**고급소프트웨어실습 9주차 과제**

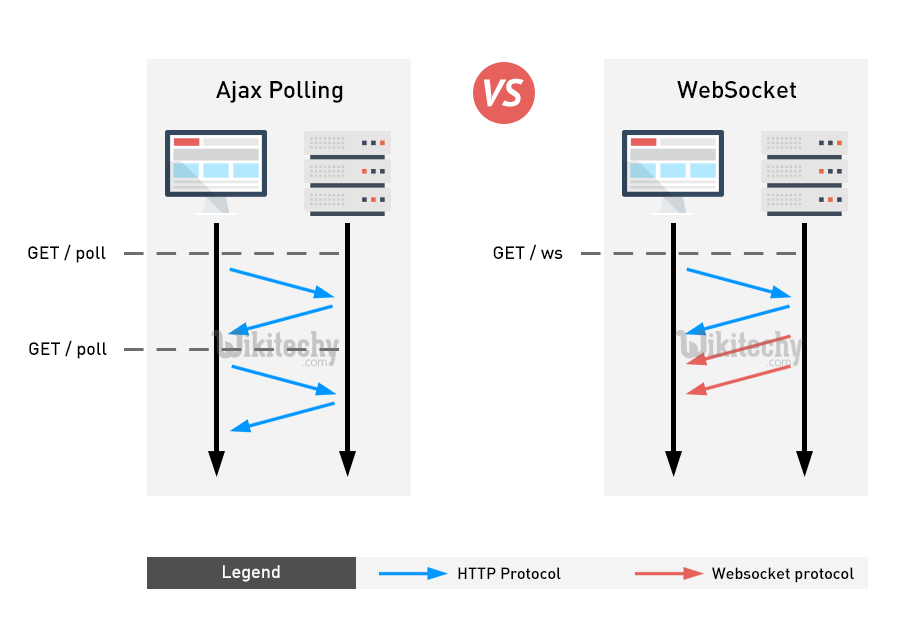
**20191657 최세은**

**1. websocket, socket.io 통신 방식이란?**

우선 웹소켓(WebSocket)은 하나의 TCP 접속에 전이중 통신 채널을 제공하는 컴퓨터 통신 프로토콜이다. 개발자는 웹 소켓 API를 통해 서버로 메시지를 보내고 서버의 응답을 위해 서버를 폴링하지 않고도 이벤트 중심 응답을 받는 것이 가능하다.

여기서 폴링(polling)이란 클라이언트에서 일정 주기마다 요청을 보내고 서버는 현재 상태를 바로 응답하는 방식이다. 이 방식은 실시간으로 반영되는 것이 중요한 서비스에는 별로 좋지 않고 서버에서 변화가 없더라도 매 요청마다 응답을 내려주기 때문에 불필요한 트래픽이 발생하게 된다.

따라서 웹소켓은 웹 서버와 웹 브라우저간 실시간 양방향 통신환경을 제공해주는 실시간 통신 기술이므로 위의 Polling 방식과 다르게 양방향으로 원할 때 요청을 보낼 수 있으며 stateless한 HTTP에 비해 오버헤드가 적으므로 유용하게 사용할 수 있다.



socket.io이란 WebSocket과 같이 클라이언트와 서버의 양방향 통신을 가능하게 해주는 모듈이다. 클라이언트에서 Event Listener로 새로운 정보를 받아 정보를 업데이트할 수 있다.

JavaScript를 이용하여 브라우저 종류에 상관없이 실시간 웹을 구현할 수 있도록 했으며, WebSocket, FlashSocket, AJAX Long Polling, AJAX Multi part Streaming, IFrame, JSONP Polling을 하나의 API로 추상화한 것이다. 즉 브라우저와 웹 서버의 종류와 버전을 파악하여 가장 적합한 기술을 선택하여 사용하는 방식이다. 가령 브라우저에 Flash Plugin v10.0.0 이상(FlashSocket 지원 버전)이 설치되어 있으면 FlashSocket을 사용하고, Flash Plugin이 없으면 AJAX Long Polling 방식을 사용한다. 개발자가 각 기술을 깊이 이해하지 못하거나 구현 방법을 잘 알지 못해도 사용할 수 있다.

Web Socket과 달리 Socket.io는 표준 기술이 아니고 Node.js 모듈로서 MIT 라이센스를 가진 오픈소스이다.

**2. 위 통신 방식들을 사용해서 실제 채팅 프로그램을 구현한다면?**

**2-1. websocket**

우선 서버쪽은 ws 모듈을 사용해서 쉽게 구현할 수 있다.

|  |
| --- |
| var WebSocket = require('ws').Server;  var wss = new WebSocketServer({ port: 3000 });    wss.on('connection', function(ws) {  ws.on('message', function(message) {  const sendData = {  event: 'response',  data: null  };    switch(message.event) {  case 'open':  console.log(message);  break;  case 'request':  sendData.data = 'some data...';  ws.send(JSON.stringify(sendData));  break;  }  });  }); |

처음에 웹소켓이 연결될 때, “wss.on('connection', callback)” 이라는 callback 함수가 실행된다.

그리고 callback함수의 인자로 클라이언트와 연결된 소켓을 paramter로 넘겨 받는데 이 변수로 메시지를 받았을때의 이벤트를 설정하고(ws.on(‘message’, callback)) 메시지를 보낼 수도 있다.(ws.send(message)) 여기서 websocket은 메시지를 보낼 때 문자열로 전송이 되므로 객체의 경우 stringify(서버) -> parse(클라이언트)를 해주어야 한다.

그리고 클라이언트 쪽은 브라우저에서 WebSocket이 기본 스펙에 있기 때문에 다른 모듈을 사용할 필요 없이 사용할 수 있다.

|  |
| --- |
| var ws = new WebSocket('ws://localhost:3000');  ws.onopen = (event) => {  let sendData = { event: 'open' }  ws.send(JSON.stringify(sendData));  }  ws.onmessage = (event) => {  let recData = JSON.parse(event.data);  switch (recData.event) {  case 'response':  console.log(recData.data);  break;  }  }    function myOnClick() {  let sendData = {  event: 'request',  data: 'some data...'  };  ws.send(JSON.stringify(sendData));  } |

서버와 비슷하게 연결되었을 때 (ws.onopen), 메시지를 받았을 때(ws.onmessage), 메시지를 보낼때(ws.send)를 각각 구현함으로써 메시지를 주고 받기가 가능해진다. 마찬가지로 메시지를 받았을 때, 보낼 때 문자열로만 전송이 가능하므로 stringify, parse를 적절하게 사용하여 데이터를 이용해야 한다.

**2-2. socket.io**

우선 서버쪽은 다음과 같이 구현할 수 있다.

|  |
| --- |
| const server = require('http').createServer();  const io = require('socket.io')(server);    const clients = [];    io.on('connection', (client) => {  clients.push(client);    client.on('message', (message) => {  // handle message...  client.forEach((c) => c.emit('message', message));  });    client.on('disconnect', () => {  console.log('client disconnect...', client.id);  // handle disconnect..  clients.filter((c) => c.id !== client.id);  });    client.on('error', (err) => {  console.log('received error from client:', client.id);  // handle error..  });  });    server.listen(5000, (err) => {  if (err) throw err;  console.log('listening on port 5000');  }); |

웹소켓에서 구현 했던 것과 비슷하게 구현한다.

io.on(‘connection’, callback) 과 같이 연결되었을 때의 이벤트를 정의하고 각 소켓에 대해서 on(‘event’, callback) 과 같이 원하는 이벤트를 작성할 수 있다. 연결될 때의 소켓값을 모두 clients에 넣어두고 누군가 message이벤트를 발생시킨다면 각 클라이언트에게 emit함수를 통해 ‘message’ 이벤트를 발생하게 한다.

클라이언트 쪽은 다음과 같이 구현할 수 있다.

|  |
| --- |
| const io = require('socket.io-client');    export default function() {  const socket = io.connect('http://localhost:5000');    function onMessageEvent(callback) {  socket.on('message', (message) => {  callback(message);  });  }    function messageSend(message) {  socket.emit('message', message);  }    function offMessageEvent() {  socket.off('message');  }    return {  onMessageEvent,  offMessageEvent  };  } |

클라이언트 부분에서는 socket.io-client 모듈을 이용하여 쉽게 서버와 연동할 수 있다. 그리고 서버에서와 마찬가지로 on, off로 이벤트 리스너를 켜고 끌 수 있으며 emit함수로 서버의 이벤트를 발생시킬 수 있다.

이렇게 구현을 하고 나면 client에서 messageSend 함수를 호출했을 때 접속되어 있는 모든 클라이언트에게 메시지가 가게 된다.