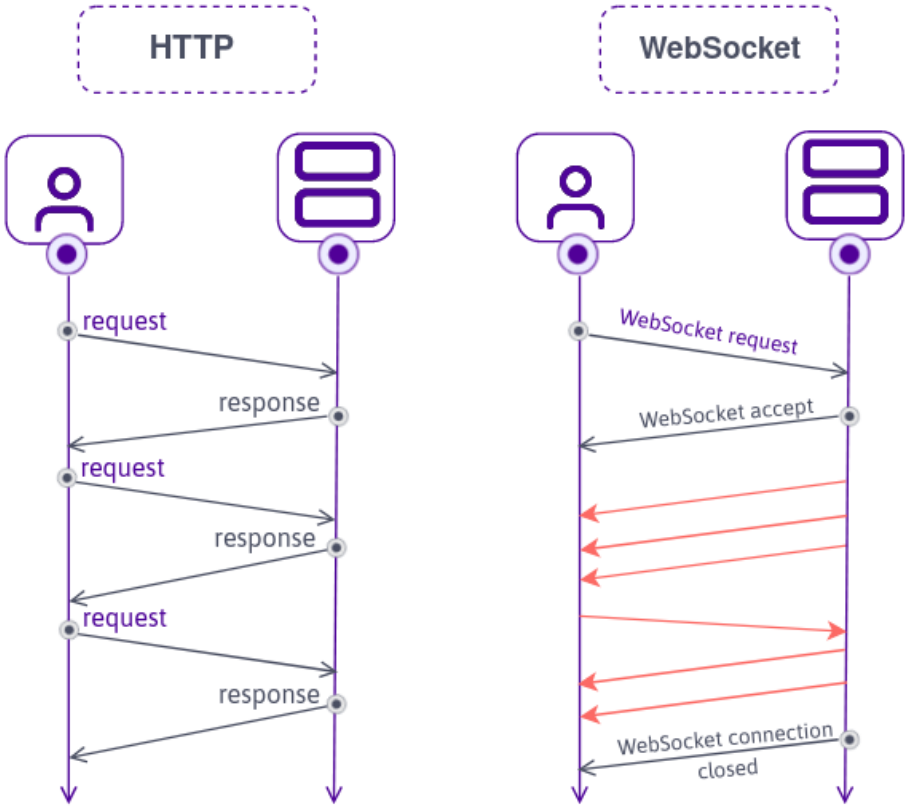
# 웹 소켓(WebSocket) 프로그래밍

웹 소켓이란?

HTML5 표준 기술로, HTTP 환경에서 클라이언트와 서버 사이에 하나의 TCP 연결을 통해 실시간으로 전이중 통신을 가능하게 하는 프로토콜이다. 여기서 전이중 통신이란, 일방적인 송신 또는 수신만이 가능한 단방향 통신과 달리 가정에서의 전화와 같이 양방향으로 송신과 수신이 가능한 것을 말한다. 양방향 통신이 아닌 단방향 통신의 예로는 텔레비전 방송, 라디오를 들 수 있는데, 데이터를 수신만 할 수 있고, TV나 라디오를 통해 데이터를 보낼 수 없다.

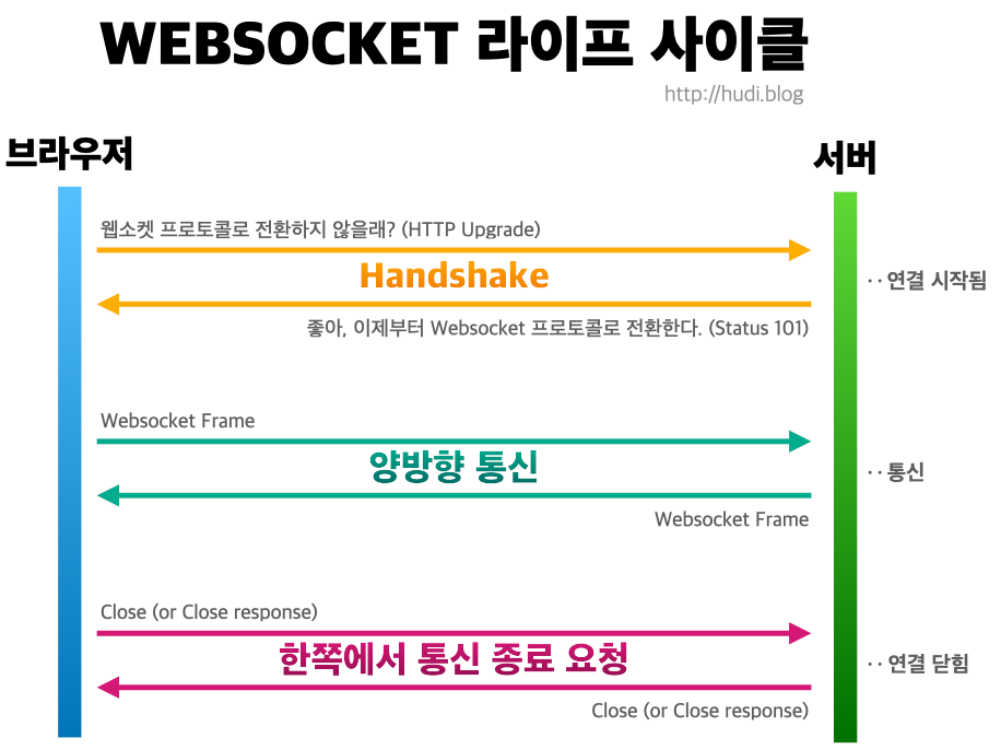


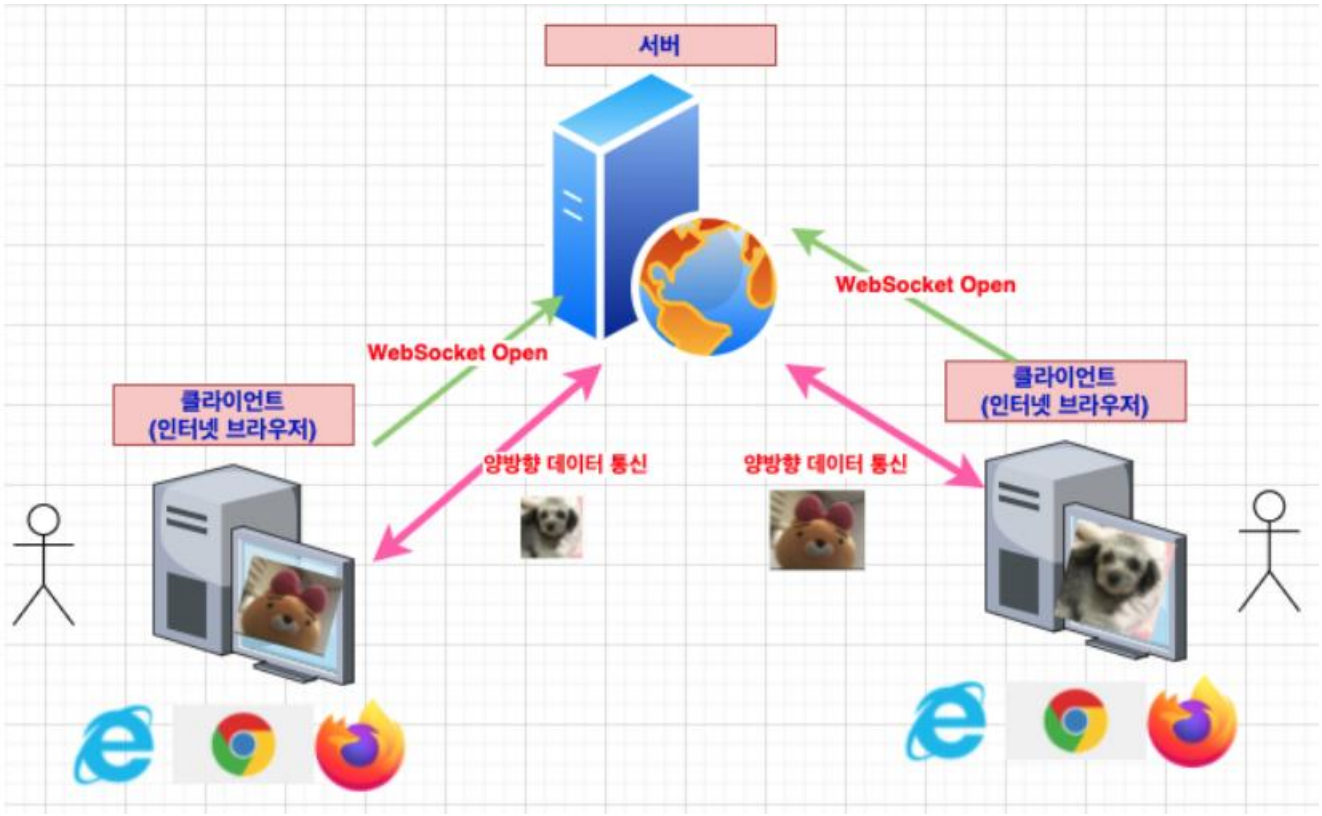

실시간 알림, 실시간 채팅 등 실시간이라는 키워드가 들어가는 기능들을 위해서는 대부분 웹 소켓 기술이 필요하다.


## 웹 소켓의 통신 방식

웹 소켓은 전 이중 통신이므로, 연속적인 데이터 전송의 신뢰성을 보장하기 위해 Handshake 과정을 진행한다.

기존의 다른 TCP 기반의 프로토콜은 TCP layer에서의 Handshake를 통해 연결을 수립하는 반면, 웹 소켓은 HTTP 요청 기반으로 Handshake 과정을 거쳐 연결을 수립한다.



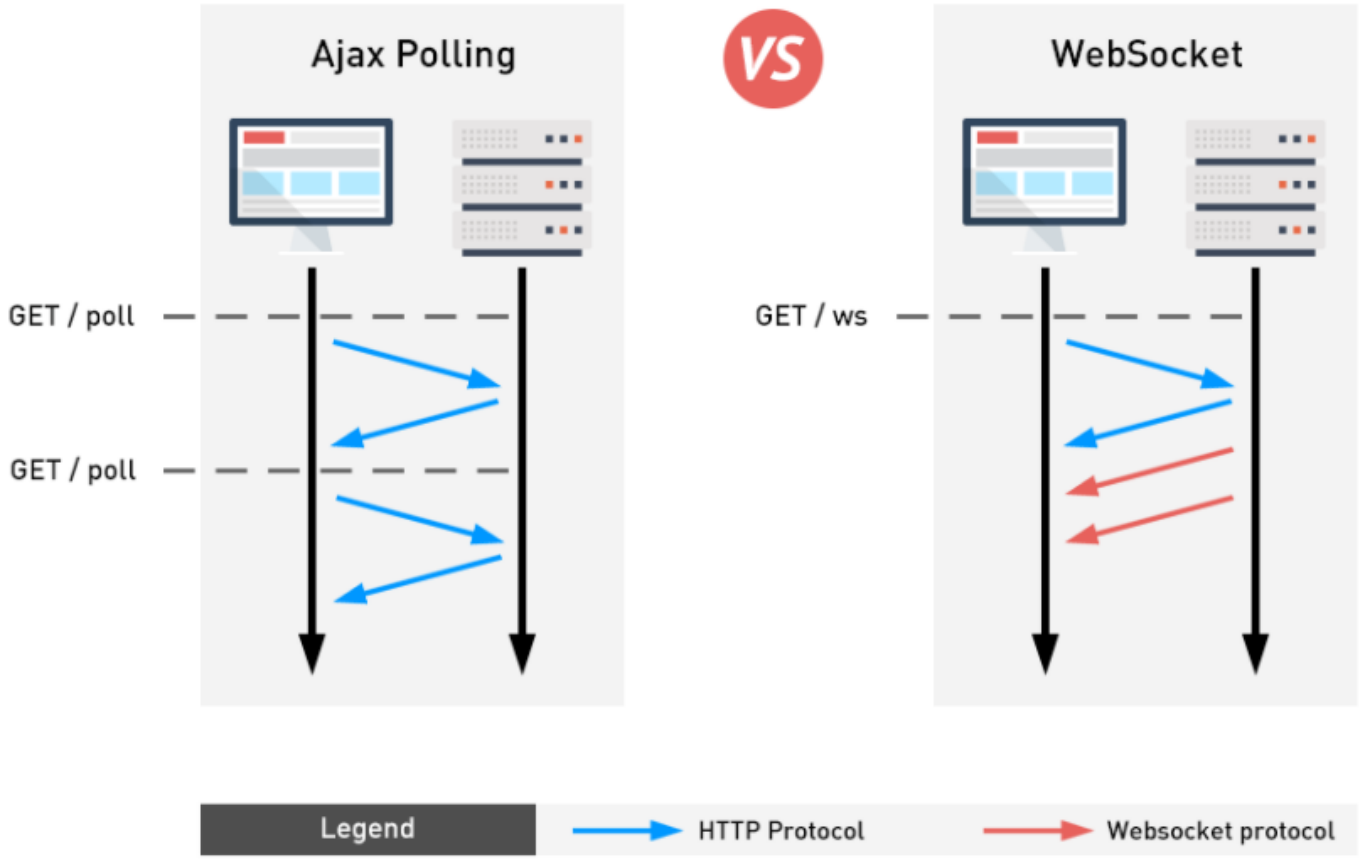



## 웹 소켓의 탄생 배경

초기 웹의 탄생 목적은 문서 전달과 하이퍼링크를 통한 문서 연결이었다. 웹을 위한 HTTP 프로토콜은 이러한 목적에 매우 부합하는 모델이다. 그러나 시대가 변하고 환경이 발전할 수록 웹에게 동적인 표현과 뛰어난 상호작용이 요구되었고 이로 인해 여러 새로운 기술이 탄생되었다.

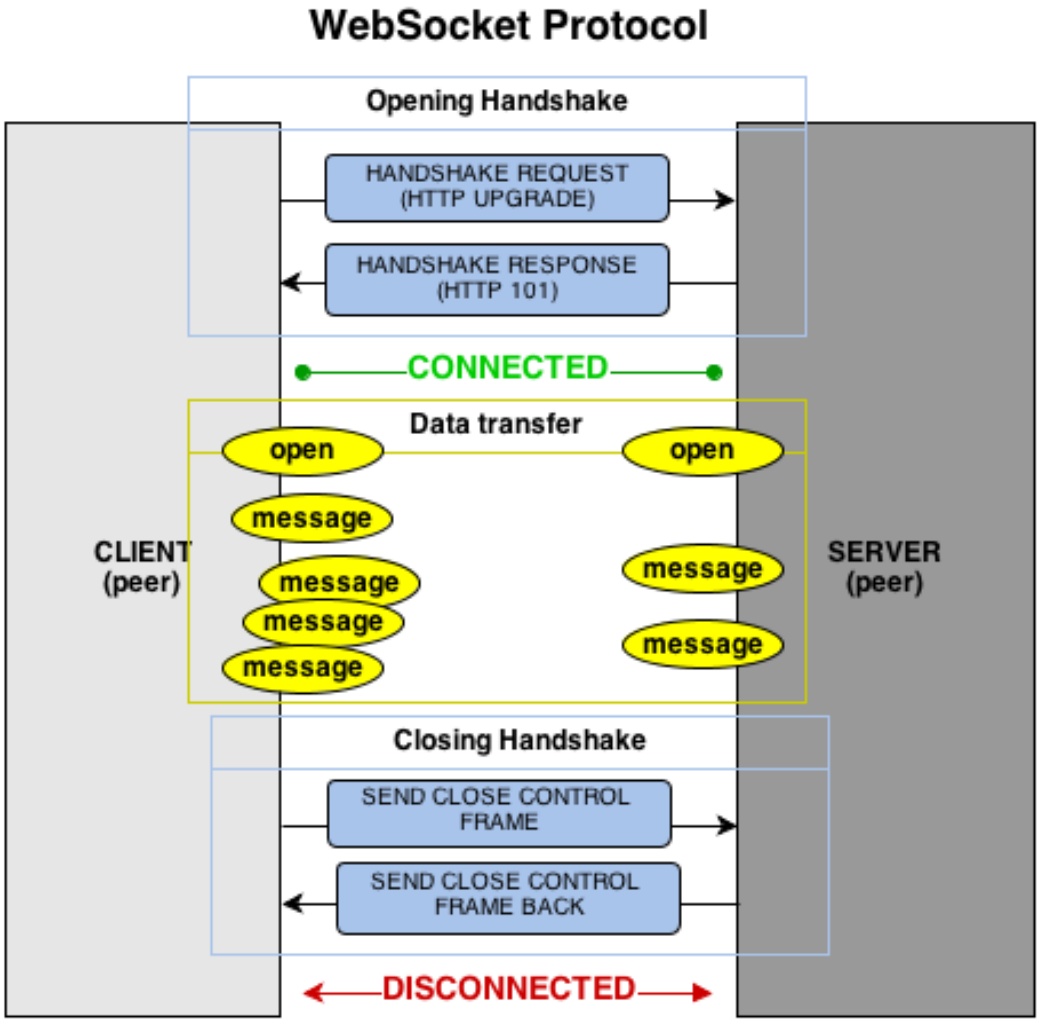
전이중 실시간 양방향 통신을 위한 스펙이 바로 웹 소켓인 것이다.

웹 소켓 기술이 없을 때는 Polling이나 Long polling 그리고 HTTP Streaming 등의 방식으로 양방향 실시간은 아니지만 그에 준하게끔 구현하여 해결했다. 지금은 웹 소켓의 등장으로 클라이언트와 서버간의 실시간 통신이 가능하게 되었다.




따라서 SNS, 멀티 플레이어 게임, 구글 공유 문서 등 실시간 웹 어플리케이션 구현을 위해 웹 소켓을 사용하고 있다.

## 웹 소켓 통신의 동작 방식




## 웹 소켓을 이용한 통신의 특징

양방향 통신이 가능하다.

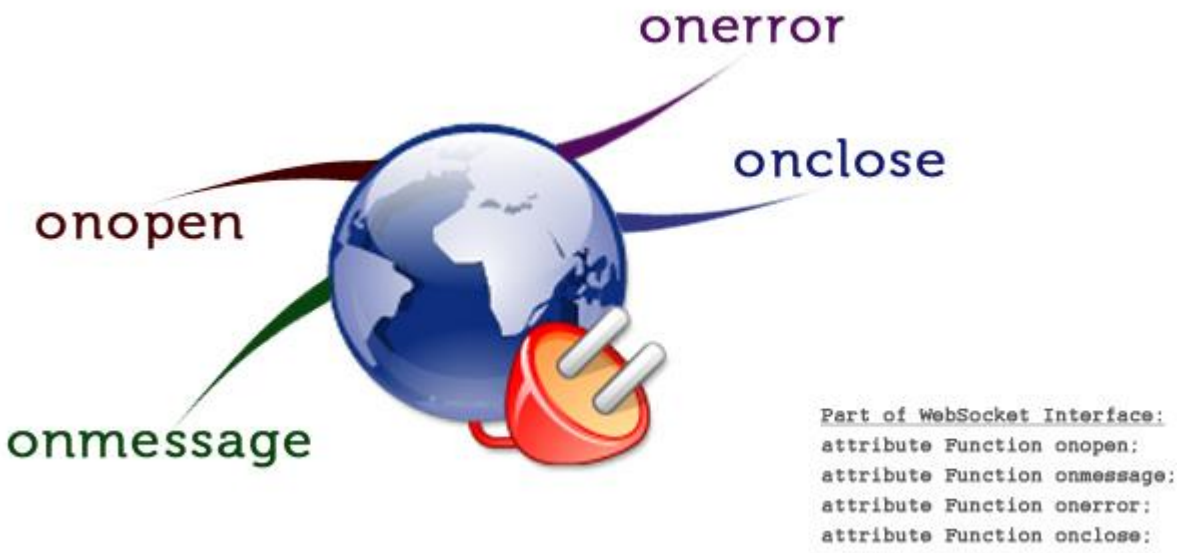
서버에서 일방적으로 클라이언트로 요청을 보낼 수 있다.

실시간 통신이 가능하다.

Connection을 유지하고 있는 동안 request-response 방식의 통신이 아닌 아닌 양방향의 실시간 데이터 통신이 가능하다.

## 웹 소켓 통신의 구현

서버와 클라이언트 관계없이 웹 소켓 관련 이벤트 핸들러를 구현하여 처리한다.




## 웹 소켓 클라이언트 구현 - 자바스크립트 + HTML5 API

- 서버연결

  HTML5가 제공하는 WebSocket 객체를 통해 서버 연결을 수행한다.

  일반 통신은 ws, 보안 통신은 wss 프로토콜을 이용한다

  ```
  let  ws = new WebSocket("웹 소켓 URL 문자열");
  ```

  웹 소켓 URL 문자열 : ws://서버주소/웹 소켓 서버 프로그램의 매핑명

- 데이터 송신

  WebSocket 객체의 send() 메서드로 데이터를 서버로 송신한다.

  ```
  ws.send ("전송하려는 메시지")
  ```

- 데이터 수신

  서버에서 전송되는 데이터를 받으려면 message 이벤트를 구현한다.

```
ws.onmessage = function(e) {
        e.data로 추출하여 수신받은 메시지 처리
}
```

- 웹 소켓 관련 이벤트

| |
|---|
| open : 웹 소켓 서버와 접속이 일어나면 발생하는 이벤트이다. |
| close : 웹 소켓 서버와 접속이 해제되면 발생되는 이벤트이다. |
| message : 웹 소켓 서버로 부터 메시지가 수신되면 발생되는 이벤트이다. |
| error : 웹 소켓 오류가 생기면 발생되는 이벤트이다. |

```
let ws = new WebSocket("ws://서버주소/웹 소켓 서버 프로그램의 매핑명");
ws.onopen = function(e) {
        //웹 소켓 서버와의 접속이 성공하면 발생하는 open 이벤트의 핸들러
}
ws.onclose = function(e) {
        //웹 소켓 서버와의 접속이 해제되면 발생하는 close 이벤트의 핸들러
}
ws.onerror = function(e) {
        //웹 소켓 통신을 하는 동안 발생할 수도 있는 error 이벤트의 핸들러
}
ws.onmessage = function(e) {
        //웹 소켓 서버로부터 메시지가 수신되면 발생하는 message 이벤트의 핸들러
}
```

[ 웹 소켓으로 구현한 채팅 클라이언트 예제 ]

```
<body>
	<div id='chatt'>
		<h1>웹 소켓 채팅</h1>
		<input type='text' id='mid' value='게스트'>
		<input type='button' value='채팅참여' id='btnJoin'>
		<br/>
		<div id='talk'></div>
		<div id='sendZone'>
			<textarea id='msg' >안녕...</textarea>
			<input type='button' value='전송' id='btnSend'>
		</div>
	</div>
	<script>
	function getId(id){
		return document.getElementById(id);
	}
	let data = {};//전송 데이터(JSON)

	let ws ;
	let mid = getId('mid');
	let btnJoin = getId('btnJoin');
	let btnSend = getId('btnSend');
	let talk = getId('talk');
	let msg = getId('msg');

	btnJoin.onclick = function(){
		ws = new WebSocket("ws://" + location.host + "/chatt");

		ws.onmessage = function(msg){
			let data = JSON.parse(msg.data);
			let css;

			if(data.mid == mid.value){
				css = 'class=me';
			}else{
				css = 'class=other';
			}
```

```
                let item = `<div ${css}>
                        <span><b>${data.mid}</b></span> [ ${data.date} ]<br/>
                        <span>${data.msg}</span>
                        </div>`;

                talk.innerHTML += item;
                talk.scrollTop=talk.scrollHeight;//스크롤바 하단으로 이동
            }
            this.style.color = 'blue';
            this.value = '채팅참여중';
        }

        msg.onkeyup = function(ev){
            if(ev.keyCode == 13){
                send();
            }
        }

        btnSend.onclick = function(){
            send();
        }

        function send(){
            if(msg.value.trim() != ''){
                data.mid = getId('mid').value;
                data.msg = msg.value;
                data.date = new Date().toLocaleString();
                let temp = JSON.stringify(data);
                ws.send(temp);
            }
            msg.value ='';
        }
        </script>
</body>
```

# 웹 소켓 서버 구현 - Spring Boot WebSocket

Spring Boot 에서 지원하는 웹 소켓 프로그래밍 기술을 이용하여 서버 프로그램을 구현하기 위해서는 WebSocket을 사용하기 위한 @ServerEndpoint 어노테이션이 선언되어 있는 클래스와 웹 소켓에 관한 환경 설정 파일로 구현할 수 있다.

- 웹 소켓에 관한 환경 설정 파일

Spring에서 빈은 싱글톤으로 관리되지만, @ServerEndpoint 클래스는 WebSocket이 생성될 때마다 인스턴스가 생성되고 관리되기 때문에 Spring의 @Autowired가 설정된 멤버들이 초기화 되지 않는다. 그러므로 클라이언트로부터 웹 소켓 접속이 요청될 때마다 @ServerEndpoint 클래스의 객체를 생성해 줄 초기화 클래스의 빈 등록이 필요하다.

```
@Component
public class WebSocketConfig {
	@Bean
	public ServerEndpointExporter serverEndpointExporter() {
		return new ServerEndpointExporter();
	}
}
```

- 웹 소켓 서버 프로그램의 기능을 구현할 @ServerEndpoint 클래스

```
@Service
@ServerEndpoint(value="/매핑명")
public class 클래스명 {
	@OnOpen
	public void onOpen(Session s) {
		// 웹 소켓 클라이언트로부터 접속 요청이 들어오면 호출되는 메서드
	}

	@OnMessage
	public void onMessage(String msg, Session session) throws Exception{
		// 메시지가 전송되었을 때 호출되는 메서드
	}
```

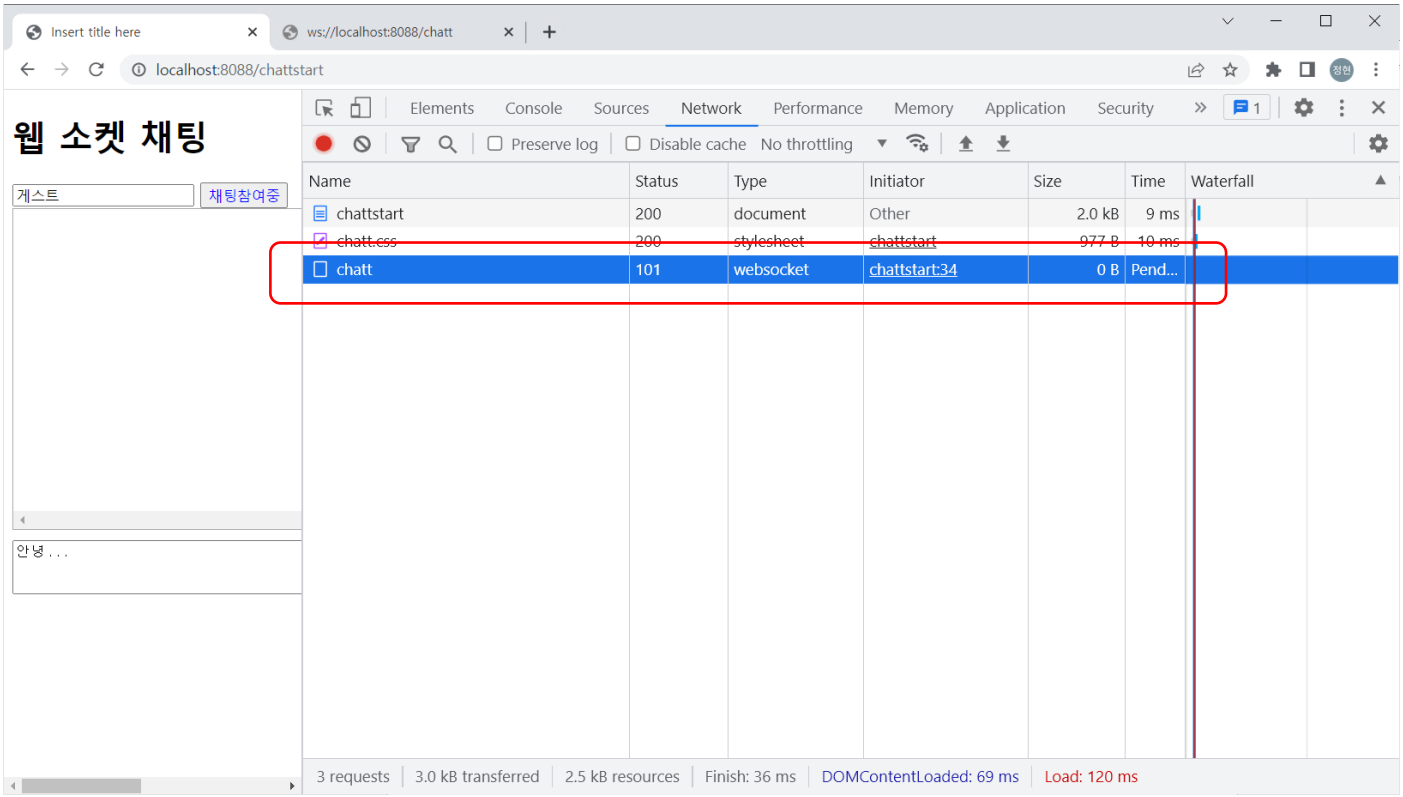
클라이언트에서 웹 소켓 통신을 요청할 때 사용되는 URL 문자열
ws://localhost:8088/chatt

```
	@OnClose
	public void onClose(Session s) {
		// 웹 소켓 통신이 종료되면서 호출되는 메서드
	}
}
```

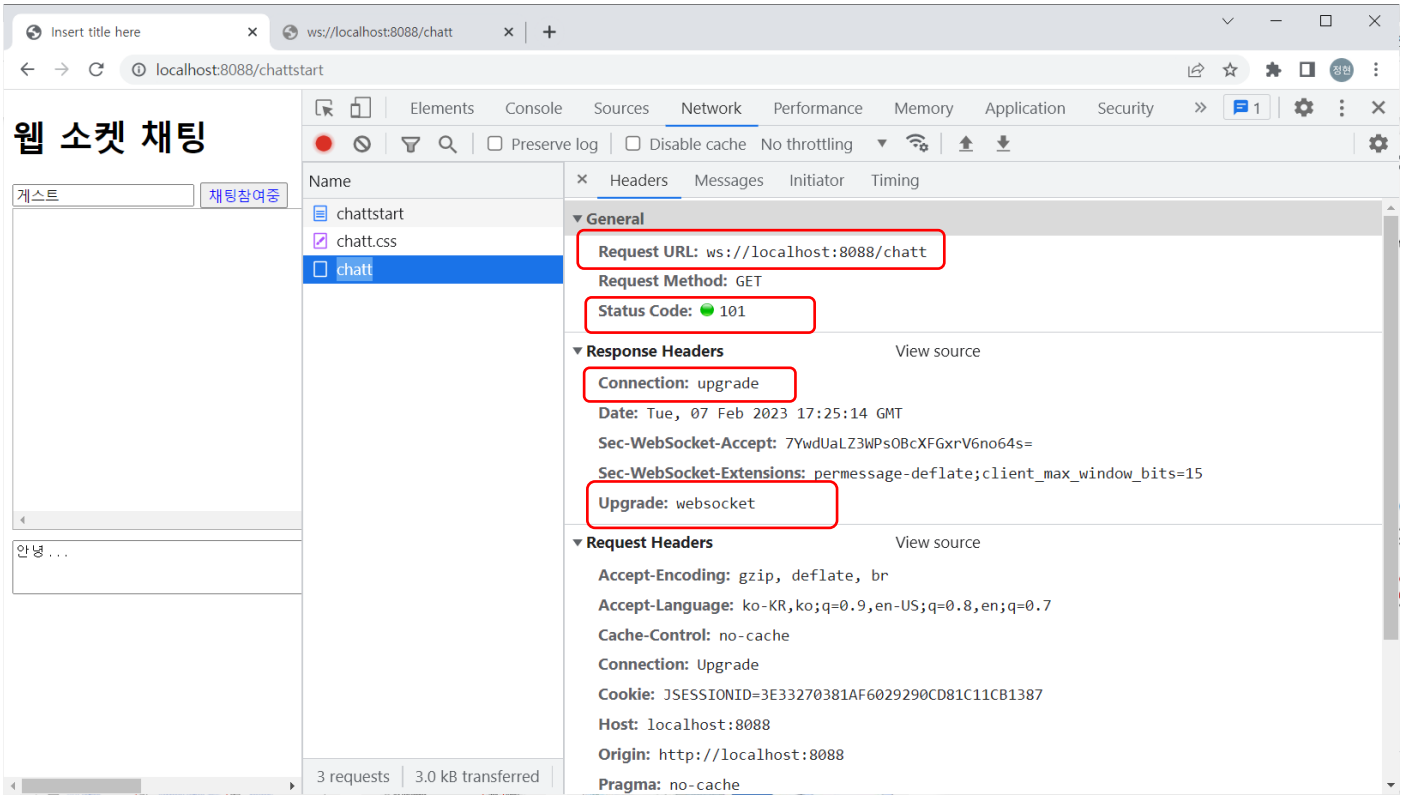
[ 웹 소켓으로 구현한 채팅 서버 예제 ]

```
@Service
@ServerEndpoint(value="/chatt")
public class WebSocketChatt {
	private static Set<Session> clientSet =
		Collections.synchronizedSet(new HashSet<Session>());
	@OnOpen
	public void onOpen(Session s) {
		if(!clients.contains(s)) {
			clientSet.add(s);
			System.out.println("[세션 오픈] " + s);
		}else {
			System.out.println("이미 연결된 세션임!!!");
		}
	}

	@OnMessage
	public void onMessage(String msg, Session session) throws Exception{
		System.out.println("[수신 메시지] " + msg);
		for(Session s : clientSet) {
			System.out.println("[송신 메시지] " + msg);
			s.getBasicRemote().sendText(msg);
		}
	}
	@OnClose
	public void onClose(Session s) {
		System.out.println("[세션 종료] " + s);
		clientSet.remove(s);
	}
}
```

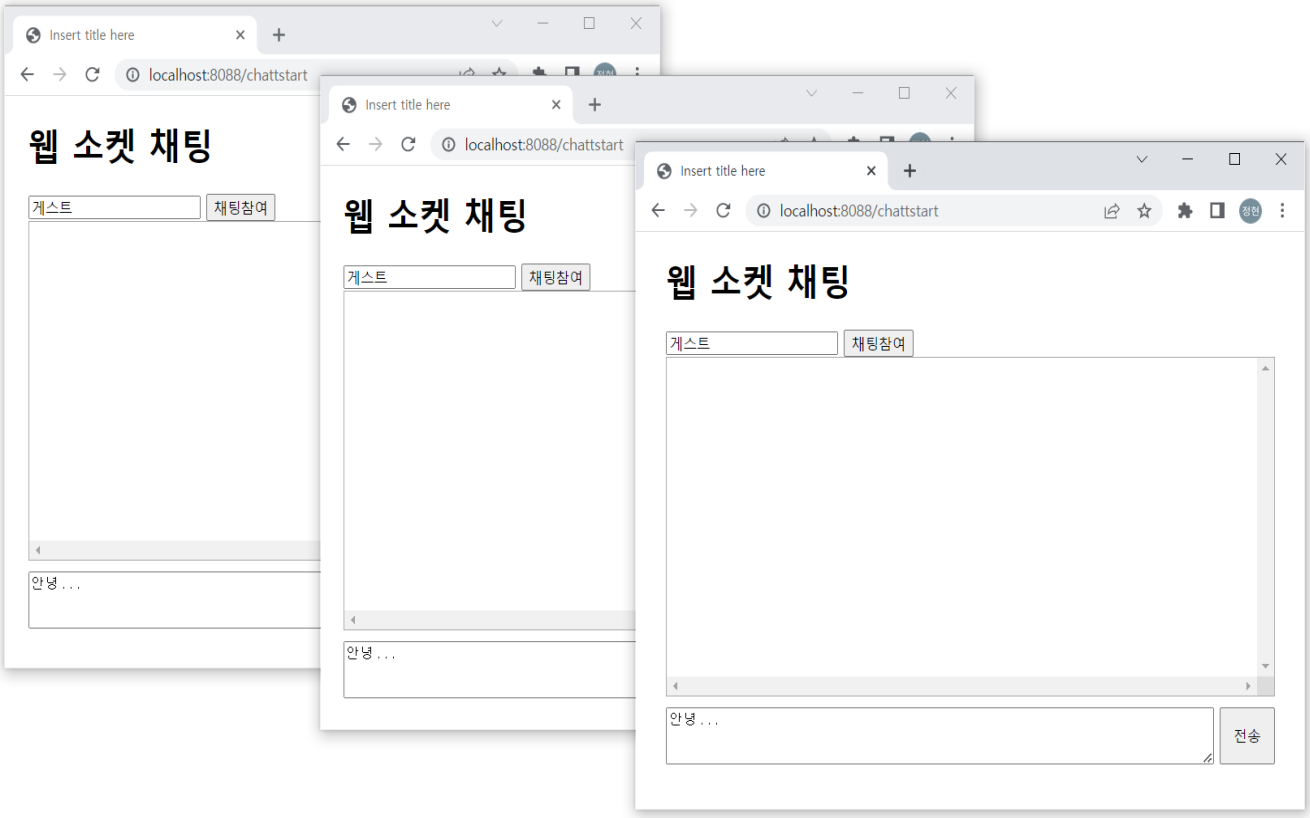


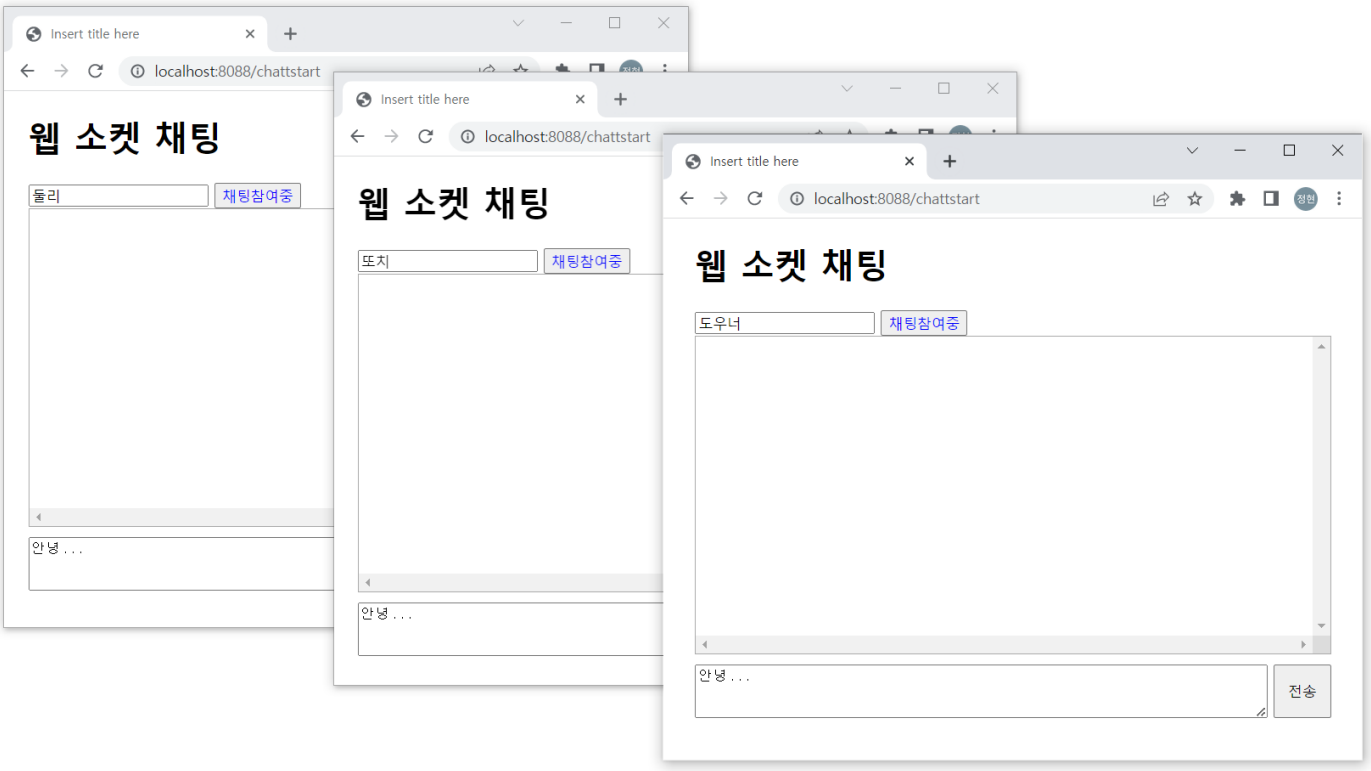

| | | |
|---|---|---|
| 101 | Switching Protocols | 서버는 클라이언트의 요청대로 Upgrade 헤더를 따라 다른 프로토콜로 바꿀 것임. (HTTP 1.1에서 처음 등장) |

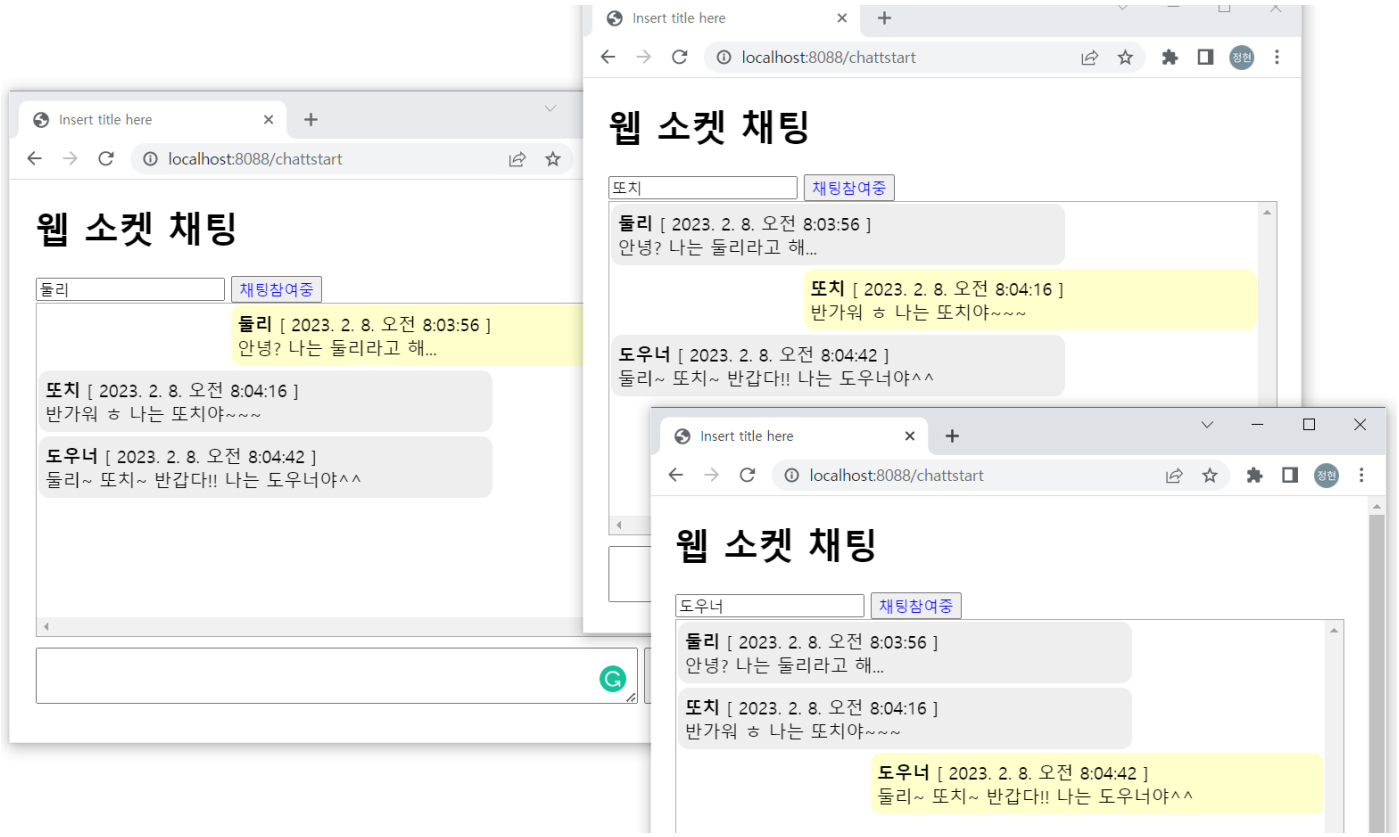


# 테스트 과정

크롬 브라우저를 3개 기동하여 각각의 브라우저에서 http://localhost:8088/chatstart를 요청한다.




Insert title here
localhost:8088/chattstart
웹 소켓 채팅
둘리
채팅참여중
둘리 [ 2023. 2. 8. 오전 8:03:56 ]
안녕? 나는 둘리라고 해...
또치 [ 2023. 2. 8. 오전 8:04:16 ]
반가워 ㅎ 나는 또치야~~~
도우너 [ 2023. 2. 8. 오전 8:04:42 ]
둘리~ 또치~ 반갑다!! 나는 도우너야^^
Insert title here
localhost:8088/chattstart
웹 소켓 채팅
또치
채팅참여중
둘리 [ 2023. 2. 8. 오전 8:03:56 ]
안녕? 나는 둘리라고 해...
또치 [ 2023. 2. 8. 오전 8:04:16 ]
반가워 ㅎ 나는 또치야~~~
도우너 [ 2023. 2. 8. 오전 8:04:42 ]
둘리~ 또치~ 반갑다!! 나는 도우너야^^
Insert title here
localhost:8088/chattstart
웹 소켓 채팅
도우너
채팅참여중
둘리 [ 2023. 2. 8. 오전 8:03:56 ]
안녕? 나는 둘리라고 해...
또치 [ 2023. 2. 8. 오전 8:04:16 ]
반가워 ㅎ 나는 또치야~~~
도우너 [ 2023. 2. 8. 오전 8:04:42 ]
둘리~ 또치~ 반갑다!! 나는 도우너야^^