HTTP

- 모든 것이 HTTP
- 클라이언트 서버 구조
- Stateful , Stateless
- 비 연결성 (connectionless)
- HTTP 메시지

HTTP (Hyper Text Transfer Protocol)

원래는 HTML 전송 목적

모든 것이 HTTP
(HTTP에 모든 것을 전송)

- HTML, TEXT
- IMAGE, 음성, 영상, file
- JSON, XML (API)
- 거의 모든 형태의 data 전송 가능
- 서버간 data 송수신 시에도 대부분 HTTP 사용

but, 지금은

HTTP 역사

- (1991년) 0.9
  : GET method만 지원, HTTP header X
- (1996년) 1.0
  : method, header 추가
- (1997년) 1.1
  : 가장 많이 사용

) 대부분의 기능

  1997 → 1999 → 2014
  이 버전 중심 ↑최신 (RFC 7230 ~ 7235)
- (2015년) 2
  : 성능 개선
- (~) 3
  : TCP 대신 UDP 사용. 성능 개선

기반 프로토콜

- TCP : HTTP/1.1, HTTP/2) TCP 프로토콜 위에서 동작
  cf) 3-way handshaking
- UDP : HTTP/3) UDP 기반

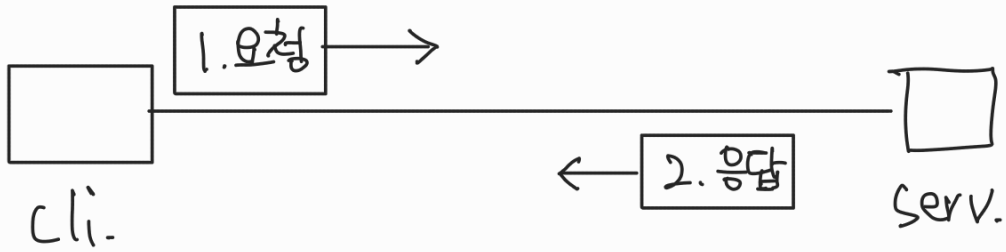
HTTP 특징

- 클라이언트 서버 구조
- 무상태 프로토콜 (stateless), 비연결성
- HTTP 메시지
- 단순함, 확장 가능

클라이언트/서버 구조

분리를 하는 게 중요

- Request Response 구조
- Client는 server에
  request 보내고 response를 대기
- server가 request에 대한
  결과 만들어서 response



UI
UX

business logic
data

⇓

각각 독립적으로 진행 가능

Stateful, Stateless
상태유지 / 유지 X

무상태 프로토콜 / Stateless

- server가 client의 상태를 보존 X
  문맥(context)
- 장점 : server 확장성 높음
- 단점 : client가 추가 data 전송

예시)

- Stateful

  고객 - 점원 & 노트북 & 개수 & 금액

  & 현금/신용카드

- Stateless

  고객이 점원1, 점원2, 점원3과 응대 받음

  ⇒ 제품 ??
  개수 ??
  결제수단 ??

  모름
  ※ 다른 점원은 이전의 context를 모르므로!

Stateless 상황에선

① 제품

② 제품 + 개수

③ 제품 + 개수 + 결제수단

but ③ 만으로도 결제가 가능

① ② 를 거쳤지만

☆ 응답 서버가 바뀜

⟹ 즉, [상태유지] 에서

점원 바뀌는 이런 상황 발생 시

☆ 장애가 발생함

but [무상태] 에선

그때그때마다

필요정보는 다 넘김!

⇒ 문제 X

⟹ 큰 확장성 (무한증식)

상태 유지 (stateful)

: 항상 같은 서버 유지 필수

- 서버 늘리기 어려움

무상태 (stateless)

: 아무 서버나 호출해도 된다.

- 특정 서버 장애 나도
  즉시 서버가 client를
  다른 서버로 연결

- scale out (수평 확장)에 유리함

But, 한계

- 무상태로 설계 불가능한 것도 존재
  - 무상태: 로그인 필요 X인 단순 서비스 소개 화면
  - 상태유지: 로그인
- user의 로그인 했다는 상태를 서버에 유지
- 일반적으로 browser cookie, server session 등을 이용
- 상태 유지는 최소한만 사용

비 연결성 (connectionless)

연결을 유지하는 모델

cf) TCP/IP 연결에선
기본적으로 연결 유지 함.

- 연결 유지 시, (여러 client들 서버에서)
서버의 자원은 계속 소모됨

연결 유지 X인 모델

- 필요한 것만 받고
연결 종료

⇒ 최소한의 자원 유지

비연결성

- HTTP는 기본적으로
연결 유지 X 모델
- 일반적으로 1/5 단위 이하의 빠른 속도로 응답
- 1시간 동안 수천명이 서비스 사용해도
실제 서버의 동시 처리 요청은
수십 개 이하

ex) web browser에서
계속 연속으로 검색 누르진 않음.

- 서버 자원 매우 효율적으로 사용 가능

비연결성 - 한계와 극복

- TCP/IP 연결을 새로 맺어야 함
  ↳ 3 way handshake 시간 추가

- web brower로 사이트 요청 시
  HTML + JS, css, images 등
  수 많은 자원이 함께 down

⇒ 지금은 HTTP 지속 연결 (Persistence Connections)로
문제 해결 cf) keep-alive

ver 2, 3에서 더 최적화

HTTP 초기 - 연결/종료 낭비
↓ But
HTTP 지속 연결

ex) html page 하나 다 받을 때까진
지속 연결 유지

Stateless를 기억

(서버 개발자들이 어려워하는 업무)

- 같은 시간에 딱 맞춰 발생하는 대용량 트래픽

ex) 선착순 event, 명절 KTX

ex) 6시 선착 1000명 치킨 할인

⇒ 대용량 요청

⇒ 최대한 stateless 하게 설계하는 게 중요

(서버 크게 늘려서 대응 가능해서)

HTTP 메시지 / 모든 것이 HTTP이다.

HTTP 메시지 구조

start-line (시작 라인)
header (헤더)
empty line (CRLF)
message body

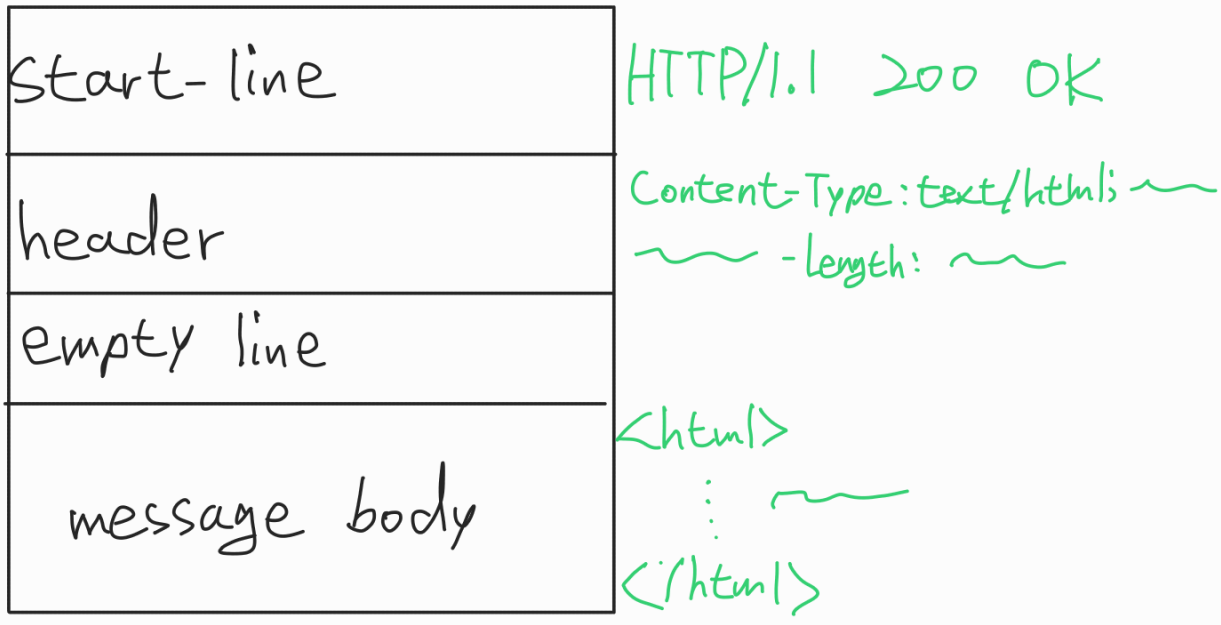
HTTP Request Message

ex)

start-line	GET /search~ HTTP/1.1
header	Host: www.~.com
empty line	
(message body)	request도 body 가질 수 있음

HTTP Response Message

ex)



---

cf) 공식 스펙 (rfc7230)

HTTP-message = start-line
* (header-filed CRLF)
CRLF
[message body]

시작라인 (Request M.)

- start-line

  = request-line

→ request-line

= method SP request-target SP HTTP-version CRLF

(path)

SP (공백)

CRLF Enter

- HTTP method (GET은 조회)

  ◦ 종류 : GET, POST, PUT, DELETE

  ◦ 서버가 수행해야 할 동작 지정

  ↳ GET : resource 조회

  ↳ POST : 요청 내역 처리

- 요청 대상 (request target)

  ◦ absolute-path [?path] (절대경로 [?query])

- HTTP version

  버전

시작 라인 (Response M.)

- start-line

  = status-line

  status-line

  =HTTP-version SP status-code SP reason-phrase CRLF

  SP (공백)

  CRLF Enter

- HTTP version

- HTTP Status Code

  client가 잘못 보낸 것.

  ex) 200: 성공 / 400: 클라이언트 요청 오류 /

  500: internel server error) 서버 장애, 오류 시

- reason-phrase

  짧은 status code 설명

HTTP header

- header-field

  = field name : OWS field-value OWS

  띄어쓰기 허용

  대/소 구분 X

• 용도?

- HTTP 전송에 필요한
  모든 부가정보

ex) cache 관리 정보, 압축, 인증, request client 정보,
(browser)
server app 정보, ...

- 표준 헤더가 너무 多

- 필요시, 임의의 헤더 추가 가능

ex) helloworld: hi

HTTP Message Body

용도?

• 실제 전송할 data

• byte로 표현 가능한
  모든 data 전송 가능
  (doc, image, JSON etc...)

단순함 / 확장 가능

⇒ 크게 성공 - 기술은

단순 but 확장 가능