〈쿠키와 캐시〉 p.208 ~

※ 쿠키·캐시 공/차 먼저 보기

공 쿠키·캐시 모두 클라이언트에서, 관리하는 데이터이며, 정보를 저장하는 목적을 가진다.

차 쿠키

- 웹서버 → PC로 보내는 작은 파일들을 저장하며, 누군가 특정 웹사이트를 접속할 때 발생
- 정보 저장을 위해 사용되며, 최종적인 목적은 사용자의 인증을 도와준다
- 만료기간이 있어서, 시간이 지나면 자동 삭제된다.

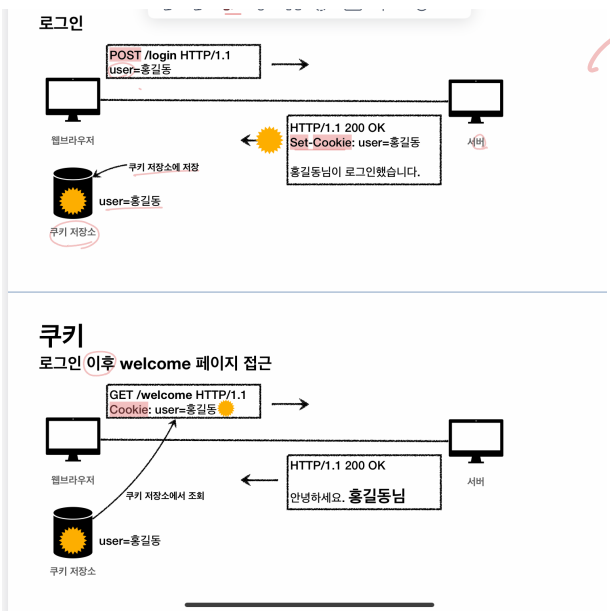
캐시

- 웹페이지 로드를 저장하기 위한 임시저장소, 그림파일 / 음악파일
- 최종적인 목적은 웹페이지가 더 빠르게 랜더링되도록 도와주는것
- 사용자가 수동으로 직접 삭제해야 한다.

⇒ 1. 쿠키

- 기본적으로 HTTP는 stateless 프로토콜이기에, 클-서버는 서로 상태를 유지 X
- e.g) "홍길동" 유저가 로그인이 끝난후, "안녕하세요, 홍길동님" 이라는 띄워줘야 한다면, 메인페이지를 요청할때 현재 사용자 정보를 또 요청해야 한다.

↳ 따라서 쿠키를 미사용한다면, 모든 요청에 정보를 넘겨야 하는 문제가...



※쿠키 작동방식

Set-Cookie : 서버 → 클라이언트로 쿠키전달

Cookie : 클라이언트가 서버에게서 받은 쿠키 저장하고, HTTP 요청시 서버로 전달.

↓

※쿠키 특징

쿠키 정보는 항상 서버에 전송되며, 네트워크 트래픽 추가유발

- 따라서 최소한의 정보만 사용하고, 보안에 민감한 데이터는 저장하지 않을것.

(• 만약 서버에 전송하지 않고, 웹브라우저 내부에 데이터를 저장하고 싶다면 웹스토리지 이용하세요)

- 예) set-cookie: **sessionId=abcde1234**; **expires**=Sat, 26-Dec-2020 00:00:00 GMT; **path**=/; **domain**=.google.com; **Secure**

쿠키 - 생명주기
Expires, max-age

- Set-Cookie: **expires**=Sat, 26-Dec-2020 04:39:21 GMT
  - 만료일이 되면 쿠키 삭제
- Set-Cookie: **max-age**=3600 (3600초)
  - 0이나 음수를 지정하면 쿠키 삭제
- 세션 쿠키: 만료 날짜를 생략하면 브라우저 종료시 까지만 유지
- 영속 쿠키: 만료 날짜를 입력하면 해당 날짜까지 유지

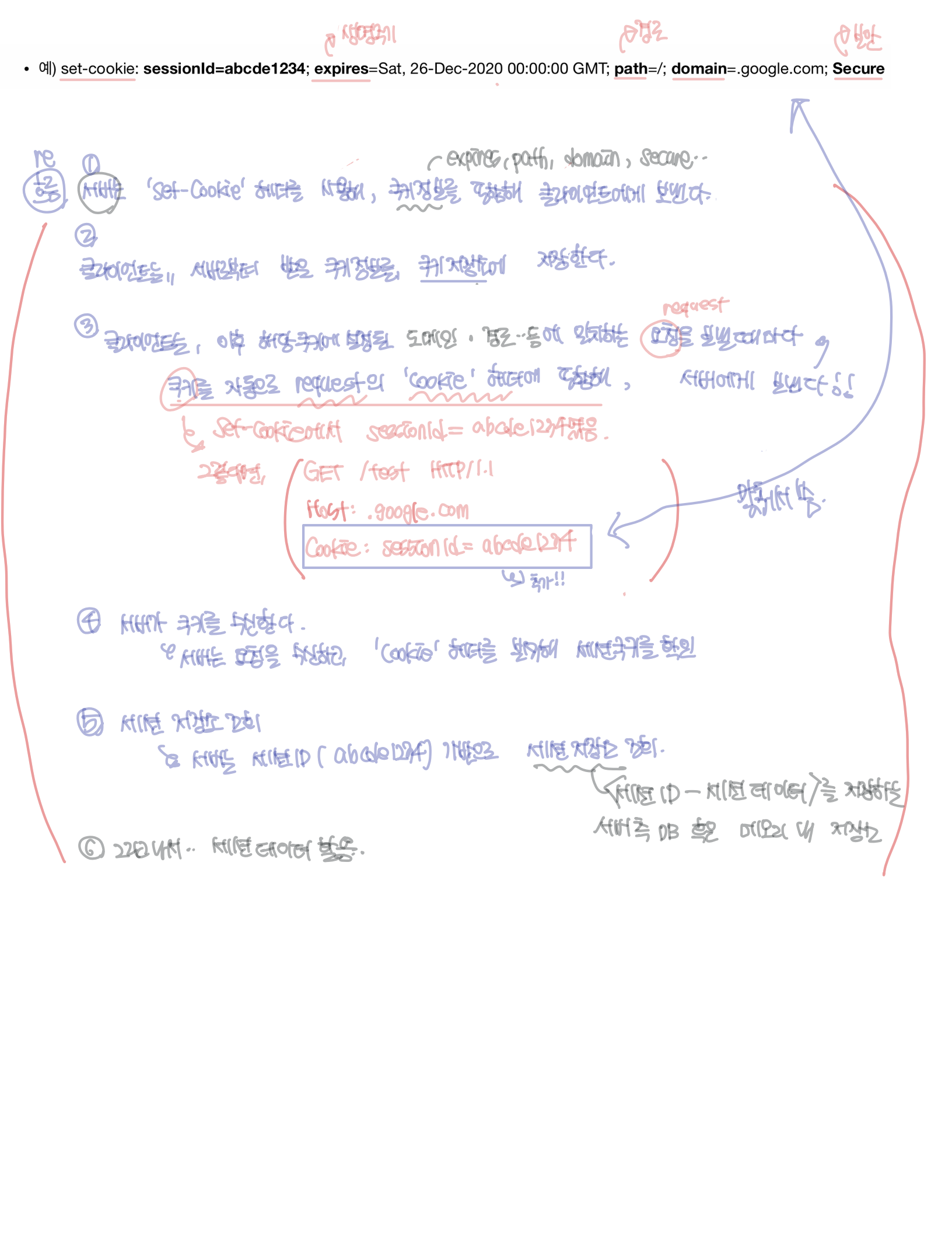
Domain

- 예) domain=example.org
- **명시: 명시한 문서 기준 도메인 + 서브 도메인 포함**
  - domain=example.org를 지정해서 쿠키 생성
    - example.org는 물론이고
    - dev.example.org도 쿠키 접근
- **생략: 현재 문서 기준 도메인만 적용**
  - example.org 에서 쿠키를 생성하고 domain 지정을 생략
    - example.org 에서만 쿠키 접근
    - dev.example.org는 쿠키 미접근

Secure, HttpOnly, SameSite

- Secure
  - 쿠키는 http, https를 구분하지 않고 전송
  - Secure를 적용하면 https인 경우에만 전송
- HttpOnly
  - XSS 공격 방지
  - 자바스크립트에서 접근 불가(document.cookie)
  - HTTP 전송에만 사용
- SameSite
  - XSRF 공격 방지
  - 요청 도메인과 쿠키에 설정된 도메인이 같은 경우만 쿠키 전송

- 예) set-cookie: **sessionId=abcde1234**; **expires**=Sat, 26-Dec-2020 00:00:00 GMT; **path**=/; **domain**=.google.com; **Secure**



2. 캐시

(캐시가 없을 때)

예를 들어, 브라우저에서 /star.jpg로 크기가 1.1M인 동일한 또 다른 star.jpg를, 서버로부터 받으려고 시도한다면?

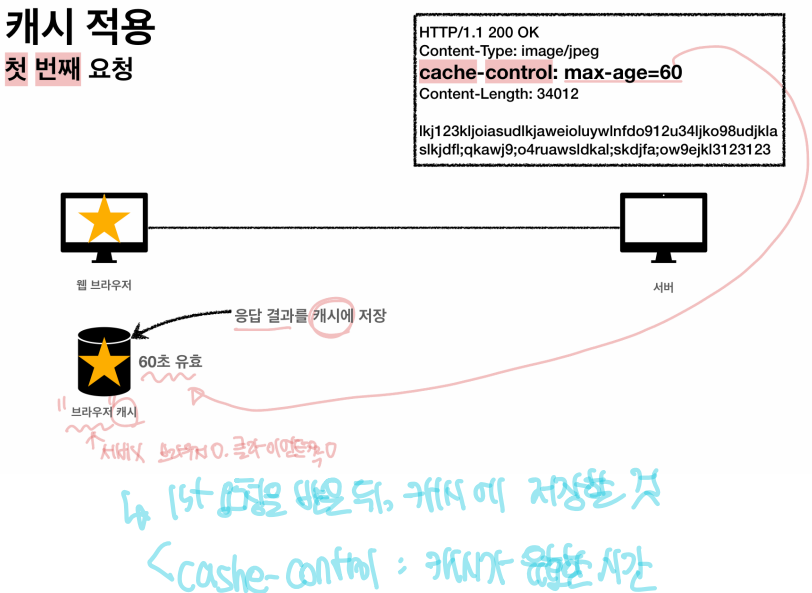
캐시가 없다면, 첫번째 요청에서 1.1M를 전송받을 것이다.

두번째 전송은? 역시나 같은 데이터임에도 불구하고 1.1M를 또다시 전송받을 것이다!

∴ 캐시가 없다면, 데이터가 똑같아도 계속 데이터를 다운받아야 해서, 굉장히 느릴 것이다.

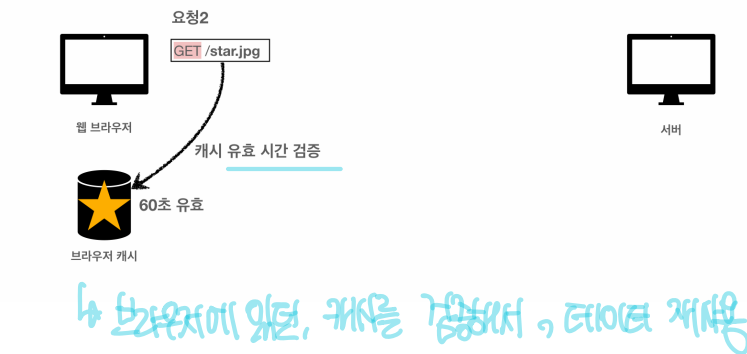
캐시를 적용한다면?

※ 캐시 - 특징



↳ 1번 요청을 했을 때, 캐시에 저장할 것

<cache-control : 캐시가 유효한 시간



↳ 브라우저에 있던, 캐시를 검증해서, 데이터 재사용

⇒ 따라서 캐시를 사용하게 되면 네트워크를 재요청할 필요 X, 브라우저 로딩속도가 빨라진다.

⇒ 그런데, 캐시 유효시간 경과했을 때 60초가 경과된 후, 같은 데이터를 또 요청하면??

당연하게도 유효기간이 끝났기에, 서버를 통해 데이터를 다시 가져온 후, 캐시에 저장한다.

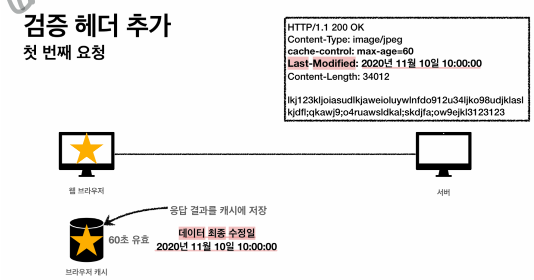
⇒ 만약 캐시 만료했어도, 서버에서 데이터를 변경하지 않은 상황에서, 데이터를 또다시 전송하지 않고, 저장되어있던 캐시를 재사용할 수는 없을까?

↳ 유효기간이 끝난 캐시를 재사용하려면 당연히도 하나의 검증과정이 필요한데, 클라이언트에서 요청한 데이터와, 서버가 가진 데이터가 같다는 사실을 확인해야 한다.

↳ 검증헤더를 추가해서 이를 판단할 수 있다.

⊕ 캐시 - 검증헤더 추가 ① (p.313)

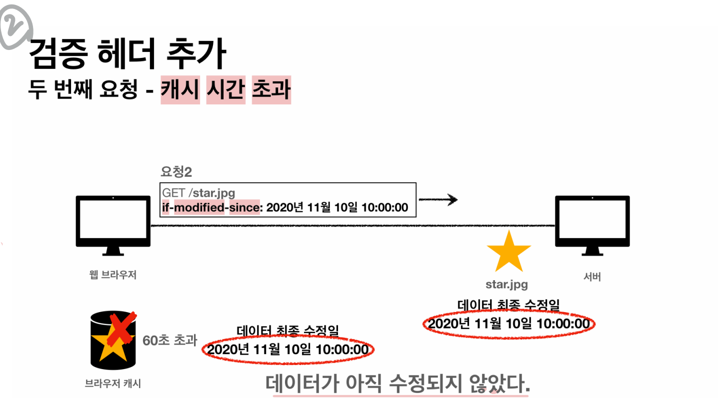
①



→ 애초에 첫번째 요청이 일어났을 때 Last-Modified ; 마지막 변경일자 를 저장해둔다.

→ 이후, 캐시 시간이 초과되어 두번째 요청이 일어난다면? (60초)

②

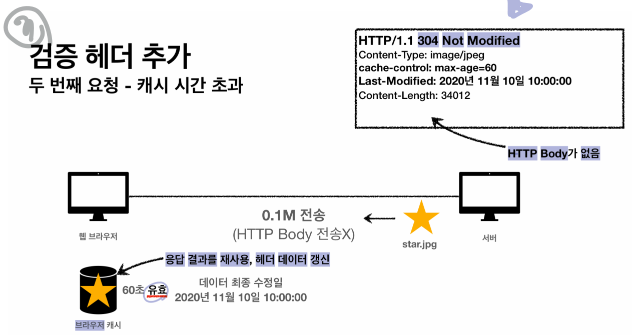


- 두번째 요청에선, 캐시가 가지고 있는 데이터 최종 수정일인 if-modified-since 를 요청에 포함

- 따라서, 서버에서 데이터 최종수정일이 동일하다고 판단하고, HTTP response에 상태코드 304 ⊕ 헤더 메타정보만 포함시켜 (이전 페이지 언급이) (HTTP body X) 클라이언트에게 보낸다.

60초 유효 (초과 X)

③



→ 브라우저는, 이제 응답결과를 재사용해 헤더 데이터를 갱신하고, 캐시에 있는 데이터를 재사용할 수 있게 됨.

정리

- If-Modified-Since: 이후에 데이터가 수정되었으면?
  - **데이터 미변경 예시**
    - 캐시: 2020년 11월 10일 10:00:00 vs 서버: 2020년 11월 10일 10:00:00
    - **304 Not Modified**, 헤더 데이터만 전송(BODY 미포함) → Body 안보내도, 캐시에 해당 내용있음
    - 전송 용량 0.1M (헤더 0.1M)
  - **데이터 변경 예시**
    - 캐시: 2020년 11월 10일 10:00:00 vs 서버: 2020년 11월 10일 11:00:00
    - **200 OK**, 모든 데이터 전송(BODY 포함) → Body에 변경된 data 포함해서 보내야함!
    - 전송 용량 1.1M (헤더 0.1M, 바디 1.0M)

⇒ 위 방법은, 캐시에 저장된 데이터를 재활용하며, 용량이 적은 헤더 정보만 다시 다운로드 받기에, 매우 실용적인 해결책 !!

⇒ 그러나, 데이터를 수정해서 날짜가 다르지만, 같은 데이터를 수정해서 데이터 결과가 똑같은 경우에도 위 방법으로 다른 데이터라 인식할 것...

또한, 서버에서 별도의 캐시 로직을 관리하고 싶기도 할 것.

⇒ 따라서, 두번째 방법인 ETag를 이용 !

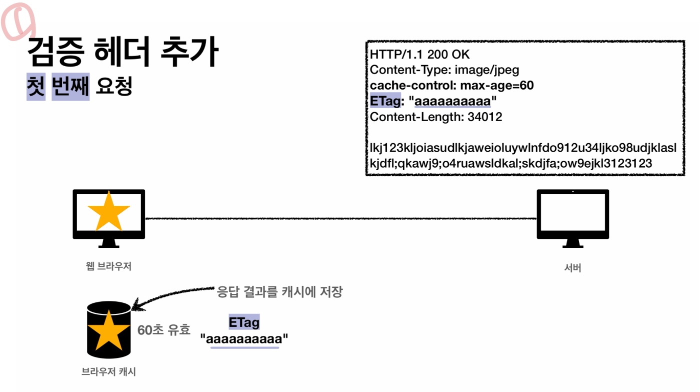
ETag

- ETag = Entity Tag
- 캐시용 데이터에 고유한 이름 달아줌.
- 데이터 변경되면, 이름도 바꾸어서 변경해줌

* ETag 보내서 같으면 유지, 다르면 데이터 다시 받기.

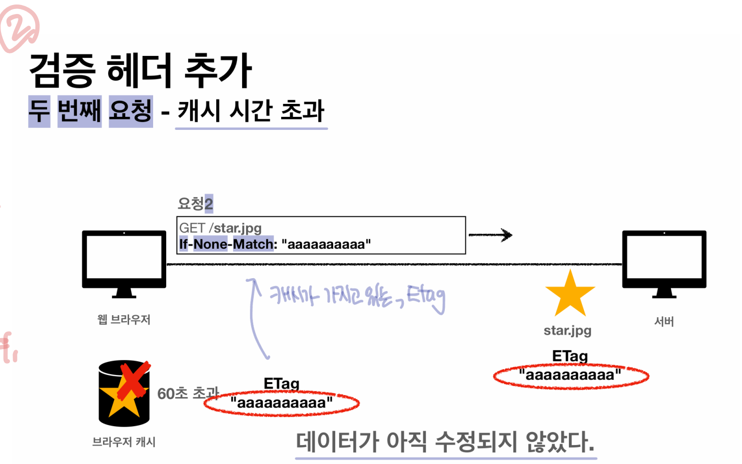
㊎ 캐시 - 검증헤더 추가 ② (p.324 -)

①



→ ETag는 서버에서 생성하므로, 첫번째 요청이 일어나면, 클라이언트는 ETag를 받아 캐시에 저장.

②



캐시 시간 지난 뒤, 두번째 요청에서는

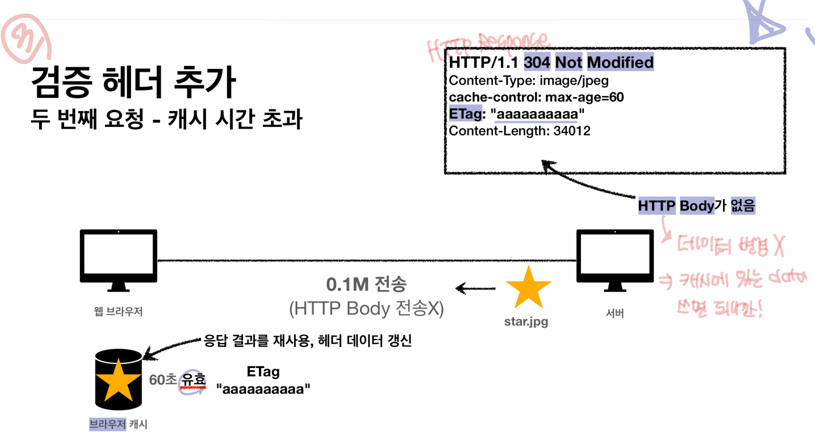
(클) If-None-Match에 캐시에 있던 ETag를 담아, 서버에 요청!

(서) 서버는, ETag가 동일해서 수정된 것을 확인하고

HTTP Response에 상태코드 304 ⊕ HTTP 메타정보만 포함해서, (HTTP 바디X)

(클)에게 보냄!

③



그러면, 브라우저는 HTTP Response를 해석해, 동일한 ETag를 보고 캐시 재사용 가능!

ㅎ 캐시 - 캐시제어 헤더.

Cache-Control (캐시 제어), Expires (캐시 만료일) ..

①

Cache-Control

캐시 지시어(directives)

- Cache-Control: max-age
  - 캐시 유효 시간, 초 단위
- Cache-Control: no-cache
  - 데이터는 캐시해도 되지만, 항상 원(origin) 서버에 검증하고 사용 ← 아래에서 설명함
- Cache-Control: no-store
  - 데이터에 민감한 정보가 있으므로 저장하면 안됨
    (메모리에서 사용하고 최대한 빨리 삭제)

②

Expires

캐시 만료일 지정(하위 호환)

- expires: Mon, 01 Jan 1990 00:00:00 GMT

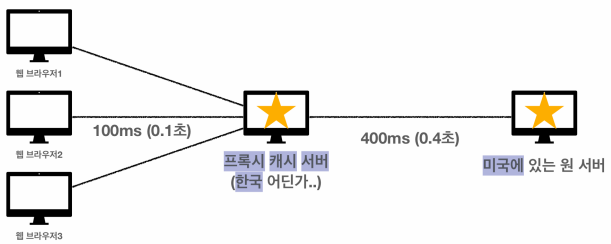
- 캐시 만료일을 정확한 날짜로 지정
- HTTP 1.0 부터 사용
- 지금은 더 유연한 Cache-Control: max-age 권장
- Cache-Control: max-age와 함께 사용하면 Expires는 무시

ㅎ 캐시 - 프록시 캐시

보통 캐시 서버는, 중간에 아래와 같은 프록시 서버를 둔다.

프록시 캐시 도입

첫 번째 요청



중간에 프록시 서버가 있더라도, 원 서버까지 갔다와야 제대로 된 데이터가 올 것이다.

- 아까 위에서 언급했던 Cache-Control : no-cache 에서 "항상 원 서버에서 검증하고 사용"은 이 상황과 같이, 이것에 있는 원 서버에서 항상 검증해야 한다는 것!

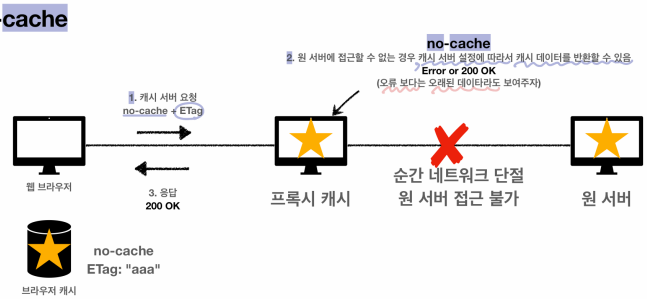
(결국 이도 캐시의 목적인, 웹브라우저 레벨의 모든 것을 위해, 클라이언트에서 정보저장..)

ㅎ Cache-Control : no-cache vs Cache-Control : must-revalidate

①

no-cache vs must-revalidate

no-cache

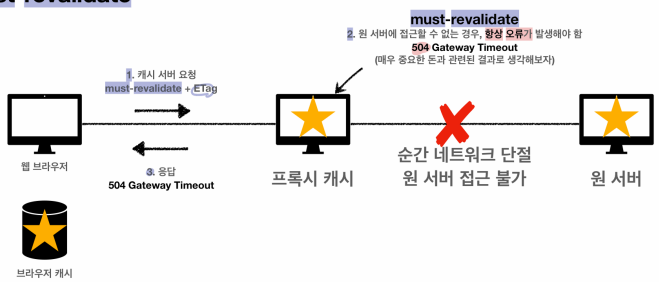


no-cache는 원 서버에 접근할 수 없는 경우,
서버 설정에 따라 캐시 데이터를 반환할 수도 있음.
(오류보다는, 오래된 데이터라도 보여주자는 마인드..)

②

no-cache vs must-revalidate

must-revalidate



must-revalidate는 원 서버에 접근할 수 없는 경우,
무조건 에러 발생! (504 error)
(ex - 돈과 관련된 매우 중요한 데이터가 원 서버에 있다면, 항상 504가 발생해야 한다!)

[서버에 저장하는 session으로, 김영한 - MVC2 - 로그인 처리 1 강의에 세션 예제 만드는 부분이] Session!

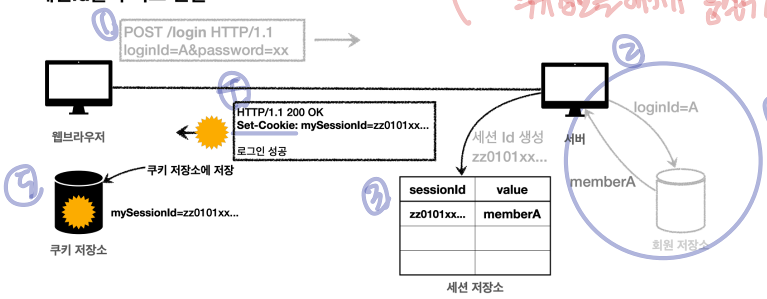
<쿠키의 보안상 문제를 해결한 session!!>

(클라이언트에서 쿠키값 변경 가능! 쿠키정보를 해커가 훔쳐간다)

세션id를 응답 쿠키로 전달

세션

세션id를 쿠키로 전달



① 클라이언트가 로그인할 때, loginID · password를 넘긴다.

② 서버는 해당 id, pw로 로그인 기능으로, 해당 사용자가 맞는지 체크

MemberRepository에 loginId 있는지, 있다면 password는 일치하는지?

로그인 성공!

③ 서버의 세션 저장소에 <sessionId, value>를 저장한다.

추정 불가능한 랜덤값! / 클이 아닌 서버에만, 회원관련 정보를 저장!!

④ 서버는 클라이언트에게 쿠키를 보낼 때, sessionId만 담아 보내준다. (value 회원정보 X)

⑤ 결과적으로, 쿠키저장소에는 sessionId만 가진다.

클라이언트와 서버는 결국 쿠키로 연결이 되어야 한다.

- 서버는 클라이언트에 `mySessionId` 라는 이름으로 세션ID 만 쿠키에 담아서 전달한다.
- 클라이언트는 쿠키 저장소에 `mySessionId` 쿠키를 보관한다.

중요

- 여기서 중요한 포인트는 회원과 관련된 정보는 전혀 클라이언트에 전달하지 않는다는 것이다.
- 오직 추정 불가능한 세션 ID만 쿠키를 통해 클라이언트에 전달한다.

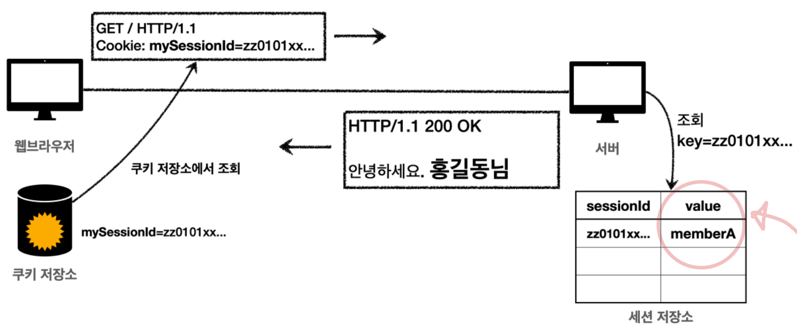
랜덤값 생성!! (추정가능하게 세션ID를 2,3.. 이런식으로 하면 안되겠지!!)

로그인하여
① 서버 세션Id - value 저장 (랜덤값, 회원정보)
② 클 쿠키에 세션Id 저장

로그인 이후?

세션

로그인 이후 접근



- 클라이언트는 요청시 항상 `mySessionId` 쿠키를 전달한다.
- 서버에서는 클라이언트가 전달한 `mySessionId` 쿠키 정보로 세션 저장소를 조회해서 로그인시 보관한 세션 정보를 사용한다.

정리

세션을 사용해서 서버에서 중요한 정보를 관리하게 되었다. 덕분에 다음과 같은 보안 문제들을 해결할 수 있다.

- 쿠키 값을 변조 가능, → 예상 불가능한 복잡한 세션Id를 사용한다.
- 쿠키에 보관하는 정보는 클라이언트 해킹시 털릴 가능성이 있다. → 세션Id가 털려도 여기에는 중요한 정보가 없다.
- 쿠키 탈취 후 사용 → 해커가 토큰을 털어가도 시간이 지나면 사용할 수 없도록 서버에서 세션의 만료시간을 짧게 (예: 30분) 유지한다. 또는 해킹이 의심되는 경우 서버에서 해당 세션을 강제로 제거하면 된다.