모든 개발자를 위한 HTTP 웹 기본 지식

- 김영한님 강좌

목록

1. 인터넷 네트워크
2. URI와 웹 브라우저 요청 흐름
3. HTTP 기본
4. HTTP 메서드
5. HTTP 메서드 활용
6. HTTP 상태코드
7. HTTP 헤더 1 - 일반헤더
8. HTTP 헤더 2 -캐시와 조건부 요청

[1장 인터넷 네트워크]

< IP >
Internet Protocol

#역할

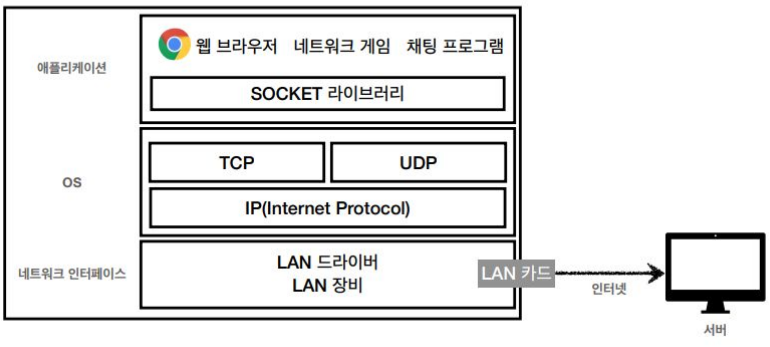
- 지정한 IP 주소에 데이터 전달
- 패킷 으로 데이터 전달

#한계

- 비연결성
- 비신뢰성
- 프로그램 구분

<TCP / UDP>
-Transmission Controller Protocol / User Datagram protocol

프로토콜 계층

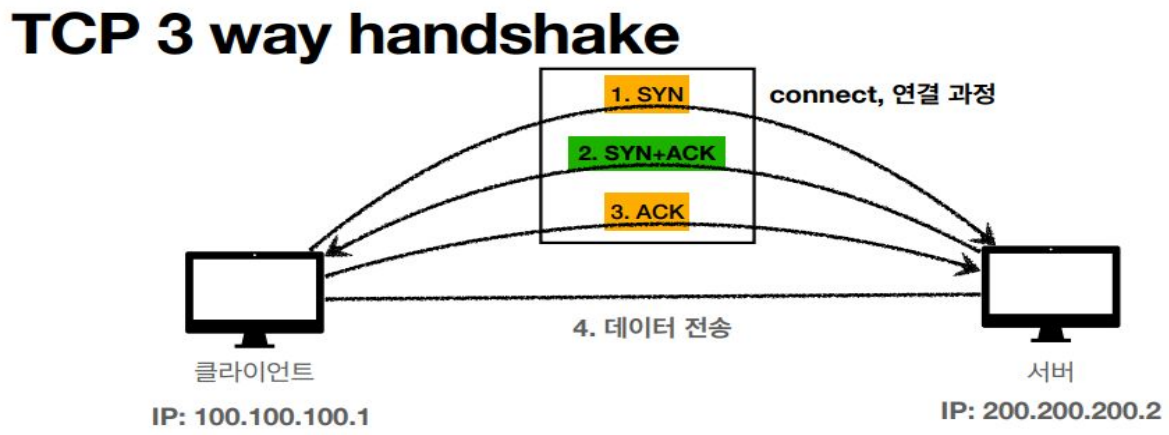


TCP/IP 패킷 정보

TCP 특징

- 연결지향 - TCP 3 way handshake (가상 연결)
- 데이터 전달 보증
- 순서 보장
  => 신뢰할 수 있는 프로토콜 / 현재는 대부분 TCP

TCP 3 way handshake



SYN: 접속 요청

ACK: 요청 수락

참고: 3. ACK와 함께 데이터 전송 가능

UDP 특징

- 기능이 거의 없음 (마치 하얀 도화지)
- 연결지향 - TCP 3 way handshake X
- 데이터 전달 보증 X
- 순서 보장 X
- 단순하고 빠름

  => IP와 거의 같다 + 포트/체크섬 정도 추가 / 애플리케이션에서 추가작업 필요

<PORT>

- 같은 IP 내에서 프로세스 구분
- 0 ~ 65535 할당 가능
- 0- ~ 1023 잘 알려진 포트

<DNS>

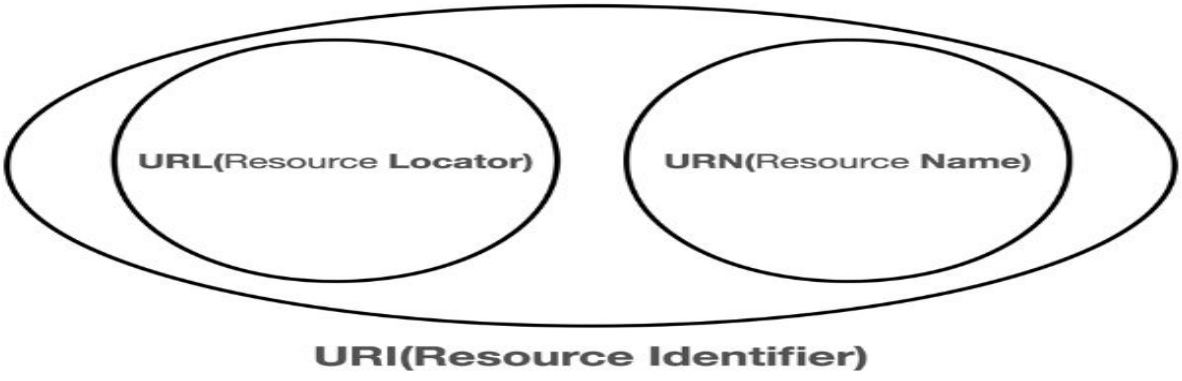
- Domain Name System
- 도메인 이름을 IP주소로 변환

[2장 URI와 웹 브라우저 요청 흐름]

<URI>
Uniform Resource Identifier

URI / URL / URN

- URI 는 로케이터(locator), 이름(Name) 또는 둘다 추가로 분류될 수 있다
- URL : Uniform Resource Locator
- URN : Uniform Resource Name



URI 단어 뜻

- Uniform : 리소스 식별하는 통일된 방식
- Resource : 자원, URI로 식별할 수 있는 모든 것(제한x)
- Identifier : 다른 항목과 구분하는데 필요한 정보

URL / URN 단어 뜻

- URL
  - Locator : 리소스가 있는 위치를 지정
- URN
  - Name : 리소스에 이름을 부여
- 위치는 변할 수 있지만 이름은 변하지 않는다
- URN 이름만으로 실제 리소스를 찾을 수 있는 방법은 보편화 X
- 통상 URI를 URL과 같은 의미

URL 문법

1) Scheme

- 주로 프로토콜 사용
- http : 80 / https : 443
- 포트는 생략가능

2) user info

- URL에 사용자정보 포함에서 인증
- 거의 사용 X

3) hos

- 호스트명
- 도메인명 또는 IP주소 직접 사용 가능

4) PORT
- 일반적으로 생략
- 접속포트

5) path
- 리소스 경로
- 계층적 구조

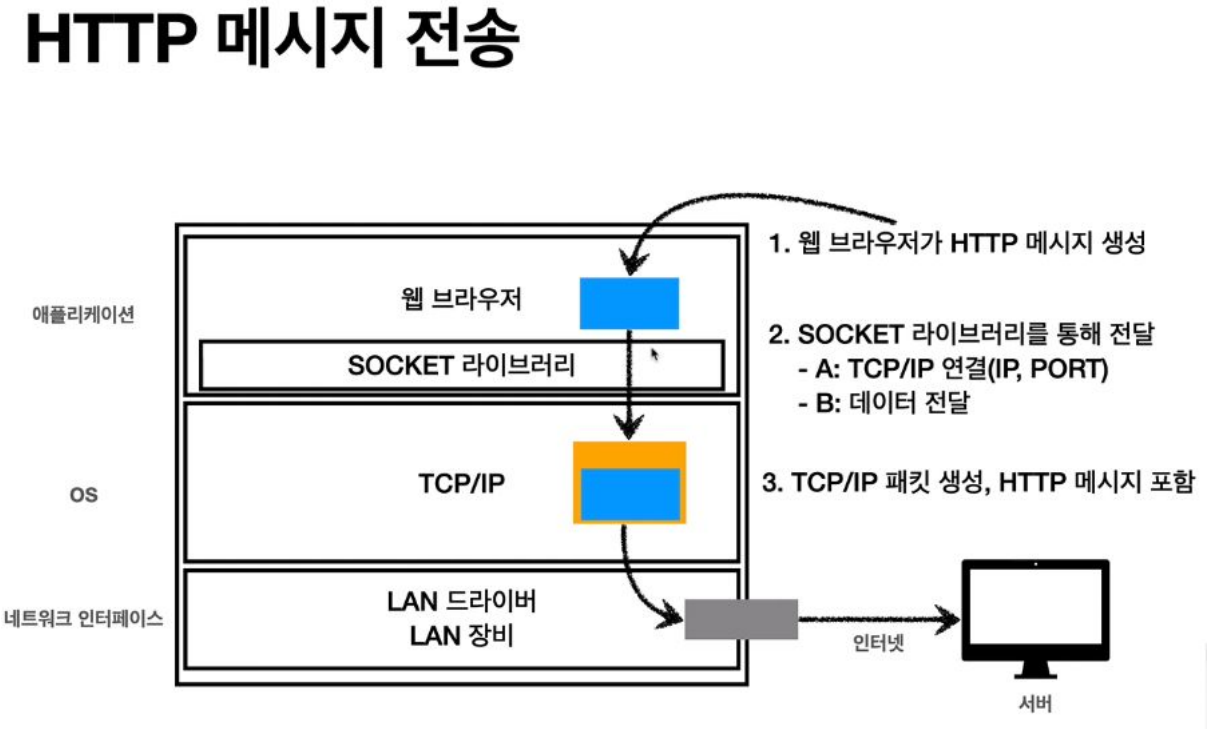
6) query
- key=value 형태
- ?로 시작 &추가 가능 ?keyA=valueA&keyB=valueB
- query parameter, query string 등으로불림 - 웹서버에 제공하는 파라미터, 문자형태

7) fragment
- html 내부 북마크 등에 사용
- 서버에 전송하는 정보 X

<웹 브라우저 요청 흐름>

1) HTTP 요청 메시지 생성
2) HTTP 요청 메시지 전송
3) HTTP 응답 메시지

HTTP 메시지 전송



패킷 생성
- 출발지 IP, PORT / 목적지 IP, PORT / 전송데이터 포함

[3장 HTTP 기본]

<모든 것이 HTTP>

#HTTP
- HyperText Transfer Protocol
- HTTP 메시지에 모든 것을 전송
    - HTML/ 텍스트 / 이미지 / 영상 파일 등
    - JSON, XML (API)
    - 거의 모든 형태의 데이터 전송 가능
    - 서버간에 데이터를 주고 받을 때도 대부분 HTTP 사용

HTTP 1.1
- 가장 많이 사용하고 가장 중요한 버전

기반 프로토콜
- TCP : HTTP/1.1 , HTTP/2
- UDP : HTTP/3
- 현재는 HTTP/1.1 주로 사용
- HTTP/2,/3 도 증가

HTTP 특징
- 클라이언트 서버 구조
- 무상태 프로토콜(스테이트리스), 비연결성
- HTTP메시지
- 단순함 / 확장 가능

<클라이언트 서버 구조>

클라이언트 서버 구조
- Request Response 구조
- 클라이언트는 서버에 요청을 보내고 대기
- 서버가 요청에 대한 결과를 만들어서 응답

<Stateful / Stateless>

무상태 프로토콜
- Stateless
- 서버가 클라이언트의 상태를 보존 X
- 장점 : 서버 확장성 높음 (스케일 아웃)
- 단점 : 클라이언트가 추가 데이터 전송

Stateless 실무 한계
- 모든 것을 무상태로 설계 할 수 있는 경우도 있고 없는 경우도 있다
- 무상태
    - ex) 로그인이 필요 없는 단순한 서비스 소개 화면

- 상태유지
    - ex) 로그인
- 로그인한 사용자의 경우 로그인 했다는 상태를 서버에 유지
- 일반적으로 브라우저 쿠키와 서버 세션등을 사용해서 상태유지
- 상태 유지는 최소한만 사용

<비연결성 connectionless

비연결성
- HTTP는 기본이 연결을 유지하지 않는 모델
- 일반적으로 초 단위의 이하의 빠른 속도로 응답
- 1시간 동안 수천명이 서비스를 사용해도 서버에서 동시에 처리하는 요청은 수십개 이하로 매우작음
    - ex) 웹 브라우저에서 계속 연속해서 검색 버튼을 누르지는 않는다
- 서버 자원을 매우 효율적으로 사용 할 수 있음

비연결성 한계와 극복
- TCP/IP 연결을 새로 맺어야함 - 3 way handshake 시간추가
- 웹 브라우저로 사이트를 요청하면 HTML 뿐 아니라 자바스크립트/css/추가 이미지 등 수 많은 자원이 함께 다운로드
- 지금은 HTTP 지속연결 (Persistent Connections)로 문제 해결
- HTTP/2 HTTP/3 에서 더 많은 최적화

<HTTP 메시지>

HTTP 메시지 구조

- 공백 라인은 무조건 있어야한다

시작라인

1) 요청 메시지
    - HTTP 메서드
    - 요청대상
    - HTTP Version
2) HTTP 메서드
    - 종류 : GET/POST/PUT/DELETE 등
    - 서버가 수행해야 할 동작 지정
3) 요청대상
    - absolute-path[?query] (절대경로[?쿼리])
    - 절대경로 = “/” 로 시작하는 경로
4) HTTP 버전
5) 응답 메시지
    - start-line = request-line / status-line
    - status-line = HTTP-version SP status-code SP reason-phrase CRLF
    - HTTP 상태코드 : 요청 성공/실패를 나타냄

HTTP 헤더

- header-field = field-name ”:” OWS field-value OWS (OWS: 띄어쓰기 가능)
- field-name은 대소문자 구분 X
- 용도
    - HTTP전송에 필요한 모든 부가정보
    - 표준헤더가 너무 많음
    - 필요시 임의의 헤더 추가 기능

HTTP 메시지 바디

- 실제 전송할 데이터
- byte로 표현할 수 있는 모든 데이터 전송 가능
    - ex) 문서 이미지 영상 JSON 등

HTTP 정리

- HTTP 메시지에 모든것을 전송
- HTTP 역사 HTTP /1.1을 기준으로 학습
- 클라이언트 서버 구조
- 무상태 프로토콜(stateless)
- Http 메시지
- 단순함 / 확장가능

[4장 HTTP 메서드]

<HTTP API를 만들어 보자>

APU URI 고민

- 리소스의 의미?
- 리소스를 어떻게 식별할 것인가?
    - 회원 조회/수정/삭제 -> 리소스 : 회원

리소스와 행위를 분리

- 가장 중요한 것은 리소스를 식별하는 것
- URI는 리소스만 식별
- 리소스와 해당 리소스를 대상으로 하는 행위를 분리
    - 리소스 : 회원
    - 행위 : 조회 / 등록 / 삭제 / 변경
- 리소스는 명사 행위는 동사

<HTTP 메서드 - GET / POST>

HTTP 메서드 종류

1) 주요메서드
    - GET : 리소스 조회
    - POST : 요청 데이터 처리 / 주로 등록에 사용
    - PUT : 리소스를 대체, 해당 리소스가 없으면 생성
    - PATCH : 리소스 부분 변경
    - DELETE : 리소스 삭제
        - ** 최근엔 리소스 -> Representation 으로 바뀜
2) 기타 메서드
    - HEAD : GET과 동일하지만 메시지 부분을 제외하고 상태줄과 헤더만 반환
    - OPTIONS : 대상 리소스에 대한 통신 가능 옵션(메서드)을 주로 설명
        - 주로 CORS에서 사용
    - CONNECT : 대상 자원으로 식별되는 서버에 대한 터널을 설정
    - TRACE : 대상 리소스에 대한 경로를 따라 메시지 루프백 테스트를 수행

GET

- 리소스 조회
- 서버에 전달하고 싶은 데이터는 query (쿼리 파라미터, 쿼리 스트링)를 통해서 전달
- 메시지 바디를 사용해서 데이터를 전달할 수 있지만 지원하지 않는 곳이 많아 권장 X

POST

- 요청 데이터 처리
- 메시지 바디를 통해 서버로 요청 데이터 전달
- 서버는 요청 데이터를 처리
- 메시지 바디를 통해 들어온 데이터를 처리하는 모든 기능을 수행한다
- 주로 전달된 데이터로 신규 리소스 등록, 프로세스 처리에 사용

POST - 요청데이터를 어떻게 처리한다는 것일까?

- 스펙 : POST 메서드는 대상 리소스가 리소스의 고유 한 의미 체계에 따라 요청에 포함된 표현을 처리하도록 요청
- ex) 다음과 같은 기능에 사용
    - HTML 양식에 입력된 필드와 같은 데이터 블록을 데이터 처리 프로세스에 제공
        - ex) HTML FORM 에 입력한 정보로 회원가입, 주문 등에서 사용
    - 게시판, 뉴스그룹, 메일링 리스트,. 블로그 또는 유사한 기사 그룹에 메시지 개시
        - ex) 게시판 글쓰기 , 댓글
    - 서버가 아직 식별하지 않은 새 리소스 생성

        - 신규 주문 생성
    - 기존 자원에 데이터 추가
        - 한 문서 끝에 용추가
- 정리
    - 이 리소스 URI에 POST 요청이 오면 요청 데이터를 어떻게 처리할지 리소스 마다 따로 정해야 한다 -> 정해진 것이 없다

POST 정리

1) 새 리소스 생성(등록)
    - 서버가 아직 식별하지 않은 새 리소스 생성
2) 요청 데이터 처리
    - 단순 데이터 생성/변경 을 넘어 프로세스를 처리해야 하는 경우
        - ex) 결제 - 배달 - 배달완료
    - POST의 결과로 새로운 리소스가 생성되지 않을 수도 있음
        - 컨트롤 URI
3) 다른 메서드로 처리하기 애매한 경우
    - JSON으로 조회 데이터를 넘겨야 하는데 GET 메서드를 사용하기 어려운 경우

<HTTP 메서드 - PUT / PATCH / DELETE >

PUT

- 리소스를 대체
    - 리소스가 있으면 대체
        - 리소스를 완전히 대체 (수정 X)
    - 리소스가 없으면 생성
    - 쉽게 이야기해서 덮어버림
- 중요! 클라이언트가 리소스를 식별
    - 클라이언트가 리소스 위치를 알고 URI 지정
    - POST와 차이점

PATCH

- 리소스 변경

DELETE

- 리소스 제거
- PATCH가 안될 경우 POST 사용

<HTTP 메서드 속성>

HTTP 메서드 속성

1) 안전(Safe Methods)
2) 멱등(Idempotent Methods)
3) 캐시가능(Cacheable Methods)

안전 Safe

- 호출해도 리소스를 변경하지 않는다
- 만약 계속 호출해서 로그 같은게 쌓인다면? -> 안전은 해당 리소스만 고려한다

멱등 Idempotent Methods
- f(f(x))=f(x)
- 1번 호출이든 100번 호출이든 결과가 같다
- 멱등 메서드
    - GET : 1번이든 2번이든 같은 결과가 조회
    - PUT : 결과를 대체한다 -> 따라서 같은 요청을 여러번해도 최종결과는 같다
    - DELETE : 결과를 삭제 -> 같은 요청을 여러번 해도 삭제된 결과는 똑같다
    - **POST** : 멱등이 아니다 -> 2번 호출하면 같은 결제가 중복해서 발생할 수 있다
- 활용
    - 자동 복구 메커니즘
- 재요청 중간에 다른 곳에서 리소스를 변경해 버리면?
    - 멱등은 외부 요인 영향까지는 고려하지 않음

캐시가능 Cacheable
- 응답 결과 리소스를 캐시해서 사용해도 되는가?
- GET / HEAD / POST / PATCH 캐시가능
- 실제로는 GET/HEAD 정도만 캐시로 사용
    - POST/PATCH는 본문 내용 까지 캐시 키로 고려해야하는데 구현이 어려움

[5장 HTTP 메서드 활용]

<클라이언트에서 서버로 데이터 전송>

데이터 전달 방식 2가지
1) 쿼리 파라미터를 통한 데이터 전송
    - GET
    - 주로 정렬필터(검색어)
2) 메시지 바디를 통한 데이터 전송
    - POST / PUT PATCH
    - 회원 가입 / 상품 주문 / 리소스 등록 / 리소스 변경

클라이언트에서 서버로 데이터 전송 4가지 상황
1) 정적 데이터 조회
2) 동적 데이터 조회
3) HTML Form 을 통한 데이터 전송
4) HTTP API를 통한 데이터 전송

정적 데이터 조회
- 일반적으로 쿼리 파라미터 없이 리소스 경로로 단순하게 조회가능
- 이미지나 정적 텍스트 문서
- 조회는 GET 사용

동적 데이터 조회
- 주로 검색, 게시판 목록에서 정렬필터
- 조회 조건을 줄여주는 필터, 조회결과를 정렬하는 정렬조건에 주로 사용
- 조회는 GET 사용
- GET은 쿼리 파라미터 사용해서 데이터를 전달

HTML Form 데이터 전송

- HTML Form submit 사용시 POST 전송
- Content-Type: application/x-www-form-urlencoded 사용
    - form의 내용을 메시지 바디를 통해서 전송(key=value, 쿼리 파라미터 형식)
    - 전송 데이터를 url encoding 처리
- HTML From은 GET 전송도 가능
- Content-Type : multipart/form-data
    - 파일 업로드 같은 바이너리 데이터 전송시 사용
    - 다른 종류의 여러 파일과 폼의 내용 함께 전송 가능(그래서 이름이 multipart)
- HTML Form 전송은 GET, POST만 지원

HTTP API 데이터 전송
- 서버 to 서버
    - 백엔드 시스템 통신
- 앱 클라이언트
    - 아이폰/안드로이드
- 웹 클라이언트
    - HTML 에서 Form 전송 대신 자바스크립트를 통한 통신에 사용(AJAX)
    - ex) React, VueJS 같은 웹 클라이언트와 API 통신
- POST / PUT / PATCH : 메시지 바디를 통해 데이터 전송
- GET : 조회, 쿼리 파라미터로 데이터 전당
- Content-Type : application/json을 주로 사용(사실상 표준)

<HTTP API 설계 예시>

HTTP API 설계 예시
1) HTTP API - 컬렉션
    - POST 기반 등록 ex) 회원관리 API 제공
2) HTTP API -스토어
    - PUT 기반 등록 ex) 정적 컨텐츠 관리, 원격 파일 관리
3) HTML FORM 사용
    - 웹 페이지 회원 관리
    - GET / POST 만 지원

POST - 신규 자원 등록 특징
- 클라이언트는 등록될 리소스의 URI를 모른다
- 서버가 새로 등록된 리소스 URI를 생성해준다
- 컬렉션
    - 서버가 관리하는 리소스 디렉토리
    - 서버가 리소스의 URI를 생성하고 관리

API 설계 - PUT 신규 자원 등록 특징
- 클라이언트가 리소스 URI 를 알고 있어야한다
- 클라이언트가 직접 리소스의 URI 지정
- 스토어
    - 클라이언트가 관리하는 리소스 저장소
    - 클라이언트가 리소스의 URI를 알고 관리

HTML FORM 사용

- GET POST 만 지원
- 컨트롤 URI
    - GET POST 만 지원해서 제약이 있음
    - 이런 제약을 해결하기 위해 동사로 된 리소스 경로 사용
    - 컨트롤 URI : POST의 /new /edit /delete
    - HTTP 메서드로 해결하기 애매한 경우 사용 (HTTP API 포함)

정리
- HTTP API - 컬렉션
    - POST 기반 등록
    - 서버가 리소스 URI 결정
- HTTP API - 스토어
    - PUT 기반 등록
    - 클라이언트가 리소스 URI 결정
- HTML FORM 사용
    - 순수 HTML + HTML form 사용
    - GET, POST 만 지원

참고하면 좋은 URI 설계 개념
- 문서(document)
    - 단일개념(파일하나, 객체 인스턴스. 데이터베이스 row)
    - ex) /members/100, /files/star.jpg
- 컬렉션(collection)
    - 서버가 관리하는 리소스 디렉토리
    - 서버가 리소스의 URI를 생성하고 관리
    - ex) /members
- 스토어(store)
    - 클라이언트가 관리하는 자원 저장소
    - 클라이언트가 리소스의 URI를 알고 관리
    - ex) /files
- 컨트롤러(controller), 컨트롤 URI
    - 문서, 컬렉션, 스토어로 해결하기 어려운 추가 프로세스 실행
    - 동사를 직접 사용
    - ex) /members/{id}/delete

[6장 HTTP 상태코드]

<HTTP 상태코드 소개>

상태코드
- 클라이언트가 보낸 요청의 처리상태를 응답에서 알려주는 기능
    - 1xx (Informational) : 요청이 수신되어 처리중
    - 2xx (Successful) : 요청 정상 처리
    - 3xx (Redirection) : 요청을 완료하려면 추가 행동이 필요
    - 4xx (Client Error) : 클라이언트 오류, 잘못된 문법등으로 서버가 요청을 수행할 수 없음
    - 5xx (Server Error) : 서버오류, 서버가 정상 요청을 처리하지 못함

1xx (Informational)

- 요청이 수신되어 처리중
- 거의 사용하지 않음

<2xx - 성공>

2xx Successful
- 클라이언트의 요청을 성공적으로 처리
    - 200 OK
        - 요청 성공
    - 201 Created
        - 요청 성공해서 새로운 리소스가 생성됨
        - 생성된 리소스는 응답의 Location 헤더 필드로 식별
    - 202 Accepted
        - 요청이 접수되었으나 처리가 완료되지 않았음
        - 배치 처리 같은 곳에서 사용
        - ex) 요청 접수 후 1시간 뒤에 배치 프로세스가 요청을 처리

    - 204 No Content
        - 서버가 요청을 성공적으로 수행했지만, 응답 페이로드 본문에 보낼 데이터가 없음
        - ex) 웹 문서 편집기에 save 버튼
        - save 버튼의 결과로 아무 내용이 없다
        - save 버튼 눌러도 같은 화면 유지
        - 결과 내용이 없어도 204 메시지(2xx)만으로 성공을 인식 할 수 있다

<3xx - 리다이렉션 >

3xx Redirection
- 요청을 완료하기 위해 유저 에이전트의 추가 조치 필요
    - 유저 에이전트 : 클라이언트 프로그램 / 주로 웹브라우저
    - 300 Multiple Choices
    - 301 Moved permanently
    - 302 Found
    - 303 See Other
    - 304 Not Modified
    - 307 Temporary Redirect
    - 308 Permanent Redirect

리다이렉션 이해
- 웹 브라우저는 3xx응답의 결과에 Location 헤더가 있으면, Location 위치로 자동 이동(리다이렉트)

리다이렉션 종류

1) 영구 리다이렉션
    - 특정 리소스의 URI 가 영구적으로 이동
        - ex) /members -> /users
        - ex) /event -> /new event
2) 일시 리다이렉션
    - 일시적인 변경
        - 주문 완료 후 주문 내역 화면으로 이동
        - PRG : Post/Redirect/Get
3) 특수 리다이렉션
    - 결과 대신 캐시를 사용

영구 리다이렉션
- 301, 308
    - 리소스의 URI 가 영구적으로 이동
    - 원래의 URL을 사용 x , 검색 엔진 등에서도 변경 인지
    - 301 Moved permanently
        - 리다이렉트 요청 메서드가 GET으로 변하고, 본문이 제거될 수 있음(MAY)
    - 308 Permanent Redirect
        - 301과 기능 같음
        - 리다이렉트시 요청 메서드와 본문 유지
        - (처음 POST를 보내면 리다이렉트도 POST 유지)

일시적인 리다이렉션
- 302, 307, 303
    - 리소스의 URI가 일시적으로 변경
    - 따라서 검색 엔진 등에서 URL을 변경하면 안됨
    - 302 Found
        - 리다이렉트시 요청 메서드가 GET으로 변하고 본문이 제거될 수 있음(MAY)
    - 307 Temporary Redirect
        - 302와 기능은 같음
        - 리다이렉트시 요청 메서드와 본문 유지
        - (요청 메서드를 변경하면 안됨.MUST NOT)
    - 303 See Other
        - 302와 기능은 같음
        - 리다이렉트 요청 메서드가 GET으로 변경

PRG : Post / Redirect / Get
일시적인 리다이렉션 - 예시
- PRG 이후 리다이렉트
    - URL이 이미 POST -> GET 으로 리다이렉트 됨
    - 새로고침 해도 GET으로 결과 화면만 조회

302/307/303 정리

- 정리
    - 302 Found -> GET으로 변할 수 있음
    - 307 Temporary Redirect -> 메서드가 변하면 안됨
    - 303 See Other -> 메서드가 GET으로 변경
- 역사
    - 처음 302 스펙의 의도는 HTTP 메서드를 유지하는 것
    - 그런데 웹 브라우저들이 대부분 GET으로 바꾸어버림(일부는 다르게 동작)
    - 그래서 모호한 302 대신하는 명확한 307/303이 등장 (301 대응 308도 등장)
- 현실
    - 307,303을 권장하지만 현실적으로 이미 많은 애플리케이션 라이브러리들이 302를 기본값으로 사용
    - 자동 리다이렉션시에 GET으로 변해도되면 그냥 302를 사용해도 큰 문제 X

기타 리다이렉션
- 300 , 304
    - 300 Multiple Choices : 안씀
    - 304 Not Modified : 많이씀
        - 캐시를 목적으로 사용
        - 클라이언트에게 리소스가 수정되지 않았음을 알려준다 따라서 클라이언트는 로컬PC에 저장된 캐시를 재사용한다 (캐시로 리다이렉트 한다)
        - 304 응답은 응답에 메시지 바디를 포함하면 안된다 (로컬 캐시를 사용해야 하므로)
        - 조건부 GET , HEAD 요청시 사용

<4xx - 클라이언트 오류 / 5xx - 서버 오

4xx Client Error
- 클라이언트 오류
    - 클라이언트의 요청에 잘못된 문법등으로 서버가 요청을 수행할 수 없음
    - 오류의 원인이 클라이언트에 있음
    - 중요! 클라이언트가 이미 잘못된 요청/데이터를 보내고 있기때문에 똑같은 재시도가 실패함

400 Bad Request
클라이언트가 잘못도니 요청을 해서 서버가 요청을 처리할 수 없음
- 요청 구문, 메시지 등등 오류
- 클라이언트는 요청 내용을 다시 검토하고., 보내야함
- 예) 요청 파라미터가 잘못되거나, API 스펙이 맞지 않을 때

401 Unauthorized
클라이언트가 해당 리소스에 대한 인증이 필요함
- 인증(Authentication) 되지 않음
- 401 오류 발생시 응답에 WWW-Authenticate 헤더와 함께 인증 방법을 설명
- 참고
    - 인증(Authentication) : 본인이 누구인지 확인 (로그인)
    - 인가 (Authorization) : 권한부여 (ADMIN 권한처럼 특정 리소스에 접근 할 수 있는 권한/인증이 있어야 인가가 있음)

    - 오류 메시지가 Unauthorized 이지만 인증되지 않음(이름이 아쉬움)

403 Forbidden
서버가 요청을 이해했지만 승인을 거부
- 주로 인증 자격 증명은 있지만 접근권한이 불충분한 경우
- ex) 관리자 등급 아닌 사용자가 로그인은 했지만 관리자 등급 리소스에 접근

404 Not Found
요청 리소스를 찾을 수 없음
- 요청 리소스가 서버에 없음
- 또는 클라이언트가 권한이 부족한 리소스에 접근 할 때 해당 리소스를 숨기고 싶을 때

5xx SErver Error
서버오류
- 서버 문제로 오류 발생
- 서버에 문제가 있기 때문에 재시도 하면 성공할 수도 있음(복구가 되거나 등등)

500 Internal Server Error
서버 문제로 오류발생, 애매하면 500 오류
- 서버 내부 문제로 오류 발생
- 애매하면 500 오류

503 Service Unavailable
서비스 이용 불가
- 서버가 일시적인 과부하 또는 예정된 작업으로 잠시 요청을 처리할 수 없음
- Retry-After 헤더 필드로 얼마뒤에 복구되는지 보낼 수도 있음
-

[7장 HTTP 헤더1 - 일반 헤더]

<HTTP 헤더 개요>

HTTP 헤더
- header-field = field-name ":" OWS field-value OWS (OWS: 띄어쓰기 가능)
- field-name은 대소문자 구분 X

HTTP 헤더 용도
- HTTP 전송에 필요한 모든 부가정보
    - ex) 메시지 바디의 내용, 메시잡디 크기, 압축, 인증, 서버정보 등등
- 표준 헤더가 너무 많음
- 필요시 임의의 헤더 추가 가능

HTTP 헤더 분류
- RFC 2616(과거)

    - General 헤더 : 메시지 전체에 적용되는 정보
    - Request 헤더 : 요청 정보
    - Response 헤더 : 응답 정보
    - Entity 헤더 : 엔티티 바디 정보

HTTP BODY

- message body -RFC 2616(과거)
    - 메시지 본문(message body)은 엔티티 본문을 전달하는데 사용
    - 엔티티 본문은 요청이나 응답에서 전달할 실제 데이터
    - 엔티티 헤더는 엔티티 본문의 데이터를 해석할 수 있는 정보 제공
        - 데이터유형, 데이터길이, 압축 정보

HTTP 표준

- RFC 2616(과거 / 폐기됨)
- 2014년 RFC 7230 ~ 7235 등장

RFC723x 변화

- 엔티티(Entity) -> 표현(Representation)
- Representation = representation Metadata + Representation Data
- 표현 = 표현 메타데이터 + 표현 데이터

HTTP BODY

- message body -RFC7230(최신)
    - message body를 통해 표현 데이터 전달
    - message body = 페이로드 payload
    - 표현은 요청이나 응답에서 전달할 실제 데이터
    - 표현헤더는 표현 데이터를 해석할 수 있는 정보 제공
        - 데이터 유형/데이터길이/압축정보 등등

<표현>

표현

- Content-Type : 표현 데이터의 형식
- Content-Encoding : 표현 데이터의 압축 방식
- Content-Language : 표현 데이터의 자연 언어
- Content-Length: 표현 데이터의 길이
- 표현 헤더는 전송/응답 둘다 사용

Content-Type

표현 데이터의 형식 설명

- 미디어 타입 / 문자 인코딩
    - ex) text/html; charset=utf-8
    - application/json
    - image/png

Content-Encoding

표현 데이터 인코딩

- 표현 데이터를 압축하기 위해 사용
- 데이터를 전달하는 곳에서 압축 후 인코딩 헤더 추가
- 데이터를 읽는 쪽에서 인코딩 헤더의 정보로 압축해제

Content-Language
표현데이터의 자연언어
- 표현데이터의 자연언어를 표현

Content-Length
표현 데이터의 길이
- 바이트 단위
- Transfer-Encoding(전송코딩)을 사용하면 Content-Length를 사용하면 안됨

<콘텐츠 협상>

협상(콘텐츠 네고시에이션)
클라이언트가 선호하는 표현 요청
- Accept : 클라이언트가 선호하는 미디어 타입 전달
- Accept-Charset : 클라이언트가 선호하는 문자 인코딩
- Accept-Encoding: 클라이언트가 선호하는 압축 인코딩
- Accept-Language: 클라이언트가 선호하는 자연언어
- 협상 헤더는 요청시에만 사용

협상과 우선순위1
Quality Values(q)
- Quality Values(q) 값 사용
- 0 ~ 1, 클수록 높은 우선순위
- 생략 시 1
- Accept-Language : ko-KR,ko:q=0.9,en-US;q=0.8,en;q=0.7
    - 1. ko-KR;q=1;(q생략)
    - 2. ko;q=0.9
    - 3. en-US;q=0.8
    - 4. en;q=0.7

협상과 우선순위2
Quality Values(q)
- 구체적인 것이 우선한다.
- Accept: text/*, text/plain, text/plain;format=flowed, */*
    1) text/plain;format=flowed
    2) text/plain
    3) text/*
    4) */*

협상과 우선순위3
Quality Values(q)

- 구체적인 것을 기준으로 미디어 타입을 맞춘다.
- text/*;q=0.3, text/html;q=0.7, text/html;level=1, text/html;level=2;q=0.4, */*;q=0.5

Media Type	Quality
text/html;level=1	1
text/html	0.7
text/plain	0.3
image/jpeg	0.5
text/html;level=2	0.4
text/html;level=3	0.7

<전송 방식>

전송 방식 설명
- 단순 전송 Content-Length
- 압축 전송 Content-Encoding
- 분할 전송 Transfer-Encoding
- 범위 전송 Range, Content-Range

<일반 정보>

일반 정보
- From : 유저 에이전트의 이메일 정보
- Referer : 이전 웹 페이지 주소
- User-Agent : 유저 에이전트 애플리케이션 정보
- Server : 요청을 처리하는 오리진 서버의 소프트웨어 정보
- Date : 메시지가 생성된 날짜

From
유저 에이전트의 이메일 정보
- 일반적으로 잘 사용되지 않음
- 검색 엔진 같은 곳에서, 주로 사용
- 요청에서 사용

Referer
이전 웹 페이지 주소
- 현재 요청된 페이지의 이전 웹 페이지 주소
- A -> B 로 이동하는 경우 B 요청시 Referer: A 를 포함해서 요청
- Referer를 사용해서 유입 경로 분석 가능
- 요청에서 사용

User-Agent :
유저 에이전트 애플리케이션 정보

- user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/ 537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36
- 클라이언트의 애플리케이션 정보(웹 브라우저 정보, 등등)
- 통계 정보
- 어떤 종류의 브라우저에서 장애가 발생하는지 파악 가능
- 요청에서 사용

#Server
요청을 처리하는 ORIGIN 서버의 소프트웨어 정보
- Server: Apache/2.2.22 (Debian)
- server: nginx
- 응답에서 사용

#Date
메시지가 발생한 날짜와 시간
- Date: Tue, 15 Nov 1994 08:12:31 GMT
- 응답에서 사용

<특별한 정보>

특별한 정보
- Host : 요청한 호스트 정보(도메인)
- Location : 페이지 리다이렉션
- Allow : 허용 가능한 HTTP 메서드
- Retry-After : 유저 에이전트가 다음 요청을 하기까지 기다려야 하는 시간

Host
요청한 호스트 정보(도메인)
- 요청에서 사용
- 필수
- 하나의 서버가 여러 도메인을 처리해야 할 때
- 하나의 IP 주소에 여러 도메인이 적용되어 있을 때

Location
페이지 리다이렉션
- 웹 브라우저는 3xx 응답의 결과에 Location 헤더가 있으면, Location 위치로 자동 이동 (리다이렉트)
- 응답코드 3xx에서 설명
- 201 (Created) : Location 값은 요청에 의해 생성된 리소스 URI
- 3xx (Redirection): Location 값은 요청을 자동으로 리디렉션하기 위한 대상 리소스를 가리킴

Allow
허용 가능한 HTTP 메서드
- 405(Method Not Allowed)에서 응답에 포함해야함
- Allow: GET, HEAD, PUT

Retry-After
유저 에이전트가 다음 요청을 하기까지 기다려야 하는 시간

- 503(Service Unavailable): 서비스가 언제까지 불능인지 알려줄 수 있음
- Retry-After : Fri, 31 Dec 1999 23:59:59 GMT(날짜 표기)
- Retry-After : 120(초단위 표기)

<인증>

인증
- Authorization : 클라이언트 인증 정보를 서버에 전달
- WWW-Authenticate : 리소스 접근시 필요한 인증 방법 정의

Authorization
클라이언트 인증 정보를 서버에 전달
- Authorization : Baisc xxxxxxxxxxxxxxxxx

WWW-Authenticate
리소스 접근 시 필요한인증 방법 정의
- 리소스 접근 시 필요한 인증 방법 정의
- 401 Unauthorized 응답과 함께 사용

<쿠키>

쿠키
- Set-Cookie : 서버에서 클라이언트로 쿠키 전달(응답)
- Cookie : 클라이언트가 서버에서 받은 쿠키를 저장하고, HTTP 요청 시 서버로 전달
    - ex) set-cookie: sessionId=abcde1234; expires=Sat, 26-Dec-2020 00:00:00 GMT; path=/; domain=.google.com; Secure
- 사용처
    - 사용자 로그인 세션 관리
    - 광고 정보 트래킹
- 쿠키 정보는 항상 서버에 전송됨
    - 네트워크 트래픽 추가 유발
    - 최소한의 정보만 사용 (세션id, 인증토큰)
    - 서버에 전송하지 않고 웹 브라우저 내부에 데이터를 저장하고 싶으면 웹 스토리지 참고
- 주의!
    - 보안에 민감한 데이터는 저장하면 안됨

쿠키 - 생명주기
- Expires, max-age
    - Set-Cookie: expires= Sat, 26-Dec-2020 00:00:00 GMT;
        - 만료일이 되면 쿠키 삭제
    - Set-Cookie: max-age=3600 (3600초)
        - 0 이나 음수 지정하면 쿠키 삭제
    - 세션 쿠키 : 만료 날짜를 생략하면 브라우저 종료 시 까지만 유지
    - 영속 쿠키 : 만료 날짜를 입력하면 해당 날짜까지 유지

쿠키 - 도메인
Domain

- domain=example.org
- 명시 : 명시한 문서 기준 도메인 + 서브 도메인 포함
    - domain=example.org를 지정해서 쿠키생성
        - example.org는 물론이고
        - dev.example.org도 쿠키접근
- 생략 : 현재 문서 기준 도메인만 적용
    - example.org에서 쿠키생성하고 domain 지정 생략
        - example.org 에서만 쿠키 접근
        - dev.example.org는 쿠키 미접근

쿠키- 경로
Path
- ex) path=/home
- 이 경로를 포함한 하위 경로 페이지만 쿠키 접근
- 일반적으로 path=/ 루트로 지정

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

[8장 HTTP 헤더 2 -캐시와 조건부 요청]

<캐시 기본 동작>

캐시가 없을 때
- 데이터가 변경되지 않아도 계속 네트워크를 통해서 데이터를 다운로드 받아야 한다
- 인터넷 네트워크는 매우 느리고 비싸다
- 브라우저 로딩 속도가 느리다
- 느린 사용자 경험

캐시 적용
- 캐시 덕분에 캐시 가능 시간동안 네트워크를 사용하지 않아도 된다
- 비싼 네트워크 사용량을 줄일 수 있다
- 브라우저 로딩 속도가 매우 빠르다
- 빠른 사용자 경험

캐시 시간 초과
- 캐시 유효 시간이 초과하면, 서버를 통해 데이터를 다시 조회하고 캐시를 갱신한다

- 이때 다시 네트워크 다운로드가 발생한다

<검증 헤더와 조건부 요청1>

캐시 시간 초과
- 캐시 유효 시간이 초과해서 서버에 다시 요청하면 다음 두 가지 상황이 나타남
    1) 서버에서 기존 데이터를 변경함
    2) 서버에서 기존 데이터를 변경하지 않음
- 캐시 만료후에도 서버에서 데이터를 변경하지 않음
- 생각해보면 데이터를 전송하는 대신에 저장해 두었던 캐시를 재사용 할 수 있다
- 단 클라이언트의 데이터와 서버의 데이터가 같다는 사실을 확인할 수 있는 방법 필요

정리
- 캐시 유효 시간이 초과해도 서버의 데이터가 갱신되지 않으면
- 304 Not Modified + 헤더 메타 정보만 응답(바디x)
- 클라이언트는 서버가 보낸 응답 헤더 정보로 캐시의 메타 정보를 갱신
- 클라이언트는 캐시에 저장되어 있는 데이터 재활용
- 결과적으로 네트워크 다운로드가 발생하지만 용량이 적은 헤더 정보만 다운로드
- 매우 실용적인 해결책

<검증 헤더와 조건부 요청2>
- 검증헤더
    - 캐시 데이터와 서버 데이터가 같은지 검증하는 데이터
    - Last-Modified, Etag
- 조건부 요청 헤더
    - 검증 헤더로 조건에 따른 분기
    - If-Modified-Since : Last-Modified 사용
    - If-None-match : Etag 사용
    - 조건이 만족하면 200 OK
    - 조건이 만족하지 않으면 304 Not Modified

검증 헤더와 조건부 요청 예시
- If-Modified-Since : 이후에 데이터가 수정되었으면 ?
- 데이터 미변경 예시
    - 캐시 : 2020년 11월 10일 10:00:00 vs 서버 : 2020 년 11월 10일 10:00:00
    - 304 Not Modified, 헤더 데이터만 전송
    - 전송 용량 0.1M(헤더 0.1M 바디 1.0M)
- 데이터 변경 예시
    - 캐시 : 2020년 11월 10일 10:00:00 vs 서버 : 2020 년 11월 10일 11:00:00
    - 200 OK, 모든 데이터 전송(Body 포함)
    - 전송 용량 1.1M(헤더 0.1M 바디 1.0M)

검증헤더와 조건부 요청
Last-Modified, If-Modified-Since 단점

- 1초 미만(0.x초) 단위로 캐시 조정이 불가능
- 날짜 기반의 로직 사용
- 데이터를 수정해서 날짜가 다르지만 같은 데이터를 수정해서 데이터 결과가 똑같은 경우
- 서버에서 별도의 캐시 로직을 관리하고 싶은 경우

ETag(Entity Tag)
- 캐시용 데이터에 임의의 고유한 버전 이름을 달아둠
    - ex) ETag: "v1.0", ETag: "a2jiodwjekjl3"
- 데이터가 변경되면 이 이름을 바꾸어서 변경함(Hash를 다시 생성)
    - ETag: "aaaaa" -> ETag: "bbbbb
- 진짜 단순하게 ETag만 보내서 같으면 유지, 다르면 다시 받기
- 캐시 제어 로직을 서버에서 완전히 관리
- 클라이언트는 단순히 이 값을 서버에 제공(클라이언트는 캐시 메커니즘을 모름)
- ex)
    - 서버는 배타 오픈 기간인 3일동안 파일이 변경되어도 ETag 동일하게 유지
    - 애플리케이션 배포 주기에 맞추어 ETag 모두 갱신

<캐시와 조건부 요청 헤더>

캐시 제어 헤더
- Cache-Control : 캐시 제어
- Pragma : 캐시 헤어(하위 호환)
- Expires : 캐시 유효 기간(하위 호환)

Cache-Control
캐시 지시어(directives)
- Cache-Control : max-age
    - 캐시 유효 기간, 초단위
- Cache-Control: no-cache
    - 데이터는 캐시해도 되지만, 항상 원(origin) 서버에 검증하고 사용
- Cache-Control : no-store
    - 데이터에 민감한 정보가 있으므로 저장하면 안됨
    - (메모리에서 사용하고 최대한 빨리 삭제)

Pragma
캐시제어(하위 호환)
- Pragma : no-cache
- HTTP1.0 하위호환

Expires
캐시 만료일 지정(하휘 호환)
- expires: Mon, 01 Jan 1990 00:00:00 GMT
- 캐시만료일을 정확한 날짜로 지정
- HTTP 1.0 부터 사용
- 지금은 더 유연한 Cache-Control : max-age 권장
- Cache-Control : max-age와 함께 사용하면 Expires 는 무시

검증헤더와 조건부 요청 헤더
1) 검증 헤더

- ETag: "v1.0", ETag: "asid93jkrh2l"
- Last-Modified: Thu, 04 Jun 2020 07:19:24 GMT

2) 조건부 요청 헤더
   - If-Match, If-None-Match: ETag 값 사용
   - If-Modified-Since, If-Unmodified-Since: Last-Modified 값 사용

<프록시 캐시>

Cache-Control
- Cache-Control : public
  - 응답이 public 캐시에 저장되어도 됨
- Cache-Control : private
  - 응답이 해당 사용자만을 위한 것임, private 캐시에 저장해야 함(기본값)
- cache-Control : s-maxage
  - 프록시 캐시에만 적용되는 max-age
- Age:60(HTTP 헤더)
  - 오리진 서버에서 응답 후 프록시 캐시 내에 머문 시간(초)

<캐시 무효화>

Cache-Control
확실한 캐시 무효화 응답
- Cache-Control : no-cache, no-store, must-revalidate
- Pragma : no-cache

Cache-Control
캐시 지시어(directives) -확실한 캐시 무효화
- Cache-Control : no-cache
  - 데이터는 캐시해도 되지만, 항상 원 서버에 검증하고 사용(이름에 주의)
- Cache-Control : no-store
  - 데이터에 민감한 정보가 있으므로 저장하면 안됨
  - (메모리에서 사용하고 최대한 빨리 삭제)
- Cache-Control : must-revalidate
  - 캐시 만료 후 최초 조회시 원 서버에 검증해야함
  - 원 서버 접근 실패시 반드시 오류가 발생해야함 -504(Gateway Timeout)
  - must-revalidate는 캐시 유효 시간이라면 캐시를 사용함
- Pragma : no-cache
  - HTTP 1.0 하위 호환