of values into a character set (such as ASCII) will be specified." Here, the specified encoding is a binary encoding where each terminal value is encoded in the specified number of bits, which varies for each field.

```
ws-frame = frame-fin ; 1 bit in length
frame-rsv1 ; 1 bit in length
frame-rsv2 ; 1 bit in length
frame-rsv3 ; 1 bit in length
frame-opcode ; 4 bits in length
frame-masked ; 1 bit in length
frame-payload-length ; either 7, 7+16,
; or 7+64 bits in
; length
[frame-masking-key ] ; 32 bits in length
frame-payload-data ; n*8 bits in
; length, where
; n >= 0
frame-fin = %x0 ; more frames of this message follow
/%x1 ; final frame of this message
; 1 bit in length
```



```
frame-rsv1 = %x0 /%x1
;1 bit in length, MUST be 0 unless
;negotiated otherwise
frame-rsv2 = %x0 /%x1
;1 bit in length, MUST be 0 unless
;negotiated otherwise
frame-rsv3 = %x0 /%x1
;1 bit in length, MUST be 0 unless
;negotiated otherwise
frame-opcode =frame-opcode-non-control /
frame-opcode-control/
frame-opcode-cont
frame-opcode-cont =%x0 ; frame continuation
frame-opcode-non-control=%x1 ; text frame
/%x2 ; binary frame
/%x3-7
;4 bits in length,
;reserved for further non-control frames
frame-opcode-control= %x8 ; connection close
/%x9 ; ping
/%xA ; pong
/%xB-F ; reserved for further control
; frames
; 4 bits in length

frame-masked = %x0
;frame is not masked, no frame-masking-key
/%x1
;frame is masked, frame-masking-key present
;1 bit in length
frame-payload-length= ( %x00-7D )
/( %x7E frame-payload-length-16 )
/( %x7F frame-payload-length-63 )
;7, 7+16, or 7+64 bits in length,
;respectively
frame-payload-length-16= %x0000-FFFF ; 16 bits in length
frame-payload-length-63= %x0000000000000000-7FFFFFFFFFFFFFFF
;64 bits in length
frame-masking-key =4( %x00-FF )
;present only if frame-masked is 1
;32 bits in length
frame-payload-data= (frame-masked-extension-data
frame-masked-application-data)
;when frame-masked is 1
/(frame-unmasked-extension-data
frame-unmasked-application-data)
;when frame-masked is 0
frame-masked-extension-data= *( %x00-FF )
;reserved for future extensibility
;n*8 bits in length, where n >= 0
frame-masked-application-data= *( %x00-FF )
;n*8 bits in length, where n >= 0
frame-unmasked-extension-data= *( %x00-FF )
;reserved for future extensibility
```

```
;n*8 bits in length, where n >= 0
frame-unmasked-application-data= *( %x00-FF )
;n*8bits in length, where n >= 0
```

### 5.3. 客户端到服务器标记

被标记的帧必须设置frame-masked域为1，如5.2节定义。标记键必须作为frame-masking-key完整包含在帧里，作为frame-masking-key，如5.2节定义。它用于标记"Payload data"。

标记键是由客户端随机选择的32位值。准备标记帧时，客户端必须从一组允许的32位值选择一个新的标记键。标记键应该是不可预测的，因此，标记键必须推断自强源的熵（strong source of entropy），给定帧的标记键必须不能简单到服务器或代理可以预测标记键是用于一序列帧的。不可预测的标记键是阻止恶意应用的作者从wire上获取数据的关键。RFC 4086 discusses what entails a suitable source of entropy for security-sensitive applications.

标记不影响"Payload data"的长度。为了把标记数据转换为非标记数据，或反转，下面的算法将被应用。同样的算法将应用，而不管转换的方向，例如同样的步骤将用于标记数据和unmask数据。

转换后的数据八位字节i（transformed-octet-i）是原始八位字节i（original-octet-i）与标记键在i对4取模（j = i % 4）后的位序的八位字节（masking-key-octet-j）的异或(XOR)。

$j = i \text{ MOD } 4$

$\text{transformed-octet-i} = \text{original-octet-i} \text{ XOR } \text{masking-key-octet-j}$

译者给出的Scala语言实现：

```
def mask(key: Array[Byte], data: Array[Byte]){
  for (i <- 0 until data.length) {
    val j = i & 3 // 对4 取模
    data(i) = (data(i) ^ key(j)).toByte
  }
}
```

帧里的有效载荷长度指示frame-payload-length，不包括masking key的长度。它是"Payload data"，如，masking key的后续字节的长度。

### 5.4. 分帧

分帧的首要目的是允许发送开始时不知道长度的消息，不需要缓存消息。如果消息不能分帧，那么终端得缓存整个消息，以便在发送前计算长度。有了分帧，服务器或中间设施可以选择一个合理大小的缓存，当缓存满时，把帧写到网络。

分帧的第二个用例是多路复用，在不值得大消息独占逻辑输出通道的地方，多路复用需要自由地把消息划分更小的帧来更好地共享输出通道。(注意，多路技术扩展不在本文档描述。)

除非另有扩展说明，分帧没有语义上的意义。中间设施可能合并和/或分隔帧，如果客户端和服务端没有协商扩展，或协商了扩展，但中间设施明白所有协商了的扩展，并知道如何合并和/或拆分帧。分帧的实现在没有扩展时，发送者和接收者必须不能依赖于帧的边界。

分帧的规则：

A、一个未分帧的消息包含单个帧，FIN设置为1，opcode非0。

B、一个分帧了的消息包含：

开始于：单个帧，FIN设为0，opcode非0；

后接：0个或多个帧，FIN设为0，opcode设为0；

终结于：单个帧，FIN设为1，opcode设为0。

一个分帧了消息在概念上等价于一个未分帧的大消息，它的有效载荷长度等于所有帧的有效载荷长度的累

加；然而，有扩展时，这可能不成立，因为扩展定义了出现的 **"Extension data"**的解释。例如，**"Extensiondata"**可能只出现在第一帧，并用于后续的所有帧，或者**"Extensiondata"**出现于所有帧，且只应用于特定的那个帧。在缺少**"Extensiondata"**时，下面的示例示范了分帧如何工作。

举例：如一个文本消息作为三个帧发送，第一帧的**opcode**是**0x1**，**FIN**是**0**，第二帧的**opcode**是**0x0**，**FIN**是**0**，第三帧的**opcode**是**0x0**，**FIN**是**1**。

C、控制帧（见5.5节）可能被插入到分帧了消息中。控制帧必须不能被分帧。

D、消息的帧必须以发送者发送的顺序传递给接受者。

E、一个消息的帧必须不能交叉在其他帧的消息中，除非有扩展能够解释交叉。

F、一个终端必须能够处理消息帧中间的控制帧。

G、一个发送者可能对任意大小的非控制消息分帧。

H、客户端和服务端必须支持接收分帧和未分帧的消息。

I、由于控制帧不能分帧，中间设施必须不尝试改变控制帧。

J、中间设施必须不修改消息的帧，如果保留位的值已经被使用，且中间设施不明白这些值的含义。

在扩展已经被协商好，且中间设施不知道已协商扩展的语义的环境，中间设施必须不修改任何消息的帧。类似地，**anintermediary that didn't see theWebSocket handshake (and wasn't notified about its content) that resulted in a WebSocketconnection MUST NOT change thefragmentation of any message of such connection.**

K、作为这些规则的结果，一个消息的所有帧属于同样的类型，由第一个帧的**opcode**指定。由于控制帧不能分帧，消息的所有帧的类型要么是文本、二进制数据或保留的操作码中的一个。

注意：如果控制帧不能插入，例如，**ping**的延迟将会很长，如果是在一个大消息后面。因此要求处理消息帧中间的控制帧。

实现注意：如果没有任何扩展，接收者不需要为了处理而缓存整个帧。例如，如果使用了流API，帧的一部分可以传递给应用程序。然而，这个假设在未来的**WebSocket**扩展中可能不成立。

## 5.5. 控制帧

控制帧由操作码标识，操作码的最高位是**1**。当前为控制帧定义的操作码有**0x8**（关闭）、**0x9**（**Ping**）和**0xA**（**Pong**），操作码**0xB-0xF**是保留的，未定义。

控制帧用来交流**WebSocket**的状态，能够插入到消息的多个帧的中间。

所有的控制帧必须有一个小于等于**125**字节的有效载荷长度，必须不能被分帧。

### 5.5.1. 关闭

关闭帧有个操作码**0x8**。

关闭帧可能包含一个主体（帧的应用数据部分）指明关闭的原因，如终端关闭，终端接收到的帧太大，或终端接收到的帧不符合终端的预期格式。如果有主体，主体的前2个字节必须是2字节的无符号整数（按网络字节序），表示以**/code/**（7.4节定义）的值为状态码。在2字节整数后，主体可能包含一个**UTF-8**编码的字符串表示原因。这些数据不一定要人类可读，但对调试友好，或给打开连接的脚本传递信息。由于不保证数据是人类可读的，客户端必须不显示给用户。

从客户端发送到服务器的关闭帧必须标记，按5.3节。

在发送关闭帧后，应用程序必须不再发送任何数据。

如果终端接收到一个关闭帧，且先前没有发送关闭帧，终端必须发送一个关闭帧作为响应。（当发送一个关闭帧作为响应时，终端典型地以接收到的状态码作为回应。）它应该尽快实施。终端可能延迟发送关闭帧，直到它的当前消息发送完成（例如，如果分帧消息的大部分已发送，终端可能在关闭帧之前发送剩余的帧）。然而，不保证已经发送了关闭帧的终端继续处理数据。

在发送和接收到关闭消息后，终端认为WebSocket连接已关闭，必须关闭底层的TCP连接。服务器必须立即关闭底层的TCP连接；客户端应该等待服务器关闭连接，但可能在发送和接收到关闭消息后关闭，例如，如果它在合理的时间间隔内没有收到TCP关闭。

如果客户端和服务器同时发送关闭消息，两端都已发送和接收到关闭消息，应该认为WebSocket连接已关闭，并关闭底层TCP连接。

#### 5.5.2. Ping

Ping帧包含操作码0x9。

一个Ping帧可能包含应用程序数据。

当接收到Ping帧，终端必须发送一个Pong帧响应，除非它已经接收到一个关闭帧。它应该尽快返回Pong帧作为响应。Pong帧在5.5.3节讨论。

终端可能在连接建立后、关闭前的任意时间内发送Ping帧。

注意：Ping帧可作为keepalive或作为验证远程终端是否可响应的手段。

#### 5.5.3. Pong

Pong帧包含操作码0xA。

5.5.2节的详细要求适用于Ping和Pong帧。

Pong 帧必须包含与被响应Ping帧的应用程序数据完全相同的数据。

如果终端接收到一个Ping 帧，且还没有对之前的Ping帧发送Pong 响应，终端可能选择发送一个Pong 帧给最近处理的Ping帧。

一个Pong 帧可能被主动发送，这作为单向心跳。对主动发送的Pong 帧的响应是不希望的。

### 5.6. 数据帧

数据帧（如非控制帧）由操作码标识，操作码的最高位是0。当前为数据帧定义的操作码有0x1（文本），0x2（二进制）。操作码0x3-0x7 为以后的非控制帧保留，未定义。

数据帧携带 应用程序层和/或扩展层 数据。操作码决定了数据的解释：

文本：

有效载荷数据是UTF-8编码的文本数据。注意，特定的文本帧可能包含部分的UTF-8 序列，然而，整个消息必须包含有效的UTF-8。重新聚合后的非法的UTF-8的处理在8.1 节描述。

二进制：

有效载荷数据是任意的二进制数据，它的解释由应用程序层唯一决定。

## 5.7. 举例

单个未标记的文本消息帧:

0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f(contains "Hello")

单个标记了的文本消息帧:

0x81 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58 (contains "Hello")

分帧了的未标记的文本消息:

0x01 0x03 0x48 0x65 0x6c (contains "Hel")

0x80 0x02 0x6c 0x6f(contains "lo")

未标记的Ping请求和标记了的Ping 响应:

0x89 0x05 0x48 0x65 0x6c 0x6c 0x6f(contains a body of "Hello", but the contents of the body are arbitrary)

0x8a 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58 (contains a body of "Hello", matching the body of the ping)

含有256字节二进制数据消息的单个未标记帧:

0x82 0x7E 0x0100 [256 bytes of binary data]

64KB二进制数据消息在单个未标记帧:

0x82 0x7F 0x00000000000010000 [65536 bytes of binary data]

## 5.8. 扩展

协议旨在允许扩展, 将给基础协议添加功能。连接的终端必须在打开握手中协商使用任意的扩展。这个规范提供了操作码0x3到0x7和0xB到0xF, 帧头的“Extension data”域, 和frame-rsv1, frame-rsv2和frame-rsv3 位供扩展使用。扩展的协商在9.1节讨论。下面是一些期望的扩展使用, 这个清单既不是完整的也不是规范。

A、“extension data”可能在“application data”之前放置。

B、保留位可能按帧的需要分配。

C、保留的操作码可以定义。

D、保留位可以分配给操作码域, 如果需要更多的操作码。

E、保留位或扩展操作码可以定义在“Payload data”外分配额外的位来定义更大的操作码或更多的特定帧位。

## 6. 发送和接收数据

### 6.1. 发送数据

为在WebSocket连接上发送一个由/data/组成的WebSocket消息, 终端必须执行下面的步骤:

1. 终端必须确保WebSocket连接处于OPEN状态。在任何时刻, 如果WebSocket连接的状态改变, 终端必须终止下面的步骤。
2. 终端必须按5.2节在WebSocket帧里封装数据。如果数据太大或者如果数据在终端准备开始发送数据时不能完整可得, 终端可能选择封装数据为一序列的帧, 按5.4节。
3. 包含数据的第一帧的操作码(frame-opcode)必须设置为恰当的值, 按5.2节, 因为这决定数据将被接收者解释为文本或二进制数据。
4. 包含数据的最后帧的FIN位(frame-fin)必须设置为1, 如5.2节定义。

5. 如果数据由客户端发送，帧必须被标记，如5.3节定义。
6. 如果为WebSocket连接协商了扩展（第9章），额外的考虑可能按定义的扩展 应用。
7. 已形成的帧必须在底层的网络连接上传输。

## 6.2. 接收数据

为了接收WebSocket数据，终端监听底层的网络连接。到来的数据必须按5.2节定义的WebSocket帧解析。如果接收到控制帧，必须按5.5节的定义处理。当接收到数据帧时，终端必须知道数据的类型，通过操作码（**frame-opcode**）定义。帧的"Application data"是定义为消息的/data/的。如果帧包含一个未分帧的消息，就是说接收到一个包含类型和数据的WebSocket消息。如果帧是分帧了的消息的一部分，应用数据是后续帧的数据的拼接。当最后帧（有FIN位指示）接收到时，就是说WebSocket消息接收到了，数据由所有帧的应用数据拼接组成，类型由第一帧指示。后续的数据帧必须解释为属于新的WebSocket消息的。

扩展（第9章）可能改变如何读取数据的语义，具体包括什么组成消息的边界。扩展，在"Applicationdata"前添加"Extensiondata"，是计入有效载荷的，也可能修改"Application data"。

服务器必须移除从客户端接收到的数据帧的标记，如5.3节。

## 7. 关闭连接

### 7.1. 定义

#### 7.1.1. 关闭WebSocket连接

为关闭WebSocket连接，终端关闭底层的TCP连接。终端应该使用能够干净地关闭TCP连接和TLS会话的方法。如果可以，丢弃可能接收到的后续的任意字节。当需要时，终端可能使用任意方法关闭连接，例如受到攻击时。

在大多数情况下，应该由服务器先关闭底层的TCP连接，这样它持有TIME\_WAIT状态，而不是客户端（因为这会导致它在2个最大段生命周期（2MSL）内不能重新打开，然而对服务器没有相应的影响，因为当遇到有更高序号的SYN时，TIME\_WAIT连接可以立即重新打开）。在不正常的情况（例如在合适的时间内没有从服务器接收到TCP关闭）下，客户端可能开始TCP关闭。这样，当服务器得到关闭WebSocket连接指示，应该马上开始TCP关闭，当客户端得到关闭WebSocket连接的指示，它应该等待来自服务器的TCP关闭。

As an example of how to obtain a clean closure in C using Berkeley sockets, one would call shutdown() with SHUT\_WR on the socket, call recv() until obtaining a return value of 0 indicating that the peer has also performed an orderly shutdown, and finally call close() on the socket.

#### 7.1.2. 开始WebSocket关闭握手

为用状态码/code/（7.4节）和可选的关闭理由/reason/（7.1.6节）开始WebSocket关闭握手，终端必须发送一个关闭帧，如5.5.1节描述。一旦终端已经发送和接收到关闭帧，终端应该关闭WebSocket连接，如7.1.1节定义。

#### 7.1.3. WebSocket关闭握手已开始

基于正在发送或接收关闭帧，这说明WebSocket关闭握手已经开始，WebSocket连接处于CLOSING状态。

#### 7.1.4. WebSocket连接已关闭

当底层TCP连接已关闭，说明WebSocket连接已关闭，处于CLOSED状态。如果TCP连接在WebSocket关闭握手完成后关闭，就是说WebSocket连接已干净地关闭。

如果WebSocket连接没能建立，这也说WebSocket连接已关闭，但是不干净。

#### 7.1.5. WebSocket连接关闭码

如5.5.1和7.4节定义，一个关闭控制帧可能包含一个状态码指示关闭的理由。一个WebSocket连接的关闭可能由

任意一方的终端发起，也可能同时。**WebSocket**连接的关闭码是作为状态码（7.4节）定义，包含在实现这个协议的应用程序接收到的第一个关闭控制帧。如果关闭控制帧不包含状态码，**WebSocket**连接的关闭码被认为是1005。如果**WebSocket**连接已关闭，且终端没有收到关闭控制帧（这可能出现在底层传输连接丢失的情况），**WebSocket**连接的关闭码被认为是1006。

注意：两个终端可能不对**WebSocket**连接的关闭码达成一致。例如，如果远程终端发送了一个关闭帧，但本地应用程序还没有从它自己的套接字接收缓存读取包含控制帧的数据，本地应用程序独自决定连接，并发送了一个关闭帧，两个终端都已发送和接收到一个关闭帧，不再发送更多的关闭帧。每个终端将看到另一端发送的状态码，作为**WebSocket**连接的关闭码。这样，在两个终端大致同时开始关闭握手的情况下，它们可能不能在**WebSocket**连接的关闭码上达成一致。

#### 7.1.6. WebSocket关闭理由

如5.5.1和7.4节定义，一个关闭帧可能包含状态码指示关闭的理由，后接UTF-8编码的数据，数据的解释留给终端，不在协议定义。**WebSocket**连接的关闭可能由两个终端的任何一方发起，也可能同时。**WebSocket**连接关闭理由定义为UTF-8编码的数据，跟在状态码（7.4节）后面，包含在实现本协议的应用程序接收到的一个关闭帧里。如果关闭帧里没有这样的数据，**WebSocket**连接关闭理由被认为是空的字符串。

注意：遵循7.1.5节提示的，两个终端可能不在**WebSocket**连接关闭理由上达成一致。

#### 7.1.7. 使WebSocket连接失败

特定的算法和规范要求终端使**WebSocket**连接失败。为此，客户端必须关闭**WebSocket**连接，可能通过恰当的方式把问题报告给用户（这对开发者特别有用）。类似地，服务器必须关闭**WebSocket**连接，并应该记录问题到日志。

如果**WebSocket**连接在终端要求使**WebSocket**连接失败之前已经建立，终端应该发生一个关闭帧，用合适的状态码，在继续关闭**WebSocket**连接之前。终端可能忽略发送关闭帧，如果它认为另一端不大可能接收和处理关闭帧，是因为导致**WebSocket**连接关闭的错误。一个终端必须不继续尝试处理来自远程终端的数据（包括响应的控制帧），在收到使**WebSocket**连接失败的指令后。

除了上面指示的或由应用程序层指定的，客户端不应该关闭连接。

### 7.2. 非正常关闭

#### 7.2.1. 客户端开始的关闭

特定的算法，特别是在打开握手的时候，要求客户端使**WebSocket**连接失败。为此，客户端必须使**WebSocket**连接失败，按7.1.7节定义。

如果在底层传输层连接意外消失的任何时刻，客户端必须使**WebSocket**连接失败。

除了上面指示的或由应用程序层指定的，客户端不应该关闭连接。

#### 7.2.2. 服务器端开始的关闭

特殊的算法要求或建议服务器中止**WebSocket**连接，在打开握手的时候。为此，服务器必须简单地关闭**WebSocket**连接（7.1.1节）。

#### 7.2.3. 从非正常关闭恢复

非正常关闭可能由任意数量的原因导致。这样的关闭可能是由于暂时的错误，在这种情况下，重连可能导致一个好的连接，恢复到正常操作。这样的关闭也可能是因为非暂时性问题，在in whichcase if each deployed client experiences an abnormal closure and immediately andpersistently tries to reconnect, the server may experience what amounts to a denial-of-service attack by a large numberof clients trying to reconnect. The end result of such a scenario could be that the service isunable to recover in a timelymanner or recovery is made much more difficult.

To prevent this,clients SHOULD use some form of backoff when trying to reconnect afterabnormal closures as described in this section.



The first reconnectattempt SHOULD be delayed by a random amount of time. The parameters by which this randomdelay is chosen are left

to the client to decide; a value chosen randomly between 0 and 5 seconds is a reasonable initial delay though clients MAY choose a different interval from which to select a delay length based on implementation experience and particular application.

Should the first reconnect attempt fail, subsequent reconnect attempts SHOULD be delayed by increasingly longer amounts of time, using a method such as truncated binary exponential backoff.

### 7.3. 正常关闭连接

服务器在需要时可能关闭WebSocket连接。客户端不应该贸然关闭WebSocket连接。在任何情况下，终端遵循开始WebSocket连接关闭握手的步骤（7.1.2节）来发起关闭。

### 7.4. 状态码

当关闭一个已建立连接时（如，在打开握手完成后，发送一个关闭帧），终端可能指示一个关闭理由。本规范未定义终端如何解释理由和给出理由要采取的动作。本规范定义了一组预定义的状态码，指定了哪些范围可能由扩展、框架、终端应用程序使用。状态码和关联的文本消息对于关闭帧是可选的。

#### 7.4.1. 状态码定义

在发送关闭帧时，终端可能使用下面预定义的状态码。

1000: 表示正常关闭，意味着连接建立的目的已完成。

1001: 表示终端离开，例如服务器关闭或浏览器导航到其他页面。

1002: 表示终端因为协议错误而关闭连接。

1003: 表示终端因为接收到不能接受的数据而关闭（例如，只明白文本数据的终端可能发送这个，如果它接收到二进制消息）。

1004: 保留。这个特定含义可能在以后定义。

1005: 保留。且终端必须不在控制帧里设置作为状态码。它是指定给应用程序而非作为状态码使用的，用来指示没有状态码出现。

1006: 同上。保留。且终端必须不在控制帧里设置作为状态码。它是指定给应用程序而非作为状态码使用的，用来指示连接非正常关闭，例如，没有发生或接收到关闭帧。

1007: 表示终端因为接收到的数据没有消息类型而关闭连接。

1008: 表示终端因为接收到的消息背离它的政策而关闭连接。这是一个通用的状态码，用在没有更合适的状态码或需要隐藏具体的政策细节时。

1009: 表示终端因为接收到的消息太大以至于不能处理而关闭连接。

1010: 表示客户端因为想和服务器协商一个或多个扩展，而服务器不在响应消息返回它（扩展）而关闭连接。需要的扩展列表应该出现在关闭帧的/reason/部分。注意，这个状态码不是由服务器使用，因为它会导致WebSocket握手失败。

1011: 表示服务器因为遇到非预期的情况导致它不能完成请求而关闭连接。

1015: 保留，且终端必须不在控制帧里设置作为状态码。它是指定用于应用程序希望用状态码来指示连接因为TLS握手失败而关闭。

#### 7.4.2. 保留的状态码范围

0 - 999 : 未使用。

1000 - 2999 : 此范围的保留给本协议使用的，它以后的校订和扩展将以永久的和容易获取 公开规范里指定。

3000 - 3999 : 此范围保留给类库、框架和应用程序使用。这些状态码直接在IANA注册。 本协议未定义如何解释这些状态码。

4000 - 4999 : 此范围保留给私用，且不能注册。这些值可以在预先达成一致的WebSocket 应用程序间使用。本协议未定义如何解释这些值。

### 8. 错误处理

#### 8.1. 处理UTF-8编码的数据的错误

当终端以UTF-8解释字节流时发现字节流不是一个合法的UTF-8流，那么终端必须使WebSocket连接失败。这个规则适用于打开握手时及后续的数据交换时。

### 9. 扩展

#### 9.1. 协商扩展

#### 9.2. 已知扩展

### 10. 安全考虑

#### 10.1. 非浏览器客户端

#### 10.2. Origin考虑

#### 10.3. Attacks On Infrastructure

#### 10.4. Implementation-Specific Limits

#### 10.5. WebSocket Client Authentication

#### 10.6. Connection Confidentiality and Integrity

#### 10.7. Handling of Invalid Data

#### 10.8. Use of SHA-1 by the WebSocketHandshake

### 11. IANA Considerations

### 12. Using the WebSocket Protocol from Other Specifications

上一篇 C#图片处理高级应用(裁剪,缩放,清晰度,水印)

下一篇 WebScket 规范 + WebSocket 协议

我的同类文章

websocket (5)

• js判断客户端是iOS还是An...

2014-06-24

阅读 921

• 使用DEFLATE的IP有效载...

2012-10-18

阅读 639

• 基于.NET的WebSocket的...

2012-10-16

阅读 527

• websocket通信协议(已更...

2012-10-15

阅读 1275

• WebScket 规范 + WebSo...

2012-10-15

阅读 8182

参考知识库



.NET 知识库  
1914 关注 | 805 收录



Scala 知识库  
972 关注 | 677 收录



算法与数据结构知识库  
10241 关注 | 2279 收录

猜你在找

全网服务器数据备份解决方案案例实践

Qt网络编程实战之HTTP服务器

Nginx服务器入门

EasyDarwin开源流媒体服务器：编译、配置、部署

阿里云ECS Linux服务器项目部署实战视频课程

RFC6455-The WebSocket protocol 之二

RFC6455-The WebSocket protocol 之八7 Closing

RFC6455-The WebSocket protocol 之十10

websocket rfc6455 中文版链接

RFC6455-The WebSocket protocol 之三WebSocket



解决车载设备的静噪问题，  
就选村田制作所的静噪元件！

更多内容 ▶

查看评论

2楼 qifali4184 2016-10-20 13:47发表

C

[java] view plain copy print ?

01. public

1楼 jcplusAres 2016-08-19 14:10发表

C

赞

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题

Hadoop

AWS

移动游戏

Java

Android

iOS

Swift

智能硬件

Docker

OpenStack

VPN

Spark

ERP

IE10

Eclipse

CRM

JavaScript

数据库

Ubuntu

NFC

WAP

jQuery

BI

HTML5

Spring

Apache

.NET

API

HTML

SDK

IIS

Fedora

XML

LBS

Unity

Splashtop

UML

components

Windows Mobile

Rails

QEMU

KDE

Cassandra

CloudStack

FTC

coremail

OPhone

CouchBase

云计算

iOS6

Rackspace

Web App

SpringSide

Maemo

Compuware

大数据

aptech

Perl

Tornado

Ruby

Hibernate

ThinkPHP

HBase

Pure

Solr

Angular

Cloud Foundry

Redis

Scala

Django

Bootstrap

