웹

Post vs get

<https://mommoo.tistory.com/60>

http vs socket

http는 요청&응답으로 서버가 먼저 메세지를 보내지x 단방향 전송

socket은 지속적으로 연결되어 양방향으로 메세지 전송

<https://velog.io/@ejchaid/http>

인터넷

:TCP/IP 기반의 네트워크가 전세계적으로 확대되어 여러 네트워크의 결합체

-www(http) : 웹서비스

-email(smtp,pop3,imap) : 이메일

-ftp(ftp) : 파일전송

-dns(tcp/udp) : 네임서비스

-news(nntp) : 인터넷 뉴스서비스

-ssh : secure shell로 서버 컴퓨터의 쉘에 접근하기 위한 프로토콜

-telnet : terminal network로 서버 컴퓨터의 쉘에 접근하기 위한 프로토콜

\*telnet과 ssh의 차이는 암호화의 차이.. telnet은 암호화가 되어있지않음.. 그러나 ssh가 암호화를 하더라도 압축기술도 사용하여서 트래픽이 심하진않음.. 리눅스는 보안에 취약한 telnet을 지양하여 주로 ssh를 사용.. 암튼 둘다 원격접속을 가능하게해주는건 맞음.. 리눅스에서도 telnet으로 연결 가능하긴함..

-ssl : secure socket layer로 보안된 소켓 통신을 위한 프로토콜..

http(hypertext transfer protocol)

\*hypertext : 웹문서를 구성하고 있는 언어인 HTML을 의미.. 기존 text를 뛰어넘어 아주 멋진(hyper) 링크, 이미지 등등 다양한것들을 표현할수 있다는 의미이다.. HTML로 이루어진 문서 만이 브라우저를 통해 웹 문서로서 읽어질 수 있음..

:웹 브라우저와 웹서버간의 통신규약

-비연결지향 : request를 서버에 보내고, 요쳥에 맞는 response를 받으면 연결 끝

-무상태 : 연결끊는순간 클라이언트와 서버의 통신끝.. => 상태정보 유지x

=> 이를 해결하기위해 Cookie와 같은 기술이 등장!

-요청 메서드

-서버에게 요청의 종류를 알려주기 위해 사용

-최초의 웹 서버는 GET 방식만 지원

-각 메서드의 사용 목적

-GET: 정보 요청 (SELECT)

-POST: 정보 밀어넣기 (INSERT)

-PUT: 정보 업데이트 (UPDATE)

-DELETE: 정보 삭제 (DELETE)

-HEAD: (HTTP) 헤더 정보만 요청하는 메서드. 해당 자원이 존재하는지 혹은 서버에 문제가 없는지를 확인

-OPTIONS: 웹 서버가 지원하는 메서드의 종류를 요청

-TRACE: 클라이언트의 요청을 그대로 반환하는 메서드. Ex) echo 서비스로 서버 상태를 확인하기 위한 목적으로 주로 사용

Http 메세지 형식

Request헤더 + 빈 줄 + Request바디

헤더구조

-General Header(공통헤더) : 요청 및 응답에 사용되는 공통헤더. Ex) Date, Connection, Cache-Control, Pragma, Trailer

-ContentType :

-application/json : json형식..

-application/x-www-form-urlencoded : key1=value1&key2=value2 ,, => 복잡한데이터 자바에서 바인딩이 어려움.. 복잡한것은 되도록이면 json으로!

Http관련하여 전체적으로 간단하게 정리되어있음 : https://gmlwjd9405.github.io/2019/04/17/what-is-http-protocol.html

Header 구조

<https://gmlwjd9405.github.io/2019/01/28/http-header-types.html>

<https://www.zerocho.com/category/HTTP/post/5b3ba2d0b3dabd001b53b9db>

<https://www.zerocho.com/category/HTTP/post/5b4c4e3efc5052001b4f519b>

쿠키 & 헤더 캐시

<https://www.zerocho.com/category/HTTP/post/5b594dd3c06fa2001b89feb9>

캐시관련

-어떤것을 나중에 유용하게 사용하기위해 임시로 저장하는것!

-반복적인 요청시 이에 해당하는 응답중 (부하가 많을수 있는 요청이라면 이를 줄이기위해) 해당 내용의 변경이 없을때에는 그대로 두고 변경시에만 가져오도록 해주는 기능..(캐시를 어떤식으로 가져올지에 따라 매번 변경을 요청할수도, 안할수도 있음..)

-자주 사용하는 쿼리도 캐싱처리가 가능하네..

캐시관련 자세한 내용: <https://feel5ny.github.io/2019/10/05/HTTP_007-2/>

웹브라우저에서 캐시가 어떻게 작동되는지 아주 잘 설명되어있음 : <https://hahahoho5915.tistory.com/33>

\*CDN : <https://ijbgo.tistory.com/32>

URI vs URL

Cookie vs Session

|  |  |
| --- | --- |
| cookie | Session |
| 클라이언트에 저장 | 같은 브라우저로부터 들어오는 요청을 하나의상태로 보고 그 상태를 **서버**에 저장 |
| 브라우저가 종료되어도 쿠키만료가 아직 안되었다면 계속 사용.. | 브라우저 종료시 끝.. |
| 비밀번호저장, 장바구니 등에 쓰임 | 로그인상태 유지에 쓰임 |
| 브라우저의 헤더(cookie)에서 가져옴  -클라이언트에 응답할때 Set-Cookie에 넣어서전송 | 브라우저의 헤더(cookie)에서 가져옴  -클라이언트에 응답할때 Set-Cookie에 넣어서전송  -cookie의 JSESSIONID에서 가져옴.. |
|  |  |
| 여러 정보들이 클라이언트에 저장되므로 보안에 취약할수 있음 | 쿠키를 이용해 Session ID(서버에서발급)만 저장하고 이 값으로 구분해서 서버에서 처리하므로 비교적 보안 괜춘 |

<https://doooyeon.github.io/2018/09/10/cookie-and-session.html>

CORS

(크로스 사이트 어쩌구)

WebSocket

-이전에 http의 요청에서 단방향으로 (request -> response) 움직일수밖에없는 상황을 해결하기 위해서 만들어짐.. 즉, 서버에서 먼저 전달할수도 있음!( ex. Push, 새로고침 계속 안해도 업뎃되는것.. )

-웹페이지와 서버간에 실시간 상호작용을 하기위해 만들어진 스펙..

=>이를 해결하기위해 꼼수로 Polling(주기적으로 요청) 방법과 streaming(한번 request한뒤 끊지 않고 필요한 메세지만 계속 보내기..)방법이 사용되어지곤했었음..

Websocket관련 역사와 개념들 정리..

<https://medium.com/@chullino/http%EC%97%90%EC%84%9C%EB%B6%80%ED%84%B0-websocket%EA%B9%8C%EC%A7%80-94df91988788>

웹소켓 잘 정리 : <https://brunch.co.kr/@adrenalinee31/2>

HTML 쉽게 정리 매우 굿 : <https://brunch.co.kr/@coveryou/14#_=_>

HttpURLConnection