Công ty Cổ phần VCCorp

BÁO CÁO TUẦN 3

Tìm hiểu về WebSocket

Tác giả: Trần Văn Toàn

Người hướng dẫn: Anh Ngô Văn Vĩ

Hà Nội, tháng 7 năm 2025

Tóm tắt nội dung

Báo cáo này trình bày về WebSocket - một giao thức quan trọng cho phép giao tiếp hai chiều liên tục giữa client và server. Nội dung báo cáo bao gồm định nghĩa, lý do sử dụng, cách thiết lập và các lưu ý khi sử dụng WebSocket trong phát triển ứng dụng web thời gian thực.

Mục lục

Tóm tắt nội dung **1**

1 Giới thiệu về WebSocket **4**

1.1 Định nghĩa WebSocket 4

1.2 Ứng dụng của WebSocket 5

2 Lý do sử dụng WebSocket **6**

2.1 Hạn chế của giao tiếp Half-Duplex trong HTTP 6

2.2 Yêu cầu của ứng dụng Web hiện đại 6

2.3 WebSocket - Giải pháp Full-Duplex 7

3 Thiết lập và sử dụng WebSocket **8**

3.1 Thiết lập kết nối WebSocket 8

3.1.1 Quá trình Handshake 8

3.2 Giao tiếp qua WebSocket 9

3.2.1 Các sự kiện (Events) 9

3.2.2 Các phương thức (Methods) 9

3.2.3 Xử lý dữ liệu nhị phân 10

3.3 Thuộc tính của WebSocket 10

4 Lưu ý khi sử dụng WebSocket **11**

4.1 Quản lý kết nối 11

4.2 Bảo mật 11
4.3 Hiệu suất 11

5 Kết luận **13**
5.1 Những điểm chính cần ghi nhớ 13
5.2 Hướng phát triển 13

6 Tài liệu tham khảo **15**

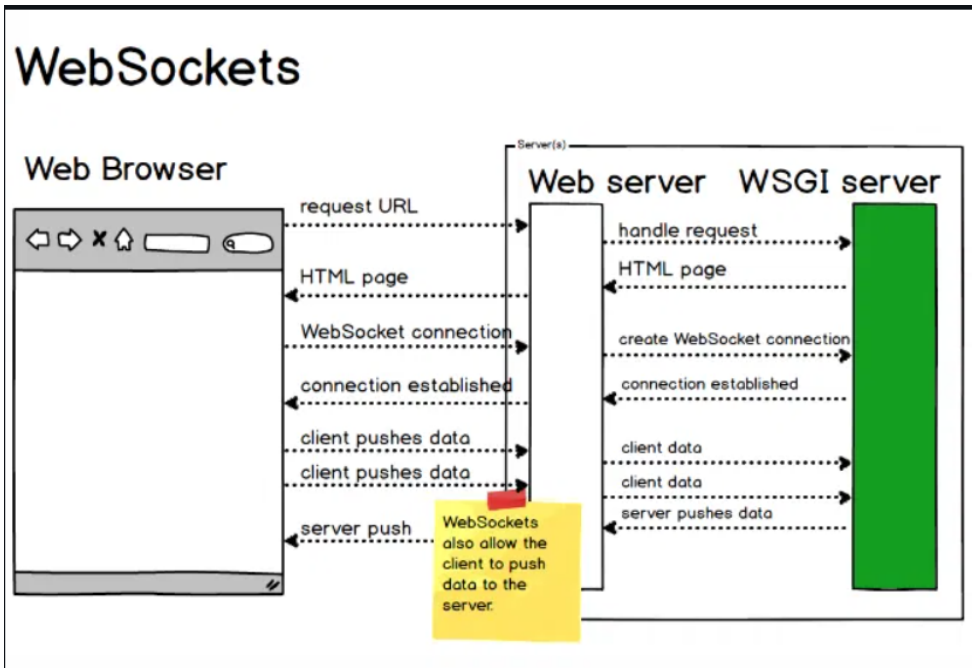
Chương 1

Giới thiệu về WebSocket

1.1 Định nghĩa WebSocket

Định nghĩa

WebSocket là một giao thức tiêu chuẩn hiện đại cho phép tạo ra kết nối hai chiều liên tục giữa client và server, hỗ trợ giao tiếp thời gian thực (real-time) trong các ứng dụng web.



Hình 1.1: Hoạt động của Client và Server với WebSocket

Trước khi WebSocket ra đời, các ứng dụng web chủ yếu sử dụng hai phương pháp

chính:

- **Polling**: Client liên tục gửi yêu cầu đến server để kiểm tra thông tin mới. Phương pháp này không hiệu quả vì tạo ra nhiều kết nối không cần thiết, đặc biệt khi tần suất tin nhắn không thể đoán trước.
- **Long Polling (Comet)**: Client gửi yêu cầu và server giữ kết nối mở cho đến khi có dữ liệu mới hoặc hết thời gian chờ. Mặc dù tốt hơn Polling, client vẫn phải liên tục kết nối lại.

WebSocket giải quyết các vấn đề của Polling và Long Polling bằng cách cung cấp một kết nối ổn định duy nhất cho việc giao tiếp hai chiều.

1.2 Ứng dụng của WebSocket

Sử dụng WebSocket, bạn có thể tạo ra những ứng dụng real-time như:

- Ứng dụng trò chuyện (chat)
- Chỉnh sửa tài liệu online (ví dụ: Google Docs)
- Giao dịch tài chính trực tuyến
- Game online nhiều người chơi
- Dashboard theo dõi dữ liệu thời gian thực

Chương 2

Lý do sử dụng WebSocket

2.1 Hạn chế của giao tiếp Half-Duplex trong HTTP

Giao thức HTTP, đặc biệt là các phiên bản 1.0 và 1.1, được xây dựng dựa trên mô hình Client-Server truyền thống. Trong mô hình này, luồng giao tiếp diễn ra theo một chiều tại một thời điểm: Client gửi yêu cầu (request), Server xử lý và gửi lại phản hồi (response).

Định nghĩa

Half-Duplex có nghĩa là dữ liệu chỉ có thể truyền theo một hướng tại một thời điểm, giống như việc sử dụng bộ đàm - một người phải nói xong thì người kia mới có thể trả lời.

Các hạn chế của Half-Duplex:

- Độ trễ cao do phải chờ phản hồi
- Chi phí tài nguyên lớn do tạo nhiều kết nối
- Không phù hợp cho ứng dụng thời gian thực
- Phức tạp trong việc triển khai

2.2 Yêu cầu của ứng dụng Web hiện đại

Các ứng dụng web ngày nay yêu cầu:

- Tương tác thời gian thực
- Trải nghiệm người dùng mượt mà
- Cập nhật dữ liệu tức thời
- Giao tiếp hai chiều hiệu quả

Các giải pháp tạm thời được sử dụng trước WebSocket:

- AJAX Long-Polling
- Server-Sent Events (SSE)
- Comet

Tuy nhiên, các phương pháp này vẫn là "cách giải quyết tạm thời" dựa trên nền tảng Half-Duplex của HTTP và có những hạn chế nhất định.

2.3 WebSocket - Giải pháp Full-Duplex

Định nghĩa

Full-Duplex là hệ thống liên lạc cho phép cả hai bên có thể gửi và nhận dữ liệu cùng một lúc.

Quy trình hoạt động của WebSocket:

1. Thực hiện "handshake" ban đầu qua HTTP
2. Nâng cấp kết nối lên WebSocket
3. Thiết lập kết nối bền vững
4. Cả Client và Server có thể chủ động gửi dữ liệu bất cứ lúc nào

Chương 3

Thiết lập và sử dụng WebSocket

3.1 Thiết lập kết nối WebSocket

Để bắt đầu sử dụng WebSocket, bạn cần tạo một kết nối bằng cách khởi tạo đối tượng WebSocket:

Listing 3.1: Tạo kết nối WebSocket

```
// Tao ket noi don gian
var ws = new WebSocket("ws://server.example.com/chat");

// Tao ket noi voi giao thuc con(sub-protocol)
var ws = new WebSocket(url, 'my-protocol');
```

3.1.1 Quá trình Handshake

Quy trình thiết lập kết nối WebSocket gồm các bước:

1. **Client gửi yêu cầu nâng cấp**: Client gửi yêu cầu HTTP với tiêu đề `Upgrade: websocket` và khóa `Sec-WebSocket-Key`
2. **Server xác nhận**: Server trả về mã trạng thái `101 Switching Protocols` và khóa `Sec-WebSocket-Accept`
3. **Kết nối được thiết lập**: Client xác nhận khóa từ server và kết nối WebSocket chính thức được mở

3.2 Giao tiếp qua WebSocket

WebSocket là một API hướng sự kiện với các thành phần chính:

3.2.1 Các sự kiện (Events)

- **onopen**: Kích hoạt khi kết nối sẵn sàng
- **onmessage**: Kích hoạt khi nhận tin nhắn từ server (dữ liệu trong `event.data`)
- **onerror**: Kích hoạt khi có lỗi xảy ra
- **onclose**: Kích hoạt khi kết nối bị đóng

3.2.2 Các phương thức (Methods)

- **ws.send(data)**: Gửi dữ liệu (chuỗi văn bản, Blob, hoặc ArrayBuffer) đến server
- **ws.close()**: Đóng kết nối

Listing 3.2: Ví dụ sử dụng WebSocket

```
// Thiet lap cac su kien
ws.onopen = function(event) {
    console.log("Ket noi da duoc thiet lap");
    ws.send("Hello Server!");
};
ws.onmessage = function(event) {
    console.log("Nhan tin nhan:", event.data);
};
ws.onerror = function(error) {
    console.log("Loi:", error);
};
ws.onclose = function(event) {
    console.log("Ket noi da dong");
};
```

3.2.3 Xử lý dữ liệu nhị phân

WebSocket có thể gửi dữ liệu dạng nhị phân để tăng hiệu suất:

Listing 3.3: Cấu hình dữ liệu nhị phân

```
// Thiet lap ding dang nhi phan
socket.binaryType = "blob";        // Mac dinh
socket.binaryType = "arraybuffer"; // Hoac arraybuffer
```

3.3 Thuộc tính của WebSocket

Các thuộc tính chỉ đọc (read-only) để theo dõi trạng thái:

- **readyState**: Trạng thái hiện tại của kết nối
  - 0: Đang kết nối (CONNECTING)
  - 1: Đã kết nối (OPEN)
  - 2: Đang đóng (CLOSING)
  - 3: Đã đóng (CLOSED)
- **bufferedAmount**: Số byte dữ liệu đang chờ được gửi
- **protocol**: Giao thức con mà server đã chọn
- **url**: URL đã dùng để tạo kết nối

Chương 4

Lưu ý khi sử dụng WebSocket

4.1 Quản lý kết nối

Lưu ý

- Luôn kiểm tra trạng thái kết nối trước khi gửi dữ liệu
- Xử lý việc mất kết nối và tự động kết nối lại
- Sử dụng heartbeat để duy trì kết nối
- Đóng kết nối khi không sử dụng để tiết kiệm tài nguyên

4.2 Bảo mật

- Sử dụng WSS (WebSocket Secure) thay vì WS trong môi trường production
- Xác thực người dùng trước khi thiết lập kết nối
- Validate dữ liệu từ client trước khi xử lý
- Giới hạn số lượng kết nối từ một IP

4.3 Hiệu suất

- Kiểm tra `bufferedAmount` để tránh gửi quá nhiều dữ liệu

- Sử dụng compression để giảm băng thông
- Tối ưu hóa tần suất gửi tin nhắn
- Sử dụng connection pooling cho server

Chương 5

Kết luận

WebSocket đã mang lại một cuộc cách mạng trong phát triển ứng dụng web thời gian thực. Với khả năng giao tiếp hai chiều liên tục, WebSocket giải quyết được các hạn chế của HTTP truyền thống và mở ra nhiều khả năng mới cho việc phát triển ứng dụng web hiện đại.

5.1 Những điểm chính cần ghi nhớ

- WebSocket cung cấp kết nối Full-Duplex, cho phép giao tiếp hai chiều đồng thời
- Hiệu quả hơn so với Polling và Long Polling
- Phù hợp cho ứng dụng thời gian thực như chat, game, dashboard
- Cần lưu ý về bảo mật và quản lý kết nối
- Sử dụng WSS trong môi trường production

5.2 Hướng phát triển

WebSocket sẽ tiếp tục đóng vai trò quan trọng trong:

- Internet of Things (IoT)
- Ứng dụng di động
- Microservices architecture

- Real-time analytics

Hiểu rõ và sử dụng thành thạo WebSocket sẽ giúp các lập trình viên xây dựng được những ứng dụng web hiện đại, mạnh mẽ và có trải nghiệm người dùng tốt.

Chương 6

Tài liệu tham khảo

1. HTTP/2 Overview. https://200lab.io/blog/http2-la-gi/
2. Introduction to WebSockets. https://www.linode.com/docs/guides/introduction-to-websocket
3. WebSocket API Documentation. https://developer.mozilla.org/en-US/docs/Web/API/WebSocket
4. RFC 6455 - The WebSocket Protocol. https://tools.ietf.org/html/rfc6455