Internet Engineering Task Force (IETF)
Request for Comments: 6455
Category: Standards Track
ISSN: 2070-1721

I. Fette
Google, Inc.
A. Melnikov
Isode Ltd.

December 2011

张开涛 [译]

WebSocket 协议

摘要

WebSocket 协议实现在受控环境中运行不受信任代码的一个客户端到一个从该代码已经选择加入通信的远程主机之间的全双工通信。用于这个的安全模型是通常由 web 浏览器使用的基于来源的安全模型。该协议包括一个打开阶段握手、接着是基本消息帧、TCP 之上的分层（layered over TCP）。该技术的目标是为需要与服务器全双工通信且不需要依赖打开多个 HTTP 连接（例如，使用 XMLHttpRequest 或<iframe>和长轮询）的基于浏览器应用的提供一种机制。

本备忘录状态

这是一个 Internet 标准跟踪文档。

本文档是互联网工程任务组（IETF）的一个产物。它代表了 IETF 社区的共识。它已接受公共审查和已经被互联网工程指导委员会（IESG）认可发布。Internet 标准的更多信息可在 RFC 5741 第 2 节 找到。

本文档的当前状态信息、任何勘误表以及如何提供它的反馈，可于 http://www.rfc-editor.org/info/rfc6455获取。

版权声明

目录

1.引言

1.1.背景

本节是非规范的。

过去，创建需要在客户端和服务之间双向通信（例如，即时消息和游戏应用）的 web 应用， 需要一个滥用的 HTTP 来轮询服务器进行更新但以不同的 HTTP 调用发生上行通知[RFC6202]。

这将导致各种各样的问题：

o 服务器被迫为每个客户端使用一些不同的底层 TCP 连接： 一个用于发送信息到客户端和一个新的用于每个传入消息。

o 线路层协议有较高的开销，因为每个客户端-服务器消息都有一个 HTTP 头信息。

o 客户端脚本被迫维护一个传出的连接到传入的连接的映射来跟踪回复。

一个简单的办法是使用单个 TCP 连接双向传输。这是为什么提供 WebSocket 协议。与 WebSocket API 结合[WSAPI]，它提供了一个 HTTP 轮询的替代来进行从 web 页面到远程服务器的双向通信。

同样的技术可以用于各种各样的 web 应用：

游戏、股票行情、同时编辑的多用户应用、服务器端服务以实时暴露的用户接口、等等。

WebSocket 协议被设计来取代现有的使用 HTTP 作为传输层的双向通信技术，并受益于现有的基础设施（代理、过滤、身份验证）。这样的技术被实现来在效率和可靠性之间权衡，因为 HTTP 最初目的不是用于双向通信（参见[RFC6202]的进一步讨论）。WebSocket 协议试图在现有的 HTTP 基础设施上下文中解决现有的双向 HTTP 技术目标；同样，它被设计工作在 HTTP 端口 80 和 443，也支持 HTTP 代理和中间件，即使这具体到当前环境意味着一些复杂性。但是，这种设计不限制 WebSocket 到 HTTP，且未来的实现可以在一个专用的端口上使用一个更简单的握手，且没有再创造整个协议。最后一点是很重要的，因为交互消息的传输模式不精确地匹配标准 HTTP 传输并可能在相同组件上包含不常见的负载。

1.2.协议概述

本节是非规范的。

本协议有两部分：握手和数据传输。

来自客户端的握手看起来像如下形式：

```
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
```

来自服务器的握手看起来像如下形式：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

来自客户端的首行遵照 Request-Line 格式。

来自服务器的首行遵照 Status-Line 格式。

Request-Line 和 Status-Line 制品定义在[RFC2616]。

在这两种情况中一个无序的头字段集合出现在首行之后。这些头字段的意思指定在本文档的第 4 章。另外的头字段也可能出现，例如 cookies[RFC6265]。头的格式和解析定义在[RFC2616]。

一旦客户端和服务器都发送了它们的握手，且如果握手成功，接着开始数据传输部分。这是一个每一端都可以的双向通信信道，彼此独立，随意发生数据。

一个成功握手之后，客户端和服务器来回地传输数据，在本规范中提到的概念单位为“消息”。在线路上，一个消息是由一个或多个帧的组成。WebSocket 的消息并不一定对应于一个特定的网络层帧，可以作为一个可以被一个中间件合并或分解的片段消息。

一个帧有一个相应的类型。属于相同消息的每一帧包含相同类型的数据。从广义上讲，有文本数据类型（它被解释为 UTF-8[RFC3629]文本）、二进制数据类型（它的解释是留给应用）、和控制帧类型（它是不准备包含用于应用的数据，而是协议级的信号，例如应关闭连接的信号）。这个版本的协议定义了六个帧类型并保留 10 以备将来使用。

1.3.打开阶段握手

本节是非规范的。

打开阶段握手目的是兼容基于 HTTP 的服务器软件和中间件，以便单个端口可以用于与服务器交流的 HTTP 客户端和与服务器交流的 WebSocket 客户端。最后，WebSocket 客户端的握手是一个 HTTP Upgrade 请求：

```
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
```

依照[RFC2616]，握手中的头字段可能由客户端按照任意顺序发送，因此在接收的不同头字段中的顺序是不重要的。

“Request-URI”的 GET 方法[RFC2616]用于识别 WebSocket 连接的端点，即允许从一个 IP 地址服务的多个域名，也允许由单台服务器服务的多个 WebSocket 端点。

客户端按照[RFC2616]在它的握手的|Host|头字段中包含主机名，以便客户端和服务器都都能验证他们同意哪一个正在使用的主机。

在 WebSocket 协议中另外的头字段可以用于选择选项。典型的选项在这个版本中可用的是子协议选择器（|Sec-WebSocket-Protocol|）、客户端支持的扩展列表（|Sec-WebSocket-Extensions|）、|Origin|头字段等。|Sec-WebSocket-Protocol|请求头字段可以用来表示客户端接受的子协议（WebSocket 协议上的应用级协议层）。服务器选择一个可接受的协议或不，并在它的握手中回应该值表示它已经选择了那个协议。

```
Sec-WebSocket-Protocol: chat
```

|Origin|头字段[RFC6454]是用于保护防止未授权的被浏览器中的使用 WebSocket

API 的脚本跨域使用 WebSocket 服务器。服务器收到 WebSocket 连接请求生成的脚本来源。如果服务器不想接受来自此来源的连接，它可以选择通过发送一个适当的 HTTP 错误码拒绝该连接。这个头字段由浏览器客户端发送，对于非浏览器客户端，如果它在这些客户端上下文中有意义，这个头字段可以被发送。

最后，服务器要证明收到客户端 WebSocket 握手的客户端，以便服务器不接受不是 WebSocket 连接的连接。这可以防止一个通过使用 XMLHttpRequest [XMLHttpRequest]或一个表单提交发送它精心制作的包欺骗 WebSocket 服务器的攻击者。

为了证明收到的握手，服务器必须携带两条信息并组合他们形成一个响应。

第一条信息源自客户端握手中的| Sec-WebSocket-Key |头信息：

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

对于这个头字段，服务器必须携带其值（出现在头字段上，如，减去开头和结尾空格的 base64-编码[RFC4648]的版本）并将这个与字符串形式的全局唯一标识符（GUID，[RFC4122]）“258EAFA5-E914-47DA-95CA-C5AB0DC85B11”连接起来，其不太可能被不理解 WebSocket 协议的网络端点使用。SHA-1 散列（160 位）[FIPS.180-3]、base-64 编码（参见 [RFC4648]第 4 章 ）、用于这个的一系列相关事物接着在服务器握手过程中返回。

具体而言，如果在上面例子中，|Sec-WebSocket-Key| 头字段的值为“ dGhlIHNhbXBsZSBub25jZQ== ”，服务器将连接字符串“ 258EAFA5-E914-47DA-95CA-C5AB0DC85B11 ”形成字符串“dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CA-C5AB0DC85B11”。服务器接着使用 SHA-1 散列这个，并产生值 0xb3 0x7a 0x4f 0x2c 0xc0 0x62 0x4f 0x16 0x90 0xf6 0x46 0x06 0xcf 0x38 0x59 0x45 0xb2 0xbe 0xc4 0xea。这个值接着使用 base64 编码（参见 [RFC4648] 第 4 章 ），产生值“ s3pPLMBiTxaQ9kYGzzhZRbK+xOo= ”。这个值将接着在 |Sec-WebSocket-Accept|头字段中回应。

来自服务器的握手比客户端握手更简单。首行是一个 HTTP Status-Line，具有状态码 101：

HTTP/1.1 101 Switching Protocols

101 以外的任何状态码表示 WebSocket 握手没有完成且 HTTP 语义仍适用。头信息遵照该状态码。

|Connection|和|Upgrade|头字段完成 HTTP 升级。|Sec-WebSocket-Accept|头字段表示服务器是否将接受该连接。如果存在，这个头字段必须包括客户端在 |Sec-WebSocket-Key|中现时发送的与预定义的 GUID 的散列。任何其他值不能被

解释为一个服务器可接受的连接。

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

这些字段由 WebSocket 客户端为脚本页面做检查。如果|Sec-WebSocket-Accept|不能匹配盼望的值、如果头字段缺失、或 HTTP 状态码不是 101，则连接将不能建立，且 WebSocket 帧将不发生。

可选的字段也可以被包含在内。在这合格版本的协议中，主要可选字段是|Sec-WebSocket-Protocol|，其表示服务器选择的子协议。WebSocket 客户端验证服务器包含的在 WebSocket 客户端握手中指定的一个值。声明多个子协议的服务器必须确保它选择一个，基于客户端握手并指定它在其握手中。

Sec-WebSocket-Protocol: chat

服务器也可以设置 cookie 相关的可选字段为_set_cookies，描述在[RFC6265]。

1.4.关闭阶段握手

本节是非规范的。

关闭阶段握手比打开阶段握手简单得多。

两个节点中的任一个都能发送一个控制帧与包含一个指定控制序列的数据来开始关闭阶段握手（详见 5.5.1 节）。在收到这样一个帧时，另一个节点在响应中发送一个 Close 帧，如果还没有发送一个。在收到那个控制帧时，第一个节点接着关闭连接，安全地知道没有更多的数据到来。

发送一个控制帧之后，表示连接将被关闭，一个节点不会发送任何更多的数据；在接收到一个控制帧之后，表示连接将被关闭，一个节点会丢弃收到的任何更多的数据。

对于两个节点同时地初始化这个握手是安全的。

关闭阶段握手目的是完成 TCP 关闭握手（FIN/ACK），基于 TCP 关闭阶段握手不总是可靠的端到端，尤其在存在拦截代理和中间件。

通过发送一个 Close 帧并等待响应中的 Close 帧，某些情况下可避免数据不必要的丢失。例如，在某些平台上，如果一个 socket 关闭了，且接收队列中有数据，一个 RST 包被发送了，这样会导致接受 RST 的一方的 recv()失败，即使有数据等待读取。

1.5.设计理念

本节是非规范的。

WebSocket 协议应该以最小帧的原则设计（唯一存在的框架是使协议基于帧而不是基于流且支持区分 Unicode 文本和二进制帧）。期望通过应用层将元数据分层在 WebSocket 之上，同样地，通过应用层将元数据分层在 TCP 之上（例如，HTTP）。

从概念上讲，WebSocket 只是 TCP 之上的一层，执行以下操作：

- 为浏览器添加一个 web 基于来源的安全模型
- 添加一个寻址和协议命名机制来支持在一个 IP 地址的一个端口的多个主机名的多个服务
- 在 TCP 之上分一个帧机制层以回到 TCP 基于的 IP 包机制，但没有长度限制
- 包括一个额外的带内（in-band）关闭阶段握手，其被设计来工作在现存的代理和其他中间件。

除此之外，WebSocket 没有添加任何东西。基本上，它的目的是尽可能接近仅暴露原始 TCP 到脚本，尽可能考虑到 Web 的约束。它也被设计为它的服务器能与 HTTP 服务器共享一个端口的这样一种方式，通过持有它的握手是一个有效的 HTTP Upgrade 请求。一个可以在概念上使用其他协议来建立客户端-服务器消息，但 WebSocket 的意图是提供一个相对简单的协议，可以与现有 HTTP 和部署的 HTTP 基础设施（例如代理）同时存在，并尽可能接近 TCP，且对于使用考虑到安全考虑的这样的基础设施同样是安全的，有针对性的补充以简化使用并保持简单的事情简单（如增加的消息语义）。

协议的目的是为了可扩展；未来版本将可能引入额外的概念如复用（multiplexing）。

1.6.安全模型

本节是非规范的。

WebSocket 协议使用浏览器使用的来源模型限制 web 页面可以与 WebSocket 服务器通信，当 WebSocket 协议是从一个 web 页面使用。当然，当 WebSocket 协议被一个独立的客户端直接使用时（也就是，不是从浏览器中的一个 web 页面），来源模型不再有用，因为客户端可以提供任意随意的来源字符串。

该协议的目的是无法与现有的协议如 SMTP[RFC5321]和 HTTP 建立一个连接，同时允许 HTTP 服务器来选择支持该协议如果想要。这是通过具有严格的和详尽的握手和通过限制在握手完成之前能被插入到连接的数据（因此限制多少服务器可以被应用）实现的。

当数据是来自其他协议时，同样的目的是无法建立连接的，尤其发送到一个 WebSocket 服务器的 HTTP，例如，如果一个 HTML“表单”提交到 WebScoket 服务器可能会发生。这主要通过要求服务器验证它读取的握手来实现，它只能做如果握手包含适当的部分，只能通过一个 WebScoket 客户端发送。尤其是，在写本规范的时侯，|Sec-|开头的字段不能由 web 浏览器的攻击者设置，仅能使用 HTML 和 JavaScript API，例如 XMLHttpRequest [XMLHttpRequest]。

1.7.与 TCP 和 HTTP 的关系

本节是非规范的。

WebSocket 协议是一个独立的基于 TCP 的协议。它与 HTTP 唯一的关系是它的握手是由 HTTP 服务器解释为一个 Upgrade 请求。

默认情况下，WebSocket 协议使用端口 80 用于常规的 WebSocket 连接和端口 443 用于 WebSocket 连接的在传输层安全（TLS）[RFC2818]之上的隧道化。

1.8.建立连接

本节是非规范的。

当一个连接到一个 HTTP 服务器共享的端口时（这种情况是很可能在传输信息到端口 80 和 443 出现），连接将出现在 HTTP 服务器，是一个正常的具有一个 Upgrade 提议的 GET 请求。在相对简单的安装，只用一个 IP 地址和单台服务器用于所有数据传输到单个主机名，这可能允许一个切实可行的办法对基于 WebSocket 协议的系统进行部署。在更复杂的安装（例如，负载均衡和多服务器），一组独立的用于 WebSocket 连接的主机从 HTTP 服务器分离出来可能更容易管理。在写该规范的时候，应该指出的是，在端口 80 和 443 上的连接有明显不同

的成功率，对于在端口 443 上的连接是明显更有可能成功，尽管这可能会随着时间而改变。

1.9.使用 WebSocket 协议的子协议

本节是非规范的。

客户端可能通过包含|Sec-WebSocket-Protocol|字段在它的握手中使用一个特定的子协议请求服务器。如果它被指定，服务器需要在它的响应中包含同样的字段和一个选择的子协议值用于建立连接。

这些子协议名字应该按照 11.5 节被注册。为了避免潜在的碰撞，推荐使用包含 ASCII 版本的子协议发明人的域名的名字。 例如，如果 Example 公司要创建一个 Chat 子协议，由 Web 上的很多服务器实现，它们可能命名它为“chat.example.com”。如果 Example 组织命名它们的竞争子协议为“chat.example.org”，那么两个子协议可能由服务器同时实现，因为服务器根据客户端发送的值动态地选择使用哪一个子协议。

通过改变子协议的名字，子协议可以以向后不兼容方式版本化，例如，要从“bookings.example.net”到“v2.bookings.example.net”。就 WebSocket 客户端而言，这些子协议将被视为是完全不同的。向后兼容的版本可以通过重用相同的子协议字符串实现，但要仔细设置实际的子协议以支持这种可扩展性。

2.一致性要求

在本规范中所有图表、示例、和注释是非规范的，以及所有章节明确地标记为非规范的。除此之外，在本规范中的一切是规范的。

该文档中的关键字“必须（MUST）”、“不能（MUST NOT）”、“需要（REQUIRED）”、“应当（SHALL）”、“不得（SHALL NOT）”、“应该（SHOULD）”、“不应该（SHOULD NOT）”、“推荐（RECOMMENDED）”、“可能（MAY）”、和“可选的（OPTIONAL）”由[RFC2119]中的描述解释。

作为算法一部分的祈使句中的要求措辞（例如“去掉任何前导空格字符”或“返回 false 并终止这些步骤” ）解释为引入算法中使用的关键字（"MUST", "SHOULD", "MAY"等）的意思。

作为算法或特定的步骤的一致性要求措辞可以（MAY）以任何形式实现，只要最终结果是相等的。（尤其是，定义在本规范中的算法目的是容易遵循而不必是高性能的）

2.1.术语和其他约定

ASCII 指定义在[ANSI.X3-4.1986]中的字符编码方案。

此文档中提到的 UTF-8 值和使用 UTF-8 标记法格式定义在 STD 63 [RFC3629]。

关键术语例如命名算法或定义是表示像_this_。

头字段名字或变量表示像|this|。

变量值表示像/this/。

本文档提及的程序_失败 WebSocket 连接_。该程序定义在 7.1.7 节。

_将字符串转换为 ASCII 小写_意思是替换 U+0041 到 U+005A（也就是，拉丁文，大写字母 A 到拉丁文大写字母 Z）范围的所有字符为 U+0061 到 U+007A（也就是，拉丁文小写字母 A 到拉丁文小写字母 Z）范围的对应的字符。

以一个_ASCII 不区分大小写_方式比较两个字符串意思是精确地比较它们，代码点对代码点，除了 U+0041 到 U+005A（也就是，拉丁文，大写字母 A 到拉丁文大写字母 Z）范围中的字符，U+0061 到 U+007A（也就是，拉丁文小写字母 A 到拉丁文小写字母 Z）范围中的对应的字符被认为也匹配。

用于本文档的术语“URI”定义在[RFC3986]。

当一个实现需要_发送_作为 WebSocket 一部分的数据，实现可能（MAY）任意地推迟实际的传输，例如，缓冲数据为了发送更少的 IP 包。

注意，该文档同时使用[RFC5234]和[RFC2616]的 ABNF 变体在不同章节。

3.WebSocket URI

本规范定义了两个 URI 方案，使用定义在 RFC5234[RFC5234]中的 ABNF 句法、和术语和由 URI 规范 RFC 3986 [RFC3986]定义的 ABNF 制品。

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>

端口组件是可选的；用于“WS”的默认端点是 80，而用于“WSS”默认端口是 443。

如果方案组件不区分大写匹配“wss”，URI 被称为“安全的”（它是说，“设置了安全标记”）。

“resource-name”（在 4.1 节也称为/resource name/）可以通过连接以下来构造：

- "/" 如果路径组件是空
- 路径组件
- "?" 如果查询组件是非空
- 查询组件

片段（译者注：# Fragment）标识符在 WebSocket URI 中是无意义的且必须不用在这些 URI 上。任何 URI 方案，字符“#”，当不表示片段开始时，必须被转义为%23。

4.打开阶段握手

4.1.客户端要求

要_建立 WebSocket 连接_，客户端打开一个连接并发送一个握手，就像本节中定义那样。一个连接最初被定义为一个 CONNECTING 状态。客户端将需要提供一个/host/、/port/、/resource name/、和/secure/标记，它们都是在第三章讨论的 WebSocket URI 的组件，连同一起使用的一个/protocols/和/extensions/列表。此外，如果客户端是一个 web 浏览器，它提供/origin/。客户端运行在一个受控环境，例如绑定到特定运营商的手机上的浏览器，可以下移（offload）连接管理到网络上的另一个代理。在这种情况下，用于本规范目的的客户端被认为包括手机软件和任何这样的代理。

当客户端要_建立一个 WebSocket 连接_，给定一组（/host/、/port/、/resource name/、和/secure/标记）、连同一起使用的一个/protocols/和/extensions/列表、和在 web 浏览器情况下的一个/origin/，它必须打开一个连接、发送一个打开阶段握手、并读取服务器响应中的握手。应如何打开连接的确切要求、在打开阶段握手应发送什么、以及应如何解释服务器响应，在本节如下所述。在下面的文本中，我们将使用第三章的术语，如定义在那章的“/host” 、和“/sucure/标记” 。

1. 传入该算法的 WebSocket URI 组件(/host/、/port/、/resource name/、和/secure/标记) 根据指定在第 3 章的 WebSocket URI 规范，必须是有效的。如果任何组件是无效的，客户端必须_失败 WebSocket 连接_并终止这些步骤。

2. 如果客户端已经有一个到通过主机/host/和端口/port/对标识的远程主机（IP 地址）的 WebSocket 连接，即使远程主机是已知的另一个名字，客户端必须等待直到连接已建立或由于连接已失败。必须不超过一个连接处于 CONNECTING 状态。如果同时多个连接到同一个 IP 地址，客户端必须 序列化它们，以致一次不多于一个连接在以下步骤中运行。

如果客户端不能决定远程主机的 IP 地址（例如，因为所有通信是通过代理服务器本身进行 DNS 查询），那么客户端必须假定这步的目的是每一个主机名引用一个不同远程主机，且相反，客户端应该限制同时挂起的连接总数为一个适当低的数（例如，客户端可能允许到 a.example.com 和 b.example.com 同时挂起连接，但如果 30 个同时连接到同一个请求的主机，那可能是不允许的）。例如，在一个 web 浏览器上下文中，客户端需要考虑用户已经打开的标签数量，在设置同时挂起的连接数量的限制时。

注意：这使得它很难仅通过打开大量的 WebSocket 连接到远程主机为脚本执行一个拒绝服务攻击。当攻击在关闭连接之前被暂停时，服务器可以进一步降低自身的负载，因为这将降低客户端重新连接的速度。

注意：没有限制一个客户端可以与单个远程主机有的已建立的 WebSocket 连接数量。服务器可以拒绝接受来自具有大量的现有连接的主机/IP 地址的连接或当遭受高负载时断开占用资源的连接。

3. _使用代理_：当有 WebSocket 协议连接主机/host/和端口/port/时，如果客户端配置使用代理，那么客户端应该连接到代理并要求它打开一个到由/host/给定主机和/port/给定端口的 TCP 连接。

例子：例如，如果客户端为所有信息传输使用一个 HTTP 代理，那么如果它试图连接到服务器 example.com 的 80 端口，它可能会发送以下行到代理服务器：

CONNECT example.com:80 HTTP/1.1
Host: example.com
如果还有密码，连接可能看起来像：
CONNECT example.com:80 HTTP/1.1
Host: example.com
Proxy-authorization: Basic ZWRuYW1vZGU6bm9jYXBlcyE=

如果客户端没有配置使用一个代理，那么应该打开一个直接 TCP 连接到由/host/给定的主机和/port/给定的端口。

注意：不暴露明确的 UI 来为 WebSocket 连接选择一个独立于其他代理的代理实现，鼓励使用 SOCKS5[RFC1928]代理用于 WebSocket 连接，如果有的话，或做不到这一点，选择为 HTTPS 连接配置代理胜过为 HTTP 连接配置代理。

为了代理自动配置脚本，传给函数的 URI 必须从/host/、/port/、/resource name/、和/secure/标记来构造，使用第三章给定的 WebSocket URI 定义。

注意：WebSocket 协议可以在代理自动配置脚本中从模式中识别（“ws”用于未加密的连接和“wss”用于加密的连接）。

4.如果连接无法打开，或者因为直接连接失败或者因为任何使用的代理返回一个错误，那么客户端必须_失败 WebSocket 连接_并终止连接尝试。

5、如果/secure/是 true，客户端必须在连接之上执行一个 TLS 握手在打开连接之后和发生握手数据之前[RFC2818]。如果这个失败了（例如，服务器的证书不能被验证），那么客户端必须_失败 WebSocket 连接_并终止连接。否则，所有在该通道上的进一步的通信必须通过加密隧道[RFC5246]。

客户端必须在 TLS 握手中使用服务器命名指示（Server Name Indication）扩展[RFC6066]。

一旦一个到服务器的连接连接（包括通过代理或在 TLS 加密隧道之上的连接），客户端必须发送一个打开阶段握手到服务器。该握手包括一个 HTTP Upgrade 请求，连同一个必需的和可选的头字段列表。该握手的要求如下所示。

1. 握手必须是像[RFC2616]指定的那样的有效的 HTTP 请求。

2. 请求方法必须是 GET、且 HTTP 版本必须是至少 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”部分必须匹配定义在第三章的（一个相对 URI）/resource name/或是一个绝对的 http/https URI，当解析时，有一个/resource name/、/host/、和/port/匹配相应的 ws/wss URI。

4. 请求必须包含一个|Host|头字段，其值包含/host/加上可选的“:”后跟/port/（当没用默认端口时）。

5. 请求必须包含一个|Upgrade|头字段，其值必须包含“websocket”关键字。

6. 请求必须包含一个|Connection|头字段，其值必须包含“Upgrade”标记。

7. 请求必须包含一个名字为| Sec-WebSocket-Key |的头字段，这个头字段的值必须是临时（nonce）组成的一个随机选择的已经 base64 编码的（参见[RFC4648]第 4 章）16 位的值。临时必须是为每个连接随机选择的。

   注意：作为一个例子，如果随机选择的值是字节序列 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10，头字段的值必须是“AQIDBAUGBwgJCgsMDQ4PEC==”

8. 如果请求来自一个浏览器客户端，请求必须包含一个名字为|Origin|[RFC6454]的头字段。如果连接是来自非浏览器客户端，如果该客户端的语义匹配描述在这的用于浏览器的使用情况时，请求可以包含这个字段。该头字段的值是在建立正运行的连接代码中的环境的 origin 的 ASCII 序列化。参考[RFC6454]获取如果构造该头字段的值的详细信息。

   作为一个例子，如果从 www.example.com 下载的代码试图建立到 ww2.example.com 的连接，该头字段的值将是“http://www.example.com”。

9. 请求必须包含一个名字为|Sec-WebSocket-Version|的头字段。该头字段的值必须是 13。

注意：尽管本文档的草案版本（-09、-10、-11、和-12）发布了（它们多不是编辑上的修改和澄清而不是改变电报协议[wire protocol]），值 9、10、11、和

12 不被用作有效的 Sec-WebSocket-Version。这些值被保留在 IANA 注册中心，但并将不会被使用。

10. 请求可以包含一个名字为|Sec-WebSocket-Protocol|的头字段。如果存在，该值表示一个或多个逗号分割的客户端想要表达的子协议，按优先顺序排列。包含该值的元素必须是非空字符串，且字符在 U+0021 到 U+007E 范围内但不包含定义在[RFC2616]中的分割字符且必须所有是唯一的字符串。用于该头字段值的 ABNF 是 1#token，其构造和规则定义在[RFC2616]给出。

11. 请求可以包含一个名字为|Sec-WebSocket-Extensions|的头字段。如果存在，该值表示客户端想要表达的协议级的扩展。该头字段的解释和格式描述在第 9.1 节。

12. 请求可以包含任意其他头字段，例如，cookie[RFC6265]和/或验证相关的头字段例如|Authorization|头字段[RFC2616]，其根据定义它们的文档处理。

一旦客户端的打开阶段握手已经发送，客户端在发送任何进一步数据之前必须等待自服务器的一个响应。客户端必须验证服务器的响应，如下所示：

1. 如果收到的服务器的状态码不是 101，客户端处理每个 HTTP[RFC2616]程序的响应。尤其是，如果收到一个 401 状态码客户端可能执行身份验证；服务器可能使用一个 3xx 状态码重定向客户端（但客户端不需要跟随他们），等等。否则，按以下步骤处理。

2. 如果响应缺少一个|Upgrade|头字段或|Upgrade|头字段包含的值不是一个不区分大小写的 ASCII 匹配值“websocket”，客户端必须_失败 WebSocket 连接_。

3. 如果想要缺少一个|Connection|头字段或|Connection|头字段不包含一个不区分大小的 ASCII 匹配值“Upgrade”符号，客户端必须_失败 WebSocket 连接_。

4. 如果想要缺少一个|Sec-WebSocket-Accept|头字段或|Sec-WebSocket-Accept|包含一个不是|Sec-WebSocket-Key|（一个字符串，不是 base64 编码的）与字符串“258EAFA5-E914-47DA-95CA-C5AB0DC85B11”但忽略任何前导和结尾空格相关联的 base64 编码的 SHA-1 值，客户端必须_失败 WebSocket 连接_。

5. 如果响应包含一个|Sec-WebSocket-Extensions|头字段且此头字段表示使用一个扩展但没有出现在客户端握手中（服务器表示的一个扩展，不是客户端请求的），客户端必须_失败 WebSocket 连接_。解析该头字段以确定请求了哪些扩展在 9.1 节讨论。

6. 如果响应包含一个|Sec-WebSocket-Protocol|头字段且该头字段表示使用一

个子协议但没出现在客户端握手中（服务器表示的一个子协议，不是客户端请求的），客户端必须_失败 WebSocket 连接_。

如果服务器响应不符合定义在本节和 4.2.2 节中的服务器握手的要求，客户端必须_失败 WebSocket 连接_。

请注意，根据[RFC2616]，所有命名在 HTTP 请求和 HTTP 响应中的头字段是不区分大小写的。

如果服务器响应验证了以上提供的，这是说，_WebSocket 连接建立了_且 WebSocket 连接处于 OPEN 状态。_使用中的扩展_被定义为一个（可能空的）字符串，其值等于服务器握手中提供的|Sec-WebSocket-Extensions|头字段的值或如果在服务器握手中没有该头字段则为 null 值。_使用中的子协议_被定义为服务器握手中的|Sec-WebSocket-Protocol|头字段的值或如果在服务器握手中没有该头字段则为 null 值。另外，如果在服务器握手中表示 cookie 应该被设置（定义在[RFC6265]）的任何头字段，这些 cookie 被称为_在服务器打开阶段握手期间的 Cookie 设置_。

4.2.服务器端要求

服务器可以下移（offload）连接管理到网络上的其他代理，例如，负载均衡和反向代理。在这样的情况下，用于本规范的目的的服务器被认为是包括服务器端基础设施的所有部分，从开始的设备到终止 TCP 连接，处理请求和发送响应的服务器的所有方式。

例如：一个数据中心可能有一个用适当的握手来响应 WebSocket 请求的服务器，并接着传递连接到另一个服务器来真正处理数据帧。对于本规范的目的，“服务器”是结合了两种计算机。

4.2.1.读取客户端的打开阶段握手

当客户端开始一个 WebSocket 连接，它发送它的打开阶段握手部分。服务器必须至少解析这个握手为了获取必要的信息来生成服务器握手部分。

客户端打开阶段握手包括以下部分。如果服务器，当读取握手时，发现客户端没有发送一个匹配下面描述的握手（注意，按照[RFC2616]，头字段顺序是不重要的），包括但不限制任何违反 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一个具有一个适当错误码的（例如 400 错误的请求）HTTP 响应。

1. 一个 HTTP/1.1 或更高版本的 GET 请求，包括一个“Request-URI”[RFC2616]应该被解释为定义在第 3 章的/resource name/（或一个包含/resource name/的绝对 HTTP/HTTPS URI）。

2. 一个|Host|头字段包含服务器的权限。

3. 一个|Upgrade|头字段包含值“websocket”，视为一个不区分大小写的 ASCII 值。

4. 一个|Connection|头字段包含符号“Upgrade”，视为一个不区分大小写的 ASCII 值。

5. 一个|Sec-WebSocket-Key|头字段，带有一个 base64 编码的值（参见[RFC4648]第 4 章），当解码时，长度是 16 字节。

6. 一个|Sec-WebSocket-Version|头字段，带有值 13。

7. 可选的，一个|Origin|头字段。该头字段由所有浏览器客户端发送。一个试图缺失此头字段的连接不应该被解释为来自浏览器客户端。

8. 可选的，一个|Sec-WebSocket-Protocol|头字段，带有表示客户端想要表达的协议的值列表，按优先顺序排列。

9. 可选的，一个|Sec-WebSocket-Extensions|头字段，带有表示客户端想要表达的扩展的值列表。此头字段的解释在 9.1 节讨论。

10. 可选的，其他头字段，例如这些用于发送 cookie 或请求服务器身份验证的。未知的头字段被忽略，按照[RFC2616]。

4.2.2.发送服务器的打开阶段握手

当客户端建议一个到服务器的 WebSocket 连接，服务器必须完成以下步骤来接受该连接并发送服务器的打开阶段握手。

1. 如果连接发送在一个 HTTPS (HTTP-over-TLS)端口上，在连接之上执行一个 TLS 握手。如果失败了（例如，在扩展的客户端 hello“server_name”扩展中的客户端指示的一个主机名，服务器对主机不可用），则关闭该连接；否则，用于该连接的所有进一步的通信必须贯穿加密的隧道[RFC5246]。

2. 服务器可以执行额外的客户端身份认证，例如，返回 401 状态码与描述在[RFC2616]中的相关的|WWW-Authenticate|头字段。

3. 服务器可以使用 3xx 状态码[RFC2616]重定向客户端。注意，此步骤可能连同，之前，或之后的可选的上面描述的身份验证步骤一起发生。

4. 建立如下信息：

/origin/

客户端握手中的|Origin|头字段表示建立连接的脚本的来源。Origin 是序列化为 ASCII 并转换为小写。服务器可以使用这个信息作为决定是否接受传入连接的一部分。如果服务器没有验证 origin，它将接受来自任何地方的连接。如果服务器不想接受这个连接，它必须返回一个适当的 HTTP 错误码（例如，403 Forbidden）并中断描述在本章中的 WebSocket 握手。更多详细信息，请参阅第 10 章。

/key/

在客户端握手中的|Sec-WebSocket-Key|头字段包括一个 base64 编码的值，如果解码，长度是 16 字节。这个（编码的）值用在创建服务器握手时来表示接受连接。服务器没必要使用 base64 解码|Sec-WebSocket-Key|值。

/version/

客户端握手中的|Sec-WebSocket-Version|头字段包括客户端试图通信的 WebSocket 协议的版本。如果该版本没有匹配服务器理解的一个版本，服务器必须中断描述在本节的 WebSocket 握手并替代返回一个适当的 HTTP 错误码（例如，426 Upgrade Required）且一个|Sec-WebSocket-Version|头字段表示服务器能理解的版本。

/resource name/

由服务器提供的服务的标识符。如果服务器提供多个服务，那么该值应该源自客户端我手中的 GET 方法的“Request-URI”中给定的资源名。如果请求的服务器不可用，服务器必须发生一个适当的 HTTP 错误码（例如 404 NotFound）并中断 WebSocket 握手。

/subprotocol/

或者一个代表服务器准备使用的子协议的单个值或者 null。选择的值必须源自客户端握手，从|Sec-WebSocket-Protocol|字段具体地选择一个值，服务器将使用它用于这个连接（如果有）。如果客户端握手不包含这样一个头字段或如果服务器不同意任何客户端请求的子协议，仅接受的值为 null。这个字段不存在等价于 null 值（意思是如果服务器不想同意任何建议的子协议，它必须在它的响应中不发送回一个|Sec-WebSocket-Protocol|头字段）。用于这些目的，空字符串与 null 值是不一样的，且它不是这个字段合法的值。用于该头字段的值 ABNF 是（符号），构造定义和规则在[RFC2616]中给出。

/extensions/

表示服务器准备使用的协议级别扩展的一个列表（可能为空）。如果服务器支持多个扩展，那么该值必须源自客户端握手，通过从|Sec-WebSocket-Extensions|字段具体地选择一个或多个值。这个字段不存在等价于 null 值。用于这些目的，空字符串与 null 值是不一样的。客户未列出的扩展必须不被列出。那些值应该被选择和解释的方法在 9.1 节讨论。

5. 如果服务器选择接受传入的连接，它必须以一个有效的表示以下的 HTTP 响应应答。

1. 一个按照 RFC2616[RFC2616]带有 101 响应码的 Status-Line。这样的响应可能看起来像“HTTP/1.1 101 Switching Protocols”。

2. 一个按照 RFC2616[RFC2616]带有值“websocket”的|Upgrade|头字段。

3. 一个带有“Upgrade”的| Connection |头字段。

4. 一个|Sec-WebSocket-Accept|头字段。该头字段的值通过连接/key/构造，它定义在 4.2.2 节第 4 步，带有字符串“258EAFA5-E914-47DA-95CA-C5AB0DC85B11”，采用 SHA-1 散列这个连接的值来获取一个 20 字节的值并 base64 编码（参考[RFC4648]第 4 章）这个 20 字节的散列。

该头字段的 ABNF[RFC2616]定义如下：

```
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 | "+" | "/"
```

注意：例如，如果客户端握手中的|Sec-WebSocket-Key|头字段的值是“ dGhlIHNhbXBsZSBub25jZQ== ”，服务器将追加字符串“ 258EAFA5-E914-47DA-95CA-C5AB0DC85B11 ”为字符串 dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CA-C5AB0DC85B11 形式。

服务器将采取 SHA-1 散列这个字符串，并给出值 0xb3 0x7a 0x4f 0x2c 0xc0 0x62 0x4f 0x16 0x90 0xf6 0x46 0x06 0xcf 0x38 0x59 0x45 0xb2 0xbe 0xc4 0xea 。这个值接着 base64 编码，给出值“s3pPLMBiTxaQ9kYGzzhZRbK+xOo=”，这将在|Sec-WebSocket-Accept|头字段中被返回。

5. 可选的，一个|Sec-WebSocket-Protocol|头字段，带有一个定义在 4.2.2 节第 4 步的值/subprotocol/。

6. 可选的，一个|Sec-WebSocket-Extensions|头字段，带有一个定义在 4.2.2 节第 4 步的值/ extensions /。如果使用多个扩展，那么可以把所有都列在一个 |Sec-WebSocket-Extensions| 头字段中或者分配到 |Sec-WebSocket-Extensions|头字段的多个实例之间。

这就完成了服务器握手。如果服务器完整这些步骤且没有中断 WebSocket 握手，服务器认为 WebSocket 连接已建立且 WebSocket 连接处于 OPEN 状态。此时，服务器可以开始发送（和接收）数据了。

4.3.为握手中使用的新的头字段整理的 ABNF

本节使用的 ABNF 语法/规范来自[RFC2616]第 2.1 节，包括“隐式*LWS 规则”。

注意，以下 ABNF 约定用于本节中。一些规则的名相当于相应的头字段名字。这样的规则表示相应的头字段值，例如 Sec-WebSocket-Key ABNF 规则描述了 |Sec-WebSocket-Key| 头字段值的语法。在名字中带有“-Client”后缀的 ABNF 规则仅用在由客户端到服务器发送请求的情况；在名字中带有“-Server”后缀的 ABNF 规则仅用在由服务器到客户端发生响应的情况。例如，ABNF 规则 Sec-WebSocket-Protocol-Client 描述了客户端到服务器发送的 |Sec-WebSocket-Protocol|头字段值的语法。

以下新的头字段可以在从客户端到服务器握手期间被发送：

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

以下新的头字段可以在服务器到客户端握手期间被发送：

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.支持多个版本的 WebSocket 协议

本节提供了在客户端和服务器中支持多个版本的 WebSocket 协议的一些指导。

使用 WebSocket 版本通知能力（|Sec-WebSocket-Version|头字段），客户端可以初始请求它选择的 WebSocket 协议的版本（这并不一定必须是客户端支持的最新的）。如果服务器支持请求的版本且握手消息是本来有效的，服务器将接受该版本。如果服务器不支持请求的版本，它必须以一个包含所有它将使用的版本的|Sec-WebSocket-Version|头字段（或多个|Sec-WebSocket-Version|头字段）来响应。此时，如果客户端支持一个通知的版本，它可以使用新的版本值重做 WebSocket 握手。

以下示例演示了上述的版本协商。

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

...

Sec-WebSocket-Version: 25

服务器的响应可能看起来像如下：

HTTP/1.1 400 Bad Request

...

Sec-WebSocket-Version: 13, 8, 7

注意服务器最后的响应可能也看起来像：

HTTP/1.1 400 Bad Request

...

Sec-WebSocket-Version: 13

Sec-WebSocket-Version: 8, 7

客户端现在可以重做符合版本 13 的握手：

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

...

Sec-WebSocket-Version: 13

5.数据帧

5.1 概述

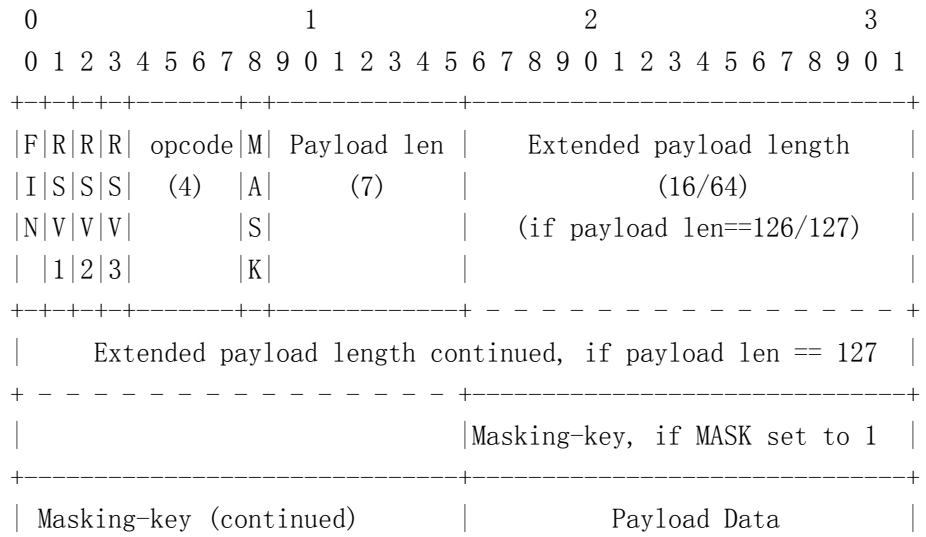
在 WebSocket 协议中，数据使用帧序列来传输。为避免混淆网络中间件（例如拦截代理）和出于安全原因，第 10.3 节进一步讨论，客户端必须掩码（mask）它发送到服务器的所有帧（更多详细信息请参见 5.3 节）。（注意不管 WebSocket 协议是否运行在 TLS 至上，掩码都要做。） 当收到一个没有掩码的帧时，服务器必须关闭连接。在这种情况下，服务器可能发送一个定义在 7.4.1 节的状态码 1002（协议错误）的 Close 帧。服务器必须不掩码发送到客户端的所有帧。如果客户端检测到掩码的帧，它必须关闭连接。在这种情况下，它可能使用定义在 7.4.1 节的状态码 1002（协议错误）。（这些规则可能在未来规范中放宽。）

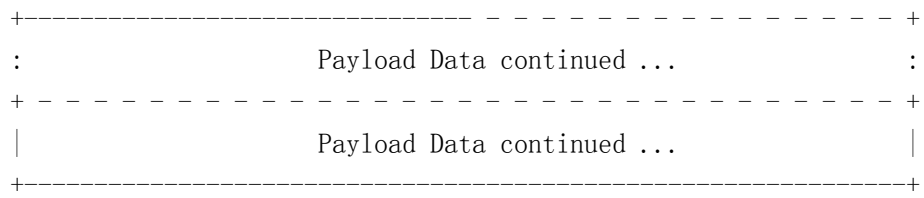
基本帧协议定义了带有操作码（opcode）的帧类型、负载长度、和用于“扩展数据”与“应用数据”及它们一起定义的“负载数据”的指定位置。某些字节和操作吗保留用于未来协议的扩展。

一个数据帧可以被客户端或者服务器在打开阶段握手完成之后和端点发送 Close 帧之前的任何时候传输（5.5.1 节）。

5.2 基本帧协议

用于数据传输部分的报文格式是通过本节中详细描述的 ABNF 来描述。（注意，不像本文档的其他章节，本节中的 ABNF 是在位（bit）组上操作。每一个位组的长度在注释中指出。在编码报文时，最重要的位是在 ABNF 的最左边。）下图给出了帧的高层次概述。在下图和在本节后边指定的 ABNF 之间冲突的，这个图表是权威的。





FIN：1 bit
指示这个是消息的最后片段。第一个片段可能也是最后的片段。

RSV1, RSV2, RSV3:　每个 1 bit

必须是 0，除非一个扩展协商为非零值定义含义。如果收到一个非零值且没有协商的扩展定义这个非零值的含义，接收端点必须_失败 WebSokcket 连接_。

Opcode:　4 bits

定义了“负载数据”的解释。如果收到一个未知的操作码，接收端点必须_失败 WebSocket 连接_。定义了以下值。

* %x0 代表一个继续帧
* %x1 代表一个文本帧
* %x2 代表一个二进制帧
* %x3-7 保留用于未来的非控制帧
* %x8 代表连接关闭
* %x9 代表 ping
* %xA 代表 pong
* %xB-F 保留用于未来的控制帧

Mask:　1 bit

定义是否“负载数据”是掩码的。如果设置为 1，一个掩码键出现在 masking-key，且这个是用于根据 5.3 节解掩码(unmask)“负载数据”。从客户端发送到服务器的所有帧有这个位设置为 1。

Payload length:　7 bits, 7+16 bits, 或者 7+64 bits

“负载数据”的长度，以字节为单位：如果 0-125，这是负载长度。如果 126，之后的两字节解释为一个 16 位的无符号整数是负载长度。如果 127，之后的 8

字节解释为一个 64 位的无符号整数（最高有效位必须是 0）是负载长度。多字节长度数量以网络字节顺序来表示。注意，在所有情况下，最小数量的字节必须用于编码长度，例如，一个 124 字节长的字符串的长度不能被编码为序列 126，0，124。负载长度是“扩展数据”长度+“应用数据”长度。“扩展数据”长度可能是零，在这种情况下，负载长度是“应用数据”长度。

Masking-key: 0 or 4 bytes

客户端发送到服务器的所有帧通过一个包含在帧中的 32 位值来掩码。如果 mask 位设置为 1，则该字段存在，如果 mask 位设置为 0，则该字段缺失。详细信息请参见 5.3 节 客户端到服务器掩码。

Payload data: (x+y) bytes

“负载数据”定义为“扩展数据”连接“应用数据”。

Extension data: x bytes

“扩展数据”是 0 字节除非已经协商了一个扩展。任何扩展必须指定“扩展数据”的长度，或长度是如何计算的，以及扩展如何使用必须在打开阶段握手期间协商。如果存在，“扩展数据”包含在总负载长度中。

Application data: y bytes

任意的“应用数据”，占用“扩展数据”之后帧的剩余部分。“应用数据”的长度等于负载长度减去“扩展数据”长度。

基本帧协议是由以下 ABNF[RFC5234]正式定义。重要的是要注意这个数据是二进制表示的，而不是 ASCII 字符。因此，一个 1 位长度的字段取值为%x0 / %x1 是表示为单个位，其值为 0 或 1，不是以 ASCII 编码代表字符“0”或“1”的完整的字节（8 位位组）。4 位长度的字段值介于%0-F 之间，是通过 4 位表示的，不是通过 ASCII 字符或这些值的完整字节（8 位位组）。[RFC5234]没有指定字符编码：“规则解析为最终值的字符串，有时候被称为字符。在 ABNF 中，一个字符仅仅是一个非负整数。在某些上下文中，一个值到一个字符集的特定映射（编码）将被指定。” 在这里，指定的编码是二进制编码，每一个最终值是编码到指定数量的比特中，每个字段是不同的。

```
ws-frame                = frame-fin           ; 1 位长度
                          frame-rsv1          ; 1 位长度
                          frame-rsv2          ; 1 位长度
```

```
      frame-rsv3              ; 1 位长度
      frame-opcode            ; 4 位长度
      frame-masked            ; 1 位长度
      frame-payload-length    ; 或者 7、 7+16、
                              ; 或者 7+64 位长度
      [ frame-masking-key ]   ; 32 位长度
      frame-payload-data      ; n*8 位长度; n>=0

frame-fin                 = %x0 ; 这条消息后续还有更多的帧
                          / %x1 ; 这条消息的最终帧
                          ; 1 位长度

frame-rsv1                = %x0 / %x1
                          ; 1 位长度，必须是 0，除非协商其他

frame-rsv2                = %x0 / %x1
                          ; 1 位长度，必须是 0，除非协商其他

frame-rsv3                = %x0 / %x1
                          ; 1 位长度，必须是 0，除非协商其他

frame-opcode              = frame-opcode-non-control /
```

frame-opcode-control /

frame-opcode-cont

frame-opcode-cont = %x0 ; 帧继续

frame-opcode-non-control= %x1 ; 文本帧

/ %x2 ; 二进制帧

/ %x3-7

; 4 位长度，保留用于未来的非控制帧

frame-opcode-control = %x8 ; 连接关闭

/ %x9 ; ping

/ %xA ; pong

/ %xB-F ; 保留用于未来的控制帧

; 4 位长度

frame-masked = %x0

; 帧没有掩码，没有 frame-masking-key

/ %x1

; 帧被掩码，存在 frame-masking-key

; 1 位长度

```
frame-payload-length        = ( %x00-7D )
                            / ( %x7E frame-payload-length-16 )
                            / ( %x7F frame-payload-length-63 )
                            ; 分别 7, 7+16, or 7+64 位长度

frame-payload-length-16 = %x0000-FFFF ; 16 位长度

frame-payload-length-63 = %x0000000000000000-7FFFFFFFFFFFFFFF
                            ; 64 位长度

frame-masking-key           = 4( %x00-FF )
                              ; 仅当 frame-masked 是 1 时存在
                              ; 32 位长度

frame-payload-data          = (frame-masked-extension-data
                                frame-masked-application-data)
                            ; 当 frame-masked 是 1
                              / (frame-unmasked-extension-data
                                 frame-unmasked-application-data)
                            ; 当 frame-masked 是 0

frame-masked-extension-data         = *( %x00-FF )
```

; 保留用于未来扩展

; n*8 位长度，n >= 0

frame-masked-application-data = *(%x00-FF)

; n*8 位长度，n >= 0

frame-unmasked-extension-data = *(%x00-FF)

; 保留用于未来扩展

; n*8 位长度，n >= 0

frame-unmasked-application-data = *(%x00-FF)

; n*8 位长度，n >= 0

5.3.客户端到服务器掩码

一个掩码的帧必须有 5.2 节定义的字段 frame-masked 设置为 1。

掩码键完全包含在帧中，5.2 节定义的 frame-masking-key。它用于掩码定义在相同章节的 frame-payload-data 中的“负载数据”，其包含“扩展数据”和“应用数据”。

掩码键是由客户端随机选择的 32 位值。当准备一个掩码的帧时，客户端必须从允许的 32 位值集合中选择一个新的掩码键。掩码键需要是不可预测的；因此，掩码键必须来自一个强大的熵源，且用于给定帧的掩码键必须不容易被服务器/代理预测用于后续帧的掩码键。掩码键的不可预测性对防止恶意应用的作者选择出现在报文上的字节是必要的。RFC 4086 [RFC4086]讨论了什么需要一个用于安全敏感应用的合适的熵源。

掩码不影响“负载数据”的长度。变换掩码数据到解掩码数据，或反之亦然，以下算法被应用。相同的算法应用，不管转化的方向，例如，相同的步骤即应用到

掩码数据也应用到解掩码数据。

变换数据的八位位组 i （"transformed-octet-i"）是原始数据的八位位组 i（"original-octet-i"）异或（XOR）i 取模 4 位置的掩码键的八位位组（"masking-key-octet-j"）：

j = i MOD 4

这个算法是很好理解的，i 指原始值

transformed-octet-i = original-octet-i XOR masking-key-octet-j

负载长度，在帧中以 frame-payload-length 表示，不包括掩码键的长度。它是“负载数据”的长度，例如，跟在掩码键后边的字节数。

5.4.分片（Fragmentation）

分片的主要目的是允许当消息开始但不必缓冲该消息时发送一个未知大小的消息。如果消息不能被分片，那么端点将不得不缓冲整个消息以便在首字节发生之前统计出它的长度。对于分片，服务器或中间件可以选择一个合适大小的缓冲，当缓冲满时，写一个片段到网络。

第二个分片的用例是用于多路复用，一个逻辑通道上的一个大消息独占输出通道是不可取的，因此多路复用需要可以分割消息为更小的分段来更好的共享输出通道。（注意，多路复用扩展在本文档中没有描述）

除非另有扩展指定，帧没有语义含义。一个中间件可能合并且/或分割帧，如果客户端和服务器没有协商扩展；或如果已协商了一些扩展，但中间件理解所有协商的扩展且知道如何去合并且/或分割在这些扩展中存在的帧。这方面的一个含义是，在没有扩展情况下，发送者和接收者必须不依赖于特定帧边界的存在。

以下规则应用到分片：

- 一个没有分片的消息由单个带有 FIN 位设置（5.2 节）和一个非 0 操作码的帧组成。

- 一个分片的消息由单个带有 FIN 位清零（5.2 节）和一个非 0 操作码的帧组成，跟随零个或多个带有 FIN 位清零和操作码设置为 0 的帧，且终止于一个带有 FIN 位设置且 0 操作码的帧。一个分片的消息概念上是等价于单个大的消息，其负载是等价于按顺序串联片段的负载；然而，在存在扩展的情况下，这个可能不适用扩展定义的“扩展数据”存在的解释。例如，“扩展数据”可能仅在首个片段开始处存在且应用到随后的片段，或 “扩展数据”可以存在于仅用于到特定片段的每个片段。在没有“扩展数据”的情况下，以下例子展示了分片如何工作。

例子：对于一个作为三个片段发送的文本消息，第一个片段将有一个 0x1 操作码和一个 FIN 位清零，第二个片段将有一个 0x0 操作码和一个 FIN 位清零，且第三个片段将有 0x0 操作码和一个 FIN 位设置。

o 控制帧（参见 5.5 节）可能被注入到一个分片消息的中间。控制帧本身必须不被分割。

o 消息分片必须按发送者发送顺序交付给收件人。

o 片段中的一个消息必须不能与片段中的另一个消息交替，除非已协商了一个能解释交替的扩展。

o 一个端点必须能处理一个分片消息中间的控制帧。

o 一个发送者可以位非控制消息创建任何大小的片段。

o 客户端和服务器必须支持接收分片和非分片的消息。

o 由于控制帧不能被分片，一个中间件必须不尝试改变控制帧的分片。

o 如果使用了任何保留的位值且这些值的意思对中间件是未知的，一个中间件必须不改变一个消息的分片。

o 在一个连接上下文中，已经协商了扩展且中间件不知道协商的扩展的语义，一个中间件必须不改变任何消息的分片。同样，没有看见 WebSocket 握手（且没被通知有关它的内容）、导致一个 WebSocket 连接的一个中间件，必须不改变这个链接的任何消息的分片。

o 由于这些规则，一个消息的所有分片是相同类型，以第一个片段的操作码设置。因为控制帧不能被分片，用于一个消息中的所有分片的类型必须或者是文本、或者二进制、或者一个保留的操作码。

注意：如果控制帧不能被插入，一个 ping 延迟，例如，如果跟着一个大消息将是非常长的。因此，要求在分片消息的中间处理控制帧。

实现注意：在没有任何扩展时，一个接收者不必按顺序缓冲整个帧来处理它。例如，如果使用了一个流式 API，一个帧的一部分能被交付到应用。但是，请注意这个假设可能不适用所有未来的 WebSocket 扩展。

5.5.控制帧

控制帧由操作码确定，其中操作码最重要的位是 1。当前定义的用于控制帧的操作码包括 0x8 （Close）、0x9（Ping）、和 0xA（Pong）。操作码 0xB-0xF 保留用

于未来尚未定义的控制帧。

控制帧用于传达有关 WebSocket 的状态。控制帧可以插入到分片消息的中间。

所有控制帧必须有一个 125 字节的负载长度或更少， 必须不被分段。

5.5.1.Close

关闭（Close）帧包含 0x8 操作码。

关闭帧可以包含内容体（“帧的“应用数据”部分）指示一个关闭的原因，例如端点关闭了、端点收到的帧太大、或端点收到的帧不符合端点期望的格式。如果有内容体，内容体的头两个字节必须是 2 字节的无符号整数（按网络字节顺序）代表一个在 7.4 节的/code/值定义的状态码。跟着 2 字节的整数，内容体可以包含 UTF-8 编码的/reason/值，本规范没有定义它的解释。数据不必是人类可读的但可能对调试或传递打开连接的脚本相关的信息是有用的。由于数据不保证人类可读，客户端必须不把它显示给最终用户。

客户端发送到服务器的关闭帧必须根据 5.3 节被掩码。

在应用发送关闭帧之后，必须不发送任何更多的数据帧。

如果一个端点接收到一个关闭帧且先前没有发送一个关闭帧，端点必须在响应中发送一个关闭帧。（当在响应中发生关闭帧时，端点通常回送它接收到的状态码）它应该根据实际情况尽快这样做。端点可以延迟发送关闭帧知道它当前消息发送了（例如，如果一个分片消息的大多数已经发送了，端点可以发送剩余的片段在发送一个关闭帧之前）。但是，不保证一个已经发送关闭帧的端点将继续处理数据。

发送并接收一个关闭消息后，一个端点认为 WebSocket 连接关闭了且必须关闭底层的 TCP 连接。服务器必须立即关闭底层 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一个关闭消息之后的任何时候关闭连接，例如，如果它没有在一个合理的时间周期内接收到服务器的 TCP 关闭。

如果客户端和服务器同时都发送了一个关闭消息，两个端点都将发送和接收一个关闭消息且应该认为 WebSocket 连接关闭了并关闭底层 TCP 连接。

5.5.2. Ping

Ping 帧包含 0x9 操作码。

Ping 帧可以包含“应用数据”。

当收到一个 Ping 帧时，一个端点必须在响应中发送一个 Pong 帧，除非它早已接收到一个关闭帧。它应该尽可能快地以 Pong 帧响应。Pong 帧在 5.5.3 节讨论。

一个端点可以在连接建立之后并在连接关闭之前的任何时候发送一个 Ping 帧。

注意：一个 Ping 即可以充当一个 keepalive，也可以作为验证远程端点仍可响应的手段。

5.5.3. Pong

Pong 帧包含一个 0xA 操作码。

5.5.2 节详细说明了应用 Ping 和 Pong 帧的要求。

一个 Pong 帧在响应中发送到一个 Ping 帧必须有在将回复的 Ping 帧的消息内容体中发现的相同的“应用数据”。

如果端点接收到一个 Ping 帧且尚未在响应中发送 Pong 帧到之前的 Ping 帧，端点可以选择仅为最近处理的 Ping 帧发送一个 Pong 帧。

一个 Pong 帧可以未经请求的发送。这个充当单向的心跳（heartbeat）。到未经请求的 Pong 帧的一个响应是不期望的。

5.6.数据帧

数据帧（例如，非控制帧）由操作码最高位是 0 的操作码标识。当前为数据帧定义的操作码包括 0x1(文本)、0x2（二进制）。操作码 0x3-0x7 保留用于未来尚未定义的非控制帧。
数据帧携带应用层和/或扩展层数据。操作码决定了数据的解释：

Text

“负载数据”是编码为 UTF-8 的文本数据。注意，一个特定的文本帧可能包括部分 UTF-8 序列；不管怎么样，整个消息必须包含有效的 UTF-8。重新组装的消息中的无效的 UTF-8 的处理描述在 8.1 节。

Binary

“负载数据”是随意的二进制数据，其解释仅仅是在应用层。

5.7.示例

- 未掩码文件消息的单个帧
  - 0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f (包含 "Hello")
- 掩码的文本消息的单个帧
  - 0x81 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58 (包含 "Hello")
- 一个分片的未掩码的文本消息
  - 0x01 0x03 0x48 0x65 0x6c (包含 "Hel")
  - 0x80 0x02 0x6c 0x6f (包含 "lo")
- 未掩码的 Ping 请求和掩码的 Ping 响应
  - 0x89 0x05 0x48 0x65 0x6c 0x6c 0x6f

    (包含内容体"Hello"、但内容体的内容是随意的)
  - 0x8a 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58

    (包含内容体"Hello"、匹配 ping 的内容体)
- 单个未掩码帧中的 256 字节的二进制消息
  - 0x82 0x7E 0x0100 [256 字节的二进制数据]
- 单个未掩码帧中的 64KB 的二进制消息
  - 0x82 0x7F 0x0000000000010000 [65536 字节的二进制数据]

5.8.可扩展性

协议被设计为允许扩展，这将增加功能到基础协议。端点的一个连接必须在打开阶段握手期间协商使用的任何扩展。本规范提供了用于扩展的操作码 0x3 到 0x7 和 0xB 到 0xF、“扩展数据”字段、和帧-rsv1、帧 rsv2、和帧 rsv3 帧头位。9.1 节进一步讨论了扩展协商。以下是一些预期使用的扩展。这个列表是不完整的也不规范的。

- “扩展数据”可以放置在“负载数据”中的“应用数据”之前。

- 保留的位可以分配给需要的每个帧。
- 保留的操作码值能被定义。
- 如果需要更多的操作码值，保留的位可以分配给操作码字段。
- 一个保留的位或一个“扩展”操作码可以定义以从“负载数据”中分配额外的位来定义更大的操作码或更多的每帧位。

6.发送和接收数据

6.1.发送数据

为了_发送一个 WebSocket 消息_，其中包括 WebSocket 连接之上的/data/，端点必须执行以下步骤。

1. 端点必须确保 WebSocket 连接处于 OPEN 状态（比较 4.1 节和 4.2.2 节）。如果在任何时刻 WebSocket 连接的状态改变了，端点必须终止以下步骤。

2. 端点必须封装/data/到定义在 5.2 节的一个 WebSocket 帧。如果要发送的数据太大或如果在端点想要开始发生数据时数据作为一个整体不可用，端点可以按照 5.2 节的定义交替地封装数据到一系列的帧中。

3. 第一个包含数据的帧的操作码（帧-opcode）必须按照 5.2 节的定义被设置为适当的值用于接收者解释数据是文本还是二进制数据。。

4. 包含数据的最后帧的 FIN 位（帧-fin）必须按照 5.2 节的定义设置位 1。

5. 如果数据正由客户端发送，帧必须按照 5.3 节的定义被掩码。

6. 如果任何扩展（第 9 章）已经协商用于 WebSocket 连接，额外的考虑可以按照这些扩展定义来应用。

7. 已成形的帧必须在底层网络连接之上传输。

6.2.接收数据

为了接收 WebSocket 数据，端点监听底层网络连接。传入数据必须按照 5.2 节的定义解析为 WebSocket 帧。如果接收到一个控制帧（5.5 节），帧必须按照 5.5 节定义的来处理。当接收到一个数据帧（5.6 节）时，端点必须注意 5.2 节由操作码（帧-opcode）定义的数据的/type/。这个帧的“应用数据”被定义为消息的/data/。如果帧由一个未分片的消息组成（5.4 节），这是说_已经接收到一个 WebSocket 消息_，其类型为/type/且数据为/data/。如果帧是一个分片消息的一部分，随后数据帧的“应用数据”连接在一起形成/data/。当接收到由 FIN 位（帧-fin）指示的最后的片段时，这是说_已经接收到一个 WebSocket 消息_，其数据为/data/（由连续片段的“应用数据”组成）且类型为/type/（分配消息的第一个帧指出）。随后的数据帧必须被解释为属于一个新的 WebSocket 消息。

扩展（第 9 章）可以改变数据如何读的语义，尤其包括什么组成一个消息的边界。扩展，除了在负载中的“应用数据”之前添加“扩展数据”外，也可以修改“应

用数据”（例如压缩它）。

服务器必须按照 5.3 节的定义为从客户端接收到的数据帧移除掩码。

7.关闭连接

7.1.定义

7.1.1.关闭 WebSocket 连接

为_关闭 WebSocket 连接_，端点需关闭底层 TCP 连接。端点应该使用一个方法完全地关闭 TCP 连接，以及 TLS 会话，如果合适，丢弃任何可能已经接收的尾随的字节。当必要时端点可以通过任何可用的手段关闭连接，例如当受到攻击时。

底层 TCP 连接，在大多数正常情况下，应该首先被服务器关闭，所以它持有 TIME_WAIT 状态而不是客户端（因为这会防止它在 2 个报文最大生存时间（2MLS）内重新打开连接，然而当一个新的带有更高的 seq number 的 SYN 时没有对应的服务器影响 TIME_WAIT 连接被立即重新打开）。在异常情况下（例如在一个合理的时间量后没有接收到服务器的 TCP Close）客户端可以发起 TCP Close。因此，当服务器被指示_关闭 WebSocket 连接_，它应该立即发起一个 TCP Close，且当客户端被知识也这么做时，它应该等待服务器的一个 TCP Close。

例如一个如何使用 Berkeley socket 在 C 中得到完全地关闭的例子，一端会在 socket 上以 SHUT_WR 调用 shutdown()，调用 recv()直到获得一个指示那个节点也已经执行了一个有序关闭的 0 返回值，且最终在 socket 上调用 close()方法。

7.1.2.启动 WebSocket 关闭阶段握手

为了_启动 WebSocket 关闭阶段握手_，其带有一个状态码（7.4 节）/code/和一个可选的关闭原因（7.1.6 节）/reason/，一个端点必须按照 5.5.1 节的描述发送一个 Close 控制帧，其状态码设置为/code/且其关闭原因设置为/reason/。一旦一个端点已经发送并接收到一个 Close 控制帧，那个端点应该按照 7.1.1 节的描述_关闭 WebSocket 连接_。

7.1.3. WebSocket 关闭阶段握手已启动

一旦发送或接收到一个 Close 控制帧，这就是说，_WebSocket 关闭阶段握手已启动_，且 WebSocket 连接处于 CLOSING 状态。

7.1.4. WebSocket 已关闭

当底层 TCP 连接已关闭，这就是说_WebSocket 连接已关闭_且 WebSocket 连接处于 CLOSED 状态。如果 TCP 连接在 WebSocket 关闭阶段我是已经完成后被关

闭，WebSocket 连接被说成已经_完全地_关闭了。

如果 WebSocket 连接不能被建立，这就是说，_WebSocket 连接关闭了_，但不是_完全的_。

7.1.5.WebSocket 连接关闭代码

按照 5.5.1 和 7.4 节的定义，一个 Close 控制帧可以包含一个表示关闭原因的状态码。一个正关闭的 WebSocket 连接可以同时由两个端点初始化。_WebSocket 连接 Close Code_定义为包含在由实现该协议的应用接收到的第一个 Close 控制帧的状态码（7.4 节）。如果这个 Close 控制帧不包含状态码，_WebSocket 连接 Close Code_被认为是 1005。如果_WebSocket 连接已经关闭_且端点没有接收到 Close 状态码（例如可能发生在底层传输连接丢失时），_WebSocket 连接 Close Code_被认为是 1006。

注意：两个端点可以有不一致的_WebSocket 连接关闭代码_。例如，如果远程端点发送了一个 Close 帧，但本地应用还没有从它的 socket 接收缓冲区中读到包含 Close 帧的数据，且本地应用独立地决定关闭连接和发送一个 Close 帧，两个端点都将发送和接收 Close 帧且将不发送更多的 Close 帧。每一个端点将看见另一端发送的以_WebSocket 连接关闭代码_结束的状态码。例如，在两个端点独立且在大致相同的时间同时_开启 WebSocket 关闭阶段握手_的情况下，两个端点可以有不一致的_WebSocket 连接关闭代码_是可能的。

7.1.6. WebSocket 连接关闭原因

按照 5.5.1 和 7.4 节的定义，一个控 Close 控制帧可以包含一个指示关闭原因的状态码，接着是 UTF-8 编码的数据，上述数据留给断点解释且本协议没有定义。WebSocket 连接的关闭可以被任何一个端点初始化，可能同时发生。_WebSocket 连接关闭原因_由跟在包含在实现该协议的应用接收到的第一个 Close 控制帧状态码（7.4 节）后边的 UTF-8 编码的数据定义。如果 Close 控制帧中没有这样的数据，_WebSocket 连接关闭原因_是空字符串。

注意：按照 7.1.5 节指出的相同的逻辑，两个端点可以有不一致的_WebSocket 连接关闭原因_。

7.1.7.失败 WebSocket 连接

某些算法和规范要求端点_失败 WebSocket 连接_。要做到这一点，客户端必须_关闭 WebSocket 连接_，并可以以适当的方式把问题报告给用户（这将对开发人员非常有用的）。

同样的，为了做到这一点，服务器必须_关闭 WebSocket 连接_，并应该记录下问

题。

如果_已建立的 WebSocket 连接_在端点需要_失败 WebsSocket 连接_之前，端点应该在处理_关闭 WebSocket 连接_之前发送一个带有适当状态码的 Close 帧（7.4 节）。

如果端点认为另一边不太可能收到并处理关闭帧可以省略发送一个关闭帧，因为错误的性质，导致 WebSocket 连接失败摆在首要位置。端点必须在被指示为_失败 WebSocket 端点_之后不继续尝试处理来自远程端点的数据（包括响应关闭帧）。

除上边指出的或由应用层指定的（例如，使用 WebSocket API 的脚本），客户端应该关闭连接。

7.2.异常关闭

7.2.1.客户端发起的关闭

某些算法，尤其在打开阶段握手期间，需要客户端_失败 WebSocket 连接_。为了做到这一点，客户端必须按照 7.1.7 节定义的那样_失败 WebSocket 连接_。

如果在任何时候，底层的传输层连接意外丢失，客户端必须_失败 WebSocket 连接_。

除上边指出的或由应用层指定的（例如，使用 WebSocket API 的脚本），客户端应该关闭连接。

7.2.2.服务端发起的关闭

某些算法需要或推荐服务端在打开阶段握手期间_中断 WebSocket 连接_。为了做到这一点，服务端必须简单地_关闭 WebSocket 连接_（7.1.1 节）。

7.2.3.从异常关闭中恢复

异常关闭可能由任何原因引起。这样的关闭可能是一个瞬时错误导致的，在这种情况下重新连接可能导致一个好的连接和一个重新开始的正常操作。这样的关闭也可能是一个非瞬时问题的导致的，在这种情况下如果每个部署的客户端遇到异常关闭并立即且持续地的尝试重新连接，服务端可能会因为大量的客户端尝试重新连接遇到的拒绝服务攻击。这种情况的最终结果可能是服务不能及时的恢复或恢复是更加困难。

为了避免这个，当客户端遇到本节描述的异常关闭之后尝试重新连接时，应该使用某种形式的补偿。

第一个重新连接尝试应该延迟一个随机的时间量。这种随机延迟的参数的选择留给客户端决定；一个可随机选择的的值在 0 到 5 秒是一个合理的初始延迟，不过客户端可以选择不同的间隔由于其选择一个延迟长度基于实现经验和特定的应用。

第一次重新连接尝试失败，随后的重新连接尝试应该延迟递增的时间量，使用的方法如截断二进制指数退避算法。

7.3.正常连接关闭

服务端在需要时可能关闭 WebSocket 连接。客户端不能随意关闭 WebSocket 连接。在这两种情况下，端点通过如下过程_开始 WebSocket 关闭握手_初始化一个关闭（7.1.2 节）。

7.4.状态码

当关闭一个已经建立的连接（例如，当在打开阶段握手已经完成后发送一个关闭帧），端点可以表明关闭的原因。由端点解释这个原因，并且端点应该给这个原因采取动作，本规范是没有定义的。本规范定义了一组预定义的状态码，并指定哪些范围可以被扩展、框架和最终应用使用。状态码和任何相关的文本消息是关闭帧的可选的组件。

7.4.1.定义的状态码

已经过期了,参见
https://www.iana.org/assignments/websocket/websocket.xhtml#close-code-number

当发送关闭帧时端点可以使用如下预定义的状态码。

1000

1000 表示正常关闭，意思是建议的连接已经完成了。

1001

1001 表示端点“离开”（going away），例如服务器关闭或浏览器导航到其他页面。

1002

1002 表示端点因为协议错误而终止连接。

1003

1003 表示端点由于它收到了不能接收的数据类型（例如，端点仅理解文本数据，但接收到了二进制消息）而终止连接。

1004

保留。可能在将来定义其具体的含义。

1005

1005 是一个保留值，且不能由端点在关闭控制帧中设置此状态码。它被指定用在期待一个用于表示没有状态码是实际存在的状态码的应用中。

1006

1006 是一个保留值，且不能由端点在关闭控制帧中设置此状态码。它被指定用在期待一个用于表示连接异常关闭的状态码的应用中。

1007

1007 表示端点因为消息中接收到的数据是不符合消息类型而终止连接（比如，文本消息中存在非 UTF-8[RFC3629]数据）。

1008

1008 表示端点因为接收到的消息违反其策略而终止连接。这是一个当没有其他合适状态码（例如 1003 或 1009）或如果需要隐藏策略的具体细节时能被返回的通用状态码。

1009

1009 表示端点因接收到的消息对它的处理来说太大而终止连接。

1010

1010 表示端点（客户端）因为它期望服务器协商一个或多个扩展，但服务器没有在 WebSocket 握手响应消息中返回它们而终止连接。 所需要的扩展列表应该出现在关闭帧的/reason/部分。

注意，这个状态码不能被服务器端使用，因为它可以失败 WebSocket 握手。

1011

1011 表示服务器端因为遇到了一个不期望的情况使它无法满足请求而终止连接。

1015

1015 是一个保留值，且不能由端点在关闭帧中被设置为状态码。它被指定用在期待一个用于表示连接由于执行 TLS 握手失败而关闭的状态码的应用中（比如，服务器证书不能验证）。

7.4.2.保留的状态码范围

0-999

0-999 范围内的状态码不被使用。

1000-2999

1000-2999 范围内的状态码保留给本协议、其未来的修订和一个永久的和现成的公共规范中指定的扩展的定义。

3000-3999

3000-3999 范围内的状态码保留给库、框架和应用使用。这些状态码直接向 IANA 注册。本规范未定义这些状态码的解释。

4000-4999

4000-4999 范围内的状态码保留用于私有使用且因此不能被注册。这些状态码可以被在 WebSocket 应用之间的先前的协议使用。本规范未定义这些状态码的解释。

8.错误处理

8.1.处理 UTF-8 编码数据的错误

当一个端点解析字节流为 UTF-8 数据，但发现字节流实际上不是一个有效的 UTF-8 流，那么端点必须_失败 WebSocket 连接_。这条规则应用在打开握手期间和随后的数据交换期间。

9.扩展

WebSocket 客户端可以请求本规范的扩展，且 WebSocket 服务器可以接受一些或所有客户端请求的扩展。服务器不必响应不是客户端请求的任何扩展。如果扩展参数包含在客户端和服务器之间的协商中，这些参数必须按照参数应用到的扩展规范来选择。

9.1.协商扩展

客户端通过包含一个|Sec-WebSocket-Extensions|头字段请求扩展，其按照正常的 HTTP 头字段规则（参考[RFC2616], 4.2 节）并且头字段的值按照以下 ABNF 定义[RFC2616]。注意本章使用的 ABNF 语法/规则来源于[RFC2616]，包括“隐式的 *LWS 规范”。如果客户端或服务器在协商阶段接收到的值不符合下边的 ABNF，这种畸形数据的接收人必须立即_失败 WebSocket 连接_。

```
Sec-WebSocket-Extensions = extension-list

extension-list = 1#extension

extension = extension-token *( ";" extension-param )

extension-token = registered-token

registered-token = token

extension-param = token [ "=" (token | quoted-string) ]

    ;当使用引用字符串的语法变种时，引用字符串之后的值必须
```

;符合’token’ABNF

注意，像其他 HTTP 头字段，这个头字段可以跨多个行分割或组合，因此，以下是等价的：

Sec-WebSocket-Extensions: foo

Sec-WebSocket-Extensions: bar; baz=2

完全等价于

Sec-WebSocket-Extensions: foo, bar; baz=2

所有使用的 extension-token 必须是一个 egistered token（参考 11.4 节）。任何给定扩展提供的参数必须被扩展定义。注意，客户端只需提供使用任何公布的扩展，除非服务器表示它希望使用使用扩展，否则必须使用它们。

注意，扩展的顺序是重要的。在多个扩展间的相互作用可以定义在定义扩展的文档中。在没有这样定义的情况下，解释是它请求中的客户端列出的头字段表示一个它希望使用的头字段的偏好，第一个列出的选项是最优选的。服务器在响应中列出的扩展表示扩展是实际正在用于连接的扩展。扩展应该修改数据和/或组帧，数据的操作顺序应该假定是与打开阶段握手期间服务器响应中列出的扩展顺序是一样的。

例如，如果有两个扩展“foo”和“bar”，且如果服务器发送的头字段|Sec-WebSocket-Extensions|有值“foo”、“bar”，那么数据上的操作将变为bar(foo(data))，是更改数据本身（如压缩）或更改可能“堆叠（stack）”的组帧。

可接受的扩展头字段（注意：为了可读性，将折叠较长行）的非规范化例子：

Sec-WebSocket-Extensions: deflate-stream

Sec-WebSocket-Extensions: mux; max-channels=4; flow-control,

deflate-stream

Sec-WebSocket-Extensions: private-extension

服务器通过包含一个容纳了一个或多个扩展的客户端请求的|Sec-WebSocket-Extensions|头字段来接受一个或多个扩展。所有扩展参数的解释，和什么构成一个有效的到客户请求的参数集的服务器响应，将由各个扩展定义。

9.2.已知扩展

扩展提供了一种机制来实现选择性加入的附加协议特性。本文档没有定义任何扩展，但实现可以使用单独定义的扩展。

10.安全注意事项

本章描述了一些适用于 WebSocket 协议的安全注意事项。具体的安全注意事项在本章的字章节描述。

10.1.非浏览器客户端

WebSocket 协议防止恶意的 JavaScript 运行在一个受信任的应用内部，比如 web 浏览器，例如，通过检查头字段|Origin|（见下文）。更多细节请参考 1.6 节 。在一个更强大的客户端的情况下，这样的假设不成立。

虽然该协议的目的是被 web 页面中的脚本使用，它也可以被主机直接使用。这样的主机按照它们自己的行为行事，因此可以发送伪造的|Origin|头字段，骗过服务器。因此服务器应该小心，假设它们是直接与来自已知源的脚本通信，且必须考虑到它们可能以非预期方式访问。尤其，服务器不应该相信任何输入是有效的。

例如：如果服务器使用输入作为 SQL 查询的一部分，所有输入文本在传输到 SQL 服务器之前应该被转义，以免服务器受到 SQL 注入攻击。

10.2. Origin 注意事项

服务器不打算处理来自任意 web 页面的输入，但仅限于网站应该验证|Origin|头是一个它们盼望的源。如果源指示是服务器不可接受的，那么它应该以一个包含 HTTP 403 Forbidden 的状态码的回复响应 WebSocket 握手。

当不受信任方通常是一个执行在受信任的客户端上下文中的 JavaScript 应用的作者时，|Origin|头字段可以保护攻击的情况。客户端本身可以与服务器联系，并通过| Origin|头字段机制，决定是否提供 JavaScript 应用的这些通信权限。目的不是为了阻止非浏览器建立连接，而是确保受信的浏览器在潜在的恶意 JavaScript 控制下不能伪造 WebSocket 握手。

10.3.攻击基础设施（掩码）

除了端点是 WebSocket 攻击的目标之外，web 基础设施的其他部分，如代理，也

可能是攻击的对象。在本协议正在开发时，进行了一个实验旨在演示一类代理上的攻击，其导致部署在野的缓存代理中毒[讨论]。一般的攻击形式是在“攻击者”的控制下建立一个到服务器的连接，执行类似于 WebSocket 协议建立连接的 HTTP 连接上的 UPGRADE，且随后在已经 UPGRADE 的连接上发送数据，看起来像一个 GET 请求一个特定的已知资源（在一次攻击中，很可能会像广泛部署的用于跟踪点击或一个广告服务网络资源的脚本）。远程服务器响应的东西看起来像是到伪造的 GET 请求的一个响应，且这个响应将被一个非零百分比的部署的中间件缓存，因此使缓存中毒了。这种攻击的静效应将是如果能说服用户去访问攻击者控制的网站，攻击者可能使用户和其他晚于相同缓存的用户的缓存中毒且在其他源运行恶意脚本，影响网络安全模型。

为避免这种部署的中间件上的攻击，前置不兼容 HTTP 的和帧一起的应用提供的数据是不够的，因为无法详尽地发现和测试每一个不符合中间件的不跳过这样的非 HTTP 帧和错误地假装帧负载。

因此，采用的防御是掩码所有从客户端到服务器端发送的数据，使远程脚本（攻击者）无法控制数据如何在电线上发送，从而无法构造一个可能被中间件误解释的作为一个 HTTP 请求的消息。

客户端必须为每一帧选择一个新的掩码密钥，使用一个不能被提供数据的终端应用预测。例如，每次掩码可以从一个强加密的随机数生成器获取。如果使用相同的密钥或存在一个可预测的模式用于选择下一个密钥，当掩码后，攻击者可以发送一个消息，可能作为一个 HTTP 请求出现（通过交换消息，攻击者希望观察线上并用下一个将被使用的掩码密钥掩码它，当客户端应用它时掩码密钥将有效的解码数据）。

一旦从客户端传输一个帧已经开始，帧的负载（应用提供的数据）必须不能被应用修改也是必要的。

否则，攻击者可能发送一个已知初始数据（如都是 0）的长帧，一收到数据的第一部分后就开始计算使用的掩码密钥，当掩码后，接着修改尚未发送的作为一个请求出现的帧中的数据（这本质上是和前面段落描述的使用一个已知的或可预测的掩码密钥是相同的问题）。

如果需要发送额外的数据或要发送的数据以某种方式修改了，新的或修改了的数据必须在一个新的帧中发送，那么需要一个新的掩码密钥。总之，一旦开始传输一个帧，对于远程脚本（应用）来说，内容必须不能是可修改的。

威胁模型用来保护客户端发送的在一个请求出现的数据。因此，需要掩码的信道是从客户端到服务器的数据。服务器到客户端的数据可以作出看起来像一个响应，但为了完成这个请求，客户端也必须有能力去伪造一个请求。因此，没必要在两个方向上掩码数据（从客户端到服务器的数据没有掩码）。

尽管掩码提供了保护，对于客户端和服务器没有掩码的这种类型的中毒攻击，非

兼容 HTTP 代理将依然是脆弱的。

10.4.实现特定限制

实现已经实现— 和/或特定平台的有关帧大小或总消息大小的限制，从多个帧重新组装后，必须保证它们自己不超过这些限制。（例如，一个恶意终端无论是通过单一的大帧（例如，2**60 大小）还是通过发送一个长流的分片消息的一部分的小帧，可以设法耗尽它的对等体端点（Peer，即要攻击的那一方）的内存或安装一个拒绝服务攻击）。这样的实现应该对帧大小和和从多个帧重组后的总消息大小加以限制。

10.5.WebSocket 客户端验证

本规范没有规定任何特定的方式在 WebSocket 握手期间服务器可以验证客户端。WebSocket 服务器可以使用任何客户端对普通 HTTP 服务器可用的验证机制，如 Cookie，HTTP 验证，或者 TLS 验证。

10.6.连接的保密性和完整性

连接的保密性和完整性是通过运行在 TLS（wss URI）上 WebSocket 协议提供的。WebSocket 实现必须支持 TLS 并应该在与它们的对等端点通信时使用它。

对于使用 TLS 的连接，TLS 提供的受益量在很大程度上取决于在 TLS 握手期间协商的算法强度。例如，一些 TLS 加密机制不提供连接的保密性。为了实现合理级别的保护，客户端应该仅适用强 TLS 算法。“Web 安全上下文：用户接口指南”[W3C.REC-wsc-ui-20100812]讨论了什么构成强 TLS 算法。[RFC5246]的 附录 A.5和 附录 D.3中提供了额外的指导。

10.7.处理无效数据

传入的数据必须始终由客户端和服务器验证。如果，在任何时候，一个端点不理解它的数据或违反了一些端点确定的安全输入标准，或当端点看到一个打开阶段握手没有符合它盼望的值（例如，在客户端请求中不正确的路径或源），端点可以终止 TCP 连接。如果在 WebSocket 握手成功后接收到了无效数据，端点应该在进行_关闭 WebSocket 连接_之前发送一个带有适当状态码（7.4 节 ）的关闭帧。使用一个带有适当状态码的关闭帧能帮助诊断问题。如果在 WebSocket 握手期间发送了无效的数据，服务器应该返回一个适当的 HTTP[RFC2616]状态码。使用错误的编码发送文本数据是通常出现的一类安全问题。本协议规定一个 Text 数据类型（而不是 Binary 或其他类型）的消息包含 UTF-8 编码的数据。虽然仍指定了长度，且实现本协议的应用应该使用长度来决定帧从哪真正结束，但以一个不当的编码发送数据仍可能打破建立在本协议之上的应用的假设，导致从误解释

数据到丢失数据或潜在的安全漏洞。

10.8.使用 SHA-1 的 WebSocket 握手

本文档中描述的 WebSocket 握手不依赖于任何 SHA-1 安全特性，例如抗碰撞性或抗第二前像攻击（如同[RFC4270]中的描述）。

11. IANA 考虑

11.1.注册新的 URI 模式

11.1.1.注册“ws“模式

一个|ws| URI 标识一个 WebSocket 服务器和资源名称。

URI 模式名称

ws

状态

永久的

URI 模式语法

使用 ABNF[RFC5234]语法和 URI 规范[RFC3986]的 ABNF 终结符：

"ws:" "//" authority path-abempty ["?" query]

<path-abempty>和<query> [RFC3986]组件形成的资源名发生给服务器来确定服务期望的类型。其他组件的含义描述在[RFC3986]。

URI 模式语义

这个模式的作用仅是使用 WebSocket 协议打开一个连接。

编码考虑

上边定义的语法不包括 host 组件中的字符，必须按照[RFC3987]从 Unicode 转换为 ASCII 或其替换。为了模式标准化的目的，国际化域名（IDN）形式的

host 组件和它们转换的域名代码（Punycode）被认为是等价的（参考[RFC3987] 5.3.3 节）。

上边定义的语法不包括其他组件中的字符，必须按照定义在 URI[RFC3986]和国际化资源标识符（IRI）[RFC3987]规范从 Unicode 编码转换为 ASCII，通过首先编码字符为 UTF-8，接着使用它们百分数编码的形式替换相应的字节。

应用/协议使用这个 URI 模式命名

WebSokcet 协议

互操作性考虑

使用 WebSocket 需要使用 HTTP 版本 1.1 或更高。

安全考虑

参考“安全考虑”章节。

联系方式

HYBI WG <hybi@ietf.org>

作者/变更管理员

IETF <iesg@ietf.org>

参考资源

RFC 6455

11.1.2.注册”wss“模式

一个|wss| URI 标识一个 WebSocket 服务器和资源名称，并表明在受 TLS 保护的连接之上通信（包括标准的 TLS 的好处，比如数据保密性和完整性和端点认证）。

URI 模式名称

wss

状态

永久的

URI 模式语法

使用 ABNF[RFC5234]语法和 URI 规范[RFC3986]的 ABNF 终结符：

"wss:" "//" authority path-abempty ["?" query]

<path-abempty>和<query> [RFC3986]组件形成的资源名发生给服务器来确定服务期望的类型。其他组件的含义描述在[RFC3986]。

URI 模式语义

这个模式的作用仅是使用 WebSocket 协议打开一个使用 TLS 的连接。

编码考虑

上边定义的语法不包括 host 组件中的字符，必须按照[RFC3987]从 Unicode 转换为 ASCII 或其替换。为了模式标准化的目的，国际化域名（IDN）形式的 host 组件和它们转换的域名代码（Punycode）被认为是等价的（参考[RFC3987] 5.3.3 节）。

上边定义的语法不包括其他组件中的字符，必须按照定义在 URI[RFC3986]和国际化资源标识符（IRI）[RFC3987]规范从 Unicode 编码转换为 ASCII，通过首先编码字符为 UTF-8，接着使用它们百分数编码的形式替换相应的字节。

应用/协议使用这个 URI 模式命名

TLS 之上的 WebSokcet 协议

互操作性考虑

使用 WebSocket 需要使用 HTTP 版本 1.1 或更高。

安全考虑

参考“安全考虑”章节。

联系方式

HYBI WG <hybi@ietf.org>

作者/变更管理员

IETF <iesg@ietf.org>

参考资源

RFC 6455

11.2.注册”WebSocket“ HTTP Upgrade 关键字

本节按照 RFC2817[RFC2817]定义了在 HTTP Upgrade 符号注册中心中注册一个关键字。

符号名称

WebSocket

作者/变更管理员

IETF <iesg@ietf.org>

参考资源

RFC 6455

11.3.注册新的 HTTP 头字段

11.3.1. Sec-WebSocket-Key

本节描述了在永久消息头字段命名注册中心[RFC3864]中注册一个头字段。

头字段名

Sec-WebSocket-Key

适用协议

http

状态

标准的

作者/变更管理员

IETF

参考资源

RFC 6455

相关信息

该头字段仅用于 WebSocket 打开阶段握手。

|Sec-WebSocket-Key| 头字段用于 WebSocket 打开阶段握手。它从客户端发送到服务器，提供部分信息用于服务器检验它收到了一个有效的 WebSocket 握手。这有助于确保服务器不接收正被滥用来发送数据给毫不知情的 WebSocket 服务器的非 WebSocket 客户端的连接（例如 HTTP 客户端）。

|Sec-WebSocket-Key| 头字段在一个 HTTP 请求中不能出现多于一个。

11.3.2. Sec-WebSocket-Extensions

本节描述了在永久消息头字段命名注册中心[RFC3864]中注册一个头字段。

头字段名

Sec-WebSocket-Extensions

适用协议

http

状态

标准的

作者/变更管理员

IETF

参考资源

RFC 6455

相关信息

该头字段仅用于 WebSocket 打开阶段握手。

|Sec-WebSocket-Extensions|头字段用于 WebSocket 打开阶段握手。它最初是从客户端发送到服务器，随后从服务器端发送到客户端，用来达成在整个连接阶段的一组协议级扩展。

|Sec-WebSocket-Extensions|头字段在 HTTP 请求中可以出现多次（逻辑上等价于单个|Sec-WebSocket-Extensions|头字段包含所有值）。

但是，|Sec-WebSocket-Extensions|头字段在一个 HTTP 响应中必须不出现多于一次。

11.3.3. Sec-WebSocket-Accept

本节描述了在永久消息头字段命名注册中心[RFC3864]中注册一个头字段。

头字段名

Sec-WebSocket-Accept

适用协议

http

状态

标准的

作者/变更管理员

IETF

参考资源

RFC 6455

规范文档

RFC 6455

相关信息

该头字段仅用于 WebSocket 打开阶段握手。

| Sec-WebSocket-Accept|头字段用于 WebSocket 打开阶段握手。它从服务器发送到客户端来确定服务器愿意启动 WebSocket 连接。

|Sec-WebSocket-Accept| 头在一个 HTTP 响应中必须不出现多于一次。

11.3.4. Sec-WebSocket-Protocol

本节描述了在永久消息头字段命名注册中心[RFC3864]中注册一个头字段。

头字段名

Sec-WebSocket-Protocol

适用协议

http

状态

标准的

作者/变更管理员

IETF

参考资源

RFC 6455

规范文档

RFC 6455

相关信息

该头字段仅用于 WebSocket 打开阶段握手。

|Sec-WebSocket-Protocol|头字段用于 WebSocket 打开阶段握手。它从客户端发送到服务器端，并从服务器端发回到客户端来确定连接的子协议。这使脚本可以选择一个子协议和确定服务器同意服务子协议。

|Sec-WebSocket-Protocol|头字段在一个 HTTP 请求中可以出现多次（逻辑上等价于单个|Sec-WebSocket-Protocol|头字段包含所有值）。

但是，|Sec-WebSocket-Protocol|头字段在一个 HTTP 响应中必须不出现多于一次。

11.3.5.Sec-WebSocket-Version

本节描述了在永久消息头字段命名注册中心[RFC3864]中注册一个头字段。

头字段名

Sec-WebSocket-Version

适用协议

http

状态

标准的

作者/变更管理员

IETF

参考资源

RFC 6455

规范文档

RFC 6455

相关信息

该头字段仅用于 WebSocket 打开阶段握手。

|Sec-WebSocket- Version |头字段用于 WebSocket 打开阶段握手。它从客户端发送到服务器端来指定连接的协议版本。这能使服务器正确解释打开阶段握手和发送数据的随后数据，如果服务器不能以安全的方式解释数据则关闭连接。当从客户端接收到不匹配服务器端理解的版本时，WebSocket 握手错误，|Sec-WebSocket-Version|头字段也从服务器端发送到客户端。在这种情况下，头字段包括服务器端支持的协议版本。

注意，如果没有期望更高版本号，必然是向下兼容低版本号。

|Sec-WebSocket-Version|头字段在一个 HTTP 响应中可以出现多次（逻辑上等价于单个|Sec-WebSocket-Version|透过自动包含所有值）。

但是，|Sec-WebSocket-Version|头字段在 HTTP 请求中必须不出现多于一次。

11.4.WebSocket 扩展名注册

本规范依据 RFC5226[RFC5226]陈述的原则，创建了一个新的 IANA 注册用于与 WebSocket 协议一起使用的 WebSocket 扩展名。

作为本注册的一部分，IANA 维护以下信息：

扩展标识符

扩展标识符，将被用在注册到本规范 11.3.2 节的|Sec-WebSocket-Extensions|头字段。其值必须符合定义在本规范 9.1 节的扩展-符号要求。

扩展通用名称

扩展名称，通常称为扩展。

扩展定义

在扩展用于的 WebSocket 协议中定义了文档参考。

已知的不兼容扩展

与此扩展是不兼容的一个扩展标识符列表。

WebSocket 扩展名受制于“先来先服务”的 IANA 注册策略 [RFC5226]。

在此注册中心没有初始值。

11.5.WebSocket 子协议名注册

本规范依据 RFC5226[RFC5226]陈述的原则，创建了一个新的 IANA 注册用于与 WebSocket 协议一起使用的 WebSocket 子协议名。

作为本注册的一部分，IANA 维护以下信息：

子协议标识符

子协议标识符，将被用在注册到本规范 11.3.4 节的|Sec-WebSocket-Protocol|头字段。其值必须符合定义在本规范 4.1 节给出的第 10 条的符号要求——也就是，其值必须是 RFC 2616 [RFC2616].定义的一个符号。

子协议通用名称

子协议名称，通常成为子协议。

子协议定义

在子协议用于的 WebSocket 协议中定义了文档参考。

WebSocket 子协议名受制于“先来先服务”的 IANA 注册策略 [RFC5226]。

11.6.WebSocket 版本号注册

本规范依据 RFC5226[RFC5226]陈述的原则，创建了一个新的 IANA 注册用于与 WebSocket 协议一起使用的 WebSocket 版本号。

作为本注册的一部分，IANA 维护以下信息：

版本号

用于|Sec-WebSocket-Version|的版本号指定在本规范 4.1 节。其值必须是一个在 0 到 255（包括）之间的非负整数。

参考

RFC 请求一个新的版本号或带版本号的草案名称（见下文）。

状态

“临时的”或“标准的”。参考下面的说明。

一个版本号被指定为“临时的”或“标准的”。

“标准的”版本号是记录在一个 RFC 中并用来识别一个主要的、稳定的 WebSocket 协议版本，例如本 RFC 定义的版本。“标准的”版本号受制于“IETF 评审”IANA 注册策略 [RFC5226]。

“Interim”的版本号记录在一个 Internet 草案中用并用于帮助实现者识别和与部署的 WebSocket 版本互操作，例如在公布这个 RFC 之前指定的版本。“临时的”版本号受制于“专家评审”IANA 注册策略 [RFC5226]，HYBI 工作组主席（或，

如果工作组关闭了，IETF 应用区域的区域董事）将是初始的指定专家。

IANA 已经添加如下初始值到注册中心：

Version Number	Reference	Status
0	draft-ietf-hybi-thewebsocketprotocol-00	Interim
1	draft-ietf-hybi-thewebsocketprotocol-01	Interim
2	draft-ietf-hybi-thewebsocketprotocol-02	Interim
3	draft-ietf-hybi-thewebsocketprotocol-03	Interim
4	draft-ietf-hybi-thewebsocketprotocol-04	Interim
5	draft-ietf-hybi-thewebsocketprotocol-05	Interim
6	draft-ietf-hybi-thewebsocketprotocol-06	Interim
7	draft-ietf-hybi-thewebsocketprotocol-07	Interim
8	draft-ietf-hybi-thewebsocketprotocol-08	Interim
9	Reserved	
10	Reserved	
11	Reserved	
12	Reserved	
13	RFC 6455	Standard

11.7.WebSocket 关闭代码注册

本规范依据 RFC5226[RFC5226]陈述的原则，创建了一个新的 IANA 注册用于 WebSocket 关闭代码。

作为本注册的一部分，IANA 维护以下信息：

状态码

状态码表示一个按照本文档 7.4 节 的 WebSocket 连接关闭的原因。状态是一个在 1000 到 4999（包括）之间的一个整数数字。

含义

状态码的含义。每一个状态码都必须有唯一的含义。

联系方式

保留状态代码实体的联系方式。

参考

稳定的文档要求状态码并定义它们的含义。在 1000-2999 范围内的状态码是必须的且推荐的状态码在 3000-3999 范围内。

WebSocket 关闭代码根据它们的范围受不同的注册要求。本协议请求使用的状态码和其后续版本或扩展受制于“标准功能”、“规定要求”（这意味着“指定专家”）或“IESG 审查”IANA 注册策略中的任何一个，且应该允许在 1000-2999 范围内。库、框架和应用请求使用的状态码受制于“先来先服务”IANA 注册策略且应该允许在 3000-3999 范围内。4000-4999 范围的状态码被指定用于私有使用。请求应该指出他们要求的状态码是用于 WebSocket 协议（或未来版本的协议）、扩展，或库/框架/应用。

IANA 已经添加如下初始值到注册中心：

Status Code	Meaning	Contact	Reference
1000	Normal Closure	hybi@ietf.org	RFC 6455
1001	Going Away	hybi@ietf.org	RFC 6455
1002	Protocol error	hybi@ietf.org	RFC 6455
1003	Unsupported Data	hybi@ietf.org	RFC 6455
1004	---Reserved----	hybi@ietf.org	RFC 6455
1005	No Status Rcvd	hybi@ietf.org	RFC 6455
1006	Abnormal Closure	hybi@ietf.org	RFC 6455

1007	Invalid frame payload data	hybi@ietf.org	RFC 6455
1008	Policy Violation	hybi@ietf.org	RFC 6455
1009	Message Too Big	hybi@ietf.org	RFC 6455
1010	Mandatory Ext.	hybi@ietf.org	RFC 6455
1011	Internal Server Error	hybi@ietf.org	RFC 6455
1015	TLS handshake	hybi@ietf.org	RFC 6455

11.8.WebSocket 操作码注册

本规范依据 RFC5226[RFC5226]陈述的原则，创建一个新的 IANA 注册用于 WebSocket 操作码。

作为本注册的一部分，IANA 维护以下信息：

操作码

操作码表示 WebSocket 帧的帧类型，定义在 5.2 节 。操作码是一个在 0 到 15（包括）之间的整数数字。

含义

状态码值的含义。

参考

规范要求的操作码。

WebSocket 状态码受制于“标准功能”IANA 注册策略[RFC5226]。

IANA 已经添加如下初始值到注册中心：

Opcode	Meaning	Reference
0	Continuation Frame	RFC 6455
1	Text Frame	RFC 6455

2	Binary Frame	RFC 6455
8	Connection Close Frame	RFC 6455
9	Ping Frame	RFC 6455
10	Pong Frame	RFC 6455

11.9.WebSocket 帧头位注册

本规范依据 RFC5226[RFC5226]陈述的原则，创建了一个新的 IANA 注册用于 WebSocket 帧头位（Framing Header Bits）。此注册控制的位分配标记为 5.2 节的 RSV1、RSV2 和 RSV3。

这些位被保留用于未来版本或本规范的扩展。

WebSocket 帧头位分配受制于“标准功能”IANA 注册测策略[RFC5226]。

12.其他规范使用 WebSocket 协议

WebSocket 协议目的是被另一个规范使用来提供一个通用机制来动态作者定义内容，例如，在一个规范中定义一个脚本 API。

这样的规范首先需要_建议一个 WebSocket 连接_，该算法是：

- 目的地，包含一个/host/和一个/port/。
- 一个/resource name/，允许在一个 host 和 port 标识多个服务。
- 一个/secure/标记，如果连接是加密的则为 true，否则为 false。
- 一个源[RFC6454]的 ASCII 序列化，负责连接。
- 可选的， 一个字符串标识一个协议，层叠在 WebSocket 连接之上。

/host/、/port/、/resource name/ 和/secure/标记通常从一个 URI 中使用该步骤解析一个 WebSocketURI 组件获得。如果没有指定一个 WebSocket，则这些步骤失败。

如果在任何时候连接将被关闭，那么规范需要使用_关闭 WebSocket 连接_算

法（7.1.1 节）。

7.1.4 节定义了什么时候_WebSocket 连接关闭_。

当打开一个连接，规范将需要处理什么时候_已经接收了一个 WebSocket 消息_的情况（6.2 节）。

要发送一些数据/data/到一个打开的连接，规范需要_发送一个 WebSocket 消息_（6.1 节）。

13.致谢

特别感谢 Ian Hickson，因为他是本协议的原始作者和编辑。本规范的初始设计受益于 WHATWG 和 WHATWG 邮件列表的许多人的参与。对规范的贡献没有按照章节跟踪，但在 WHATWG HTML 规范 http://whatwg.org/html5中给出了给规范贡献的所有人列表。

也特别感谢 John Tamplin 为本规范的“数据帧”提供了大量的文字。

也特别感谢 Adam Barth 提供了大量的文字和“数据掩码”章节的背景研究。

特别感谢 Lisa Dusseault 为应用区（Apps Area）复审（和帮助启动这个工作），Richard Barnes 为 Gen-Art 复审，和 Magnus Westerlund 为传输区（Transport Area）复审。特别感谢 HYBI WG 和过去和现在的 WG 主席：Joe Hildebrand、Salvatore Loreto 和 Gabriel Montenegro，他们不知疲倦地在幕后工作，直到这项工作完成。最后，但并非最不重要的，特别感谢负责区域董事（Area Director）Peter Saint-Andre。

感谢在 HYBI WG 邮件列表参与讨论的和贡献想法和/或提供细节复审的以下人员（列表并不是完整的）：Greg Wilkins, John Tamplin, Willy Tarreau, Maciej Stachowiak, Jamie Lokier, Scott Ferguson, Bjoern Hoehrmann, Julian Reschke, Dave Cridland, Andy Green, Eric Rescorla, Inaki Baz Castillo, Martin Thomson, Roberto Peon, Patrick McManus, Zhong Yu, Bruce Atherton, Takeshi Yoshino, Martin J. Duerst, James Graham, Simon Pieters, Roy T. Fielding, Mykyta Yevstifeyev, Len Holgate, Paul Colomiets, Piotr Kulaga, Brian Raymor, Jan Koehler, Joonas Lehtolahti, Sylvain Hellegouarch, Stephen Farrell, Sean Turner, Pete Resnick, Peter Thorson, Joe Mason, John Fallows, 和 Alexander Philippou。注意，上边列出的人员并不一定赞同这些工作的最终结果。

14.参考资料

14.1.参考标准

[ANSI.X3-4.1986]
American National Standards Institute, "Coded Character Set - 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[FIPS.180-3]
National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-3, October 2008,<http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf>.

[RFC1928]
Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928,March 1996.

[RFC2119]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616]
Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "HypertextTransfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2817]
Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.

[RFC2818]
Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3629]
Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3864]
Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

[RFC3986]
Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier

(URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC3987]
Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

[RFC4086]
Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4648]
Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5226]
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234]
Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5246]
Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6066]
Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6454]
Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

14.2.参考资料

[RFC4122]
Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC4270]
Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, November 2005.

[RFC5321]
Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC6202]
Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.

[RFC6265]
Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[TALKING]
Huang, L-S., Chen, E., Barth, A., Rescorla, E., and C. Jackson, "Talking to Yourself for Fun and Profit", 2010, <http://w2spconf.com/2011/papers/websocket.pdf>.

[W3C.REC-wsc-ui-20100812]
Roessler, T. and A. Saldhana, "Web Security Context: User Interface Guidelines", World Wide Web Consortium Recommendation REC-wsc-ui-20100812, August 2010, <http://www.w3.org/TR/2010/REC-wsc-ui-20100812/>.

Latest version available at <http://www.w3.org/TR/wsc-ui/>.

[WSAPI]
Hickson, I., "The WebSocket API", W3C Working Draft WD-websockets-20110929, September 2011, <http://www.w3.org/TR/2011/WD-websockets-20110929/>.

Latest version available at<http://www.w3.org/TR/websockets/>.

[XMLHttpRequest]
van Kesteren, A., Ed., "XMLHttpRequest", W3C Candidate Recommendation CR-XMLHttpRequest-20100803, August 2010,<http://www.w3.org/TR/2010/CR-XMLHttpRequest-20100803/>.

Latest version available at <http://www.w3.org/TR/XMLHttpRequest/>.

作者地址

Ian Fette
Google, Inc.

EMail: ifette+ietf@google.com
URI: http://www.ianfette.com/

Alexey Melnikov
Isode Ltd.
5 Castle Business Village
36 Station Road
Hampton, Middlesex TW12 2BX
UK

EMail: Alexey.Melnikov@isode.com