HTTP 웹 기본 지식

1. 인터넷 네트워크

인터넷 통신

IP(인터넷 프로토콜)

- IP 인터넷 프로토콜 역할
  - 지정한 IP 주소에 데이터 전달
  - 패킷(Packet)이라는 통신 단위로 데이터 전달
  -
- IP 프로토콜의 한계
  - 비연결성 : 패킬을 받을 대상이 없거나 서비스 불능 상태여도 패킷 전송
  - 비신뢰성 : 중간에 패킷이 사라지거나 순서대로 오지 않을 경우
  - 프로그램 구분 : 같은 IP를 사용하는 서버에서 통신하는 애플리케이션이 둘 이상이면?

TCP/UDP

> IP 프로토콜의 한계를 넘어서기 위해 만들어진 기술

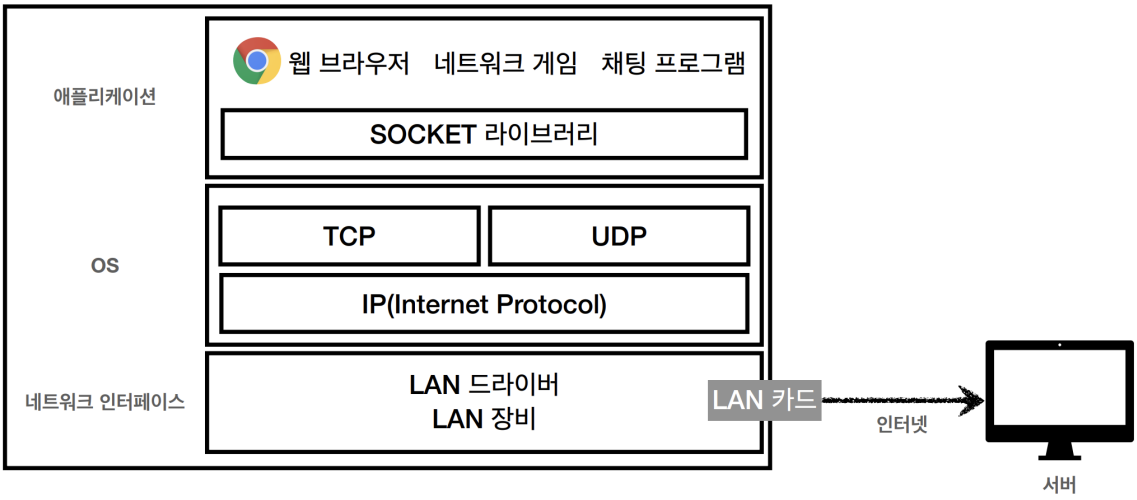
인터넷 프로토콜 스택의 4계층

애플리케이션 계층 - HTTP, FTP

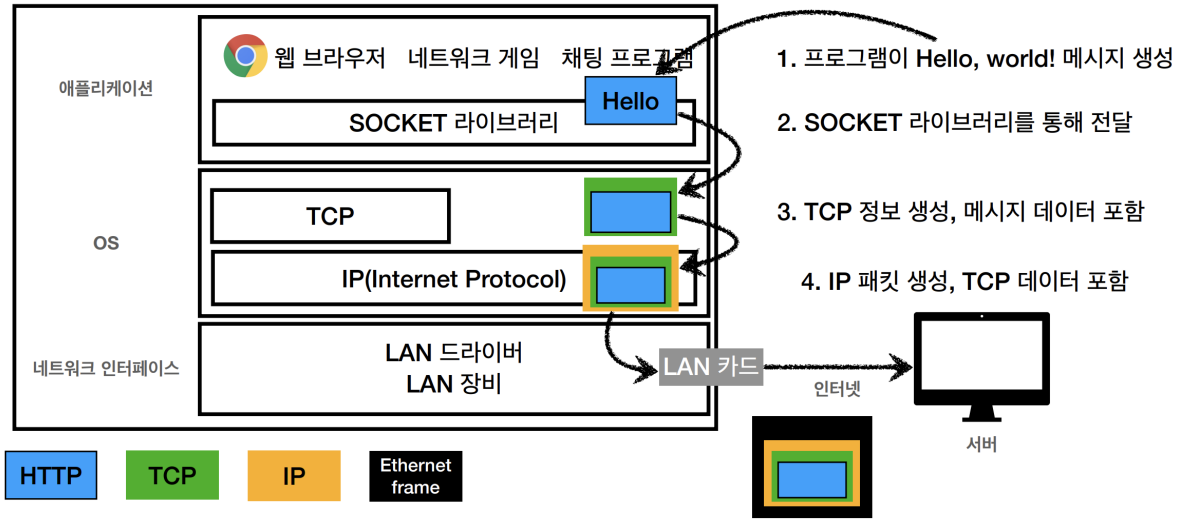
전송 계층 - TCP, UDP

인터넷 계층 - IP

네트워크 인터페이스 계층



출처 : 김영한님 HTTP 웹 기본지식 강의



출처 : 김영한님 HTTP 웹 기본지식 강의

- TCP(Transmission Control Protocol, 접속 제어 프로토콜) 특징
  - 연결 지향 -TCP 3 way handshake

    - 데이터 전달 보증 (못받은걸 알려줌)
    - 순서 보장
    - 신뢰할 수 있는 프로토콜
    - 현재는 대부분 TCP 사용
    - 3 way handshake
        - SYN(클라이언트→서버) → SYN/ASK(서버→클라이언트) → ACK(+데이터) (클라이언트→서버)
- UDP(User Datagram Protocol, 사용자 데이터그램 프로토콜) 특징
    - 하얀 도화지에 비유(기능이 거의 없음)
    - 연결지향 - TCP 3 way handshake X
    - 데이터 전달 보증 X
    - 순서 보장 X
    - 데이터 전달 및 순서가 보장되지 않지만, 단순하고 빠름
    - 정리
        - IP와 거의 같다. + PORT +체크섬 정도만 추가
        - 애플리케이션에서 추가 작업 필요

PORT

> 같은 IP 내에서 프로세스 구분

- 0 ~ 65535 할당 가능
- 0 ~ 1023 : 잘 알려진 포트, 사용하지 않는 것이 좋음
    - FTP - 20, 21
    - TELNET - 23
    - HTTP - 80
    - HTTPS - 443

DNS(Domain Name System)

---

2. URI와 웹 브라우저 요청 흐름

URI(Uniform Resource Indentifier)

URI는 로케이터(**L**OCATOR), 이름(**N**AME) 또는 둘다 추가로 분류될 수 있다.

URL (Reource **Locator**)

URN (Resource **Name**)

URI (Resource **Identifier**)

- URI 단어 뜻
  - Uniform : 리소스 식별하는 통일된 방식
  - Resource : 자원, URI로 식별할 수 있는 모든 것(제한 없음)
  - 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:isbn:8960777331
  - URN 이름만으로 실제 리소스를 찾을 수 있는 방법이 보편화 되지 않음
- URL 전체 문법
  - scheme://[userinfo@]host[:port][/path][?query][#fragment]
  - 예) https://www.google.com:443/search?q=hello&hl=ko
    - 프로토콜(https)
      - 어떤 방식으로 자원에 접근할 것인가 하는 약속 규칙
      - https는 http에 보안 추가(HTTP Secure)

- userinfo
  - URL에 사용자 정보를 포함하여 인증할때 사
- 호스트명(www.google.com)
- 포트번호(443)
  - 일반적으로 생략, 생략시 http는 80, https는 443
- 패스(/search)
  - 리소스 경로(path), 계층적 구조
- 쿼리 파라미터(q=hello&hl=ko)
  - key=value 형태
  - ?로 시작, &로 추가 가능
  - query parameter, query string 등으로 불림
- fragment
  - html 내부 북마크 등에 사용
  - 서버에 전송하는 정보 아님

웹 브라우저 요청 흐름

DNS 조회 → hppts port 조회 → http 요청 메시지 생성 → SOCKET 라이브러리를 통해 전달(A: TCP/IP 연결(IP, PORT), B:데이터 전달) → TCP/IP패킷 생성, HTTP 메시지 포함 → 인터넷을 통한 패킷 전달 → 요청 패킷을 확인 → HTTP 응답 메시지 생성 → 응답 메시지 전달 → 응답 패킷 도착 → 응답 메시지 확인 후 웹 브라우저 HTML 렌더링

---

3. HTTP 기본

모든 것이 HTTP

- **HTTP**(**H**yper**T**ext **T**ransfer **P**rotocol)
- http 메시지에 모든 것을 전송
- HTTP/1.1 1997년

- RFC2068(1997) → RFC2616(1999) → RFC7230~7235 (2014)

기반 프로토콜

- TCP : HTTP/1.1, HTTP/2
- UDP : HTTP/3
- 현재 HTTP/1.1 주로 사용
- HTTP 2,3 도 점점 사용 중

클라이언트 서버 구조

- Request Response 구조
- 클라이언트는 서버에 요청을 보내고, 응답을 대기
- 서버가 요청에 대한 결과를 만들어서 응답

이전에는 클라이언트와 서버가 한 곳에 존재했지만, 현재는 클라이언트와 서버가 따로 구성되어 있다. 비즈니스 로직에 관련된 것은 클라이언트에서 처리하며 데이터 통신에 관련된 것은 서버에서 처리한다.

진행하는 서비스의 요청이 급격하게 늘어날 경우 클라이언트는 수정할 필요가 없으며, 서버만 수정하면 된다.

Stateful, Stateless

무상태 프로토콜(Stateless)

- 서버가 클라이언트의 상태를 보존하지 않는다.
- 장점 : 서버 확장성이 높음(스케일 아웃)
- 단점 : 클라이언트가 추가 데이터 전송

상태 유지(Stateful)

- 서버가 클라이언트의 상태를 유지

상태 유지와 무상태의 차이

- 상태 유지 : 중간에 다른 컨텍스트로 변경되면 안된다.

- 무상태 : 중간에 다른 컨텍스트로 바뀌어도 된다? / 무상태는 응답 서버를 쉽게 바꿀 수 있다.

실무 한계

모든 것을 무상태 상태로 설계 할 수 있는 경우도 있고 없는 경우도 있다.

로그인 같은 경우, 로그인 했다는 상태를 서버에 유지해야 한다. 일반적으로 브라우저 쿠키와 서버 세션등을 사용해서 상태 유지를 해야한다. 상태 유지는 최소한만 사용하는 것이 좋다.

(나의 주관) 여기서 상태라는건 클라이언트와 서버가 통신시 서로 통신할 때 가지고 있던 정보에 대한 보관을 이야기한다. 상태?와 보관?은 좀 다르긴 하지만, 클라이언트가 서버로 요청했을때 전달해준 정보들을 서버가 보관하고 있다면 상태 유지이다.

하지만, 클라이언트가 서버에 보낸 정보에 대해서 서버가 해당 정보를 보관하지 않고 단순히 비즈니스 로직만 처리하며 응답만 주는 형태는 무상태 이다.

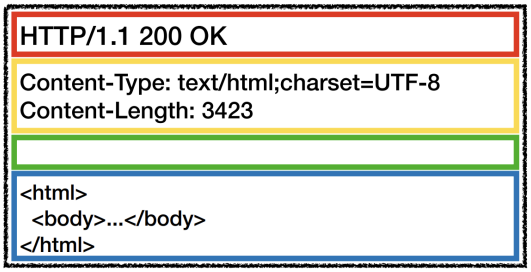
비 연결성(connectionless)

- 장점
  - 서버 입장에서는 최소한의 자원을 사용할 수 있다.
  - HTTP는 기본 연결을 유지하지 않는 모델
  - 일반적으로 초 단위 이하의 빠른 속도 응답
- 단점
  - TCP/IP 연결을 새로 맺어야 함 - 3 way handshake 시간 추가
  - 웹 브라우저로 사이트를 요청하면 html 뿐만 아니라 기타 자원이 함께 다운로드
  - 지금은 HTTP 지속 연결(Persistent Connections) 로 문제 해결
    (Kepp-Alive라고 이전에 많이 쓰지만 현재는 이걸 더 많이 씀)
  - HTTP/2, HTTP/3에서 더 많은 최적화
  -

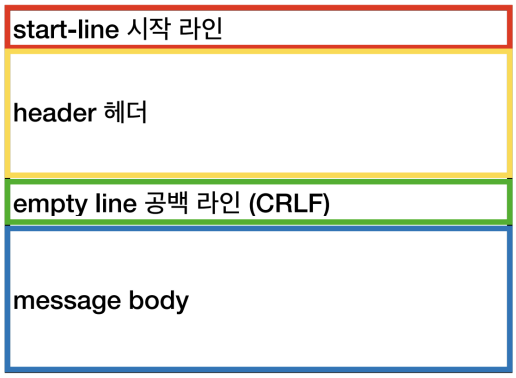
HTTP 메시지

예) HTTP 요청 메시지

요청 메시지도 body 본문을 가질 수 있음



예) HTTP 응답 메시지



HTTP 메시지 구조

START-LINE

- 전송

  HTTP.METHOD(띄어쓰기)절대경로(띄어쓰기)HTTP.VERSION

- 응답

  HTTP.VERSION(띄어쓰기)HTTP.STATUS(띄어쓰기)이유문구

  - HTTP.STATUS
    - 200 : 성공
    - 400 : 클라이언트 요청 오류
    - 500 : 서버 내부 오류
  - 이유문구
    - 사람이 이해할 수 있는 짧은 상태 코드 설명

HTTP 헤더의 용도

- HTTP 전송에 필요한 모든 부가정보가 포함되어 있음
- 표준 헤더는…많다.

- 필요시 임의의 헤더 추가 가능

HTTP 메시지 body 용도

- 실제 전송할 데이터
- BYTE로 표현할 수 있는 모든 데이터가 들어간다.

---

4. HTTP 메서드

API URI 고민

URI(Uniform Resource Identifier)

- 리소스의 의미
  - domain 개념 자체가 리소스이다.
- 리소스는 어떻게 식별하는 것이 좋을 까?
  - domain 이라는 리소스만 식별하면 된다 → domian 리소를 URI에 매칭

리소스와 행위를 분리

가장 중요한 것은 리소스를 식별하는것

- URI는 리소스만 식별
- 리소스와 해당 리소스를 대상으로 하는 행위를 분리

HTTP 메서드 종류(주요 메서드)

- **GET** : 리소스 조회(요새는 리소스대신 Representation라고 불린다.)
- **POST** : 요청 데이터 처리 , 주로 등록에 사용
- **PUT** : 리소스를 대체, 해당 리소스가 없으면 생성
- **PATCH** : 리소스 부분 변경
- **DELETE** : 리소스 삭제

HTTP 메서드 종류(기타 메서드)

- **HEAD** : GET과 동일하지만 메시지 부분을 제외하고, 상태 줄과 헤더만 반환
- **OPTIONS** : 대상 리소스에 대한 통신 가능 옵션(메서드)을 설명
  (주로 **CORS(corss-origin resource sharing)**에서 사용)
- **CONNECT** : 대상 자원으로 식별되는 서버에 대한 터널을 설정
- **TRACE** : 대상 리소스에 대한 경로를 따라 시지 루프백 테스트를 수행

GET

- 리소스 조회
- 서버에 전달하고 싶은 데이터는 query를 통해서 전달
- 메시지 바디를 사용해서 데이터를 전달할 수 있지만, 지우너하지 않는 곳이 많아서 장하지 않음

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

(생성시 코드는 201 응답문구는 Created로 보낸다.)

- 스펠 : POST 메서드는 대살 리소스가 리소스의 고유 한 의미 체계에 따라 요청에 포함된 표현을 처리하도록 요청합니다.
- 컨트롤 URI : 동사형 URI
  - EX) POST /oders/{orderId}/start-delivery

PUT

- 리소스를 전부 대체!
- **클라이언트가 리소스를 식별**

PATCH

- 리소스를 부분 변경

- 지원하지 않는 서버도 있으므로 그땐 POST를 쓴다.

DELETE

- 리소스를 제거

HTTP 속성

요약표

HTTP 메소드	RFC	요청에 Body가 있음	응답에 Body가 있음	안전	멱등(Idempotent)	캐시 가능
GET	RFC 7231	아니요	예	예	예	예
HEAD	RFC 7231	아니요	아니요	예	예	예
POST	RFC 7231	예	예	아니요	아니요	예
PUT	RFC 7231	예	예	아니요	예	아니요
DELETE	RFC 7231	아니요	예	아니요	예	아니요
CONNECT	RFC 7231	예	예	아니요	아니요	아니요
OPTIONS	RFC 7231	선택 사항	예	예	예	아니요
TRACE	RFC 7231	아니요	예	예	예	아니요
PATCH	RFC 5789	예	예	아니요	아니요	예

출처: https://ko.wikipedia.org/wiki/HTTP

- 안전(Safe Methods)
  - 호출해도 리소스를 변경하지 않는다.
- 멱등(Idempotent Methods)
  - $f(f(x)) = f(x)$
    한 번 호출하든 두 번 호출하든 100번 호출하든 결과가 똑같다.
  - 활용
    - 자동 복구 메커니즘
    - 서버가 timeout 등으로 정상 응답을 못주었을 때, 클라이언트가 같은 요청을 다시 해도 되는가? 판단 근거
  - 멱등은 외부 요인으로 중간에 리소스가 변경되는 것 까지는 고려하지 않는다.
- 캐시가능(Cacheable Methods)
  - 응답 결과 리소스를 캐시해서 사용해도 되는가?
  - GET, HEAD, POST, PATCH 캐시는 가능하지만 실제로는 GET, HEAD 정도만 캐시로 사

5. HTTP 메서드 활용

클라이언트에서 서버로 데이터 전송 방법

HTML Form 데이터 전송

- HTML Form submit시 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 Form은 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)
- 참고 : HTML Form 전송은 GET, POST만 지원

HTTP API 데이터 전송

- 서버 to 서버
  - 백엔드 시스템 통신
- 앱 클라이언트
  - 아이폰, 안드로이드
- 웹 클라이언트
  - HTML에서 Form 전송 대신 자바 스크립트를 통한 통신에 사용(AJAX)
  - 예) React, VueJs 같은 웹 클라이언트와 API 통신
- POST, PUT, PATCH : 메시지 바디를 통해 데이터 전송
- GET : 조회, 쿼리 파라미터로 데이터 전달
- Content-TYep : application/json을 주로 사용(사실상 표준)
  - TEXT, XML, JSON 등등..

HTTP API 설계 예시

POST - 신규 자원 등록 특징(회원 관리 시스템)

- 클라이언트는 등록될 리소스의 URI를 모른다.
- 서버가 새로 등록된 리소스 URI를 생성해준다.
- 컬렉션(Collection)
  - 서버가 관리하는 리소스 디렉토리
  - 서버가 리소스의 URI를 생성하고 관리
  - 여기서 컬렉션은 /domains

PUT - 신규 자원 등록 특징(파일 관리 시스템)

- 클라이언트가 리소스 URI를 알고 있어야 한다.
  - 파일 등록 /files/{filename} → PUT
  - PUT /files/star.jpg
- 클라이언트가 직접 리소스의 URI를 지정한다.
- 스토어(Store)
  - 클라이언트가 관리하는 리소스 저장소
  - 클라이언트가 리솟의 URI를 알고 관리
  - 여기서 스토어는 /files

HTML Form 사용

- HTML FORM은 GET, POST만 지원
- AJAX 같은 기술을 사용해서 해결 가능
- 컨트롤 URI
  - GET, POST만 지원하므로 제약이 있음
  - 이런 제약을 해결하기 위해 동사로 된 리소스 경로 사용
  - /new, /edit, /delete가 컨트롤 URI
  - 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 리소스의 URI를 알고 관리
- HTML FORM 사용
  - 순서 HTML + HTML form 사용
  - GET, POST만 지원

참고하면 좋은 URI 설계 개념

- 문서(document)
  - 단일 개념(파일 하나, 객체 인스턴스, 데이터베이스 row)
  - 예) /members/100, /files/star.jsp
- 컬렉션(collection)
  - 서버가 관리하는 리소스 디렉터리
  - 서버가 리소스의 URI를 생성하고 관리
  - 예) /members
- 스토어(store)
  - 클라이언트가 관리하는 자원 저장소
  - 클라이언트가 리소스의 URI를 알고 관리
  - 예) /files
- 컨트롤러(controller), 컨트로 URI
  - 문서, 컬렉션, 스토어로 해결하기 어려운 추가 프로세스 실행
  - 동사를 직접 사용
  - 예) /members/{id}/delete
- 참고 싸이트(https://restfulapi.net/resource-naming)

6. HTTP 상태코드

HTTP 상태코드 소개

상태코드

클라이언트가 보낸 요청의 처리 상태를 응답에서 알려주는 기능

- 1XX : 처리중
- 2XX : **SUCCESSFUL**요청 정상 처리
  - **200** OK
  - **201** Created
  - **202** Accepted
    - 요청이 접수되었으나 처리되지 않음
    - 배치 처리 같은 곳에서 사용
  - **204** No Content
    - 서버가 요청을 성공적으로 수행했지만, 응답 페이로드 본문에 보낼 데이터가 없음
- 3XX : **Redirection**요청을 완료하려면 추가 행동이 필요
  웹 브라우저는 3XX 응답의 결과에 Location 헤더가 있으면, Location 위치로 자동 이동
  - **300** Multiple Choices
  - **301** Moved Permanently(페이지 이동됨, 영구 리다이렉션)
    - 요청 메서드가 GET으로 변경, 본문이 제거될 수 있음
  - **302** Found(일시적인 리다이렉션)
    - 요청 메서드가 GET으로 변경, 본문이 제거될 수 있음
  - **303** See Other
    - 리다이렉트시 요청 메서드가 GET으로 변
  - **304** Not Modified
    - 캐시를 목적으로 사용

        - 클라이언트에게 리소스가 수정되지 않았음을 알려준다.
          따라서 클라이언트는 로컬PC에 저장된 캐시를 재사용한다.
          (캐시로 리다이렉트 한다.)
        - 304 응답은 응답에 메시지 바디를 포함하면 안된다.
        - 조건부 GET, HEAD 요청시 사용
    - **307** Temporary Redirect(일시 리다이렉션)
        - 리다이렉트시 요청 메서드와 본문 유지(요청 메서드를 변경하면 안된다.)
    - **308** Permanent Redirect(특수 리다이렉션, 결과 대신 캐시를 사용)
    - PRG : Post / Redirect / Get
        - Post 로 등록 요청 → Redirect 응답 (Location : get page) → Get page 호출 → 200 응답
- 4XX : **CLIENT ERROR**클라이언트 오류, 잘못된 문법등으로 서버가 요청을 수행할 수 없음
    - 오류의 원인이 클라이언트에 있음
    - **400** Bad Request
        - 요청 구문, 메시지 등등 오류
        - 클라이언트는 요청 내용을 다시 검토하고, 보내야함
    - **401** Unauthorized
        - 클라이언트가 해당 리소스에 대한 인증이 필요함
        - 응답에 WWW-Atuthenticate 헤더와 함께 인증 방법을 설명해야 함
    - **403** Forbidden
        - 서버가 요청을 이해 했지만 승인을 거부함
        - 주로 인증 자격 증명은 있지만, 접근 권한이 불충분한 경우
    - **404** Not Found
        - 요청 리소스를 찾을 수 없음
- 5XX : **SERVER ERROR**서버 오류, 서버가 정상 요청을 처리하지 못함
    - 서버 문제로 오류 발생
    - **500** Internal Server Error
        - 서버 내부 문제

- **503** Service Unavailablwe
  - 서비스 이용 불가
  - 서버가 일시적인 과부하 또는 예정된 작업으로 잠시 요청을 처리할 수 없음
  - Retry-After 헤더 필드로 얼다뮈에 복귀도는지 보낼 수 있음

7. HTTP 헤더1- 일반 헤더

HTTP 헤더 개요

- HTTP 전송에 필요한 모든 부가정보
- 필요시 임의의 헤더 추가 가능
- 표준 헤더 정보는 많음..

HTTP BODY(과거)

> RFC2616

- 메시지 본문(MESSAGE BODY)은 엔티티 본문(entity body)을 전달하는데 사용
- 엔티티 본문은 요청이나 응답에서 전달할 실제 데이터
- 엔티티 헤더는 엔티티 본문의 데이터를 해석할 수 있느 ㄴ정보 제공
  - 데이터 유형(html, json), 데이터 길이, 압축 정보 등

HTTP BODY(최신)

> RFC7230
>
> 엔티티(Entity) → 표현(Representation)

- 메시지 본문(MESSAGE BODY)을 통해 표현 데이터 전달
- 메시지 본문 = 페이로드(PAYLOAD)
- 포현은 요청이나 응답에서 전달할 실제 데이터
- 표현 헤더는 표현 데이터를 해석할 수 있는 정보 제공
  - 데이터 유형(HTML, JSON), 데이터 길이, 압축 정보 등등

표현

- Content-Type : 표현 데이터의 형식
- Content-Endcoding : 표현 데이터의 압축 방식
  - 표현 데이터를 압축하기 위해 사
  - 데이터를 전달하는 곳에서 압축 후 인코딩 헤더 추가
  - 데이터를 읽는 쪽에서 인코딩 헤더의 정보를 이용하여 압축 해제
- Content-Language : 표현 데이터의 자연 언어
- Content-Length : 표현 데이터의 길이(명확하게는 payload 헤더)
  - 바이트 단위
  - Transfer-Encoding(전송 코딩)을 사용하면 사용하면 안됨
- 표현 헤더는 전송, 응답 둘다 사용

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 자연 언어
- 협상 헤더는 요청시에만 사용

협상과 우선순위

> Quality Values(q)

- 0~1, 클수록 높은 우선순위
- 생략하면 1
- Accept—Language : ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7

> 구체적인 것이 우선한다.

- Accept: text/*, text/plain, text/plain;format=flowed, */*
    1. text/plain;format=flowed
    2. text/plain
    3. text/*
    4. */*

전송 방식

- 단순 전송
- 압축 전송(Transfer-Encoding: gzip)
- 분할 전송(Transfer-Encoding: chunked)
- 범위 전송

일반 정보

- From
    - 유저 에이전트의 이메일 정보
    - 일반적으로 잘 사용되지 않음
- Referer
    - 이전 웹 페이지의 주소
    - 참고 : referer는 단어 referrer의 오타
- User-Agent
    - 유저 에이전트 애플리케이션 정보
    - 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
        - 나의 요청을 처리하는 마지막 서버

    - 응답에서 사용
    - Server: Apache/2.2.22 (Debian)
    - server: nginx
- Date
    - 메시지가 발생한 날짜와 시간
    - 응답에서 사용

특별한 정보

- 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
    - 300대 말고 201에서도 사용(요청에 의해 생성된 리소스 URI)
- Allow
    - 허용 가능한 HTTP 메서드
    - 405(Method Not Allowed) 에서 응답에 포함해야 함
- Retry-After
    - 503 : 서비스가 언제까지 불능인지 알려줄 수 있음
- 인증
    - Authorization
        - 클라이언트 인증 정보를 서버에 전달
    - WWW-Authenticate
        - 리소스 접근시 필요한 인증 방법 정의
        - 401 Unauthorized 응답과 함께 사용

쿠키

- Set-Cookie : 서버에서 클라이언트로 쿠키 전달(응답)
- Cookie : 클라이언트가 서버에서 받은 쿠키를 저장하고, HTTP 요청시 서버로 전달
- 쿠키 정보는 항상 서버에 전송됨
- 쿠키 - 생명주기(Expires, max-age)
  - 세션 쿠키 : 만료 날짜를 생략하면 브라우저 종료시 까지만 유지
  - 영속 쿠키 : 만료 날짜를 입력하면 해당 날짜까지 유지
- 쿠키 - 도메인
  - domain=example.org
  - 명시 : 명시한 문서 기준 도메인 + 서브 도메인 포함
  - 생략 : 현재 문서 기준 도메인만 접근
- 쿠키 - 경로(path)
  - 이 경로를 포함한 하위 경로 페이지만 쿠키 접근
  - 일반적으로 path=/ 루트로 지정
- 쿠키 - 보안
  - 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 경우에만 쿠키 전송

8. HTTP 헤더2 캐시와 조건부 요청

캐시가 없을 때

- 데이터가 변경되지 않아도 계속 네트워크를 통해서 데이터를 다운받아야 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 유효 시간이 초과해도, 서버의 데이터가 ㄱ갱신되지 않을 경우
- 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 실용적

검증 헤더와 조건부 요청

검증 헤더

- 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-Math : ETag 사용
- 조건이 만족하면 200 OK
- 조건이 만족하지 않으면 304 Not Modified

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, If-None-Match

- 진짜 단순하게 ETag만 보내서 같으면 유지, 다르면 다시 받기
- 캐시 제어 로직을 서버에서 완전히 관리
- 클라이언트는 단순이 이 값을 서버에 제공
  클라이언트는 캐시 메커니즘을 모름

캐시와 조건부 요청 헤더

캐시 제어 헤더

- Cache-Control : 캐시 제어
  - max-age
    - 캐시 유효 시간, 초 단위
  - no-cache
    - 데이터는 캐시하도 되지만, 항상 origin 서버에 검증하고 사용
  - no-store
    - 데이터에 민감한 정보가 있으므로 저장하면 안됨
- Pragma : 캐시 제어(하위 호환)
  - no-cache

    - HTTP 1.0 하위 호환
- Expires : 캐시 유효 기간(하위 호환)
    - 캐시 만료일을 정확한 날짜로 지정 지정
    - HTTP 1.0부터 사용

프록시 캐시

Cache-Control

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

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 만료후 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

no-cache 와 must-revalidate의 차이

no-cache는 원 서버에 접근할 수 없는 경우에도 캐시 서버 설ㅈ어에 따라 캐시 데이터를 반환할 수 있음

must-revalidate는 원 서버에 접근할 수 없는 경우 항상 오류가 발생해야 함(504 Gateway Timeout)