



03. Cookie와 Session

작성자 : 백성애

본 문서는 학습용으로만 사용하고 웹이나 다른 곳의 배포를
금합니다

Session과 Cookie

◇◇◇ HTTP프로토콜과 쿠키 및 세션 ◇◇◇

웹의 근간이 되는 HTTP(HyperText Transfer Protocol) 프로토콜은 웹페이지나 이미지를 가져오는데 매우 이상적인 프로토콜이지만, 서버측 응용프로그램(CGI, Servlet JSP, ASP, PHP)에서 생성되는 상호 대화적인 내용을 처리하는데는 여러 가지 문제가 있다.

HTTP 프로토콜은 브라우저에서 요청이 있을 때 마다 새로운 네트워크 연결이 이루어지기 때문에 사용자의 요청에 네트워크 연결이 지속되지 않으므로 클라이언트의 요청을 서로 구분 할수 없으며, 클라이언트의 상태정보를 저장 할 수 없다.

상태가 없으면 웹을 이용한 응용프로그램(대표적으로 쇼핑몰 구현)을 개발하기 어려워 지는데, 이것은 사용자가 쇼핑몰을 방문해서 원하는 상품을 선택할 때마다 이전에 선택한 상품정보는 잃어버리는 문제가 발생하기 때문이다.

이런한 문제를 해결(상태정보 유지)하기 위해서 쿠키와 세션이 등장하게 되었다.

Cookie란?

- ▶ 1) 쿠키란?
- ▶ - 쿠키는 클라이언트(웹브라우저)에 저장되는 간단한 정보를 의미.
- ▶ - 쿠키는 웹서버와 웹브라우저 양쪽에서 생성할 수 있으며,
- ▶ - 웹서버는 웹브라우저가 전송한 쿠키를 사용해 필요한 데이터를 읽어올 수 있다.
- ▶ - 쿠키는 그 크기가 하나에 4KB 이하로 제한이 되어 있으며, 총 300개까지 정보를 저장할 수 있다.
- ▶ 따라서,최대로 저장가능한 쿠키의 용량은 1200KB 즉 1.2MB

Cookie란?

❶ 쿠키

쿠키는 클라이언트의 상태 정보를 유지하기 위한 방법으로 네스케이프에서 제안된 기술로 웹응용프로그램에서 필요에 따라

- 1) 클라이언트의 정보를 텍스트 형태로 클라이언트(웹브라우저)에게 전송하고(응답메세지의 헤더를 통해),
- 2) 웹 브라우저는 전송받은 텍스트파일(쿠키)을 메모리에 로딩(저장)하고 있다가, 다시 쿠키를 전송한 해당 사이트에 방문할 경우 웹서버에게 제출하도록 한다.(요청메세지의 헤더를 통해)
- 3) 웹서버는 전달 받은 쿠키의 값을 읽어 들여 적절한 작업을 수행할 수 있다.

● HTTP 응답 메세지의 헤더 형태

Set-Cookie : name=value; expires=date; path=path; domain=domain; secure

- ▶ **name=value** : 쿠키의 이름과 값을 기술한다.([] () = , " / ? @ ; 등의 특수문자는 사용할 수 없다)
- ▶ **expires=date** : 쿠키의 유효기간을 명시한다. 디폴트는 현재의 브라우저가 실행되는 동안에만 유효.
- ▶ **path=path** : 쿠키를 사용할 수 있는 URL 패스를 기술한다.
기술하지 않으면 쿠키를 설정한 URL에 방문할 경우에만 쿠키를 제출한다.
- ▶ **domain=domain** : domain 속성을 기술하게 되면 해당 도메인의 웹서버에 접근하는 경우에도 쿠키를 제출한다.
디폴트는 쿠키를 설정한 컴퓨터가 된다.
- ▶ **secure** : secure라고 명시적으로 기술하면 SSL(Secure Socket Layer: 인증 암호화 프로토콜)을 사용하는 웹서버인 경우에만 쿠키를 제출한다.

Cookie

- HTTP 요청 메시지의 헤더 형태(이름과 값의 쌍)

Cookie : name1=value1; name2=value2; name3=value3;

- ★ 이것 아세요??

쿠키는 클라이언트측에 파일로 저장되는 경우도 있고, 저장되지 않은 경우도 있다.

웹응용프로그램에서 쿠키를 설정할 때 expires 속성을 설정하지 않는 경우에는 브라우저가 실행되는 경우에만 유효하지만, 만일 expire를 초단위로 설정하게 되면 파일로 저장된다.

(최대 300개, 최대 4KB)

저장되는 위치는 익스플로러의 경우 Cookies 디렉토리에 저장되고,
네스케이프의 경우 cookie.txt 파일에 저장된다.

Cookie의 종류

- 1>하드에 저장되는 쿠키

- 2>브라우저 메모리에 존재하는 쿠키(브라우저가 켜져있을 때는 항상 존재)

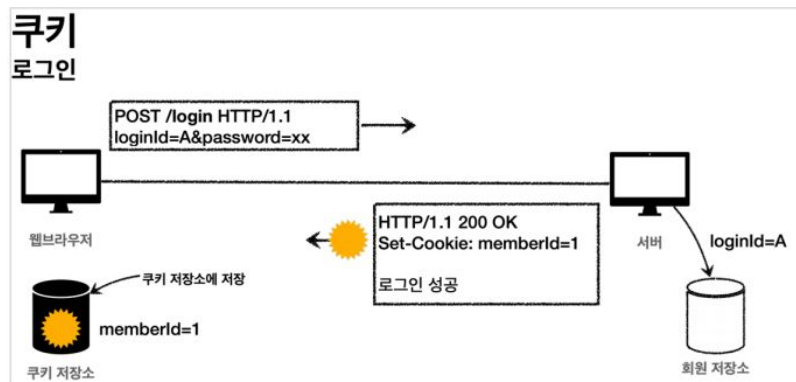
Cookie 동작 원리

1> 클라이언트가 처음 서버에 접근하면 **서버는 클라이언트에 쿠키를 설정** 한다.

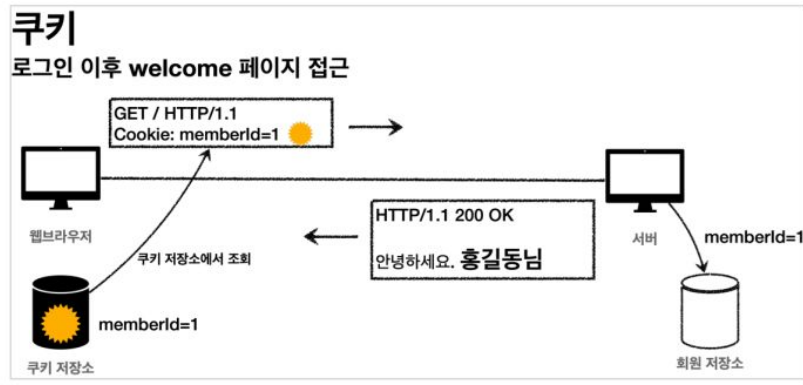
2> 후에 사용자가 그 사이트를 다시 방문하는 경우 **웹브라우저는 쿠키 정보를 서버에 전달** 한다.

3> 서버는 이 쿠키 정보를 이용해 적절한 작업을 수행 한다

쿠키 생성



쿠키 전달



Cookie 동작 원리

쿠키

모든 요청에 쿠키 정보 자동 포함



웹브라우저



쿠키 저장소

memberId=1

GET /welcome HTTP/1.1
Cookie: memberId=1

GET /board HTTP/1.1
Cookie: memberId=1

GET /order HTTP/1.1
Cookie: memberId=1

GET /xxx... HTTP/1.1
Cookie: memberId=1

위에서 말했듯 서버로 보내는 모든 요청 메시지에 쿠키 정보가 자동으로 포함된다. 따라서 서버는 요청을 한 클라이언트가 누구인지, 정보를 알 수 있다.

하지만 이러한 쿠키 사용엔 심각한 문제점이 있다.

바로 쿠키를 탈취당했을 경우인데, 위의 쿠키의 key는 memberId이지만 만약 쿠키의 key가 주민번호나 신용카드 번호라면 심각한 상황에 놓이게 된다. 따라서 Session을 사용하는데 세션의 동작은 아래에서 알아보자.



cookie 설정/꺼내기/지우기 예제

- express + cookie-parser 사용 예제

먼저 쿠키 파서를 설치한다

- `npm install cookie-parser`

쿠키 설정

응답(res)객체에 cookie()함수를 이용하여
key값과 value값을 설정한다 옵션 설정도 함께

```
ex27_cookie-parser.js > ...
1 //npm i cookie-parser
2
3 const express = require('express');
4 const cookieParser = require('cookie-parser');
5
6 const app = express();
7 const PORT = 3333;
8 // cookie-parser 미들웨어 등록
9 app.use(cookieParser());
10 //클의 요청을 보낼때 쿠키를 같이 보낸다. 이를 req.headers.cookie(파싱되지 않은 원시쿠키 문자열 key=value;key2=value2)를
11 //통해 쿠키 값에 접근하게 되는데 이를 추출하기 위해 쿠키 파서를 사용하자
12 //쿠키 파서가 없다면 추출과정이 복잡함
13 //cookie-parser 미들웨어를 이용하면 "req.cookies.키값" 로 추출한다 (파싱됨)
14
15 // 쿠키 설정 라우트
16 app.get('/set-cookie', (req, res) => {
17   res.cookie('username', 'Tom_brown', {
18     maxAge: 1000 * 60 * 60, // 1시간 동안 유지
19     httpOnly: true, // 클라이언트에서 js로 접근 못함
20     secure: false, // HTTPS만 허용하고 싶다면 true
21   });
22   res.send('<h1쿠키가 설정되었습니다.</h1>');
23 });
```

개발자 도구에서 확인

크롬 F12 눌러 Application 탭에서 확인

← → ↻ ⓘ localhost:3333/set-cookie

쿠키가 설정되었습니다.

DevTools is now available in Korean [Don't show again](#) [Always match Chrome's language](#) [Switch DevTools to Korean](#)

Elements Console Sources Network Performance Memory **Application** Privacy and security Lighthouse Recorder Redux R

Application

- Manifest
- Service workers
- Storage
 - Local storage
 - http://localhost:...
 - Session storage
 - Extension storage
 - IndexedDB
 - Cookies**
 - http://localhost:...
 - Private state tokens
 - Interest groups

Filter

| Name | Value | Domain | Path | Expires / Max-Age | Size | Ht... |
|----------|-----------|-----------|------|----------------------|------|-------|
| username | Tom_brown | localh... | / | 2025-06-16T13:20:... | 17 | ✓ |

쿠키 꺼내기

req.cookies에서 꺼낸다

```
24
25 // 쿠키 확인 라우트
26 app.get('/get-cookie', (req, res) => {
27   const { username } = req.cookies;
28   if (username) {
29     res.send(`<h1>쿠키에서 가져온 사용자명: ${username}</h1>`);
30   } else {
31     res.send(`<h1>쿠키가 없습니다.</h1>`);
32   }
33 });
34
```

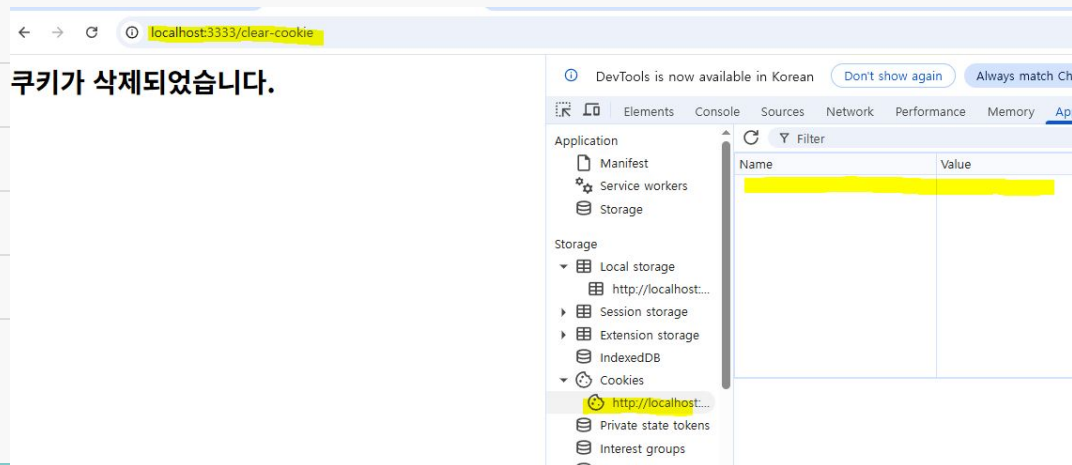
← → 🔍 localhost:3333/get-cookie

쿠키에서 가져온 사용자명: **Tom_brown**

쿠키 삭제

res.clearCookie(키값)

```
34
35 // 쿠키 삭제 라우트
36 app.get('/clear-cookie', (req, res) => {
37   res.clearCookie('username');
38   res.send('<h1> 쿠키가 삭제되었습니다.</h1>');
39 });
40
41 app.listen(PORT, () => {
42   console.log(`서버 실행 중: http://localhost:${PORT}`);
43 });
44
```



Session이란?

◇◇◇ 세션 ◇◇◇

세션이란 일정 시간 동안(디폴트 30분) 같은 사용자(정확히 말하면 같은 브라우저)로 부터 들어오는
일련의 요구들 하나의 상태로 보고 그 상태를 일정하게 유지시키는 기술이다.

예를 들어서 쇼핑몰에서 장바구니를 구현할 때 매우 유용하게 사용될 수 있다. 즉 쇼핑몰에서 사용자가
상품을 주문할 때마다 그 사용자에게 대한 상품을 세션으로 저장해 두면 나중에 한 번에 구매한 물품을
세션에 저장된 정보에서 불러올 수 있는 것입니다.

세션 동작 원리

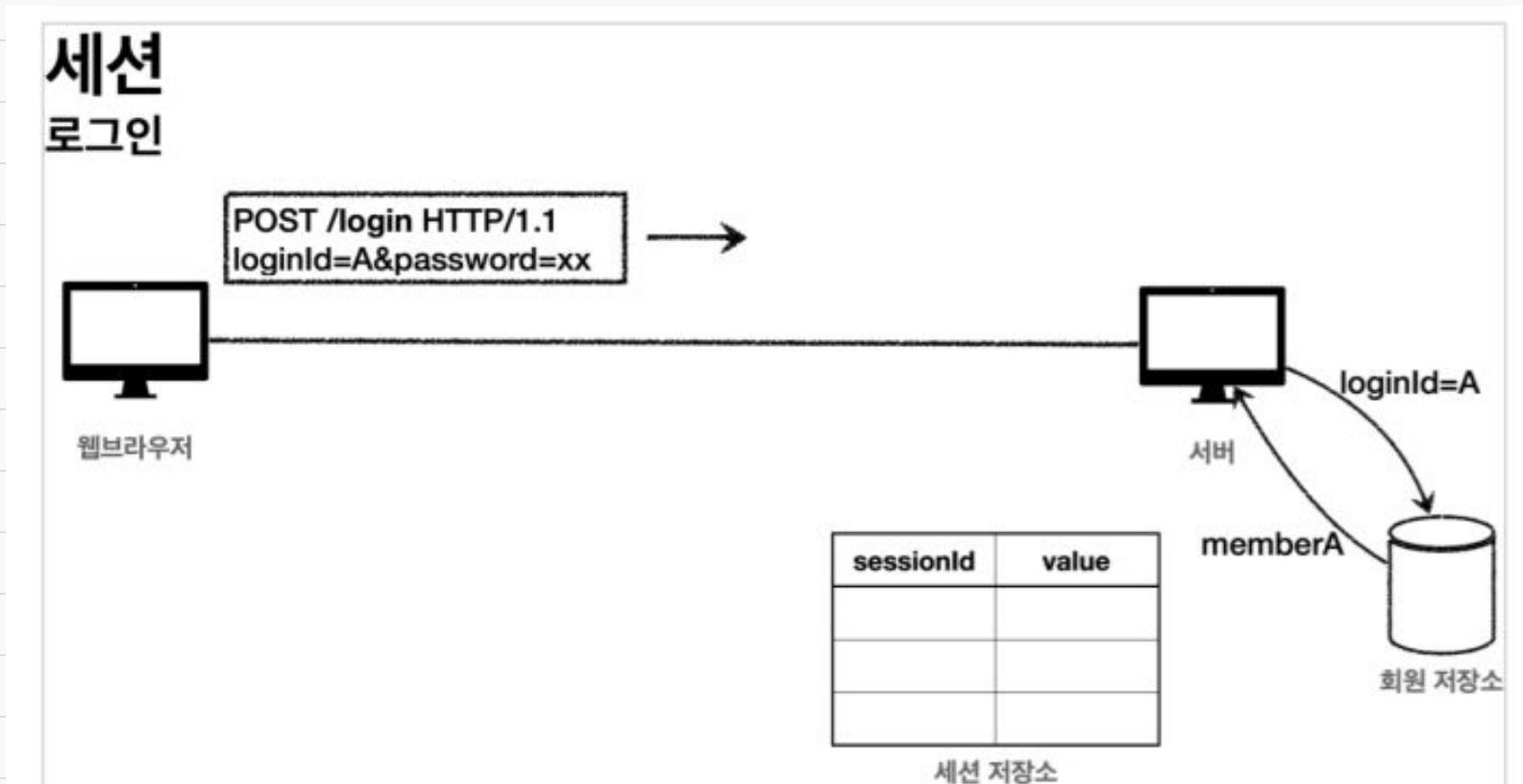


그림 출처:

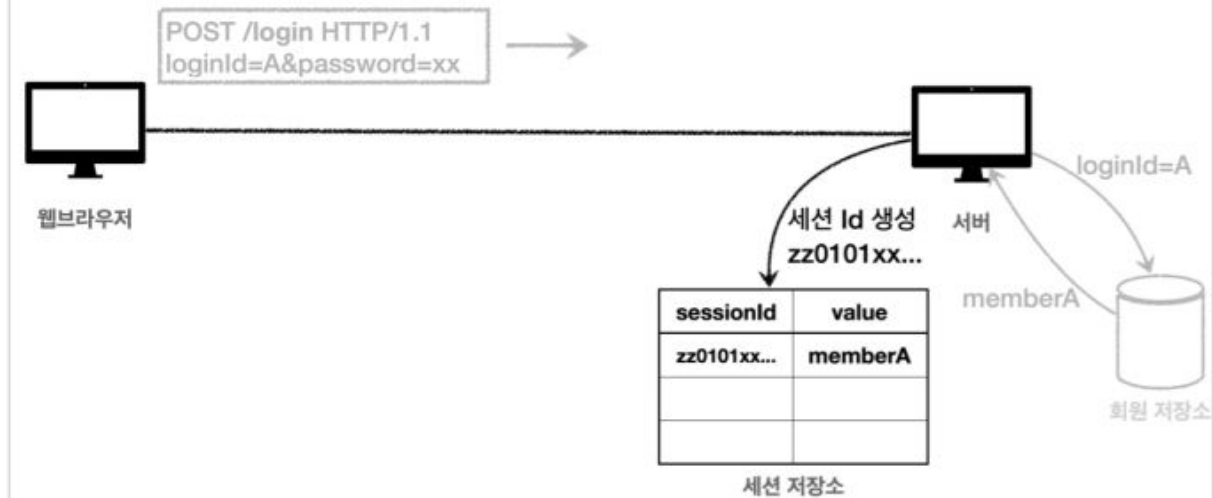
<https://velog.io/@rlfrkdms1/%EC%BF%A0%ED%82%A4%EC%99%80-%EC%84%B8%EC%85%98%EC%9D%98-%EB%8F%99%EC%9E%91-%EC%9B%90%EB%A6%AC%EC%99%80-%EC%84%B8%EC%85%98%EC%9D%98-%EA%B5%AC%EC%A1%B0>

Session 생성

세션 생성

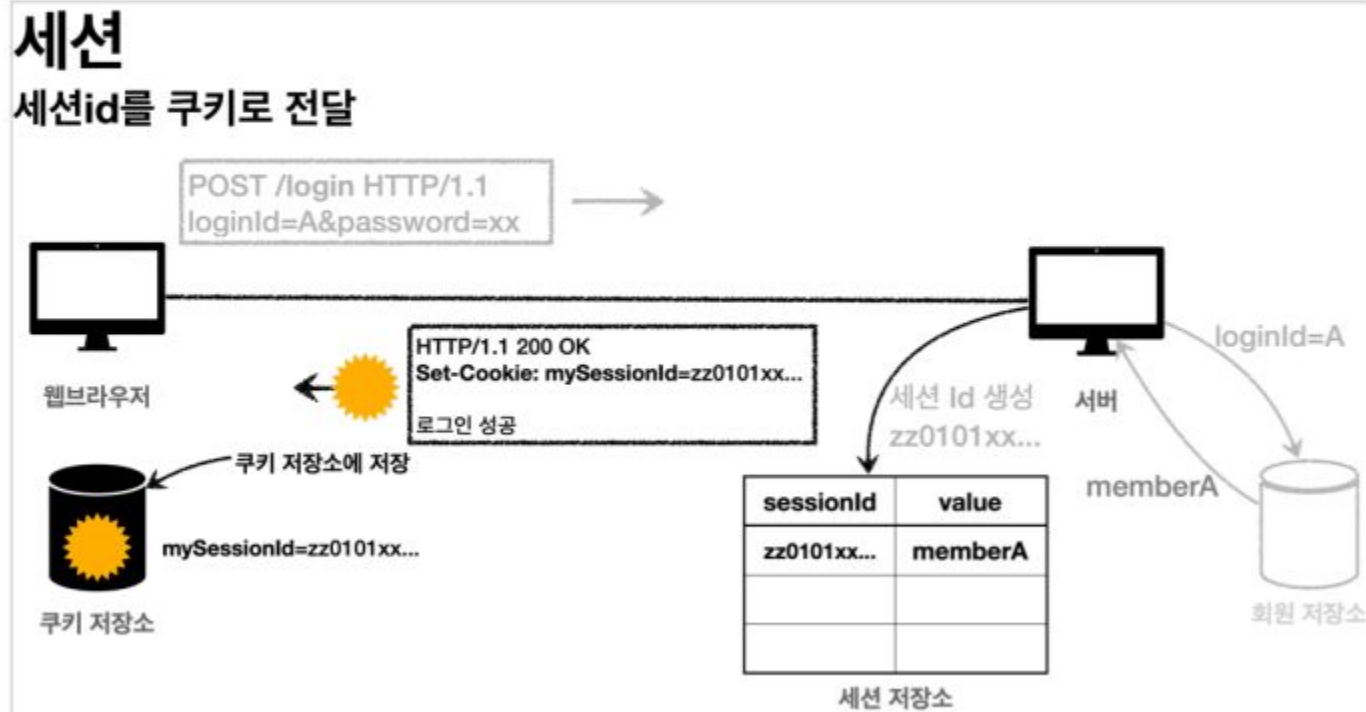
세션

세션 관리



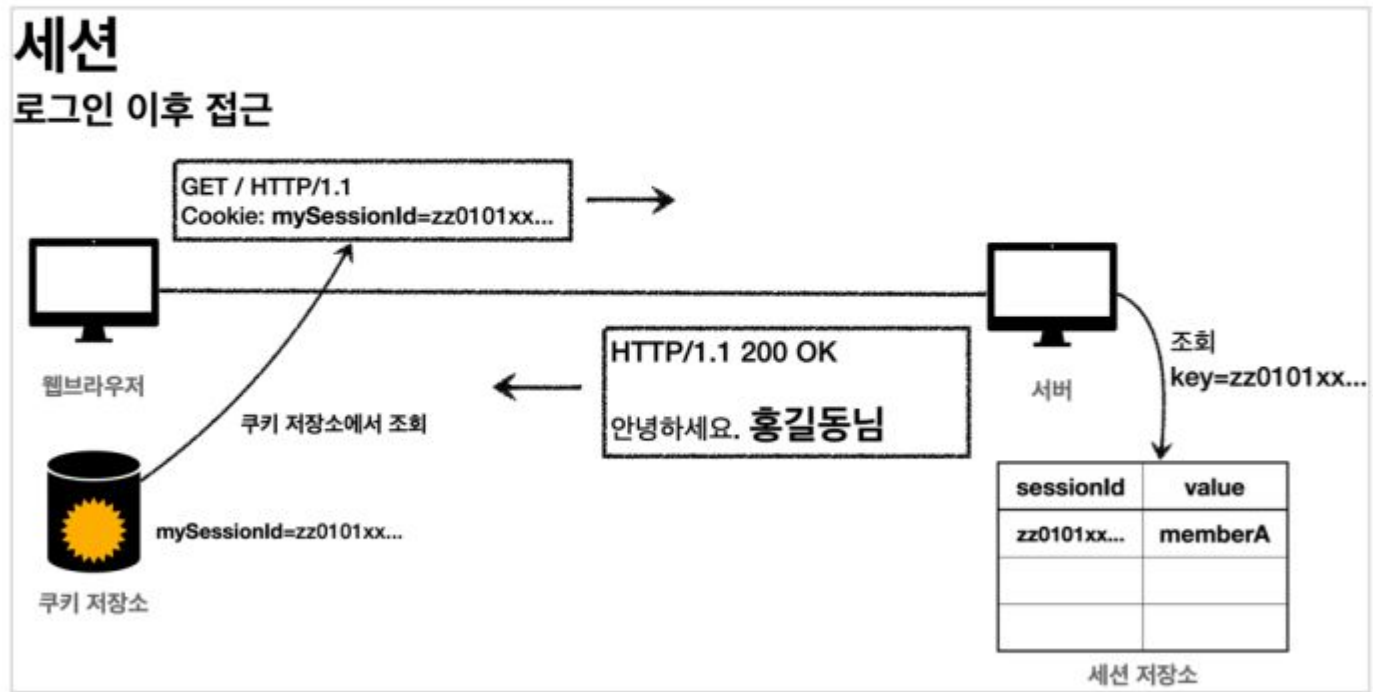
Session Id 쿠키 전송


세션 id를 응답 쿠키로 전달



요청시 Session Id 쿠키 전달

클라이언트의 세션 id 쿠키 전달





session 을 이용한 예제

- express + express-session 사용 예제

express-session 패키지를 설치하자

- npm install express-session**

세션에 로그인한 사용자 정보 저장

ex28_session.js > ...

```
1 //express-session 패키지 설치
2 //npm i express-session
3 const express = require('express');
4 const session = require('express-session');
5 const app = express();
6 const PORT = 3333;
7 // 세션 미들웨어 설정
8 app.use(
9   session({
10     secret: 'mySecretKey12!@#', // 세션 암호화에 사용되는 키-서명키 (세션 id에 디지털 서명을 추가해 위조를 방지하기 위해 설정함)
11     resave: false, // 요청마다 세션을 다시 저장할지 여부(동일한 내용일 때는 다시 저장안해도 되도록 false설정)
12     saveUninitialized: true, // 초기화되지 않은 세션도 저장할지 여부
13     cookie: {
14       maxAge: 1000 * 60 * 10, // 쿠키 만료 시간: 10분 =>세션을 식별하는 쿠키(세션 ID)의 생존 기간을 설정
15     },
16   })
17 );
18 // 세션 저장
19 app.get('/login', (req, res) => {
20   req.session.user = {
21     id: 1,
22     name: 'Dooly',
23   };
24   res.send('<h1>로그인 처리 완료: 세션 저장됨</h1>');
25 });
```

← → ↻ ⓘ localhost:3333/login

로그인 처리 완료: 세션 저장됨

로그인한 사용자 정보 확인

```
26
27 // 세션 값 읽기
28 app.get('/profile', (req, res) => {
29   if (req.session.user) {
30     res.send(`<h1 style='color:green'>현재 사용자: ${req.session.user.name}</h1>`);
31   } else {
32     res.send(`<h1>로그인한 사용자가 없습니다.</h1>`);
33   }
34 });
35
```

← → ↻ ⓘ localhost:3333/profile

현재 사용자: Dooly

로그아웃 처리

```
36 // 세션 삭제 (로그아웃)
37 app.get('/logout', (req, res) => {
38   req.session.destroy((err) => {
39     if (err) {
40       return res.send('세션 삭제 중 오류 발생');
41     }
42     res.clearCookie('connect.sid'); // 기본 세션 쿠키 이름 (세션을 식별하는 용도로 사용)
43     // Express에서 express-session 미들웨어를 사용할 때 기본적으로 사용하는 세션 쿠키 이름
44     res.send('<h1>로그아웃: 세션 삭제됨</h1>');
45   });
46 });
47
48 app.listen(PORT, () => {
49   console.log(`서버 실행 중: http://localhost:${PORT}`);
50 });
```

localhost:3333/logout

로그아웃: 세션 삭제됨

localhost:3333/profile

로그인한 사용자가 없습니다.

참고사이트

<https://velog.io/@rlfrkdms1/%EC%BF%A0%ED%82%A4%EC%99%80-%EC%84%B8%EC%85%98%EC%9D%98-%EB%8F%99%EC%9E%91-%EC%9B%90%EB%A6%AC%EC%99%80-%EC%84%B8%EC%85%98%EC%9D%98-%EA%B5%AC%EC%A1%B0>