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

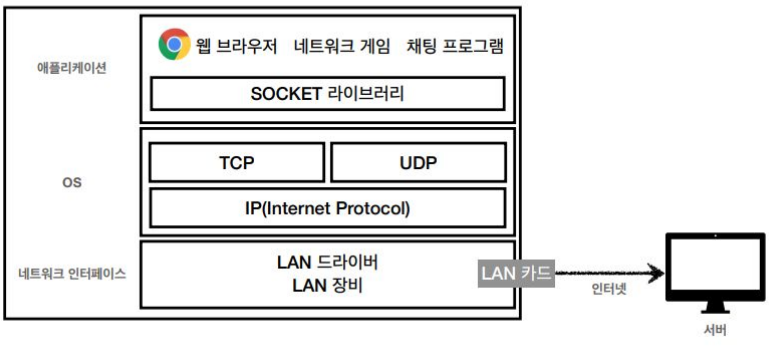
- 지정한 IP 주소에 데이터 전달
- 패킷 으로 데이터 전달

#한계

- 비연결성
- 비신뢰성
- 프로그램 구분

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

프로토콜 계층

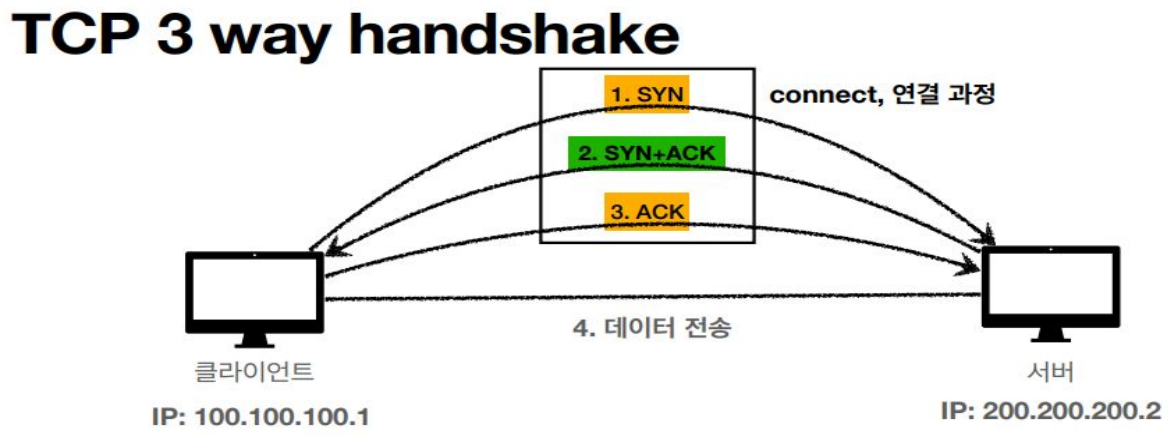


TCP/IP 패킷 정보

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>

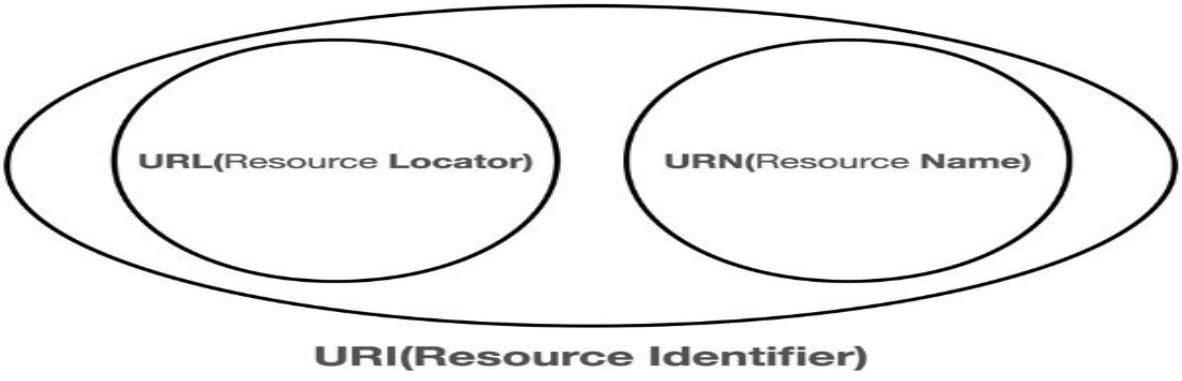
- Domain Name System
- 도메인 이름을 IP주소로 변환

[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

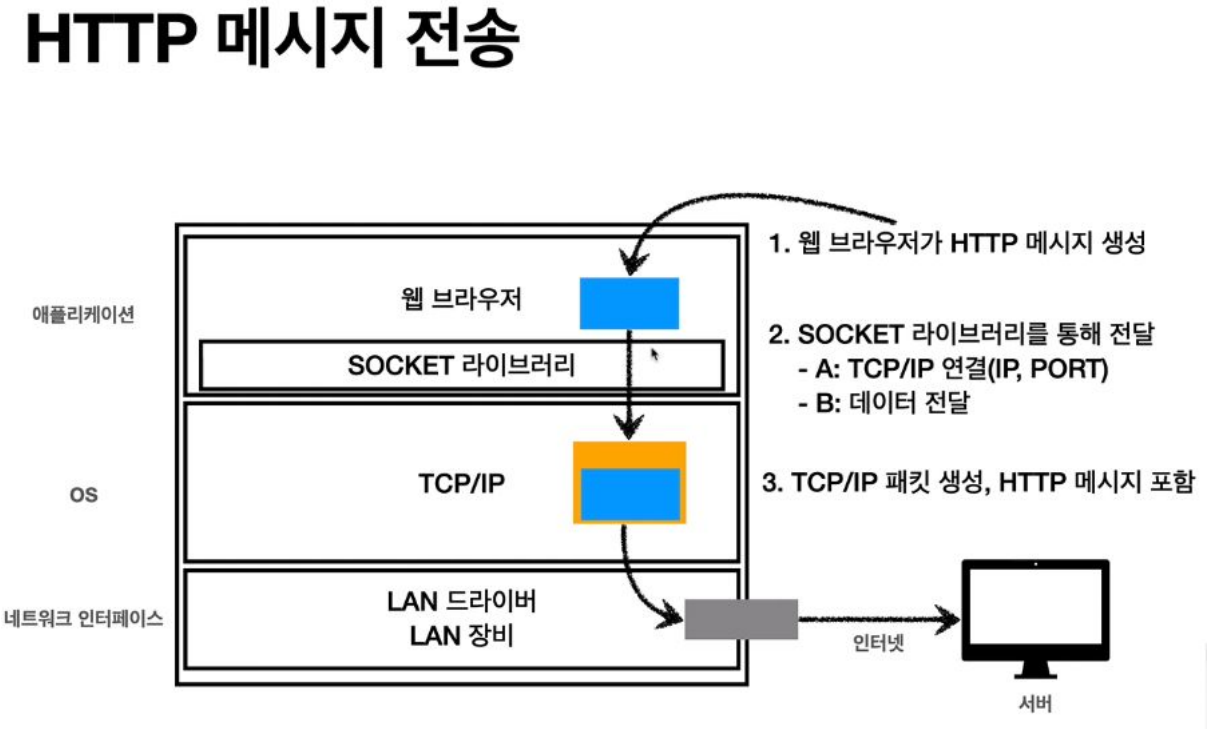
6) query
- key=value 형태
- ?로 시작 &추가 가능 ?keyA=valueA&keyB=valueB
- query parameter, query string 등으로불림 - 웹서버에 제공하는 파라미터, 문자형태

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>

[6장 HTTP 상태코드]
<HTTP 상태코드 소개>
<2xx - 성공>
<3xx - 리다이렉션 1 >
<3xx - 리다이렉션 2 >
<4xx - 클라이언트 오류 / 5xx - 서버 오
[7장 HTTP 헤더1 - 일반 헤더]
<HTTP 헤더 개요>
<표현>
<컨텐츠 협상>
<전송 방식>
<일반 정보>
<특별한 정보>
<인증>
<쿠키>

[8장 HTTP 헤더 2 -캐시와 조건부 요청]
<캐시 기본 동작>
<검증 헤더와 조건부 요청1>
<검증 헤더와 조건부 요청2>

<캐시와 조건부 요청 헤더>
<프록시 캐시>
<캐시 무효화>