通常 Web 应用的交互模式是由客户端向服务端发送 HTTP 请求,
服务端根据客户端的的请求返回相应的数据, 在这样的交互模式下,
通信双方并不是对等的, 因为所有的请求都是由客户端主动发起, 对于 HTTP/1.x
协议 [RFC 1945], [RFC 2616] 来说,
协议本身并不提供服务端向客户端主动推送数据的机制, 因此基于 HTTP/1.x 的
Web 应用, 若需要获取服务端的数据或状态只能采用不断轮询 (Long Polling)
的方式, 最典型的例子如持续集成软件 Jenkins, 在 Job
构建过程中需要在浏览器向用户展示实时的 Console Output,
如果你在构建过程中进入浏览器的开发者模式便可以看到 Jenkins
采用周期性地向服务端发送请求以拉取实时的 Console 输出数据, 再例如一些基于
Web 的网络游戏, 例如 FPS 类游戏, 客户端需要知道当前实时的全局状态,
如其它玩家当前的坐标, 装备等, 如果使用 HTTP/1.x
协议则只能采用不断轮询服务器的方式以获得最新的状态数据,
这种方式一方面效率不高, 而且不够实时, 消息的实时性取决于两次轮询的时间差
(Gap), 最坏情况下需要晚于 1 个 Gap 才能拉到最新的数据,
另一方面频繁地轮询也增加了服务端额外的负载,
客户端需要单独维持一个连接用于轮询服务器状态, WebSocket
协议便是为了解决这个问题, WebSocket 协议提供了一种全双工的通信机制,
服务端可以主动向客户端推送数据, WebSocket 协议采用了 HTTP 协议来握手, 与
HTTP 使用相同的默认端口, 这一切都是为了兼容现有的 HTTP 组件或代理, 但
WebSocket 与 HTTP 是相互独立的协议, 二者并不存在上下的层级关系,
WebSocket 的正式协议文档为 [RFC 6455], 本文全面讨论 WebSocket
协议的设计与工作原理

1. WebSocket 协议概述

WebSocket 协议主要为了解决基于 HTTP/1.x 的 Web 应用无法实现服务端向客户端主动推送的问题, 为了兼容现有的设施, WebSocket 协议使用与 HTTP 协议相同的端口, 并使用 HTTP Upgrade 机制来进行 WebSocket 握手, 当握手完成之后, 通信双方便可以按照 WebSocket 协议的方式进行交互

WebSocket 使用 TCP 作为传输层协议, 与 HTTP 类似, WebSocket 也支持在 TCP 上层引入 TLS 层, 以建立加密数据传输通道, 即 WebSocket over TLS, WebSocket 的 URI 与 HTTP URI 的结构类似, 对于使用 80 端口的 WebSocket over TCP, 其 URI 的一般形式为 ws://host:port/path/query 对于使用 443 端口的 WebSocket over TLS, 其 URI 的一般形式为 wss://host:port/path/query

在 WebSocket 协议中, 帧 (frame) 是通信双方数据传输的基本单元, 与其它网络协议相同, frame 由 Header 和 Payload 两部分构成, frame 有多种类型, frame 的类型由其头部的 Opcode 字段 (将在下面讨论) 来指示, WebSocket 的 frame 可以分为两类, 一类是用于传输控制信息的 frame (如通知对方关闭 WebSocket 连接), 一类是用于传输应用数据的 frame, 使用 WebSocket 协议通信的双方都需要首先进行握手, 只有当握手成功之后才开始使用 frame 传输数据

2. WebSocket 握手

当客户端想要使用 WebSocket 协议与服务端进行通信时, 首先需要确定服务端是否支持 WebSocket 协议, 因此 WebSocket 协议的第一步是进行握手, WebSocket 握手采用 HTTP Upgrade 机制, 客户端可以发送如下所示的结构发起握手 (请注意 WebSocket 握手只允许使用 HTTP GET 方法):

```
GET /chat HTTP/1.1Host: server.example.comUpgrade: websocket
Connection: UpgradeSec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ
==Origin: http://example.comSec-WebSocket-
Protocol: chat, superchatSec-WebSocket-Version: 13
```

在 HTTP Header 中设置 Upgrade 字段, 其字段值为 websocket, 并在 Connection 字段指示 Upgrade, 服务端若支持 WebSocket 协议, 并同意握手, 可以返回如下所示的结构:

```
HTTP/1.1 101 Switching ProtocolsUpgrade: websocketConnection
: UpgradeSec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=S
```

```
ec-WebSocket-Protocol: chatSec-WebSocket-Version: 13
```

我们来详细讨论 WebSocket 的握手细节, 客户端发起握手时除了设置 Upgrade 之外, 还需要设置其它的 Header 字段

- | Sec-WebSocket-Key |, 必传, 由客户端随机生成的 16 字节值, 然后做 base64 编码, 客户端需要保证该值是足够随机, 不可被预测的 (换句话说, 客户端应使用熵足够大的随机数发生器), 在 WebSocket 协议中, 该头部字段必传, 若客户端发起握手时缺失该字段, 则无法完成握手
- | Sec-WebSocket-Version |, 必传, 指示 WebSocket 协议的版本, RFC 6455 的协议版本为 13, 在 RFC 6455 的 Draft 阶段已经有针对相应的 WebSocket 实现, 它们当时使用更低的版本号, 若客户端同时支持多个 WebSocket 协议版本, 可以在该字段中以逗号分隔传递支持的版本列表 (按期望使用的程序降序排列), 服务端可从中选取一个支持的协议版本
- | Sec-WebSocket-Protocol |, 可选, 客户端发起握手的时候可以在头部设置该字段, 该字段的值是一系列客户端希望在于服务端交互时使用的子协议 (subprotocol), 多个子协议之间用逗号分隔, 按客户端期望的顺序降序排列, 服务端可以根据客户端提供的子协议列表选择一个或多个子协议
- | Sec-WebSocket-Extensions |, 可选, 客户端在 WebSocket 握手阶段可以在头部设置该字段指示自己希望使用的 WebSocket 协议拓展

服务端若支持 WebSocket 协议, 并同意与客户端握手, 则应返回 101 的 HTTP 状态码, 表示同意协议升级, 同时应设置 Upgrade 字段并将值设置为 websocket, 并将 Connection 字段的值设置为 Upgrade, 这些都是与标准 HTTP Upgrade 机制完全相同的, 除了这些以外, 服务端还应设置与 WebSocket 相关的头部字段:

- | Sec-WebSocket-Accept |, 必传, 客户端发起握手时通过 | Sec-WebSocket-Key | 字段传递了一个将随机生成的 16 字节做 base64 编码后的字符串, 服务端若接收握手, 则应将该值与 WebSocket 魔数 (Magic Number) "258EAFA5-E914-47DA- 95CA-C5AB0DC85B11" 进行字符串连接, 将得到的字符串做 SHA-1 哈希, 将得到的哈希值再做 base64 编码, 最终的值便是该字段的值, 举例来说, 假设客户端传递的 Sec-WebSocket-Key 为 "dGhlIHNhbXBsZSBub25jZQ==", 服务端应首先将该字符串与 WebSocket 魔数进行字符串拼接, 得到 "dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CA-C5AB0DC85B11", 然后对该字符串做 SHA-1 哈希运算得到哈希值 0xb3 0x7a 0x4f 0x2c 0xc0 0x62 0x4f 0x16 0x90 0xf6 0x46 0x06 0xcf 0x38

0x59 0x45 0xb2 0xbe 0xc4 0xea, 然后对该哈希值做 base64 编码,
最终得到 Sec-WebSocket-Accept 的值为
s3pPLMBiTxaQ9kYGzzhZRbK+xOo=,
当客户端收到服务端的握手响应后,
会做同样的运算来校验该值是否符合预期, 以便于判断服务端是否真的支持 WebSocket 协议, 设置这个环节的目的就是为了最终校验服务端对 WebSocket 协议的支持性, 因为单纯使用 Upgrade 机制,
对于一些没有正确实现 HTTP Upgrade 机制的 Web Server,
可能也会返回预期的 Upgrade, 但实际上它并不支持 WebSocket, 而引入 WebSocket
魔数并进行这一系列操作后可以很大程度上确定服务端确实支持 WebSocket 协议

- | Sec-WebSocket-Protocol |, 可选, 若客户端在握手时传递了希望使用的 WebSocket 子协议,
则服务端可在客户端传递的子协议列表中选择其中支持的一个,
服务端也可以不设置该字段表示不希望或不支持客户端传递的任何一个 WebSocket 子协议
- | Sec-WebSocket-Extensions |, 可选, 与 Sec-WebSocket-Protocol 字段类似, 若客户端传递了拓展列表,
可服务端可从中选择其中一个做为该字段的值,
若服务端不支持或不希望使用这些扩展, 则不设置该字段
- | Sec-WebSocket-Version |, 必传, 服务端从客户端传递的支持的 WebSocket 协议版本中选择其中一个, 若客户端传递的所有 WebSocket 协议版本对服务端来说都不支持, 则服务端应立即终止握手, 并返回 HTTP 426 状态码, 同时在 Header 中设置 | Sec-WebSocket-Version | 字段向客户端指示自己所支持的 WebSocket 协议版本列表

服务端若接收客户端的握手, 便按上述所表述的规则向客户端返回握手响应,
客户端对服务端返回的握手响应做校验, 若校验成功, 则 WebSocket 握手成功,
之后双方就可以开始进行双向的数据传输。客户端在发起握手后必须处于阻塞状态,
换句话说, 客户端必须等待服务端发回响应之后才允许开始数据传递,
客户端对服务端的握手响应的校验机制如下:

- 客户端应首先检查服务端返回的状态码是否为 101, 只有在 HTTP 状态码为 101 时才代表服务端同意了协议升级, 对于其它类型的状态码,
客户端应根据 HTTP 状态码的语义做相应的处理
- 客户端应检查服务端返回的响应是否包含 Upgrade 字段, 若缺失, 代表 Upgrade 未成功, 客户端应终止 WebSocket 握手
- 客户端应检查 Upgrade 字段的值是否为 websocket

(该字段是大小写不敏感的, 如 websocket, WebSocket, webSocket 等都是合法的), 若不是, 客户端应终止 WebSocket 握手
- 客户端应采用如上所表述的方式校验服务端返回的 Sec-WebSocket-Accept 字段的值是否合法, 若该字段不存在或值不符合预期, 则客户端应终止 WebSocket 握手
- 若服务端返回的 Header 中包含 Sec-WebSocket-Extensions, 但其字段的值并不在客户端最初向服务端发起握手时传递的 Sec-WebSocket-Extensions 的值列表中, 则客户端应终止 WebSocket 握手
- 若服务端返回的 Header 中包含 Sec-WebSocket-Protocol, 但该字段的值并不在客户端最初向服务端发起握手时传递的 Sec-WebSocket-Protocol 的值列表中, 则客户端应终止 WebSocket 握手

若客户端校验服务端的握手响应通过, 则 WebSocket 握手阶段完成, 接下来双方就可以进行 WebSocket 的双向数据传输了

3. WebSocket 数据帧 (frame)

WebSocket 以 frame 为单位传输数据, frame 是客户端和服务端数据传输的最小单元, 当一条消息过长时, 通信方可以将该消息拆分成多个 frame 发送, 接收方收到以后重新拼接、解码从而还原出完整的消息, 在 WebSocket 中, frame 有多种类型, frame 的类型由 frame 头部的 Opcode 字段指示, WebSocket frame 的结构如下所示: