

05

소켓 연결 프로그램

05-4. WebSocket

Web APIs

이게 왜 필요한지에 대해
이해하는 것이 중요하다!

기본 : 동기통신 (request 1회 → response 1회 대응) - 요청을 하고, 응답이 올때까지 브라우저가 대기상태에 빠진다. (1) js 코드 실행 안 됨. (2) 유저 입력 안 됨

WebSocket

HTTP 프로토콜의 한계

네트워킹은 기본적으로 ‘클라이언트-서버’ 구조다.

클라이언트 : 서버의 네트워크 정보(IP 주소)를 알고, 서버에 연결 요청을 보내는 역할자.

서버 : 항상 실행되고 있으며, 연결 요청을 완성해주는 역할자.

개발자가 만든 리소스들은
서버에 저장되고, 클라이언트가 요청을 하면 서버에서
리소스 파일과 데이터를 꺽어온다.

web의 Stateless라는 것은 1 request – 1 response가 끝나면,
클라이언트-서버 연결이 끊어진다는 것.
매번 연결 요청을 할 때마다 연결을 다시 한다.

- HTTP 프로토콜의 특징을 이야기 할 때 항상 화자되는 중요 특징이 Connection Oriented 와 Stateless 입니다.
- Connection Oriented 라는 특징은 네트워킹을 통해 데이터를 주고 받고자 하는 두 애플리케이션이 상호 연결된 상태에서 데이터 송신/수신이 이루어 진다는 특징입니다.
- HTTP 프로토콜은 Connection Oriented 하다고 이야기 합니다.
- 우리가 관심을 가지고 있는 웹 애플리케이션으로 이야기 하자면 브라우저와 웹 서버가 네트워크에서 연결된 상태에서 상호 데이터를 주고 받으면서 브라우저에 백엔드 데이터가 출력되는 것입니다.

그렇기 때문에 단점도 있다. 단점을 개발자 코드로 극복해야 한다. 그게 세션, 쿠키 프로그램이다(백엔드).

(1) 클라이언트 상태 유지 불가

- 로그인을 성공했던 커넥션이 끊어졌다.
- 새로운 요청을 하면 새로운 커넥션이 연결되는데, 그 때 요청하는 유저가 로그인 했는지 안 했는지 프로토콜 정보로는 구별을 할 수 없다.

왜 이렇게 할까? <서버 경량화>

서버 입장에서 커넥션을 유지한다는 건, 상당히 많은 메모리가 필요한 일이다.

- 데이터를 ‘주고받는 동안’에는 커넥션이 유지되고, 클라이언트가 데이터를 ‘사용하는 동안’에는 커넥션을 끊는다.
- 그렇게 함으로써 서버에 연결된 커넥션 숫자를 줄일 수 있다.
- 약 10배 정도 많은 동시 접속자를 감당할 수 있게 된다.

- (2) Realtime Server Push를 구현할 수 없다.
- http는 클라이언트 realtime push는 가능한데, 서버 realtime push는 불가능.
 - 커넥션이 끊어져 있기에, 서버 입장에서 먼저 요청하는 것이 불가능한 구조.

이걸 해결하기 위해 추가된 기술이 web socket이다.

http를 기본으로 하고, real time push가 필요한 경우에만 web socket을 이용해야 한다.

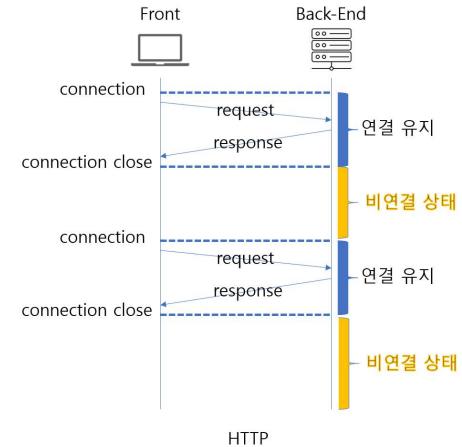
WebSocket

예) 채팅, 계속 정보를 업데이트해줘야 하는 증권 등

그렇지 않으면 서버 부하가 너무 늘어난다.

HTTP 프로토콜의 한계

- HTTP 프로토콜이 가지는 또 하나의 중요 특징이 Stateless 입니다.
- Stateless 는 웹 애플리케이션을 개발할 때 중요한 특징이며 여走路에 영향을 미치지만 네트워킹과 관련된 부분만 살펴보도록 하겠습니다.
- Stateless 란 상태를 유지하지 않는다는 말이며 네트워킹 측면에서 살펴보면 데이터 송수신을 위한 연결을 지속적으로 유지하지 않는다는 의미입니다.
- HTTP 프로토콜은 서버와 연결을 하며 이 연결을 이용해 데이터 송수신을 합니다.
- 그런데 송수신이 끝나면 자동으로 연결을 끊어 버리는 특징이 있습니다.



WebSocket

HTTP 프로토콜의 한계

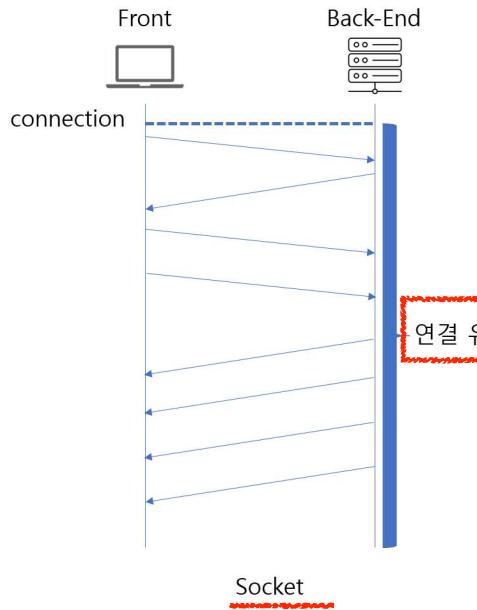
“server side realtime push가 불가능하다.”

- 비연결 상태에 있을 때 Back-end 의 데이터를 Front 에 전달할 수 없다는 점입니다.
- 이를 흔히 Realtime Server Push 라고 하는데 Server 즉, Back-end 에서 어떤 데이터가 발생하고 그 즉시 Front 에게 전송해야 하는 하지만 네트워크 연결이 되지 않았기 때문에 전송할 수 없게 됩니다.
- Front 가 원할때는 언제든지 연결해서 데이터를 주고 받을 수 있지만 Back-end 가 원할 때 데이터를 주고 받을 수 없게 됩니다.
- Front 가 원할 때는 언제든지 연결을 통해 데이터를 주고 받을 수 있으며 데이터 송수신이 끝나면 연결을 자동으로 끊음으로서 지속적으로 연결을 유지하지 않아도 되기 때문에 서버 경량화 측면에서는 이점이 있는 프로토콜입니다.

WebSocket

WebSocket 이란

- WebSocket 은 HTTP 프로토콜이 연결을 유지하지 않음으로서 Realtime Server Push 를 구현할 수 없는 문제를 해결하기 위해서 웹에서 Socket 연결을 통한 데이터 송수신을 지원하기 위한 프로토콜입니다.



1. 에러가 나거나
2. 클라이언트/서버에서 close 시키면
연결은 끊어진다.

WebSocket – WebSocket Client

- 브라우저에서 실행되는 프론트 웹 애플리케이션이 WebSocket 클라이언트 프로그램이 되며 백엔드 웹 애플리케이션이 WebSocket 서버 프로그램이 됩니다.

서버 연결

ip번호, 도메인, 포트번호

- 서버 연결을 위해 WebSocket 객체를 생성해야 하며 매개변수로 서버의 네트워크 정보를 설정해야 합니다.
- 프로토콜 명은 WebSocket 의 약어인 ws 이며 만약 보안 프로토콜을 적용하겠다면 wss 이어야 합니다.

```
서버 연결  
1 webSocket = new WebSocket("ws://localhost:3000")
```

이게 연결 정보.
 얘를 이용해서 데이터를 송수신한다.

WebSocket – WebSocket Client

데이터 송신

- 연결된 서버에 데이터를 송신하거나 수신해야 하는데 먼저 데이터 송신을 하겠다면 WebSocket 객체의 send() 함수를 이용하면 됩니다.
- 만약 JSON 데이터를 서버에 전송하겠다면 `JSON.stringify()`로 JSON 문자열을 만들어 전송하면 됩니다.

```
webSocket.send(` ${nickname} => ${msg}`)
```

WebSocket – WebSocket Client

데이터 수신

- 수신은 언제 데이터가 넘어올지 모르기 때문에 **이벤트 모델**을 이용합니다.
- 즉 데이터가 수신될 때 실행될 이벤트 콜백 함수를 등록해 놓으면 실제 데이터가 수신될 때 콜백함수가 실행되게 하는 구조입니다.

```
webSocket.onmessage = (event) => {  
    let data = event.data  
};
```

(1) .onmessage

(2) 이벤트 리스너 ‘message’

연결 해제

- 서버와 연결을 해제하고 싶다면 close() 함수를 호출해 주면 됩니다.

```
webSocket.close()
```

WebSocket – WebSocket Server

- WebSocket Server 프로그램은 다양한 언어로 개발할 수 있습니다.
- Java, C#, Javascript 등을 이용해 서버 프로그램을 작성할 수 있으며 이곳에서는 Node.js 기반의 Javascript로 WebSocket Server 프로그램을 작성하는 방법에 대해 살펴보겠습니다.
- Node.js 기반의 Javascript로 WebSocket 서버를 구현하기 위한 websocket, ws, socket.io 등 많은 모듈이 있습니다.
- 이곳에서는 ws 모듈을 이용해 WebSocket 서버 프로그램을 작성하는 방법에 대해 소개해 보겠습니다.
- >npm i ws

WebSocket – WebSocket Server

서버는 while 문으로 무한루프를 돌아 항상 요청에 대기하게끔 만든다.

WebSocket 서버 구동

- 서버 프로그램이 실행되면서 WebSocket 으로 들어오는 연결 요청을 처리할 WebSocket 서버를 구동시켜야 합니다.

```
const { WebSocketServer } = require('ws')

const sockserver = new WebSocketServer({ port: 3000 })
```

3000번 포트로 들어오는 연결 요청만을 처리하겠다.

연결 요청 처리

- 연결 요청이 들어오면 처리해 주어야 합니다. 이는 connection 이라는 이벤트를 등록해 주면 되며 연결요청이 들어오는 순간 이벤트가 발생하여 등록한 콜백 함수가 실행되게 됩니다.

```
sockserver.on('connection', ws => {
  console.log('New client connected!')
  ws.on('close', () => console.log('disconnected!'))

  ws.onerror = function () {
    console.log('websocket error')
  }
})
```

이벤트 콜백 실행

WebSocket – WebSocket Server

데이터 수신

- 데이터 수신은 message 이벤트를 이용합니다.

```
ws.on('message', data => {
    sockserver.clients.forEach(client => {
        console.log(`send message: ${data}`)
        client.send(`${data}`)
    })
})
```

데이터 송신

- 데이터 송신은 클라이언트와 연결된 연결객체의 send() 함수를 이용합니다.

[ws.send\('message...'\)](#)



감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.



윌리엄 셰익스피어

William Shakespeare