互联网工程任务组（IETF）
征求意见：6455Google
类别：标准轨道A
ISSN:

.Fette
, Inc.
.梅尔尼科夫
2070-
1721Isode Ltd.

2011 年 12 月

WebSocket 协议

摘要

WebSocket 协议实现了客户端与远程主机之间的双向通信，远程主机在受控环境中运行不受信任的代码，并选择接受来自该代码的通信 该协议使用的安全模型是 Web 浏览器常用的基于来源的安全模型 该协议包括一个开局握手，然后是基本的消息框架，分层覆盖在 TCP 上。 该技术的目标是为需要与服务器进行双向通信的基于浏览器的应用程序提供一种机制，这种机制不依赖于打开多个 HTTP 连接（例如，使用 XMLHttpRequest 或 <iframe>s 和长时间轮询）。

本备忘录的现状

这是一份互联网标准跟踪文件。

本文档是互联网工程任务组 (IETF) 的产品。 它代表了 IETF 社区的共识。 它已接受公众审查，并已由互联网工程指导小组 (IESG)批准发布。 有关互联网标准的更多信息，请参阅 RFC 5741 第 2 节。

有关本文件的当前状态、任何勘误以及如何提供反馈的信息，请访问 http://www.rfc-editor.org/info/rfc6455。

版权声明

读这些文件，因为它们描述了您对本权利和限制　　从本文档中提取的代码组件必须

目录

1. 导言

1.1. 背景介绍

本节不具规范性。

一直以来，创建需要在客户端和服务器之间进行双向通信的网络应用程序（如即时消息和游戏应用程序）时，都需要滥用 HTTP 来轮询服务器进行更新，同时以不同的 HTTP 调用 [RFC6202] 发送上游通知。

这导致了各种问题：

- 服务器被迫为每个客户端使用多个不同的底层 TCP 连接：一个用于向客户端发送信息，另一个用于接收每条信息。
- 有线协议的开销很大，每个客户端到服务器的信息都有一个 HTTP 头。
- 客户端脚本被迫维护从传出连接到传入连接的映射，以跟踪回复。

这就是 WebSocket 协议提供的功能。 结合 WebSocket API [WSAPI]，它为网页与远程服务器之间的双向通信 提供了 HTTP 轮询的 替代方案

同样的技术可用于各种网络应用程序：游戏、股票行情、多用户同时编辑应用程序、实时公开服务器端服务的用户界面等。

WebSocket 协议旨在取代现有的双向通信技术，这些技术将 HTTP 用作传输层，并从现有的基础架构（代理、过滤、身份验证）中获益。 这些技术是在效率和可靠性之间权衡后实现的，因为 HTTP 最初并不是用于双向通信的（更多讨论请参阅 [RFC6202]）。

WebSocket 协议试图在现有 HTTP 基础架构的背景下实现现有双向 HTTP 技术的目标；因此，它被设计为可在 HTTP 端口 80 和 443 上运行，并支持 HTTP 代理和中介，即使这意味着当前环境特有的一些复杂性 不过，WebSocket 的设计并不局限于 HTTP，未来的实现可以在 HTTP 端口 80 和 443上使用更简单的握手。

最后这一点很重要，因为交互式消息传输的流量模式标准 HTTP 流量并不完全一致，可能会对某些组件造成异常负载。

1.2. 协议概述

本节不具规范性。

协议包括两个部分：握手和数据传输。客户端的握手过程如下：

```
GET /chat HTTP/1.1
主机：server.example.com 升
级：websocket 连接：升级
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== 起
源：http://example.com
Sec-WebSocket 协议：聊天、超级聊天 Sec-
WebSocket 版本：13
```

服务器的握手过程如下：HTTP/1.1 101 交换协议

```
升级：websocket 连接：升级
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

来自客户端的前导行遵循请求行格式。
来自服务器的前导行遵循 Status-Line 格式。

两种情况下，前导行后都会出现一组无序的标头字段。这些标头字段的含义在本文档第 4 节中有具体说明。也可能出现其他标头字段，如 cookies [RFC6265]。标头的格式和解析如 [RFC2616] 中所定义。

一旦客户端和服务器都发送了握手信息，如果握手成功，那么数据传输部分就开始了。这是一个双向通信通道，每一方都可以独立于另一方随意发送数据。

握手成功后，客户端和服务器以概念单位来回传输数据，在本规范中称为 "报文"。

WebSocket 报文并不一定与特定的网络层成帧相对应，因为分片报文可能会被中间人合并或拆分。

属于同一报文的每个帧都包含相同类型的数据概括地说，有文本数据类型（解释为 UTF-8 [RFC3629] 文本）、二进制数据类型（解释由应用程序决定）和控制帧类型（不打算为应用程序携带数据，而是用于协议级信令，如关闭连接的信号）。 该版本的协议定义了六种帧类型，并为将来使用保留了十种。

1.3. 开场握手

本节不具规范性。

为此，WebSocket 客户端的握手方式是HTTP 升级请求：

```
GET /chat HTTP/1.1
主机：server.example.com 升
级：websocket 连接：升级
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== 起
源：http://example.com
Sec-WebSocket 协议：聊天、超级聊天 Sec-
WebSocket 版本：13
```

根据 [RFC2616]，客户端可按任何顺序发送握手中的标头字段，因此接收不同标头字段的顺序并不重要。

GET 方法 [RFC2616] 的 "Request-URI "用于标识 WebSocket 连接的端点，这样既可以从一个 IP 地址为多个域提供服务，也可以由一个服务器为多个 WebSocket 端点提供服务。

根据 [RFC2616]，客户端会在握手的 |Host| 标头字段中包含主机名，这样客户端和服务器就能确认它们同意使用的是哪台主机。

该版本中可用的典型选项包括子协议选择器（|Sec-WebSocket-Protocol|）、客户端支持的扩展列表（|Sec-WebSocket-Extensions|）、|Origin|头信息字段等。 |Sec-WebSocket-Protocol|请求头信息字段可用于指明客户端可接受哪些子协议（WebSocket 协议上分层的应用级协议）。
服务器从可接受的协议中选择一个或一个都不选，并在握手中回传该值，以表明它选择了该协议。

Sec-WebSocket 协议：聊天

|Origin| 头字段 [RFC6454] 用于防止使用 WebSocket API的脚本在未经授权的情况下跨源使用 WebSocket 服务 服务器会被告知生成 WebSocket 连接请求的脚本源 如果服务器不希望接受来自该脚本源的连接，它可以选择发送一个适当的 HTTP 错误代码来拒绝该连接。 该标头字段由浏览器客户端发送；对于非浏览器客户端，如果该标头字段对这些客户端的上下文有意义，也可以发送。

最后，服务器必须向客户端证明它收到了客户端的 WebSocket 握手，这样服务器就不会接受非 WebSocket 连接。 这可以防止攻击者通过使用 XMLHttpRequest [XMLHttpRequest] 或表单提交向 WebSocket 服务器发送精心制作的数据包来欺骗服务器

第一条信息来自客户端握手中的 |Sec-WebSocket-Key| 标头字段：

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

对于该标头字段，服务器必须获取其值（如标头字段中的值，例如Base64-encoded [RFC4648] 版本减去任何前导和尾部空白），并将其与字符串形式的全球唯一标识符（GUID，[RFC4122]）"258 EAFA5-E914-47DA- 95CCA-C5AB0DC85B11 "连接起来。 然后在服务器的握手过程中返回该连接的 SHA-1 hash（160 位）[FIPS.180-3] base64 编码（参见 [RFC4648] 第 4 节）。

具体来说，如果在上面的示例中，|Sec-WebSocket-Key| 头信息字段的值为 "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"。 然后，服务器将对其进行 SHA-1 散列，得到 0xb3 0x7a 0x4f 0x2c 0xc0 0x62 0x4f 0x16 0x90 0xf6 0x46 0x06 0xcf 0x38 0x59 0x45 0xb2 0xbe 0xc4 0xea。 然后，对该值进行 base64 编码（参见 [RFC4648] 第 4 节），得到值 "s3pPLMBiTxaQ9kYGzzhZRbK+xOo=" 。

第一行是 HTTP 状态行 ，状态代码为 101：

HTTP/1.1 101 交换协议

101 以外的任何状态代码都表示 WebSocket 握手尚未完成，HTTP 语义仍然适用 。

连接 "和 "升级 "头字段完成 HTTP 升级 。"Sec-WebSocket-Accept "头字段表示服务器是否愿意接受连接 如果存在，该头字段必须包括Sec-WebSocket-Accept "头字段中发送的客户端 nonce 的哈希以及预定义的 GUID。 任何其他值都不得被解释为服务器接受连接。

HTTP/1.1 101 协议转换升级：websocket
连接：升级
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

WebSocket 客户端会检查这些字段以获取脚本页面。如果 |Sec-WebSocket-Accept| 值与预期值不匹配，如果缺少标头字段，或者 HTTP 状态代码不是 101，连接将不会建立，WebSocket 框架也不会发送。

在此版本的协议中，主要选项字段是 |Sec-WebSocket-Protocol|，表示服务器选择的子协议。 WebSocket 客户端会验证服务器是否包含了 WebSocket 客户端握手中指定的值之一。 使用多种子协议的服务器必须确保它根据客户端的握手选择了其中一种，并在其握手中指定了该

Sec-WebSocket 协议：聊天

如 [RFC6265] 所述，服务器还可以设置与 cookie 相关的选项字段来 _set_ cookie。

1.4. 闭幕式握手

本节不具规范性。

结束握手比开始握手简单得多。

任一对等方均可发送包含指定控制序列数据的控制帧，以开始关闭握手（详见 在收到这样一个帧后，另一个对等方会发送一个关闭帧作为回应（如果它还没有发送过的话）在收到该控制帧后，第一个对等方会关闭连接，并确信不会再收到任何数据。

在发送表示连接应关闭的控制帧后，对等端不会再发送任何数据；在接收表示连接应关闭的控制帧后，对等端会丢弃收到的任何其他数据。

双方同时启动握手是安全的。

关闭握手是对 TCP 关闭握手（FIN/ACK）的补充，因为 TCP 关闭握手并不总是可靠的端到端，特别是在存在拦截代理和其他中介的情况下。

例如，在某些平台上，如果套接字在接收队列中有数据的情况下 被关闭，就会发送一个 RST 数据包，这将导致收到 RST 的一方的 recv() 失败，即使还有数据等待读取。

1.5. 设计理念

本节不具规范性。

WebSocket 协议的设计原则是应尽量减少框架（唯一的框架是使协议基于框架而不是基于流，并支持区分 Unicode 文本框架和二进制框架）。 预计元数据将由应用程序分层到 WebSocket 上。

层，就像元数据被应用层（如 HTTP）分层到 TCP 上一样。

从概念上讲，WebSocket 其实只是 TCP 的一个顶层，它的功能如下：

- 为浏览器添加了基于网络起源的安全模型
- 增加了寻址和协议命名机制，以支持一个端口上的多个服务和一个 IP 地址上的多个主机名
- 在 TCP 上建立框架机制，回到 TCP 所基于的 IP 数据包机制，但没有长度限制
- 包括额外的带内关闭握手功能，该功能可在代理和其他中介存在的情况下使用

外，WebSocket 没有增加任何东西。 基本上它的目的是在网络的限制条件下，尽可能接近于将原始 TCP 暴露给脚本。 它的设计方式还使其服务器可以与 HTTP 服务器共享一个端口，方法是让其握手成为一个有效的 HTTP 升级请求。 从概念上讲，我们可以使用其他协议来建立客户端-服务器消息传递，但 WebSockets 的目的是提供一个相对简单的协议，该协议可以与 HTTP 和已部署的 HTTP 基础设施（如代理）共存考虑到安全因素的情况下，该协议尽可能接近 TCP，可安全地与此类基础设施有针对性地添加了一些内容，以简化使用并保持简单（如添加消息语义）。

该协议具有可扩展性，未来版本可能会引入更多概念，如多路复用。

1.6. 安全模式

本节不具规范性。

，WebSocket 协议由专用客户端直接使用时（即不是通过网页浏览器从网页使用），起源模型就没有用了，因为客户端可以提供任何任意的起源字符串。

该协议旨在避免与 SMTP [RFC5321] 和 HTTP 等现有协议的服务器建立连接，同时允许 HTTP 服务器在以下情况下选择支持该协议

要做到 这一点，需要进行严格而细致的握手，并限制在握手结束前可插入连接的数据（从而限制服务器可受影响的程度）。

同样，当来自其他协议（尤其是 HTTP）的数据被发送到 WebSocket服务器HTML "表单 " 被提交到 WebSocket 服务器时可能发生时，WebSocket 也会无法建立连接。 这主要是通过要求服务器证明它读取了握手信息来实现的，而服务器只有在握手包含适当部分时才能做到而这些部分只能通过WebSocket 客户端发送 特别是，在编写本规范时，攻击者无法从仅使用 HTML 和 JavaScript API（如 XMLHttpRequest [XMLHttpRequest]）的 Web 浏览器中设置以 |Sec-| 开头的字段。

1.7. 与 TCP 和 HTTP 的关系

本节不具规范性。

WebSocket 协议是一个独立的基于 TCP 的协议。它与 HTTP 的唯一关系是，HTTP 服务器将其握手解释为升级请求。

默认情况下，WebSocket 协议使用 80 端口进行常规 WebSocket 连接，使用 443 端口进行通过传输层安全（TLS）[RFC2818] 隧道传输的 WebSocket 连接。

1.8. 建立联系

本节不具规范性。

当连接到一个 HTTP 服务器共享的端口时（这种情况很可能发生在端口 80 和 443 的流量上），该连接在 HTTP 服务器看来将是一个普通的 GET 请求，并附带一个升级提议。
在相对简单设置中只需一个 IP 地址和一个服务器来处理到一个主机名的所有流量，这可能是部署基于 WebSocket 协议的系统的一种实用方法 在更平器），服务器
在编写本规范时，应注意端口 80 和端口100 的连接可能会出现问题。
443 端口的连接成功率明显不同，443 端口的连接成功率明显更高，但随着时间的推移，这种情况可能会发生变化。

1.9. 使用 WebSocket 协议的子协议

本节不具规范性。

客户端可以通过在握手中加入 |Sec-WebSocket-Protocol| 字段来请求服务器使用特定的子协议。 如果指定了该字段，服务器需要在其响应中加入相同的字段和所选子协议值之一才能建立连接。

这些子协议名称应按照第 11.5 节的规定进行注册。 为避免潜在的冲突，建议使用包含子协议发起者域名的 ASCII 版本的名称。 例如，如果示例公司要创建一个聊天子协议，由 Web 上的许多服务器来执行，他们可以将其命名为 "chat.example.com"。如果示例组织将其竞争子协议命名为 "chat 如果 Example 组织将其竞争性子协议命名为 "chat.example.org"，那么这两个子协议可以由服务器同时执行，服务器会根据客户端发送的值动态选择使用哪个子协议。

通过更改子协议名称，子协议可以向后不兼容的方式进行版本控制，例如，从 "bookings.example.net "到 "v2.bookings.example.net"。 这些子协议将被 WebSocket 客户端视为完全独立的协议。 向后兼容的版本控制可以通过重复使用相同的子协议字符串来实现，但要精心设计实际的子协议以支持这种可扩展性。

2. 一致性要求

本规范中的所有图表、示例和注释以及明确标注为非规范性的所有章节均为非规范性。本规范中的其他内容均为规范性内容。

中的关键词 "MUST"、"MUST NOT"、"REQUIRED"、"SHALL"、"SHALL NOT"、"SHOULD"、"SHOULD NOT"、"RECOMMENDED"、"MAY "和 "OPTIONAL"（可选）。 文件的解释与 [RFC2119] 中所述相同。

作为算法一部分的命令式要求（如 "去掉任何前导空格字符 "或 "返回假值并中止这些步骤"），应根据介绍算法时使用的关键词（"MUST"、"SHOULD"、"MAY "等）的含义来解释。

以算法或具体步骤表述的一致性要求可以以任何方式实现，只要最终结果是等同的即可（特别是，本规范中定义的算法旨在简单易懂，而不是为了提高性能）。

2.1. 术语和其他约定

ASCII 应指 [ANSI.X3-4.1986] 中定义的字符编码方案。

本文档参考了 STD 63 [RFC3629] 中定义的 UTF-8 值，并使用了 UTF-8 符号格式。

关键术语，如已命名的算法或定义，如
这个

标题字段或变量的名称用 |this| 表示。变量值用 /this/ 表示。

本文档引用了 _Fail the WebSocket Connection_ 的过程， 该过程在第 7.1.7 节中进行了定义。

将字符串转换为 ASCII 小写字母_是指将 U+0041 至 U+005A（即大写拉丁字母 A 至大写拉丁字母 Z）范围内的所有字符替换为 U+0061 至 U+007A（即小写拉丁字母 A 至小写拉丁字母 Z）范围内的相应字符。

以_ASCII 大小写不敏感_的方式比较两个字符串，是指完全按码位进行比较，但 U+0041 至 U+005A 范围内的字符（即大写拉丁字母 A 至大写拉丁字母 Z）和 U+0061 至 U+007A 范围内的相应字符（即小写拉丁字母 A 至小写拉丁字母 Z）除外。
字母 Z）也视为匹配。

本文档中使用的术语 "URI "与 [RFC3986] 中的定义相同。

当实现需要_发送_数据作为 WebSocket 协议的一部分时，实现可以任意延迟实际传输，例如，缓冲数据以减少 IP 数据包的发送量。

请注意，本文档在不同章节中使用了 ABNF 的 [RFC5234] 和 [RFC2616] 变体。

3. WebSocket URI

本规范使用 RFC 5234 [RFC5234] 中定义的 ABNF 语法，以及 URI 规范 RFC 3986 [RFC3986] 中定义的术语和 ABNF 产品，定义了两种 URI 方案。

```
ws-URI = "ws:""//" host [ ":" port ] path [ "?" query ]
wss-URI = "wss:""//" host [ ":" port ] path [ "?" query ]

host = <主机，定义见 [RFC3986]，第 3.2.2 节 > port =
<端口，定义见 [RFC3986]，第 3.2.3 节 >
path = <path-abempty，定义于 [RFC3986]，第 3.3 节 > query =
<query，定义于 [RFC3986]，第 3.4 节 >
```

端口组件是可选的，"ws "默认为 80 端口，而 "wss "默认为 443 端口。

如果方案组件与 "wss "不区分大小写，则 URI 称为 "安全"（也可以说 "安全标志已设置"）。

资源名称"（在第 4.1 节中也称为"/资源名称/"）可由以下文字连接而成：

- 如果路径组件为空，则为"/"
- 路径组件
- "？"，如果查询组件是非空的
- 查询组件

片段标识符在 WebSocketURI的上下文中毫无意义，因此不得在这些 URI 中使用。
与任何 URI 方案一样，当字符 "#"不表示片段的开始时，必须转义为 %23

4. 开场握手

4.1. 客户要求

要_建立 WebSocket 连接_，客户机需要打开一个连接并发送本节中定义的握手。 连接被定义为初始处于 CONNECTING 状态。 客户机需要提供/host/、

/port/、/resource name/ 和/secure/标志，它们是第 3 节中讨论的 WebSocket URI 的组成部分，以及要使用的/protocols/和/extensions/列表。
此外，如果客户端是网络浏览器，它还会提供 /origin/。

在受控环境中运行的客户端，例如与特定运营商绑定的手机上的浏览器，可以将连接的管理卸载给网络上的另一个代理。 在这种情况下，本规范中的客户端被视为包括手机软件和任何此类代理。

当客户端要_建立一个 WebSocket 连接_时，必须给定一组（/host/、/port/、/resource name/ 和/secure/标志），以及 使用的/protocols/和/extensions/列表，如果是网络浏览器，还必须给定/origin/，然后打开一个连接，发送开局握手，并读取服务器的握手回应。 本节将如何打开连接、在开场握手中应发送哪些内容以及如何解释服务器的响应等具体要求。 在下文中，我们将使用第 3 节中的术语，如该节中定义的"/host/"和"/secure/标志"。

1. 根据第 3 节中指定WebSocket URI 规范，传入该算法的 WebSocket URI 的组件（/host/、/port/、/resource name/ 和/secure/标志）必须有效。 如果任何组件无效，客户端必须 _Fail the WebSocket Connection_ 并中止这些步骤

2. 如果客户端已经与 /host/ 和端口 /port/ 对标识的远程主机（IP 地址）建立了 WebSocket 连接，即使远程主机以其他名称为人所知，客户端也必须等待该连接建立或连接失败。 处于 CONNECTING（连接）状态的连接不得超过一个。 如果同时尝试与同一 IP 地址建立多个连接，客户端必须将它们序列化，以便通过以下步骤一次运行的连接不得超过一个。

   如果客户机无法确定远程主机的 IP 地址（例如，因为所有通信都是通过代理服务器进行的，而代理服务器本身会执行 DNS 查询），则客户机必须在本步骤中假定每个主机名都指向一个不同的远程主机，而客户机应将同时待建连接的总数限制在一个合理的较低水平（例如，客户机可能允许同时待建连接到 a.example.com 和 b.example.com，但如果请求同时连接到一台主机的数量达到 30 个，则可能不被允许）、客户端可能允许同时与 a.example.com 和 b.example.com 进行待处理连接，但如果请求同时与一台主机进行 30 次连接，则可能不允许）。 例如，在网络浏览器中，客户端在设置同时待处理连接数限制时需要考虑用户打开的标签页数量。

注意：这样脚本就很难通过打开大量 WebSocket 连接到远程主机来执行拒绝服务攻击。 服务器可以在关闭连接前暂停，这样可以降低客户端重新连接的速度，从而在受到攻击时进一步减轻自身的负载

服务器可以拒绝接受来自现有连接数过多的主机/IP 地址的连接，或在负载过高时断开占用资源的连接。

3. 使用代理：在使用 WebSocket 协议连接主机/host/和端口时，如果客户端被配置为使用代理
/port/，那么客户端就应该连接到该代理，并要求它打开一个与 /host/ 给定的主机和 /port/ 给定的端口的 TCP 连接。

举例说明例如，如果客户端使用 HTTP 代理来处理所有流量，那么如果它要尝试连接到服务器 example.com 上的 80 端口，它可能会向代理服务器发送以下行文：

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

如果有密码，连接可能如下所示：CONNECT example.com:80 HTTP/1.1
主机：example.com
代理授权：Basic ZWRuYW1vZGU6bm9jYXBlcyE=

如果客户端未配置为使用代理，则应向 /host/ 指定的主机和 /port/ 指定的端口打开直接 TCP 连接。

注意：如果实现没有提供明确的用户界面来选择 WebSocket 连接的代理（与其他代理分开），我们鼓励实现为 WebSocket 连接使用 SOCKS5 [RFC1928] 代理（如果有），如果没有，则优先使用为 HTTPS 连接配置的代理，而不是为 HTTP 连接配置的代理。

就代理自动配置脚本而言，传递函数的 URI 必须由 /host/、/port/ 构建、/resource name/ 和 /secure/ 标志，使用第 3 节中给出的 WebSocket URI 定义。

注意：代理自动配置脚本可通过方案（"ws "表示未加密连接，"wss "表示加密连接）识别 WebSocket 协议。

4. 如果由于直接连接失败或使用的代理返回错误而导致连接无法打开，那么客户端必须 _Fail the WebSocket Connection_（WebSocket 连接失败）并放弃连接尝试。

5. 如果 /secure/ 为 true，客户端必须在打开连接后和发送握手数据之前通过连接执行 TLS 握手 [RFC2818]。 如果失败（例如，服务器证书无法验证），则客户端必须 _Fail the WebSocket Connection_ 并中止连接。 否则，该通道上的所有进一步通信都必须通过加密隧道进行 [RFC5246]。

   客户端必须在 TLS 握手[RFC6066]中使用服务器名称指示扩展。

一旦建立了与服务器的连接（包括通过代理或 TLS 加密隧道建立的连接），客户端必须向服务器发送开局握手信息。 握手信息由 HTTP 升级请求以及必填和可选标头字段列表组成 握手信息的要求如下。

1. 握手必须是 [RFC2616] 规定的有效 HTTP 请求。

2. 请求方法必须是 GET，HTTP 版本必须至少是 1.1。

   例如，如果 WebSocket URI 为 "ws://example.com/chat"，则发送的第一行应为 "GET /chat HTTP/1.1"。

3. 请求的 "Request-URI "部分必须与第 3 节中定义的/resource name/（相对 URI）相匹配，或者是一个绝对的 http/https URI，在解析时，其/resource name/、/host/和/port/与相应的 ws/wss URI 相匹配。

4. 请求必须包含一个 |Host| 头信息字段，其值包含 /host/ 和可选的":"，后面跟 /port/（如果不使用默认端口）。

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

6. 请求必须包含一个 |Connection| 标头字段，其值必须包括 "升级 "标记。

7. 请求必须包含一个名称为 |Sec-WebSocket-Key|。该标头字段的值必须是一个由随机选择的 16 字节值（ ）组成的 nonce ，该值已经过 base64 编码（参见 [RFC4648] 第 4 节）。 nonce 必须为每个连接随机选择。

   注：举例来说，如果随机选择的值是字节 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 的序列，那么报头字段的值将是 "AQIDBAUGBwgJCgsMDQ4PEC=="。

8. 如果请求来自浏览器客户端 ，则请求必须包含一个名称为 |Origin| [RFC6454] 的头域。 如果连接来自非浏览器客户端，则如果该客户端的语义与此处描述的浏览器客户端用例相符，则请求可以包含该头域。 该头域的值是建立连接的代码运行的上下文的 ASCII 序列化 origin有关如何构建该头域值的详细信息， 请参阅 [RFC6454]。

   例如，如果从 www.example.com 下载的代码试图建立与 ww2.example.com 的连接，则标头字段的值为 "http://www.example.com"。

9. 请求必须包含一个名称为 |Sec-WebSocket-Version|。该标头字段的值必须是 13。

   注：尽管本文档的草案版本（-09、-10、-11 和-12）已发布（主要是编辑上的修改和说明，而非对有线协议的修改），但值 9、10、11 和 12 并未用作 Sec-WebSocket-Version 的有效值。 这些值在 IANA 注册表中保留，但过去和将来都不会使用。

10. 请求可以包含一个名称为 | 如果存在，该值表示客户端希望使用的一个或多个以逗号分隔的子协议，按优先级排序。 构成该值的元素必须是非空字符串，字符范围为 U+0021至 U+007E，不包括 [RFC2616] 中定义的分隔符，并且必须都是唯一字符串。 该标头字段值的 ABNF 为 1#token，其中构造和规

则的定义如 [RFC2616] 所述。

11. 请求可以包含一个名称为 |Sec-WebSocket-Extensions|. 如果存在，该值表示客户端希望使用的协议级扩展。

12. 请求可以包含任何其他标头字段，例如 Cookie [RFC6265] 和/或与身份验证相关的标头字段，如 |Authorization| 标头字段 [RFC2616]，这些标头字段将根据定义它们的文档进行处理。

一旦发送了客户端的开局握手，客户端必须等待服务器的响应，然后再发送任何数据。客户端必须按以下方式验证服务器的响应：

1. 如果从服务器收到的状态代码不是 101，客户端将按照 HTTP [RFC2616] 程序处理响应。特别是，如果收到 401 状态代码，客户端可能会执行身份验证；服务器可能会使用 3xx 状态代码重定向客户端（但不要求客户端遵循这些代码），等等。否则，请按以下

2. 如果响应中没有 |Upgrade| 头信息字段，或者 |Upgrade| 头信息字段包含的值不是 "websocket "值的 ASCII 大小写不敏感匹配值，客户端必须 _Fail the WebSocket Connection_（WebSocket 连接失败）。

3. 如果响应缺少|连接|标头字段或
如果 |Connection| 标头字段不包含与 "Upgrade"（升级）值进行 ASCII 大小写不敏感匹配的标记，则客户端必须 _Fail the WebSocket Connection_（WebSocket 连接失败）。

4. 如果响应缺少|Sec-WebSocket-Accept|头域，或者|Sec-WebSocket-Accept|包含的值不是由|Sec-WebSocket-Accept|头域和|Sec-WebSocket-Accept|头域连接的基于64编码的SHA-1值。
Key|（作为字符串，未进行 base64 解码）为 "258EAFA5- E914-47DA-95CA-C5AB0DC85B11"，但忽略任何前导和尾部空白，客户端必须_使 WebSocket 连接失败_。

5. 如果响应包含一个 |Sec-WebSocket-Extensions| 头信息字段，并且该头信息字段表明使用了客户端握手过程中没有的扩展（服务器表明使用了客户端未请求的扩展

），则客户端必须 _Fail the WebSocket Connection_（失败 WebSocket 连接）。（第 9.1 节将讨论如何解析该头信息字段以确定请求了哪些扩展）。

6. 如果响应包含一个 |Sec-WebSocket-Protocol| 头字段，并且该头字段表明使用了客户端握手中不存在的子协议（服务器表明使用了客户端未请求的子协议），则客户端必须 _Fail the WebSocket Connection_（失败 WebSocket 连接）。

如果服务器的响应不符合本节和第 4.2.2 节中对服务器握手的要求，客户端必须_使 WebSocket 连接失败_。

请注意，根据 [RFC2616]，HTTP 请求和 HTTP 响应中的所有标头字段名都不区分大小写。

使用中的扩展名（Extensions In Use）被定义为一个（可能为空）字符串，其值等于服务器握手提供的|Sec-WebSocket-Extensions|头域如果没有该头域，则为空值。 使用中的子协议（_Subprotocol In Use_）被定义为服务器握手中的|Sec-WebSocket-Protocol|标头字段的值，如果服务器握手中没有该标头字段，则为空值。
此外，如果服务器握手中的任何标头字段表明应设置 cookie（如 [RFC6265] 所定义），则这些 cookie 被称为_服务器开启握手期间设置的 cookie_。

4.2. 服务器端要求

在这种情况下，就本规范而言，服务器被视为包括服务器端基础设施的所有部分，从第一个终止 TCP 连接的设备一直到处理请求和发送响应的服务器。

举例说明： 在本规范中，"服务器 "是两台计算机的组合。

4.2.1. 读懂客户的开场白

当客户端启动 WebSocket 连接时，它会发送自己的部分开启握手。 服务器必须至少解析该握手的一部分，以便获取必要的信息来生成握手的服务器部分

如果服务器在读取握手过程中发现客户端发送的握手不符合下面的描述（注意，根据 [RFC2616]，头域的顺序并不重要），包括但不限于任何违反为握手组件指定的 ABNF 语法的行为，服务器必须停止处理客户端的握手，并返回带有适当错误代码（如 400 Bad Request）的 HTTP 响应。

1. HTTP/1.1 或更高版本的 GET 请求，包括一个 "Request-URI" [RFC2616]，应被解释为第 3 节中定义的/资源名称/（或包含/资源名称/的绝对 HTTP/HTTPS URI）。

2. 包含服务器权限的 |Host| 头信息字段。

3. 包含 "websocket "值的 |Upgrade| 标头字段，作为 ASCII 大小写不敏感值处理。

4. 包含标记 "Upgrade "的 |Connection| 标头字段，被视为 ASCII 大小写不敏感值。

5. 一个 |Sec-WebSocket-Key| 头信息字段，带有一个 base64 编码（见 [RFC4648] 第 4 节）值，解码后长度为 16 字节。

6. 一个 |Sec-WebSocket-Version| 标头字段，值为 13。

7. 可选择|Origin|头域。 所有浏览器客户端都会发送此头域 。 缺少此头域的连接尝试不应被解释为来自浏览器客户端。

8. 可选择|Sec-WebSocket-Protocol|标头字段，该字段包含一个值列表，表示客户端希望使用的协议，按优先级排序。

9. 该标头字段的解释将在第 9.1 节中讨论。

10. [RFC2616]，未知标头字段将被忽略。

4.2.2. 发送服务器开局握手信息

当客户端与服务器建立 WebSocket 连接时，服务器必须完成以下步骤以接受连接并发送服务器的开局握手。

1. 如果连接发生在 HTTPS（HTTP-over-TLS） 端口上，则在连接上执行 TLS 握手。 如果失败（例如，客户端在扩展客户端 hello "server_name "扩展名中指定了一个主机名，而该主机不在服务器上），则关闭连接；否则，连接的所有进一步通信（包括服务器的握手）都必须通过加密隧道 [RFC5246]

2. 服务器可执行额外的客户端身份验证，例如通过返回 401 状态代码和相应的 |WWW-Authenticate|标头字段，如 [RFC2616] 所述。

3. 服务器可以使用 3xx 状态代码[RFC2616]重定向客户端。 请注意，这一步骤可以与上述可选的身份验证步骤同时、之前或之后进行。

4. 建立以下信息：

   /origin/
   客户端握手中的||Origin|头字段表示建立连接的脚本 的来源。 来源被序列化为 ASCII 并转换为小写。 服务器可使用此信息作为是否接受传入连接的决定的一部分。 如果服务器不验证来源，它将接受来自任何地方的连接。 如果服务器不想接受此连接，它必须返回一个适当的 HTTP 错误代码（如，并中止本节所述的 WebSocket 握手、403 Forbidden），并中止本节所述的 WebSocket 握手。 更多详情，请参阅 第 10 节.

   /key/
   客户端握手中的|Sec-WebSocket-Key|标头字段包含一个 base64 编码值，如果解码，长度为 16 字 该（编码）值用于创建服务器握手，以表示接受连接。 服务器无需对|Sec-WebSocket-Key|值进行 base64 解码。

/version/
客户端握手中的|Sec-WebSocket-Version|标头字段包括客户端试图与之通信的 WebSocket 协议版本。 如果该版本服务器能理解的版本不匹配，服务器必须中止本节所述的 WebSocket 握手，并发送一个适当的 HTTP 错误代码（如 426 Upgrade Required）和一个|Sec-WebSocket-Version|标头字段，指明服务器能理解的版本。

/资源名称/
服务器提供的服务的标识符。 如果服务器提供多种服务，那么该值应来自客户端握手中在 GET 方法的 "Request-URI" [RFC2616] 中给出的资源名称 如果请求的服务不可用，服务器必须发送适当的 HTTP 错误代码（如 404 Not Found）并中止 WebSocket 握手

/subprotocol/
代表服务器准备使用的子协议的单个值或空值。 所选值必须来自客户端的握手，特别是从服务器愿意用于此连接（如果有）的|Sec-WebSocket-Protocol|字段中选择一个值。 如果客户端的握手不包含这样的头字段，或者服务器不同意客户端请求的任何子协议，则唯一可接受的值是空值。 不包含此类字段等同于空值（这意味着如果服务器不希望同意建议的子协议之一，则不得在其响应中发回头信息字段）。 空字符串与这些目的下的空值不同，也不是该字段的合法值。 该头信息字段值的 ABNF 为 (token)，其中构造和规则的定义如 [RFC2616] 所述

/ 扩展名/
代表服务器准备使用的协议级扩展的列表（可能为空）。 如果服务器支持多个扩展，则必须从客户端的握手中获取该值，特别是从|Sec-WebSocket-Extensions|字段中选择一个或多个值。 没有该字段 则等同于空值。 空字符串与这些扩展的 空值不同。

这些值的选择和解释方法将在第 9.1 节中讨论。

5. 如果服务器选择接受传入的连接，它必须回复一个有效的 HTTP 响应，说明以下内容。

   1. 根据 RFC 2616 [RFC2616]，状态行的响应代码为 101， 这样的响应可能类似于 "HTTP/1.1 101 交换协议"。

   2. 根据 RFC 2616 [RFC2616]，|Upgrade| 标头字段的值为 "websocket"。

   3. 连接 "标头字段，值为 "升级"。

   4. Sec-WebSocket-Accept| 头信息字段。 该头信息字段的值是由上文第 4.2.2 节步骤 4 中定义的/key/与字符串 "258EAFA5- E914-47DA-95CA-C5AB0DC85B11" 连接对该连接值进行 SHA-1 哈希运算得到一个 20 字节的值，并对该 20 字节的哈希值进行 base64 编码（参见 [RFC4648] 第 4 节）。

      该标头字段的 ABNF [RFC2616] 定义如下：

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

注意举例来说，如果客户端握手中的 |Sec-WebSocket-Key| 头域值为 "dGhlIHNhbXBsZSBub25jZQ==", 服务器将附加字符串 "258EAFA5-E914-47DA-95CA-C5AB0DC85B11"，形成字符串 "dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CA- C5AB0DC85B11"。 然后，服务器将对该字符串进行 SHA-1 哈希运算，得到 0xb3 0x7a 0x4f 0x2c 0xc0 0x62 0x4f 0x160xf6 0x46 0x06 0xcf 0x38 0x59 0x45 0xb2 0xbe 0xc4 0xea 的值。然后该值进行 base64 编码，得到 "s3pPLMBiTxaQ9kYGzzhZRbK+xOo="的值，该值将在

"s3pPLMBiTxaQ9kYGzzhZRbK+xOo="中返回
|Sec-WebSocket-Accept| 头信息字段。

5. 可选择|Sec-WebSocket-Protocol|标头字段，其值为第 4.2.2 节步骤 4 中定义的/subprotocol/。

6. 如果要使用多个扩展，可以在一个|Sec-WebSocket-Extensions| 头信息字段中列出所有扩展，也可以在|Sec-WebSocket-Extensions| 头信息字段的多个实例之间分割。

如果服务器完成这些步骤而没有中止 WebSocket 握手，则服务器认为 WebSocket 连接已建立，WebSocket 连接处于 OPEN 状态 此时，服务器可以开始发送（和接收）数据。

4.3. 握手时使用的新标头字段 ABNF 汇编

本节使用 [RFC2616] 第 2.1 节中的 ABNF 语法/规则，包括 "隐含 *LWS 规则"。

请注意，本节使用了以下 ABNF 约定。 此类规则表达了相应头域的值，例如，Sec-WebSocket-Key ABNF 规则描述了|Sec-WebSocket-Key| 头域值的语法。 ABNF 规则中带有"。
名称中带有"-Client "后缀的 ABNF 规则仅用于客户端发送给服务器的请求；名称中带有"-Server "后缀的 ABNF 规则仅用于服务器发送给客户端的响应。
例如，ABNF 规则 Sec-WebSocket-Protocol-Client 描述了客户端向服务器发送的 |Sec-WebSocket-Protocol| 头字段值的语法。

在客户端与服务器的握手过程中，可以发送以下新的标头字段：

```
Sec-WebSocket-Key = base64-value-non-non-
empty Sec-WebSocket-Extensions =
extension-list Sec-WebSocket-Protocol-
Client = 1#token Sec-WebSocket-Version-
Client = version

base64-value-non-nempty = (1*base64-data [ base64-padding ]) |
                          base64-padding
base64-data=     4base64 字符
base64-padding=  (2base64 字符"==") |
                   (3base64-character "=")
base64-character = ALPHA | DIGIT | "+" | "/"
extension-list = 1#extension
extension = 扩展令牌 *( ";" 扩展参数 ) extension-token
```

= 注册令牌
注册令牌 = 标记

```
extension-param = token [ "=" (token | quoted-string) ]
     当使用引号字符串语法变体时，值为
     后的字符串必须符合
     ; 'token' ABNF.
NZDIGIT       =  "1" | "2" | "3" | "4" | "5" | "6" |
                 "7" | "8" | "9"
版本 = DIGIT | (NZDIGIT DIGIT) | (NZDIGIT DIGIT)
          ("1 "位数字) | ("2 "位数字)
          仅限于 0-255 范围，不含前导零
```

在服务器与客户端的握手过程中，可以发送以下新的标头字段：

```
Sec-WebSocket-Extensions = 扩展列表
                         Sec-WebSocket-Accept=
base64-value-non-non-empty Sec-WebSocket-
Protocol-Server = token
Sec-WebSocket-Version-Server = 1#version
```

4.4. 支持多种版本的 WebSocket 协议

本节将就客户端和服务器如何支持多个版本的 WebSocket 协议提供一些指导。

使用 WebSocket 版本广告功能（... |Sec-WebSocket-Version|标头字段），客户端最初可以请求其偏好的 WebSocket 协议版本（不一定是客户端支持的最新版本）。 如果服务器支持所请求的版本，且握手消息在其他方面有效，服务器将接受该版本。 如果服务器不支持所请求的版本，则必须响应一个 |Sec-WebSocket-Version|标题字段（或多个 | 此时，如果客户端支持其中一个广告版本，就可以使用新的版本值重复 WebSocket 握手。

下面的示例演示了上述版本协商：

```
GET /chat HTTP/1.1
主机：server.example.com 升级
：websocket 连接：升级
...
Sec-WebSocket-Version: 25
```

服务器的响应可能如下：

```
HTTP/1.1 400 坏请求
...
Sec-WebSocket 版本：13、8、7
```

请注意，服务器最后的响应可能也是这样的：HTTP/1.1 400 坏请求

```
...
Sec-WebSocket-Version: 13
Sec-WebSocket 版本：8、7
```

客户端现在重复符合版本 13 的握手：GET /chat HTTP/1.1

主机：server.example.com 升级

：websocket 连接：升级

```
...
Sec-WebSocket-Version: 13
```

5. 数据框架

5.1. 概述

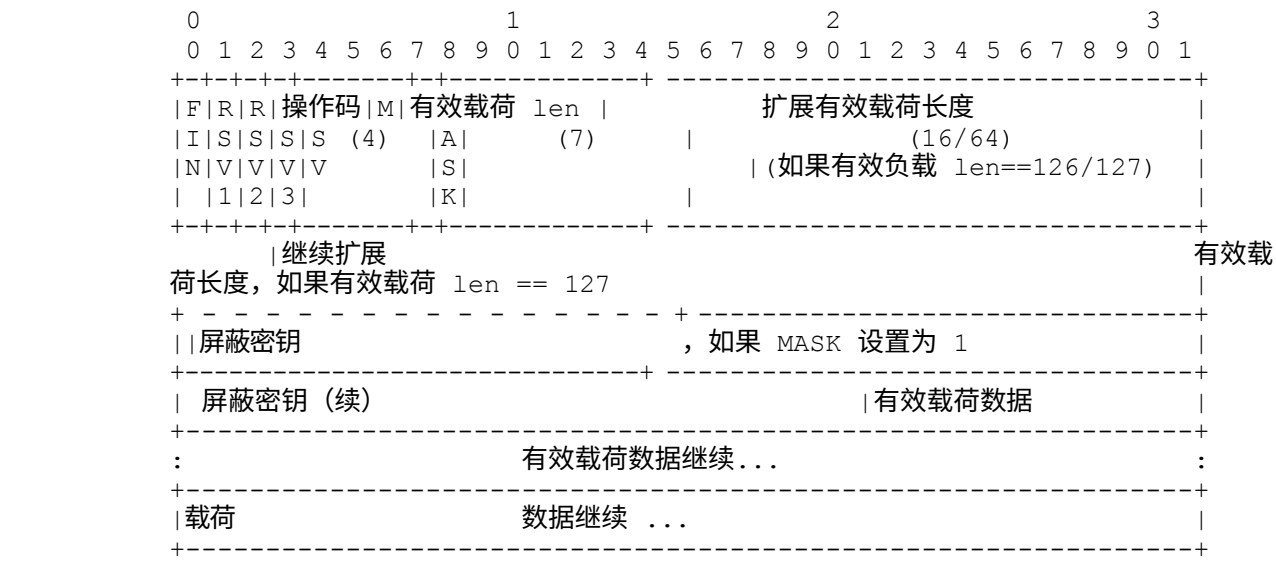
在 WebSocket 协议中，数据是通过帧序列传输的。 为了避免混淆网络中间人（如拦截代理），并出于安全原因（第 10.3 节将进一步讨论），客户端必须屏蔽它发送给服务器的所有帧（更多详情请参见第 5.3 节）。 （请注意，无论 WebSocket 协议是否通过 TLS 运行都要进行屏蔽）。 服务器在收到未屏蔽的帧时必须关闭连接。 在这种情况下，服务器可以发送状态代码为 1002（协议错误）的关闭帧（如第 7.4.1 节所定义）。 服务器不得屏蔽它发送给客户端的任何帧 在这种情况下，它可以使用第 7.4.1 节中定义的状态代码 1002（协议错误）。 （这些规则可能会在未来的规范中放宽）。

基本成帧协议定义了一种带有操作码、有效载荷长度以及 "扩展数据 "和 "应用数据 "指定位置的帧类型，这两种数据共同定义了 "有效载荷数据"。
某些比特和操作码是为协议的未来扩展而保留的。

客户端或服务器可在开放握手完成后、端点发送关闭帧之前的任何时间传输数据帧（第 5.5.1 节）。

5.2. 基本成帧协议

本节详细的ABNF [RFC5234]描述了数据传输部分的导线格式 （请注意，与本文档其他部分不同，本节中的 ABNF 以比特组为单位。 每组比特的长度在注释中标明。 在电线上编码时，ABNF 中最重要的位在最左边）。 下图是 框架的高级概览。 如果下图与本节后面指定的 ABNF 有冲突，则以下图为准。



FIN: 1 位

表示这是报文中的最后一个片段。 第一个片段也可能是最后一个片段。

RSV1、RSV2、RSV3：各 1 位

必须为 0，除非协商的扩展定义了非零值 如果接收到非零值，且协商的扩展均未定义该非零值的含义，则接收端点必须_使 WebSocket连接失败_。

操作码： 4 位

定义了对 "有效载荷数据 "的解释。 如果收到 未知操作码，接收端点必须 _Fail the WebSocket Connection_。 定义了以下值

* %x0 表示继续帧
* %x1 表示文本框
* %x2 表示二进制帧
* %x3-7 预留给其他非控制帧
* %x8 表示连接关闭
* %x9 表示 ping
* %xA 表示庞
* %xB-F 为后续控制帧预留 屏蔽： 1 位

定义是否对 "有效载荷数据 "进行 设置为 1 时，密钥（masking-key）中存在一个屏蔽密钥，根据第 5.3 节该用于解除对 "有效载荷数据 " 所有从客户端发送到服务器的帧都将该位设置为 1。

有效载荷长度： 7 位、7+16 位或 7+64 位

有效载荷数据 "的长度，以字节为单位：如果是 0-125，即有效载荷。 如果是 126，则以下 2 个字节解释为 16 位无符号整数，即为有效载荷长度。 如果是 127，则以下 8 个字节解释为 6

注意，在所有情况下，必须使用最小字节数对长度进行编码，例如，124 字节长的字符串的长度不能编码为序列 126、0、124 有效负载长度为 "扩展数据 "长度 + "应用程序数据 "长度 "扩展数据 "长度可能为

零，在这种情况下，有效负载长度为 "应用程序数据 "长度。

屏蔽密钥： 0 或 4 个字节

从客户端发送到服务器的所有帧都会被一个包含在帧中的 32 位值屏蔽。 该字段为

有关客户端到服务器屏蔽的更多信息，请参见第 5.3 节。

有效载荷数据： （x+y）字节

有效载荷数据 "被定义为 "扩展数据 "与 "应用数据 "的连接。

扩展数据： x 字节

任何扩展都必须 规定 "扩展数据 "的 长度，或规定如何计算该长度，以及如何在开 始握手时协商扩展的使用。如果存在，"扩展数据 "将包含在总有效载荷长度中。

应用数据： y 字节

任意 "应用数据"，占帧中 "扩展数据 "之后的剩余部分。 "应用数据 "的长度等于有效载荷长度减去 "扩展数据 "的长度。

[RFC5234]正式定义。 值得注意的是，这些数据是二进制数据，而不是 ASCII 字符因此， 长度为 1 位、值为 %x0 / %x1字段用单个位表示，其值为 0 或 1，而不是 ASCII 编码中代表字符 "0 "或 "1 "的全字节（八进制）。 长度为 4 位、值在 %x0-F 之间的字段同样用 4 位表示，而不是用 ASCII 字符或具有这些值的完整字节（八进制位组） ： "在 ABNF 中，字符只是一个非负整数 。
在某些情况下，将指定值与字符集（如 ASCII）的特定映射（编码）。" 在这里，指定的编码是二进制编码，其中每个终端值以指定的位数编码每个字段的位数各不相同。

```
    ws-frame                  frame-fin             长度为 1 位 frame-
                                rsv1                ; 长度为 1 位
                                frame-rsv2          ; 长度     为 1 位
                                frame-rsv3          ; 长度     为 1 位
                                帧操作码            ; 长度     为 4 位
                                frame-masked（帧屏蔽）    ; 长度为 1 位
                                frame-payload       -length（帧有效载荷长
                                度）                   ; 7 或 7+16、
                                                       或 7+64 位
                                                       长度
                                [ frame-masking-key ]  长度为 32 位的
                                frame-payload-data     ; 长度 为 n*8 位。
                                                       长度，其中
                                                       ; n >= 0

                              frame-fin= %x0 ; 此信息的更多帧随后出现
                              / %x1 ; 此信息的最后一帧
                                    长度为 1 位

    frame-rsv1=               %x0 / %x1
                                长度为 1 位，必须为 0，除非
                                否则另行协商

    frame-rsv2=               %x0 / %x1
                                长度为 1 位，必须为 0，除非
                                否则另行协商

    frame-rsv3=               %x0 / %x1
                                长度为 1 位，必须为 0，除非
                                否则另行协商

                              frame-opcode= frame-opcode-
                                non-control / frame-
                                opcode-control / frame-
                                opcode-cont

                              frame-opcode-cont= %x0 ;

    帧继续 frame-opcode-non-control= %x1 ; 文本框
                              / %x2 ; 二进制帧
                              / %x3-7
                              长度为 4 位、
                              保留给其他非控制帧

    frame-opcode-control=     %x8 ; 关闭     连接
                              / %x9 ; ping
```

```
   / %xA ; pong
   / %xB-F ; 保留用于进一步控制
           框架
           长度为 4 位
```

```
帧屏蔽= %x0
                            帧未屏蔽，无帧屏蔽键
                            / %x1
                            帧已屏蔽，存在帧屏蔽密钥
                            长度为 1 位

frame-payload-length=    ( %x00-7D )
                            / ( %x7E frame-payload-length-16 )
                            / ( %x7F frame-payload-length-63 )
                            长度为 7、7+16 或 7+64 位、
                            分别为

frame-payload-length-16 = %x0000-FFFF ; 长度为 16 位 frame-

payload-length-63 = %x0000000000000000-7FFFFFFFFFFFFFFF
                            长度为 64 位

帧屏蔽键=                    4( %x00-FF )
                              仅当帧屏蔽为 1 时才存在
                              长度为 32 位

frame-payload-data=          （帧屏蔽扩展数据
                               帧屏蔽的应用程序数据）
                            帧屏蔽为 1 时
                              /（框架无掩码扩展数据 框架无掩码应用数据
                               ）
                            帧屏蔽为 0 时

帧屏蔽扩展数据=                      *( %x00-FF )
                            保留用于未来扩展
                            长度为 n*8 位，其中 n >= 0

帧屏蔽应用数据=                      *( %x00-FF )
                            长度为 n*8 位，其中 n >= 0

帧无掩码扩展数据=                    *( %x00-FF )
                            保留用于未来扩展
                            长度为 n*8 位，其中 n >= 0

帧无掩码应用数据 = *( %x00-FF )
                            长度为 n*8 位，其中 n >= 0
```

5.3. 客户端对服务器屏蔽

根据第 5.2 节的定义，屏蔽帧必须将帧屏蔽字段设置为 1。

屏蔽密钥完全包含在帧中，如第 5.2 节中定义屏蔽密钥（frame-masking-key 用于屏蔽同一节中定义的 "有效载荷数据"（frame-payload-data），其中包括 "扩展数据 "和 "应用数据"。

屏蔽密钥是客户端随机选择的 32 位值。在准备屏蔽帧时，客户端必须从一组允许的 32 位值中选择一个新的屏蔽密钥。 必须是不可预测的；因此，屏蔽密钥必须来自强熵源，而且给定帧的屏蔽密钥不能让服务器/代理机构轻易预测后续帧的屏蔽密钥。 4086 [RFC4086] 讨论了什么是安全敏感型应用的合适熵源。

。 要将屏蔽数据转换为未屏蔽数据，或反之亦然， 以下算法。无论转换的方向如何，都适用相同的算法，例如，屏蔽数据和解除屏蔽数据的步骤相同。

转换后数据的第 i 个八位位组（"转换后-八位位组-i"）是原始数据的第 i 个八位位组（"原始-八位位组-i"）与索引处的八位位组的 XOR i modulo 4 的屏蔽密钥（"masking-key-octet-j"）：

j= i MOD 4
转换八位字节-i = 原始八位字节-i XOR 屏蔽关键八位字节-j

它是 "有效载荷数据 "的长度，屏蔽密钥之后的字节数

5.4. 碎片化

如果信息不能分片，那么端点就必须缓冲整个信息，以便在发送第一个字节前计算其长度。有了分片服务器或中间人就可以选择一个合理大小的缓冲区在缓冲区满时向网络写入一个片段

分片的另一个用例是多路复用，在这种情况下，一个逻辑通道上的大信息不希望垄断输出通道，因此多路复用需要是自由的

将报文分割成更小的片段，以便更好地共享输出通道。 （请注意，本文件未对多路复用扩展进行说明。）

除非扩展另有规定，否则帧没有语义意义。 如果客户端和服务器没有协商扩展，或者协商了一些扩展，但中间人理解所有协商的扩展，并知道如何在存在这些扩展的情况下凝聚和/或分割帧，那么中间人就可以凝聚和/或分割帧。 这意味着，在没有扩展的情况下，发送方和接收方不得依赖于特定帧边界的存在。

以下规则适用于分片：

- 未分片报文由设置了 FIN 位（第 5.2 节）的单帧和 0 以外的操作码组成。

- 片段式报文由 FIN 位清零且操作码不为 0 的单帧、FIN 位清零且操作码设置为 0 的零帧或多帧组成，最后由 FIN 位设置为 0 且操作码为 0 的单帧结束。 片段式报文在概念上等同于一个较大的报文，其有效载荷等于按顺序排列的片段有效载荷的总和；但是，如果存在扩展，则可能不成立，因为扩展定义了对所存在的 "扩展数据 "的解释。 例如，"扩展数据 "可能只出现在第一个片段的开头，并适用于随后的片段，也可能在每个片段中都有 "扩展数据"，但只适用于该特定片段 在没有 "扩展数据 "下面的示例演示了片段是如何工作的。

  举例说明：对于以三个片段形式发送的文本信息，第一个片段的操作码为 0x1，FIN 位清零；第二个片段的操作码为 0x0，FIN 位清零；第三个片段的操作码为 0x0，FIN 位设置。

- 控制帧（见第 5.5 节）可以注入到分片报文的中间。 控制帧本身不得分片。

- 信息片段必须按照发送方发送的顺序传送给收件人。

- 一个报文的片段不得交错在另一个报文的片段之间，除非已协商出一个可解释交错的扩展。
- 端点必须能够处理碎片报文中间的控制帧。
- 发送方可为非控制信息创建任意大小的片段。
- 客户端和服务器必须支持接收碎片和非碎片报文。
- 由于控制帧不能被分片，因此中间人不得试图改变控制帧的分片。
- 如果使用了任何保留位值，而中间人又不知道这些值的含义，则中间人不得更改报文的分片。
- 同样，中介机构如果没有看到导致 WebSocket 连接的 WebSocket 握手（也未被告知其内容），也不得更改此类连接中任何信息的片段。
- 由于控制帧不能被分片，因此报文中所有片段的 类型必须是文本、二进制或某个保留操作码 。

注：如果不能插入控制帧，例如，如果后面有大量信息，ping 的延迟就会非常长。因此，需要在碎报文中间处理控制帧。

实施注意事项：在没有任何扩展的情况下，接收方不必为了处理整个帧而缓冲整个 帧。例如， ，如果使用流 API，帧的一部分就可以传送给应用程序。 但请注意，这一假设可能不会适用于所有未来的 WebSocket 扩展

5.5. 控制框

当前定义的控制操作码包括 0x8（Close）、0x9（Ping）和 0xA（Pong）。 操作码0xB-0xF 保留给尚未定义的其他控制帧。

控制帧用于交流 WebSocket 的状态。控制帧可以在零散信息的中间插入。

所有控制帧的有效载荷长度必须小于或等于 125 字节，且不得分片。

5.5.1. 关闭

关闭帧包含一个 0x8 的操作码。

关闭帧可能包含一个正文（帧的"应用数据 "部分），表示关闭的原因，如端点关闭、端点收到的帧太大或端点收到的帧不符合端点预期的格式。 如果有正文，正文的前两个字节必须是一个 2 字节无符号整数（按网络字节顺序排列），代表第 7.4 节定义的状态代码，其值为 /code/。在 2 字节整数之后，主体可能包含UTF-8 编码的数据，其值为 /reason/，本规范未定义其解释。 这些数据不一定是人类可读的，但可能有助于调试或传递与打开连接的脚本相关的信息。 由于不能保证这些数据是人类可读的，客户端不得将其显示给最终用户

从客户端发送到服务器的关闭帧必须按照第 5.3 节进行屏蔽。

发送关闭帧后，应用程序不得再发送数据帧。

如果端点收到关闭帧，但之前没有发送关闭帧，则端点必须发送关闭帧作为响应。 （当发送关闭帧作为响应时，端点通常会回声它收到的状态代码。） 应在可行的情况下 尽快发送关闭帧。端点可以延迟发送关闭帧，直到其当前信息完毕（例如，如果已发送了大部分碎片信息，端点可以在发送关闭帧之前发送剩余碎片）。 但是，不能保证已发送关闭帧的端点会继续处理数据。

服务器必须立即底层 TCP 连接；客户端应该等待，但可以在发送和接收关闭消息后任何时间例如，如果客户端在合理的时间内没有收到服务器的 TCP 关闭消息。

如果客户端和服务器同时发送关闭消息，则两个端点都将发送和接收关闭消息，并认为 WebSocket 连接已关闭，同时关闭底层 TCP 连接。

5.5.2. 平

Ping 帧包含一个 0x9 的操作码。

Ping 帧可能包括 "应用数据"。

收到 Ping 帧后，端点必须发送 Pong 帧作为回应，除非它已收到关闭帧。 端点应在可行的情况下尽快发送 Pong作为回应。 第 5.5.3 节中讨论

端点可在连接建立后、连接关闭前的任何时间发送 Ping 帧。

注：Ping 帧既可作为保持回应，也可作为验证远程端点是否仍有响应的手段。

5.5.3. 庞

Pong 帧包含一个 0xA 的操作码。

第 5.5.2 节详细介绍了适用于 Ping 和 Pong 帧的要求。

为响应 Ping 帧而发送的 Pong 帧必须具有与被响应 Ping 帧的信息体相同的 "应用数据"。

如果端点接收到 Ping 帧，但尚未发送 Pong 帧响应之前的 Ping 帧，则端点可以选择只为最近处理的 Ping 帧发送 Pong 帧。

可以不经请求发送 Pong 帧。 作为单向心跳 不期望对不经请求的 Pong 帧做出响应。

5.6. 数据帧

数据帧（如非控制帧）由操作码标识，其中操作码的最有效位为 0。 目前定义的 数据帧操作码包括 0x1（文本）、0x2（二进制）。 操作码 0x3-0x7 保留给尚未定义的其他非控制帧。

数据帧携带应用层和/或扩展层数据， 操作码决定数据的解释：

文本

有效载荷数据 "是以 UTF-8 编码的文本数据。 注意，特定文本帧可能包含部分 UTF-8 序列；但整个报文必须包含有效的 UTF-8。 重新组装报文中的无效 UTF-8 将按第 8.1 节所述处理。

二进制

有效载荷数据 "是任意二进制数据，其解释完全由应用层决定。

5.7. 实例

- 单帧无屏蔽文本信息
  - 0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f（包含 "Hello"）。
- 单帧屏蔽文本信息
  - 0x81 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58 （包含 "Hello"）。
- 零散的未屏蔽文本信息
  - 0x01 0x03 0x48 0x65 0x6c（包含 "Hel"）。

* 0x80 0x02 0x6c 0x6f（包含 "lo）

- 非屏蔽 Ping 请求和屏蔽 Ping 响应
  - 0x89 0x05 0x48 0x65 0x6c 0x6c 0x6f（包含 "Hello "正文，但正文内容是任意的）
  - 0x8a 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58（包含 "Hello "正文，与 ping 正文相匹配）
- 单个无掩码帧中的 256 字节二进制信息
  - 0x82 0x7E 0x0100 [256 字节二进制数据]
- 单个无掩码帧中的 64KiB 二进制信息
  - 0x82 0x7F 0x0000000000010000 [65536 字节二进制数据]

5.8. 可扩展性

连接的端点必须在开局握手过程中协商使用任何扩展。 本规范提供了操作码 0x3 至 0x7 和 0xB 至 0xF、"扩展数据 "字段以及帧头的 frame-rsv1、frame-rsv2 和 frame-rsv3 位供扩展使用。 扩展的协商将在第 9.1 节中进一步详细讨论。 以下是扩展的一些预期用途 该列表既不完整，也不规范。

- "扩展数据 "可放在 "有效载荷数据 "中的 "应用数据 "之前。
- 预留位可根据每个帧的需要进行分配。
- 可定义保留的操作码值。
- 如果需要更多操作码值，可将保留位分配给操作码字段。
- 可定义保留位或 "扩展 "操作码，从 "有效载荷数据 "中分配额外的位，以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

要通过 WebSocket 连接_发送由 /data/ 组成的 WebSocket 消息_，端点必须执行以下步骤。

1. 端点必须确保 WebSocket 连接处于打开状态（参见第 4.1 节和第 4.2.2 节）如果WebSocket 连接的状态在任何时候发生变化，端点必须终止以下步骤。

2. 端点必须按照第 5.2 节中的定义将 /data/ 封装在 WebSocket 框架中。 如果要发送的数据量较大，或者在端点希望开始发送数据时数据还未完整可用，端点可以按照第 5.4 节中的定义将数据封装在一系列框架中。

3. 对于接收方将解释为文本或二进制数据的数据，包含数据的第一个帧的操作码（帧操作码）必须设置为第 5.2 节中的适当值。

4. 如第 5.2 节所述，包含数据的最后一帧的 FIN 位（frame-fin）必须设为 1。

5. 如果数据由客户端发送，则必须按照第 5.3 节的规定对帧进行屏蔽。

6. 如果为 WebSocket 连接协商了任何扩展（第 9 节），则可能需要根据这些扩展的定义进行额外的考虑。

7. 已形成的帧必须通过底层网络连接进行传输。

6.2. 接收数据

要接收 WebSocket 数据，端点必须监听底层网络连接。 接收到的数据必须按照第 5.2 节的定义解析为 WebSocket帧。 如果接收到控制帧（第 5.5 节），则必须按照第 5.5 节的定义处理该帧。 接收到数据帧（第 5.6 节）时，端点必须注意以下内容 / 该帧中的 "应用数据为报文/data/ 如果该帧由未分片的报文组成（第 5.4 节），则表示已收到 _A WebSocket 报文_，其类型为 /type/，数据为 /data 当接收到 FIN 位（frame-fin）指示的最后一个片段时，则表示_已收到一条 WebSocket 报文_，其数据为 /（由各片段的 "应用 "连接而成）和 /data/（由各片段的 "应用数据 "连接而成）。

类型/type/（从片段报文的第一帧开始记录）。随后的数据帧必须被解释为属于新的 WebSocket 报文。

扩展（第 9 节）可能会改变数据读取方式的语义，特别是包括信息边界的构成。扩展除了在有效载荷中的 "应用数据 "前添加 "扩展数据 "外，还可以修改 "应用数据"（如对其进行压缩）。

服务器必须按照第 5.3 节所述，为从客户端接收的数据帧去除屏蔽。

7. 关闭连接

7.1. 定义

7.1.1. 关闭 WebSocket 连接

要_关闭 WebSocket 连接_，端点应底层 TCP 端点应使用一种方法来干净利落地关闭 TCP 连接以及 TLS 会话（如适用），并丢弃可能已接收到的任何尾随字节。 端点可在必要时（如 受到攻击时）通过任何可用手段关闭连接。

在大多数正常情况下，底层 TCP 连接应首先由服务器关闭，以便服务器而不是客户端保持 TIME_WAIT 状态（因为这将防止服务器在最大网段生命周期（2MSL）重新打开连接，而服务器不会受到相应的影响，因为 TIME_WAIT 连接会在新的 SYN（具有更高的序列号）出现时立即重新打开）。 在非正常情况下（如在合理的时间后仍未收到服务器的 TCP 关闭），客户端可以启动 TCP 关闭。 因此，当服务器收到关闭 WebSocket 连接的指示时，它应立即启动 TCP 关闭；当客户端收到同样的指示时，它应等待服务器的 TCP 关闭。

举例说明如何在 C 语言中使用伯克利套接字获得干净利落的关闭，方法是在套接字上调用带有 SHUT_WR 的 shutdown()，然后调用 recv()，直到获得 0 的返回值，表明对等程序也已有序关闭，最后在套接字上调用 close()。

7.1.2. 启动 WebSocket 关闭握手

使用状态代码（第 7.4 节）/code/和可选的关闭原因（第 7.1.6 节）_开始 WebSocket 关闭握手
/ 一旦端点发送并接收了关闭控制帧，该端点就应该第 7.1.1 节中的定义_关闭WebSocket 连接_。

7.1.3. 开始 WebSocket 结束握手

在发送或接收到关闭控制帧后，就表示 _The WebSocket Closing Handshake is Started_（WebSocket 关闭握手开始），WebSocket 连接处于关闭状态。

7.1.4. WebSocket 连接已关闭

如果TCP 连接是在 WebSocket 关闭握手完成后的，则表示 WebSocket 连接已_完全。

如果 WebSocket 连接无法建立，也会说 _The WebSocket Connection is Closed_（WebSocket 连接已关闭），但不是_Cleanly_（关闭）。

7.1.5. WebSocket 连接关闭代码

根据第 5.5.1 节和第 7.4 节的定义，关闭控制帧可能包含一个状态代码，表示关闭的原因。 WebSocket 连接的关闭可能由任一端点发起，也可能同时发起。WebSocket 连接关闭代码_被定义为实现协议应用程序收到的第一个关闭控制帧中包含的状态代码（第 7.4 节）。如果该关闭控制帧不包含任何状态代码，_TheWebSocket 连接关闭代码）视为 1005。 如果_The WebSocket Connection is Closed_（WebSocket 连接已关闭）且端点没有收到任何关闭控制帧（如底层传输连接丢失时可能发生的情况），_The WebSocket Connection Close Code_（WebSocket 连接关闭代码）将被视为1006。

举例来说，如果远程端点发送了关闭帧，但本地应用程序尚未从其套接字的接收缓冲区读取包含关闭帧的数据，而本地应用程序独立决定关闭连接并发送关闭帧，那么两个端点都将发送和接收一个

每个端点都会将另一端发送的状态代码视为 _The WebSocket Connection Close Code_（WebSocket 连接代码）。 因此，如果两个端点_Start the WebSocket Closing Handshake_（WebSocket 连接关闭代码）在大致相同的时间独立启动，那么这两个端点有可能在_The WebSocket Connection Close Code_（WebSocket 连接关闭代码）的值上意见不一致。

7.1.6. WebSocket 连接关闭原因

根据第 5.5.1 节和第 7.4 节的定义，关闭控制帧可能包含一个表示关闭原因的状态代码，后面是UTF-8 编码的数据，对上述数据的解释由端点决定，本协议未作定义WebSocket 连接的关闭可能由任一端点发起，也可能同时发起。_The WebSocket Connection Close Reason_（WebSocket 连接关闭原因）被定义为 执行本协议的应用程序收到的第一个关闭控制帧中包含的状态代码（第 7.4 节）之后的 UTF-8 编码数据 如果关闭控制帧中没有此类数据，_The WebSocket Connection Close Reason_（WebSocket 连接关闭原因）即为空字符串。

注意：按照第 7.1.5 节所述的相同逻辑，两个端点可能无法就_The WebSocket Connection Close Reason_（WebSocket 连接关闭原因）达成一致。

7.1.7. 使 WebSocket 连接失败

某些算法和规范要求端点 _Fail the WebSocket Connection_关闭 WebSocket。为此，客户端必须 _Close the WebSocketConnection_（关闭 WebSocket连接），并可以适当的方式向用户报告问题（这对开发人员尤其有用）。同样，服务器也必须_关闭 WebSocket 连接_并记录问题。

如果在要求端点_使 WebSocket 连接失败_的点之前_WebSocket 连接已建立_，端点应在继续_关闭 WebSocket 连接_之前发送带有适当状态代码（第 7.4 节）的关闭帧。如果端点认为对方由于导致 WebSocket 连接失败的错误的性质而不太可能接收和处理关闭帧，那么端点可以省略发送关闭帧。 端点在收到_关闭 WebSocket 连接_的指示后不得继续尝试处理来自远程端点的数据（包括响应的关闭帧）。

除上述情况或应用层（如使用 WebSocket API 的脚本）指定的情况外，客户端不应关

闭连接。

7.2. 异常闭合

7.2.1. 客户主动关闭

某些算法，特别是在开场握手过程中，要求客户端 _Fail the WebSocket Connection_（中断 WebSocket 连接）。 要做到这一点，客户端必须_Fail the WebSocket Connection_（中断 WebSocket 连接），其定义请参阅
第 7.1.7 节.

如果底层传输层连接在任何时候意外丢失，客户端必须_使 WebSocket 连接失败_。

除上述情况或应用层（如使用 WebSocket API 的脚本）指定的情况外，客户端不应关闭连接。

7.2.2. 服务器启动关闭

某些算法要求或建议服务器在打开握手过程中_中止 WebSocket 连接 要做到这一点，服务器必须简单地_关闭 WebSocket 连接_（第 7.1.1 节）。

7.2.3. 从异常闭合中恢复

异常关闭可能由多种原因造成。 这种关闭可能是瞬时错误造成的，在这种情况下，重新连接可能会导致连接良好并恢复正常运行。这种 关闭也可能是非瞬时性问题造成的，在这种情况下， 如果每个部署的客户端都遇到异常关闭，并立即持续尝试重新连接 ，服务器就可能受到大量尝试重新连接的客户端的拒绝服务攻击。 这种情况的最终结果可能是服务无法及时恢复，或恢复变得更加困难

为防止出现这种情况，客户端在异常关闭后尝试重新连接时，应使用本节所述的某种形式的回退。

第一次重新连接尝试应随机延迟一段时间。选择随机延迟时间的参数由客户机决定；在 0 至 5 秒之间随机选择一个值是一个合理的初始延迟时间，但客户机可以根据实施经验和特

定应用选择不同的时间间隔来选择延迟时间长度。

如果第一次重新连接尝试失败，随后的重新连接尝试应该使用截断二进制指数回退等方法，延迟越来越长的时间。

7.3. 连接正常关闭

服务器可以根据需要随时关闭 WebSocket 连接。客户端不应随意关闭 WebSocket 连接。在这两种情况下，端点都可以通过以下步骤启动关闭 _启动 WebSocket 关闭握手_（第 7.1.2 节）。

7.4. 状态代码

在关闭已建立的连接时（例如，在开局握手结束后发送关闭帧时），端点可以指出关闭的原因。本规范未定义端点对该原因的解释以及端点在该原因下应采取的行动本规范定义了一组预定义的状态代码，并规定了扩展、框架和终端应用程序可以使用的范围。状态代码和任何相关的文本消息都是关闭帧的可选组件。

7.4.1. 定义的状态代码

发送关闭帧时，端点可以使用以下预定义状态代码。

1000

1000 表示正常关闭，这意味着建立连接的目的已经达到。

1001

1001 表示一个端点正在 "消失"，例如服务器宕机或浏览器已从页面导航离开。

1002

1002 表示端点因协议错误而终止连接。

1003

1003 表示端点因收到无法接受的数据类型而终止连接（例如，只能理解文本数据的端点在收到二进制信息时可能会发送此信息）。

1004

保留。 具体含义可能会在今后 确定。

1005

1005 是保留值，端点在关闭控制帧时不得将其设置为状态代码。 指定用于期待状态代码的应用程序，以表示实际上没有状态代码。

1006

1006 是保留值，不得在端点的关闭控制帧中设置为状态代码。 指定用于期望用状态代码表示连接非正常关闭的应用程序，例如，在没有发送或接收关闭控制帧

1007

1007 表示端点终止连接，因为它收到的报文数据与报文类型不符（如文本报文中的非 UTF-8 [RFC3629] 数据）。

1008

1008 表示端点因收到违反其策略的信息而终止连接。 这是一个通用状态代码，当没有其他更合适的状态代码（如 1003 或 1009）或需要隐藏策略的具体细节时，就会返回该代码。

1009

1009 表示端点正在终止连接，因为它收到的报文太大，无法处理。

1010

1010 表示端点（客户端）正在终止连接，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 握手的 响应消息中返回这些。

应出现在关闭框架的 /reason/ 部分。请注意，服务器不会使用此状态代码，因为它会导致 WebSocket 握手失败。

1011

1011 表示服务器正在终止连接，因为它遇到了意外情况，无法满足请求。

1015

015 是保留值，不得在关闭控制帧中被端点设置为状态代码。它被指定用于期望状态代码的应用程序，以指示由于 TLS 握手失败（如服务器证书无法验证）而导致连接关闭

7.4.2. 保留状态代码范围 0-999

不使用范围在 0-999 之间的状态代码。

1000-2999

1000-2999 范围内的状态码保留给本协议、本协议未来的修订版以及永久性公开规范中指定的扩展部分。

3000-3999

范围在 3000-3999 之间的状态代码保留给库、框架和应用程序使用。 这些状态代码直接在 IANA 注册。 本协议未定义这些代码的解释。

4000-4999

WebSocket 应用程序之间可通过事先协议使用这些代码。 本协议未定义这些代码的解释。

8. 错误处理

8.1. 处理 UTF-8 编码数据中的错误

当一个端点要将字节流解释为 UTF-8，但发现该字节流实际上不是有效的 UTF-8 流时，该端点必须 _Fail the WebSocket Connection_。 该规则既适用于打开握手过程，也适用于随后的数据交换过程。

9. 扩展

WebSocket 客户端可以请求对本规范的扩展，WebSocket 服务器可以接受客户端请求的部分或全部扩展。 服务器不得响应客户端未请求的任何扩展。 如果扩展参数包含在客户端和服务器之间的协商中，则必须根据参数适用的扩展规范选择这些参数。

9.1. 协商延期

客户端通过包含一个 |Sec-WebSocket- Extensions| 头域来请求扩展，该头域遵循 HTTP 头域的常规规则（见 [RFC2616]，第 4.2 节），头域的值由以下 ABNF [RFC2616] 定义。 请注意，本节使用的是 [RFC2616] 中的 ABNF 语法/规则，包括"隐含 *LWS 规则" 如果客户端或服务器在协商过程中接收到不符合以下 ABNF 的值，则此类畸形数据的接收方必须立即 _Fail the WebSocket Connection_（中断 WebSocket 连接）。

```
Sec-WebSocket-Extensions = extension-list
extension-list = 1#extension
extension = 扩展令牌 *( ";" 扩展参数 ) extension-token
= 注册令牌
注册令牌 = 标记
extension-param = token [ "=" (token | quoted-string) ]
    当使用引号字符串语法变体时，值为
    在引号字符串解转码后，必须符合
    ;'令牌' ABNF.
```

请注意，与其他 HTTP 头信息字段一样，该头信息字段也可以在多行中拆分或组合。因此，以下内容是等价的：

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

完全等同于

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

使用的扩展令牌必须是已注册的令牌（见 与任何给定扩展一起提供的参数是为该扩展定义 注意，客户端只提供使用任何已公布的扩展，除非服务器表示希望使用该扩展，否则不得使用这些扩展。

请注意，扩展的顺序很重要。 多个扩展之间的任何交互都可能在定义扩展的文档中进行定义。 如果没有此类定义，则解释为客户端在请求中列出的标头字段代表其希望使用的标头字段的优先选择，其中列出的第一个选项最可取。 服务器在响应中列出的扩展名代表连接实际使用的扩展名。 如果扩展名修改了数据和/或构架，对数据的操作顺序应假定为与服务器在开局握手响应中列出的扩展名顺序相同。

例如，如果有两个扩展名 "foo "和 "bar"，如果服务器发送的头字段|Sec-WebSocket-Extensions|的值为 "foo, bar"，那么对数据的操作将以 bar(foo(data))的形式进行，无论是对数据本身的更改（如压缩），还是对可能 "堆叠 "的框架的更改。

可接受的扩展标头字段的非规范性示例（注意，为便于阅读，长行被折叠）：

```
Sec-WebSocket 扩展名：deflate-stream
Sec-WebSocket-Extensions: mux; max-channels=4; flow-control,
 deflate-stream
Sec-WebSocket 扩展名：私有扩展名
```

服务器通过包含一个
头信息字段，其中包含客户端请求的一个或多个扩展。

任何扩展参数，以及服务器对客户端请求的参数集的有效响应，将由每个此类扩展来定义。

9.2. 已知扩展

本文档没有定义任何但实现者可以使用单独定义的扩展。

10. 安全考虑因素

本节介绍了适用于WebSocket 协议一些安全注意事项。 具体的安全注意事项将在本节的小节中介绍。

10.1. 非浏览器客户端

WebSocket 协议通过检查 |Origin| 头域（见下文）来防止在网络浏览器等可信应用程序内运行的恶意 JavaScript 更多详情请参见第 1.6 节。 在客户端能力更强的情况下，上述假设并不成立。

这些主机以自己的名义行事，因此可以发送虚假的 |Origin| 头域，误导服务器因此服务器在假定它们直接与来自已知来源的 脚本对话时应小心谨慎必须考虑它们可能会以意想不到的方式被访问。 尤其是，服务器不应相信任何输入都是有效的。

示例：如果服务器使用输入作为 SQL 查询的一部分，则所有输入文本在传递到 SQL 服务器之前都应转义，以免服务器受到 SQL 注入的影响。

10.2. 原产地考虑因素

如果服务器不打算处理来自任何网页的输入，而只打算处理某些网站的输入，则应验证 |Origin| 字段是否为其所期望的 起源。 如果服务器无法接受所指示的起源，则应响应 WebSocket 握手，回复包含 HTTP 403 Forbidden 状态代码的内容。

客户端本身可以联系服务器，并通过"|Origin||Origin|)(|Origin|)(|Origin||)(|Origin||)

这样做的目的不是阻止非浏览器建立连接，而是确保受潜在恶意 JavaScript 控制的可信浏览器无法伪造 WebSocket 握手。

10.3. 对基础设施的攻击（屏蔽）

除了端点会成为 WebSockets 攻击的目标外，网络基础设施的其他部分（如代理）也可能成为攻击的目标。

在开发该协议的过程中，有人在进行了一次实验，以演示对代理服务器的一类攻击，这种攻击会导致部署在野外的缓存代理服务器中毒[TALKING]。 攻击的一般形式是与"攻击者 "控制下的服务器建立连接，在 HTTP 连接上执行类似 WebSocket 协议建立连接的 UPGRADE 操作，然后通过 UPGRADEd 连接发送数据，这些数据看起来像是对特定已知资源（在攻击中可能是广泛部署的用于跟踪命中率 的脚本或广告服务网络上的资源）的 GET 请求。

这种攻击的净效果是，如果用户被说服访问攻击者控制的网站，攻击者就有可能毒化用户和同一缓存后面的其他用户的缓存，并在其他来源上运行恶意从而破坏网络安全模型

为了避免对已部署的中介机构进行此类攻击，仅在应用程序提供的数据前加上不符合 HTTP 协议的框架是不够的，因为不可能详尽无遗地发现和测试每个不符合 HTTP 协议的中介机构不会跳过此类非 HTTP 框架并对框架有效载荷采取不正确的行动。
因此，采用的防御方法是屏蔽从客户端到服务器的所有数据，这样远程脚本（攻击者）就无法控制发送的数据在线路上的显示方式，也就无法构建一个可能被中间人误解为 HTTP 请求的信息。

客户端必须使用提供数据的终端应用程序无法预测的算法，为每一帧选择新的屏蔽密钥。

例如，每次屏蔽可以从密码学上很强的随机数生成器中提取。 如果使用相同的密钥或存在可破译的模式来选择下一个密钥，攻击者就可以发送一个信息，该信息在屏蔽后可能看起来像是："......"。

HTTP 请求（通过获取攻击者希望在网络上看到的信息，并使用下一个屏蔽密钥对其进行屏蔽，当客户端使用屏蔽密钥时，屏蔽密钥将有效地解除对数据的屏蔽）。

此外，一旦开始传输来自客户端的帧，该帧的有效载荷（应用程序提供的数据）就不能被应用程序修改。
否则，攻击者可以发送一个初始数据为已知值（如全为零）的长帧，在收到第一部分数据时计算出正在使用的屏蔽密钥，然后修改帧中尚未发送的数据，使其在屏蔽后显示为 HTTP请求（这与上一段中描述的使用已知或可预测屏蔽密钥的问题基本相同）。
简而言之，一旦开始传输帧，远程脚本（应用程序）就不能修改其内容。

因此，需要屏蔽的通道是从客户端到服务器的数据。
从服务器发送到客户端的数据可以伪造成响应的样子，但要完成这一请求，客户端也必须能够伪造请求。 因此，我们认为没有必要对两个方向的数据都进行屏蔽（从服务器发送到客户端的数据不进行屏蔽）。

尽管屏蔽提供了保护，但不合规的 HTTP 代理仍容易受到未应用屏蔽的客户端和服务器的此类中毒攻击。

10.4. 具体实施限制

（例如，恶意端点可通过发送单个大帧（如大小为 2**60 的帧，或发送作为片段报文帧长流）来耗尽对等设备的内存或发起拒绝服务攻击 这种实现应限制帧的大小和从多个帧 重新组装后的总信息大小。

10.5. WebSocket 客户端验证

WebSocket 服务器可以使用通用 HTTP 服务器可用的任何客户端身份验证机制，如 Cookie、HTTP 身份验证或 TLS 身份验证。

10.6. 连接保密性和完整性

WebSocket 实现必须支持 TLS，并应在与其对等设备通信时使用 TLS。

对于使用 TLS 的连接，TLS 带来的好处在很大程度上取决于 TLS 握手过程中协商的算法强度 例如，某些 TLS 密码机制不提供连接保密性。 要达到合理的保护 级别，客户端应只使用强 TLS 算法。 "Web 安全环境：用户界面指南 [RFC5246] 在附录 A.5 和附录 D.3 中提供了更多指导。

10.7. 处理无效数据

客户端和服务器必须始终对接收到的数据进行验证。如果端点在任何时候遇到无法理解的数据或违反端点确定输入安全性的某些标准的数据，或者端点看到的开放握手与它所期望的值不符（例如，客户端请求中的路径或来源不正确），端点可能会放弃 TCP 连接、incorrect path or origin in the client request), the endpoint MAY drop the TCP connection. If the invalid data was received after a successful WebSocket handshake, the endpoint SHOULD send a Close framewith an appropriate status code (Section 7. 如果无效数据是在 WebSocket 握手期间发送的，服务器应该返回适当的 HTTP [RFC2616] 状态代码

本协议规定，文本数据类型（而不是二进制或其他类型）的报文包含编码的数据。 虽然长度仍有标示，而且执行本协议的应用程序应使用长度来确定帧的实际结束位置，但以不正确编码发送数据会导致安全问题

编码仍有可能打破建立在该协议之上的应用程序可能做出的假设，导致从数据误读到数据丢失或潜在安全漏洞等各种问题。

10.8. 通过 WebSocket 握手使用 SHA-1

本文档中描述的 WebSocket 握手不依赖于 SHA-1 的任何安全特性，例如抗碰撞性或抗第二次预映像攻击（如 [RFC4270] 中所述）。

11. IANA 考虑因素

11.1. 注册新的 URI 计划

11.1.1. 注册 "ws "计划

|ws| URI 可标识 WebSocket 服务器和资源名称。

URI 方案名称 ws

现状
永久性

URI 方案语法
使用 URI 规范 [RFC3986] 中的 ABNF [RFC5234] 语法和 ABNF 终端：

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

<path-abempty> 和 <query> [RFC3986] 组件构成了发送给服务器的资源名称，用于标识所需的服务类型。
其他组件的含义参见 [RFC3986]。

URI 方案语义
该方案的唯一操作是使用 WebSocket 协议打开一个连接。

编码注意事项
主机组件中被上文定义的语法排除在外的字符必须按照或其替代版本的规定从 Unicode 转换为 ASCII 为实现基于方案的规范化，主机组件的国际化域名 (IDN) 形式及其转换为 punycode 的形式被视为等价（参见 [RFC3987] 第 5.3.3 节）。

其他组件中被上述语法排除在外的字符必须从 Unicode 转换为 ASCII，方法是首先将这些字符编码为 UTF-8，然后使用 URI [RFC3986] 和国际化资源标识符 (IRI) [RFC3987] 规范中定义的百分比编码形式替换相应的字节。

使用此 URI 方案的应用程序/协议名称 WebSocket 协议

互操作性考虑因素
使用 WebSocket 需要使用 HTTP 1.1 或更高版本。

安全考虑因素
请参阅 "安全注意事项 "部分。

联系方式
HYBI WG <hybi@ietf.org>

作者/变更控制人 IETF
<iesg@ietf.org>

参考 RFC 6455

11.1.2. 注册 "wss "计划

|wss| URI 可识别 WebSocket 服务器和资源名称，并表明通过该连接的流量将通过 TLS 保护（包括 TLS 的标准优势，如数据保密性和完整性以及端点验证）。

URI 方案名称 wss

现状
永久性

URI 方案语法
使用 URI 规范 [RFC3986] 中的 ABNF [RFC5234] 语法和 ABNF 终端：

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

<path-abempty> 和 <query> 组件构成发送到服务器的 资源名称，用于标识所需的服务类型 。

URI 方案语义
该方案的唯一操作是使用 TLS 加密的 WebSocket 协议打开连接。

编码注意事项
主机组件中被上述语法排除在外的字符，必须按照或其替代版本[RFC3987]的规定从 Unicode 转换为 ASCII。 在基于方案的规范化中，主机组件的 IDN 形式及其转换为 punycode 的形式被认为是等价的（参见 [RFC3987] 第 5.3.3 节）。

其他组件中被上述语法排除在外的字符必须从 Unicode 转换为 ASCII，方法是首先将这些字符编码为 UTF-8，然后使用 URI [RFC3986] 和 IRI [RFC3987] 规范中定义的百分比编码形式替换相应的字节。

使用此 URI 方案的应用程序/协议名称 WebSocket Protocol over TLS

互操作性考虑因素
使用 WebSocket 需要使用 HTTP 1.1 或更高版本。

安全考虑因素
请参阅 "安全注意事项 "部分。

联系方式
HYBI WG <hybi@ietf.org>

作者/变更控制人 IETF
<iesg@ietf.org>

参考 RFC 6455

11.2. 注册 "WebSocket "HTTP 升级关键字

本节定义了根据 RFC 2817 [RFC2817] 在 HTTP 升级标记注册表中注册的关键字。

令牌名称
WebSocket

作者/变更控制人 IETF
<iesg@ietf.org>

联系方式
HYBI <hybi@ietf.org>

参考 RFC 6455

11.3. 注册新的 HTTP 标头字段

11.3.1. Sec-WebSocket-Key

本节介绍在永久报文头字段名注册表 [RFC3864] 中注册的一个报文头字段。

标头字段名 Sec-
WebSocket-Key

适用协议 http

现状
标准

作者/变更控制者 IETF

规范文件 RFC 6455

相关信息
该标头字段仅用于 WebSocket 开启握手。

|Sec-WebSocket-Key| 标头字段用于 WebSocket 开启握手。 它从客户端发送到服务器，提供服务器用来证明收到有效 WebSocket 开启握手的部分信息。 这有助于确保服务器不接受来自非 WebSocket 客户端（如 HTTP 客户端）的连接，这些连接被滥用来向毫无戒心的 WebSocket 服务器发送数据。

在 HTTP 请求中，|Sec-WebSocket-Key| 头信息字段不得出现超过一次。

11.3.2. Sec-WebSocket 扩展

本节介绍在永久报文标头字段名称注册表 [RFC3864] 中注册的标头字段。

标题字段名称
Sec-WebSocket 扩展

适用协议 http

现状
标准

作者/变更控制者 IETF

规范文件 RFC 6455

相关信息
该标头字段仅用于 WebSocket 开启握手。

Sec-WebSocket-Extensions|标头字段用于WebSocket 开启握手过程它最初从客户端发送到服务器，然后再从服务器发送到客户端，以商定在连接期间使用的一组协议级扩展。

在 HTTP 请求中，|Sec-WebSocket-Extensions|（Sec-WebSocket 扩展名）标头字段可以出现多次（这在逻辑上与单个
包含所有值的 |Sec-WebSocket-Extensions| 头信息字段。但是，|Sec-WebSocket-Extensions| 头域在 HTTP 响应中出现的次数不得超过一次。

11.3.3. Sec-WebSocket-Accept

本节介绍在永久报文头字段名注册表 [RFC3864] 中注册的一个报文头字段。

标题字段名称
Sec-WebSocket-Accept

适用协议 http

现状
标准

作者/变更控制者 IETF

规范文件 RFC 6455

相关信息
该标头字段仅用于 WebSocket 开启握手。

Sec-WebSocket-Accept| 标头字段用于WebSocket 开启握手过程。它由服务器发送给客户端，以确认服务器愿意启动 WebSocket连接。

在 HTTP 响应中，|Sec-WebSocket-Accept| 头不得出现超过一次。

11.3.4. Sec-WebSocket 协议

本节介绍在永久报文头字段名注册表 [RFC3864] 中注册的一个报文头字段。

标题字段名称
Sec-WebSocket 协议

适用协议 http

现状
标准

作者/变更控制者 IETF

规范文件 RFC 6455

相关信息
该标头字段仅用于 WebSocket 开启握手。

|Sec-WebSocket-Protocol| 标头字段用于 WebSocket 开启握手。 它从客户端发送到服务器，再从服务器发送回客户端，以确认连接的子协议。这使脚本既能选择子协议，又能确保服务器同意提供该子协议。

在 HTTP 请求中，|Sec-WebSocket-Protocol|（网络套接字协议）标头字段可以出现多次（这在逻辑上与单个
包含所有值的 |Sec-WebSocket-Protocol| 头信息字段）。但是，|Sec-WebSocket-Protocol| 头域在 HTTP 响应中出现的次数不得超过一次。

11.3.5. Sec-WebSocket 版本

本节介绍在永久报文头字段名注册表 [RFC3864] 中注册的一个报文头字段。

标题字段名称
Sec-WebSocket 版本

适用协议 http

现状
标准

作者/变更控制者 IETF

规范文件 RFC 6455

相关信息
该标头字段仅用于 WebSocket 开启握手。

|Sec-WebSocket-Version|标头字段用于 WebSocket 开启握手 它从客户端发送到服务器，用于指示连接的协议版本。 这使服务器能够正确理解开启握手和随后从数据中发送的数据，并在无法以安全的方式解释数据关闭连接。 当从客户端收到的版本与服务器理解的版本不匹配时，|Sec-WebSocket- Version| 标头字段也会在 WebSocket 握手出错时从服务器发送给客户端。 在这种情况下，标头字段包括服务器支持的协议版本。

请注意，我们并不期望较高的版本号一定向后兼容较低的版本号。

Sec-WebSocket-Version|标头字段可以在 HTTP 响应中出现多次（这在逻辑上与单个包含所有值的 |Sec-WebSocket-Version| 头信息字段）。但是，|Sec-WebSocket-Version| 头信息字段在 HTTP 请求中出现的次数不得超过一次。

11.4. WebSocket 扩展名注册表

本规范根据 RFC 5226 [RFC5226] 中规定的原则，为 WebSocket 协议使用的 WebSocket 扩展名创建了一个新的 IANA 注册表。

作为该注册表的一部分，IANA 维护以下信息：扩展标识符
扩展名的标识符，将用于
中注册的|Sec-WebSocket-Extensions|标头字段。
该值必须符合本规范第 9.1 节对扩展标记的要求。

扩展通用名称
扩展名的名称，通常指扩展名。

扩展定义
对文档的引用，该文档定义了与 WebSocket 协议一起使用的扩展。

已知不兼容扩展
已知与该扩展不兼容的扩展标识符列表。

WebSocket 扩展名应遵守 "先到先得 "的 IANA 注册政策 [RFC5226]。

该注册表中没有初始值。

11.5. WebSocket 子协议名称注册表

本规范为 WebSocket 子协议名称创建了一个新的 IANA 注册表，以便根据 RFC 5226 [RFC5226] 中规定的原则与 WebSocket 协议一起使用。

作为注册表的一部分，IANA 维护以下信息：

子协议标识符
子协议的标识符，将用于
| 该值必须符合本规范第 4.1 节第 10 项中的要求--即
该值必须是 RFC 2616 [RFC2616] 中定义的标记。

子协议通用名称
子协议的名称，通常指子协议。

子协议定义
对定义 WebSocket 协议所用子协议的文档的引用。

WebSocket 子协议名称应遵守 "先到先得 "的 IANA 注册政策 [RFC5226]。

11.6. WebSocket 版本号注册表

本规范根据 RFC 5226 [RFC5226] 中规定的原则，为 WebSocket 协议使用的 WebSocket 版本号创建了一个新的 IANA 注册表。

作为注册表的一部分，IANA 维护以下信息：版本号
本规范第 4.1 节规定了|Sec-WebSocket-Version|中使用的版本号。 该值必须是介于 0 和 255（含）之间的非负整数。

参考资料
请求新版本号的 RFC 或带有版本号的草案名称（见下文）。

现状
说明 见下文。

版本号被指定为 "临时 "或 "标准"。

标准 "版本号记录在 RFC 中，用于标识 WebSocket 协议的主要稳定版本，如 本 RFC 所定义的 "标准 "版本号受 "IETF 审查 "IANA 注册政策 [RFC5226]。

临时 "版本号记录在 Internet 草案中，用于帮助实施者识别 WebSocket 协议的已部署版本并与之互操作，例如在本 RFC 发布之前开发的版本 "临时 "版本号须遵守 "专家审查 "IANA 注册政策 [RFC5226]，HYBI 工作组主席（或者，如果工作组关闭，则 IETF 应用程序领域的区域总监）是初始指定专家

IANA 在注册表中添加了以下初始值。

版本编号	版本	状态
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	保留	
10	保留	
11	保留	
12	保留	
13	RFC 6455	标准

11.7. WebSocket 关闭代码编号注册表

本规范根据 RFC 5226 [RFC5226] 中规定的原则，为 WebSocket 连接关闭代码号创建了一个新的 IANA 注册表。

作为注册表的一部分，IANA 维护以下信息：状态代码

状态代码表示 WebSocket 连接关闭的原因，如本文档第 7.4 节 状态代码是介于 1000 和 4999 之间的整数。

意义

状态代码的含义。 每个状态代码都必须有唯一的含义。

联系方式

预订状态代码的实体的联系人。

参考资料

稳定文件请求状态代码并定义其含义。对于范围在 1000-2999 之间的状态代码，需要提供该文件；对于范围在 3000-3999 之间的状态代码，建议提供该文件。

本协议及其后续版本或扩展所使用的状态代码申请应遵守 "标准行动"、"规范要求"（这意味着 "指定专家"）或 "IESG 审核 " 中的任何一项 IANA 注册政策，并应在 1000-2999 范围内批准。

供库、框架和应用程序使用的状态代码申请应遵守 "先到先得 "的 IANA 注册政策，并应在 3000-3999 范围内批准。4000-4999 的状态代码范围指定为私人使用。 申请时应说明是申请 WebSocket 协议（或协议的未来版本）、扩展或库/框架/应用程序使用的 状态代码。

IANA 在注册表中添加了以下初始值。

状态代码	含义	联系人	参考资料
1000	正常闭合	hybi@ietf.org	RFC 6455
1001	Going Away	hybi@ietf.org	RFC 6455
1002	协议错误	hybi@ietf.org	RFC 6455
1003	不支持的数据	hybi@ietf.org	RFC 6455
1004	---Reserved----	hybi@ietf.org	RFC 6455
1005	No Status Rcvd	hybi@ietf.org	RFC 6455
1006	非正常闭合	hybi@ietf.org	RFC 6455
1007	无效帧有效载荷数据	hybi@ietf.org	RFC 6455
1008	Policy Violation	hybi@ietf.org	RFC 6455
1009	信息太大	hybi@ietf.org	RFC 6455
1010	Mandatory Ext.	hybi@ietf.org	RFC 6455
1011	内部服务器错误	hybi@ietf.org	RFC 6455
1015	TLS 握手	hybi@ietf.org	RFC 6455

11.8. WebSocket 操作码注册表

本规范根据 RFC 5226 [RFC5226] 中规定的原则，为 WebSocket 操作码创建了一个新的 IANA 注册表。

作为该注册表的一部分，IANA 维护以下信息：操作码

操作码表示 WebSocket 框架的框架类型，定义见第 5.2 节 操作码是介于 0 和 15 之间的整数，包括 0 和 15。

意义

操作码值的含义。

参考资料
要求操作码的规范。

WebSocket Opcode 编号受 "标准行动 "IANA 注册政策 [RFC5226] 约束。

IANA 在注册表中添加了以下初始值。

操作码	含义	参考文献
RFC 6455		
1	文本框	RFC 6455
RFC 6455		
8	连接关闭	帧 RFC 6455
9	Ping	Frame RFC 6455
10	Pong	Frame RFC 6455

11.9. WebSocket 成帧头位注册表

本规范根据 RFC 5226 [RFC5226] 中规定的原则为 WebSocket 成帧标题位创建了一个新的 IANA 注册表。

这些位将保留给本规范的未来版本或扩展版本使用。

WebSocket 成帧标题位的分配受 "标准行动 "IANA 注册政策 [RFC5226] 约束。

12. 从其他规范中使用 WebSocket 协议

WebSocket 协议的目的是被其他规范使用。
规范，为作者定义的动态内容提供通用机制，例如在定义脚本 API 的规范中。

这样的规范首先需要_建立一个 WebSocket 连接_，为该算法提供

- 目的地，由 /host/ 和 /port/ 组成。

- 资源名称/，允许在一个主机和端口上识别多个服务。
- 一个 /secure/ 标志，如果要对连接进行加密，则该标志为 true，否则为 false。
- 负责连接的起源[RFC6454]的 ASCII 序列化。
- 可选字符串，用于标识要在 WebSocket 连接上分层的协议。

如果 URI 未指定 WebSocket，这些步骤就会失败。

如果在任何时候需要关闭连接，则规范需要使用 _Close the WebSocket Connection 算法
（第 7.1.1 节）。

第 7.1.4 节定义了何时关闭_WebSocket 连接。

当连接打开时，规范需要处理_已收到 WebSocket 消息_的情况（第 6.2 节）。

要向打开的连接发送数据/data/，规范需要_发送 WebSocket 消息_（第 6.1 节）。

13. 致谢

本规范的最初设计得益于WHATWG 和 WHATWG 邮件列表中许多人的参与。 对该规范的贡献没有按章节进行跟踪，但在 http://whatwg.org/html5 的 WHATWG HTML 规范中列出了对该规范做出贡献的所有人的名单。

此外，还要特别感谢 John Tamplin 为本规范的 "数据构架 "部分提供了大量文本。

还要特别感谢 Adam Barth 为本规范的 "数据屏蔽 "部分提供了大量的文本和背景研究。

特别感谢 Lisa Dusseault 负责应用程序领域的审核（并帮助启动这项工作），Richard Barnes 负责 Gen-Art 领域的审核，Magnus Westerlund 负责传输领域 特别感谢 HYBI WG 前任和现任 WG 主席，他们在幕后不懈努力，推动这项工作的完成： 最后特别感谢负责该领域的主任 Peter Saint-Andre。

感谢以下人员参与了 HYBI WG 邮件列表的讨论，并提出了想法和/或提供了详细的评论（该列表可能不完整）：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 FarrellSean Turner、Pete Resnick、Peter Thorson、Joe Mason、John Fallows 和 Alexander Philippou。 注意，以上所列人员并不一定认可本作品的最终结果。

14. 参考资料

14.1. 规范性参考文献

[ANSI.X3-4.1986]
美国国家标准学会，"编码字符集--7 位美国信息交换标准代码"，ANSI X3.4，1986 年。

[FIPS.180-3]
美国国家标准与技术研究院，"安全散列标准"，FIPS PUB 180-3，2008 年 10 月、
<http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf>。

[RFC1928] Leech，M.、Ganis，M.、Lee，Y.、Kuris，R.、Koblas，D.和 L.Jones，"SOCKS 协议版本 5"，RFC 1928，1996 年 3 月
。

[RFC2119]Bradner , S., "RFC 中用于指示要求级别的关键词", BCP 14 , RFC 2119, 1997 年 3 月。

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

[RFC2817] Khare、R. 和 S. Lawrence，"在 HTTP/1.1 内升级到 TLS"，RFC 2817，2000 年 5 月。

[RFC2818] Rescorla，E.，"HTTP Over TLS"，RFC 2818，2000 年 5 月。

[RFC3629] Yergeau，F.，"UTF-8，ISO 10646 的一种转换格式"，STD 63，RFC 3629，2003 年 11 月。

[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 3986，2005 年 1 月。

[RFC3987] Duerst、M. 和 M. Suignard，"国际化资源标识符（IRIs）"，RFC 3987，2005 年 1 月。

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

[RFC4648] Josefsson，S.，《Base16、Base32 和 Base64 数据编码》，RFC 4648，2006 年 10 月。

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

[RFC5234] Crocker, D. 和 P. Overell, "Augmented BNF for Syntax Specifications：ABNF"，STD 68，RFC 5234，2008 年 1 月。

[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：扩展定义"，RFC 6066，2011 年 1 月。

[RFC6454] Barth，A.，"网络起源概念"，RFC 6454，2011 年 12 月。

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. 和 B. Schneier，"对互联网协议中加密哈希值的攻击"，RFC 4270，2005 年 11 月。

[RFC5321] Klensin，J.，"简单邮件传输协议"，RFC 5321，2008 年 10 月。

[RFC6202] Loreto，S.、Saint-Andre，P.、Salsano，S. 和 G. Wilkins，"在双向 HTTP 中使用长轮询和流的已知问题和最佳实践"，RFC 6202，2011 年 4 月。

[RFC6265] Barth，A.，"HTTP 状态管理机制"，RFC 6265，2011 年 4 月。

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

[W3C.REC-wsc-ui-20100812]
Roessler，T. 和 A. Saldhana，"网络安全背景：用户界面指南"，万维网联盟建议书 REC-wsc-ui-201812，2010 年 8 月：用户界面指南》，万维网联盟建议书 REC-wsc-ui-20100812，2010 年 8 月、
<http://www.w3.org/TR/2010/REC-wsc-ui-20100812/>.

最新版本见
<http://www.w3.org/TR/wsc-ui/>。

[WSAPI] Hickson，I.，"The WebSocket API"，W3C 工作草案 WD-websockets-20110929，2011 年 9 月、
<http://www.w3.org/TR/2011/WD-websockets-20110929/>.

最新版本见
<http://www.w3.org/TR/websockets/>。

[XMLHttpRequest]（扩展标记语言请求
van Kesteren，A.，Ed.，"XMLHttpRequest"，W3C 候选建议 CR-XMLHttpRequest-20100803，2010 年 8 月、
<http://www.w3.org/TR/2010/CR-XMLHttpRequest-20100803/>。

最新版本见
<http://www.w3.org/TR/XMLHttpRequest/>。

作者地址

伊恩-费特 谷歌公司

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

阿列克谢-梅尔尼科夫-伊索德有限公司
5 城堡商业村
车站路 36 号
Hampton, Middlesex TW122BX
UK

EMail: Alexey.Melnikov@isode.com