개요

WebSocket은 웹 애플리케이션이 서버와의 양방향 통신을 가능하게 하는 프로토콜입니다. 이 프로토콜은 단일 TCP(Transmission Control Protocol) 연결을 통해 실시간 데이터를 주고받을 수 있는 효율적인 방법을 제공합니다. WebSocket은 2011년에 IETF(Internet Engineering Task Force)에 의해 RFC 6455로 표준화되었습니다.

주요 특징

1. **양방향 통신**: WebSocket은 클라이언트와 서버 간의 실시간 상호작용을 지원합니다. 클라이언트와 서버는 언제든지 메시지를 주고받을 수 있습니다.
2. **경량화**: HTTP 요청-응답 방식과 달리, WebSocket은 연결을 유지하면서 데이터 전송 시 오버헤드가 적습니다. 이는 실시간 애플리케이션에서 중요한 요소입니다.
3. **비동기성**: WebSocket은 비동기적으로 작동하여 클라이언트가 서버로부터 메시지를 수신할 때까지 기다릴 필요가 없습니다.
4. **프로토콜 호환성**: WebSocket은 HTTP와 호환되도록 설계되어 있어, 기존의 HTTP 인프라(예: 방화벽, 프록시)를 통해 통신할 수 있습니다.

주의 사항

1. **서버 자원 소모**:
    - WebSocket 연결이 활성화되어 있는 동안 서버에서 해당 연결을 유지해야 하므로, 많은 클라이언트가 동시에 연결될 경우 서버 자원이 많이 소모될 수 있습니다.
2. **보안 문제**:
    - WebSocket은 동일 출처 정책의 제한을 받지 않으므로, 공격자가 악용할 수 있는 가능성이 있습니다. 따라서 올바른 인증 및 보안 조치를 취해야 합니다.
3. **복잡한 구현**:
    - WebSocket은 상태 유지를 위해 추가적인 코드와 관리가 필요할 수 있으며, HTTP에 비해 구현이 더 복잡합니다.
4. **네트워크 상태 의존성**:
    - 클라이언트와 서버 간의 연결이 지속적으로 유지되기 때문에, 네트워크가 불안정한 경우 연결이 끊길 수 있습니다. 이 경우 클라이언트는 재연결 로직을 구현해야 합니다.
        - 연결이 비정상적으로 끊긴 경우, 지수 백오프 알고리즘을 사용하여 지연 시간을 점점 늘려가며 재연결을 시도하는 알고리즘을 구현하여 재연결 로직을 구현할 수 있습니다.
5. **브라우저 지원 문제**:

- 대부분의 현대 브라우저에서 WebSocket을 지원하지만, 오래된 브라우저에서는 지원되지 않을 수 있습니다. 따라서 모든 사용자가 최신 브라우저를 사용하는 것은 아닙니다.

WebSocket의 동작 과정

1. 오프닝 핸드셰이크

오프닝 핸드셰이크는 클라이언트와 서버 간의 WebSocket 연결을 수립하는 과정입니다.

과정:

1. **클라이언트 요청**:
   - 클라이언트가 WebSocket 서버에 연결하기 위해 HTTP GET 요청을 보냅니다. 이 요청은 다음과 같은 헤더를 포함합니다.

```
GET /socket HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Version: 13
Origin: http://example.com
```

- **헤더 설명**:
  - `Host` : 요청할 서버의 호스트 이름.
  - `Upgrade` : WebSocket 프로토콜로 전환하겠다는 의사 표시.
  - `Connection` : 연결을 업그레이드할 것이라는 지시.
  - `Sec-WebSocket-Key` : 클라이언트가 생성한 랜덤 키. 서버는 이를 바탕으로 응답을 생성.
  - `Sec-WebSocket-Version` : 클라이언트가 지원하는 WebSocket 버전.
  - `Origin` : 클라이언트의 출처를 명시.

2. **서버 응답**:
   - 서버는 클라이언트의 요청을 받고, 이를 검증한 후 응답합니다. 성공적인 응답은 다음과 같습니다:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
```

- **헤더 설명**:
  - `101 Switching Protocols` : WebSocket 프로토콜로의 전환을 의미.
  - `Sec-WebSocket-Accept` : 클라이언트가 보낸 `Sec-WebSocket-Key` 를 바탕으로 생성된 해시값. 이를 통해 클라이언트와 서버 간의 안전한 연결을 확인.

이 단계가 완료되면 클라이언트와 서버 간의 WebSocket 연결이 수립됩니다.

2. 데이터 전송

WebSocket 연결이 성공적으로 수립되면, 클라이언트와 서버는 실시간으로 데이터를 주고받을 수 있습니다. 이 데이터 전송은 두 가지 유형의 메시지를 포함합니다: **데이터 메시지**와 **제어 메시지**.

데이터 메시지

- **구조**: 각 메시지는 하나 이상의 프레임으로 구성됩니다. 프레임 구조는 다음과 같습니다:
  - **FIN**: 최종 프레임 여부 (1 비트).
  - **Opcode**: 메시지의 유형 (4 비트).
    - 텍스트 메시지(1) 또는 바이너리 메시지(2).
  - **Masked**: 클라이언트가 보낸 프레임인 경우 마스킹 여부 (1 비트).
  - **Payload length**: 메시지의 길이.
  - **Payload**: 실제 데이터.
- **클라이언트 메시지 전송**:
  - 클라이언트는 WebSocket 객체의 `send()` 메서드를 사용하여 데이터를 전송합니다.
  - ex) `ws.send("Hello, server!");`
- **서버 메시지 전송**:
  - 서버는 클라이언트와 마찬가지로 데이터를 전송할 수 있습니다. 서버가 클라이언트에게 메시지를 보내면, 클라이언트는 `onmessage` 이벤트 리스너를 통해 이를 수신합니다.

제어 메시지

- **Ping** 및 **Pong** 메시지를 통해 연결의 상태를 확인하고, 유지할 수 있습니다.
- **Close** 메시지는 연결 종료 요청을 나타내며, 정상적인 종료 절차를 따릅니다.

3. 클로징 핸드셰이크

클로징 핸드셰이크는 WebSocket 연결을 종료하는 과정입니다.

과정:

- **클라이언트 또는 서버의 종료 요청**:
  - 클라이언트 또는 서버가 연결을 종료하고자 할 경우, `close()` 메서드를 호출합니다.
  - ex) `ws.close(1000, "Normal closure");`
- **종료 프레임 전송**:
  - 종료 요청을 한 쪽은 `Close` 프레임을 전송합니다. 이 프레임에는 종료 코드와 이유가 포함될 수 있습니다.
- **상대방의 종료 확인**:
  - 종료 요청을 받은 쪽은 `Close` 프레임을 응답으로 보냅니다. 이 프레임도 종료 코드와 이유를 포함할 수 있습니다.
- **연결 종료**:
  - 두 쪽의 종료 프레임이 수신되면, WebSocket 연결은 완전히 종료됩니다. 이 시점에서 클라이언트와 서버는 더 이상 메시지를 주고받을 수 없습니다.

사용 사례

- **실시간 웹 애플리케이션**: 예를 들어, 채팅 애플리케이션, 게임, 주식 거래 시스템 등.
- **소셜 미디어 피드**: 실시간으로 업데이트되는 뉴스 피드나 알림.
- **모니터링 대시보드**: 서버나 애플리케이션의 상태를 실시간으로 모니터링.

예제 코드

```
// WebSocket 서버에 연결
const ws = new WebSocket("ws://example.com/socket");

ws.onopen = () => {
    console.log("연결이 열림");
    ws.send("서버에 메시지 전송");
};

ws.onmessage = (event) => {
    console.log("데이터 수신:", event.data);
};

ws.onclose = () => {
    console.log("연결이 종료됨");
};

ws.onerror = (error) => {
```

```
    console.error("WebSocket 오류:", error);
};
```