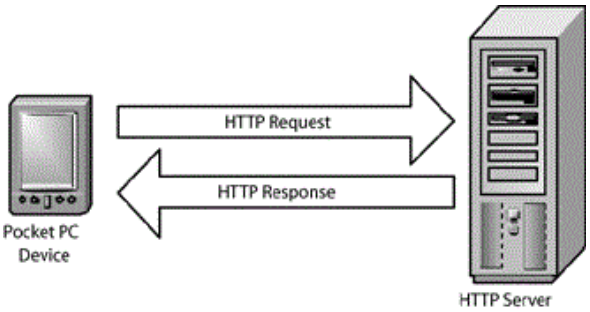
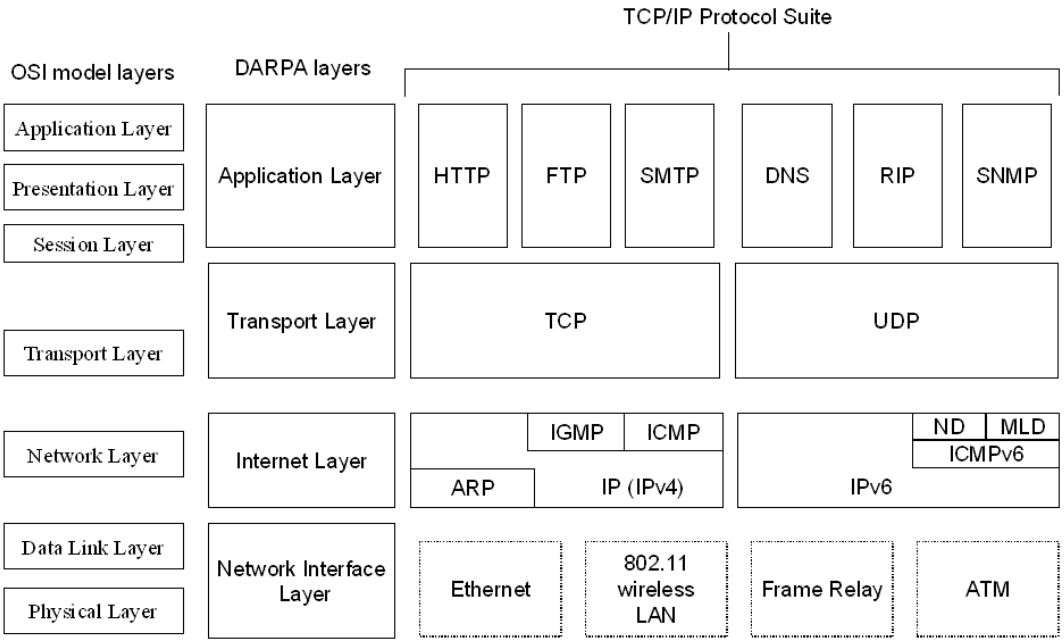
HTTP을 사용한 통신 과정

☑ HTTP 란?

- ☞ HTTP는 Hypertext Transfer Protocol의 약자로 인터넷 상에서 데이터를 주고받기 위한 프로토콜이다.
- ☞ **데이터**는 오디오 / 비디오 / 이미지 / 텍스트 등 어떠한 데이터의 종류를 가리지 않는다.
- ☞ 모두 HTTP 프로토콜을 이용해 전달하고 전달 받을 수 있다.
- ☞ 브라우저는 HTTP 통신을 통해서 사이트 문서를 가져오고 이를 해석해 화면에 출력하게 된다.



- ☞ 이때 TCP/IP 계층의 순서로 네트워크에 접근하게 되는데 HTTP 프로토콜은 4계층(Application)에 속해 있다.
- ☞ 4계층에서 **정보**를 만들어 전달하면 3계층(Transport)에서 **통신 노드**를 **연결**하고, 2계층(Internet)에서 통신노드간 **패킷**을 **전송**과 **라우팅**하고, 1계층(Network Interface)에서 **전기적 신호**로 **변환**하여 실제로 전달한다.
- ☞ 받는 쪽에서는 반대로 해석한다.



☞ 모든 컴퓨터(Client)와 Server는 네트워크를 통해 외부에서 접근하려면 IP 주소가 필요하다.
☞ 하지만 사용자는 Browser를 이용해 문자 URL을 전달한다.
☞ 때문에 이를 해석하기위해 **DNS Server**로 접근하여 해당 Domain Name에 맞는 IP를 받아온다.
☞ 이 과정을 이름 해결이라 부르며 UDP 통신을 한다.
☞ 받은 IP를 가지고 한번에 Server에 접근할수 없으며 IP를 알고있는 다른 Server에 접근하면서 경유하여 접근하게 된다.

[참고] DNS Server

❖ DNS Server의 역할

인터넷에서는 컴퓨터를 식별하기 위해 IP 주소를 사용한다. 그러나 이 숫자만으로는 무엇에 사용되고 있는지 알 수 없다. 따라서 인터넷에서는 IP 주소에 Domain Name이라는 이름을 붙혀 알기 쉽게 한다.
IP 주소와 Domain Name을 서로 교환하는 장치를 DNS(Domain Name System)이라 한다.

❖ Domain Name은 Tree 구조로 되어 있다.

Domain Name은 'www.semyung.ac.kr'과 같이 점(.)으로 구분된 문자열로 구성되어 있다.
이 하나하나의 문자열을 라벨이라고 하며, 오른쪽부터 순서대로 '탑레벨 도메인', '제 2레벨 도메인', '제 3레벨 도메인'과 같이 부른다. 즉 트리 모양의 계층 구조로 되어 있다.

❖ DNS Server의 종류

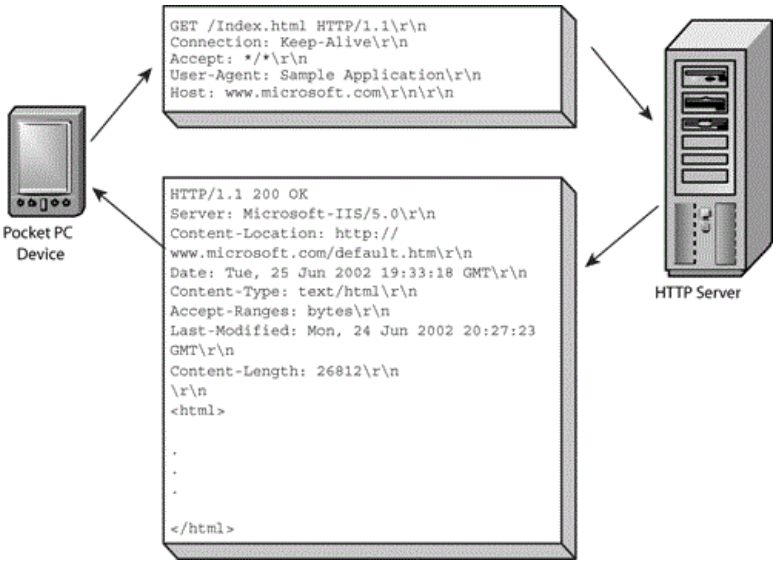
DNS Server는 Cache Server와 Content Server로 크게 나눈다.

☞ Cache Server는 LAN 안에 있는 Client로부터 조회를 받아 Client를 대신하여 인터넷에 조회해 주는 DNS Server 이다. Client가 인터넷에 Access할 때 사용한다.

☞ Content Server는 외부 Host로부터 자신이 관리하는 Domain에 관한 조회를 받는 DNS Server 이다. 자신의 Domain 내의 Host Name은 zone 파일이라는 DB에서 관리한다.
☞ Client로부터 조회를 받은 Cache Server는 받은 Domain Name을 오른쪽부터 순서대로 검색하여 해당 Domain Name을 관리하는 Content Server를 찾는다. 거기까지 도달하면 해당 Content Server에 대해 Host Name + Domain Name에 대응하는 IP주소를 가르쳐 준다.
☞ 이러한 동작을 이름 해결이라 한다.

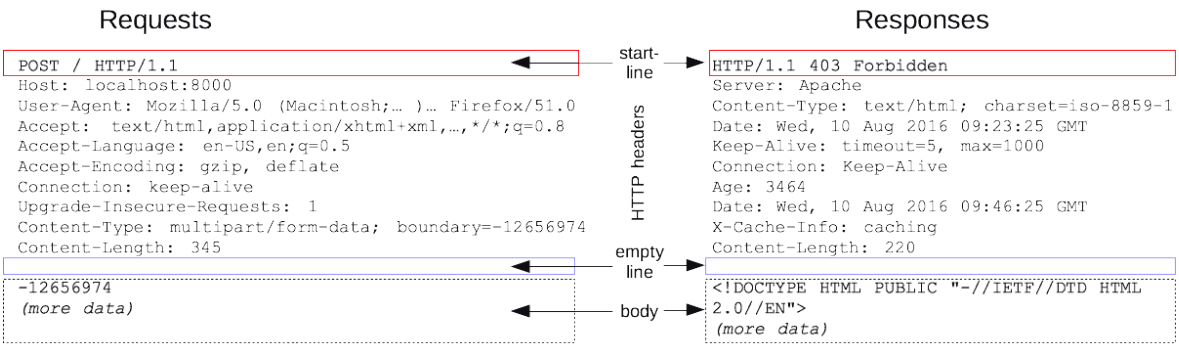
☑ Browser는 Server에게 어떻게 데이터를 받아올까?

☞ Browser는 Server에게 Web Site 문서를 받아오기 위해 Client에서 Request(요청)을 만들어 Server에게 전달한다.
☞ Server는 Client의 Request를 해석하고, 요청에 해당하는 Response(응답)를 전달하게 된다.
☞ Client에서는 Server에서 전달해준 Response를 이용해 화면에 표현하게 된다.



☞ HTTP 프로토콜의 데이터 형식은 크게 **Header**와 **Body**로 구성되어 있다.

☞ Header에는 **Server**가 인식할 수 있는 **약속된 형식**을 따라야 한다.



☑ Request Header

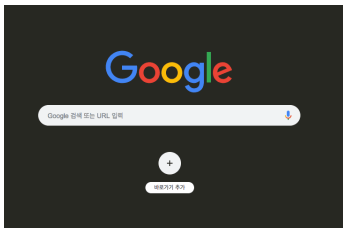
☞ 해당 형식으로 표현된 데이터를 전달하면 Server에서는 해당 형식을 해석하고 Response를 전달한다.

① GET / HTTP/1.1 : **HTTP 전송 방법**과 프로토콜 버전
② Host : 요청하는 Server 주소
③ User-Agent : OS/브라우저 정보
④ Accept : Client 이해 가능한 컨텐츠 타입
⑤ Accept-Language : Client 인식 언어
⑥ Accept-Encoding : Client 인코딩 방법
⑦ Connection : 전송 완료후 접속 유지 정보 (keep-alive)
⑧ Upgrade-Insecure-Requests : 신호를 보낼때 데이터 암호화 여부
⑨ Content-Type : Client에게 반환되어야하는 컨텐츠 유형
⑩ Content-Length : 본문크기

☑ Response Header

① HTTP/1.1 200 ok : 프로토콜 버전과 **응답 상태**
② Access-Control-Allow-Origin : Server에 타 사이트의 접근을 제한하는 방침
③ Connection : 전송 완료후 접속 유지 정보 (keep-alive)
④ Content-Encoding : 미디어 타입을 압축한 방법
⑤ Date : 헤더가 만들어진 시간
⑥ ETag : 버전의 리소스를 식별하는 식별자
⑦ Keep-Alive : 연결에대한 타임아웃과 요청 최대 개수 정보
⑧ Last-Modified : Web 시간을 가지고 있다 수정되었을때만 데이터 변경 (캐시연관)
⑨ Server : Web Server로 사용되는 프로그램 이름
⑩ Set-Cookie : **쿠키 정보**
⑪ Transfer-Encoding : 인코딩 형식 지정
⑫ X-Frame-Options : frame/iframe/object 허용 여부

☑ 사용자가 Web Browser를 통해 Server에 이미지를 요청해서 사용자에게 보여주기까지 과정



⊙ google.png 이미지를 브라우저에서 보기까지 과정에 대해 알아보자.

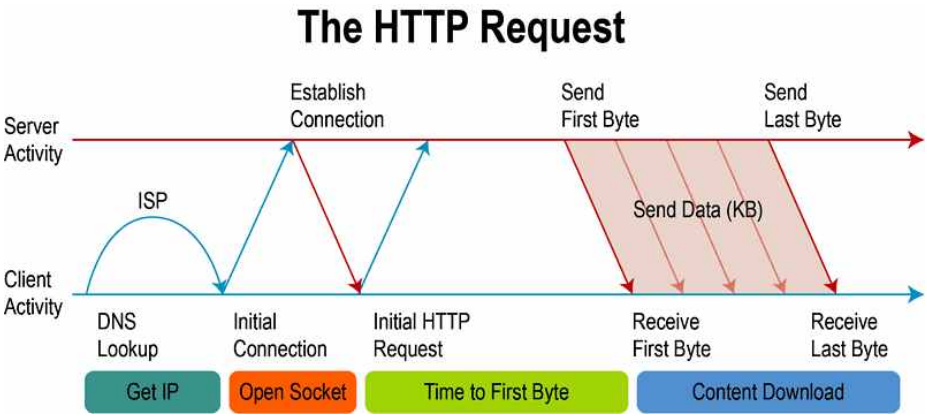
☞ www.google.com에 접속하면 index.html을 받아온다.
☞ 여기 안에 이미지로 https://www.google.com/images/google.png가 들어있다.
☞ Web Browser는 html파일을 읽어서 해석을 하는데 **이미지 주소**가 나오면 해당 **URL로 Server에 request(요청)**을 보낸다.
☞ request를 주고받는 과정을 일련의 **트랜잭션(transaction)**이라고 칭한다.
☞ 위 구글 첫 화면이 나오는 과정에서 이미지 요청을 보내고 받았기 때문에 트랜잭션이 일어났다고 볼 수 있다.
☞ 이미지에 대한 트렌잭션뿐만 아니라 html page도 트랜잭션이 이뤄졌다.
☞ 이처럼 1개의 페이지를 보여주는데도 **많은 수의 트랜잭션**이 발생한다.
☞ 여기선 HTTP 통신을 이용하기 때문에 **HTTP Transaction** 이라 부른다.
☞ HTTP Transaction은 **HTTP Message**라고 불리는 데이터 덩어리를 이용해 이루어진다.

이걸 좀 정리를 해보면 다음과 같다.

① Web Browser가 https://www.google.com/images/google.png로 이미지를 요청해야 한다는 것을 인지한다.
② Web Browser는 URL을 이용해 Server의 IP를 추출한다. DNS Server에 요청하게 되고 이 때 IP를 찾기 위한 이름 해결 과정은 UDP 통신으로 이루어 진다.
③ 이미지를 요청하기 위한 HTTP Message를 만든다.

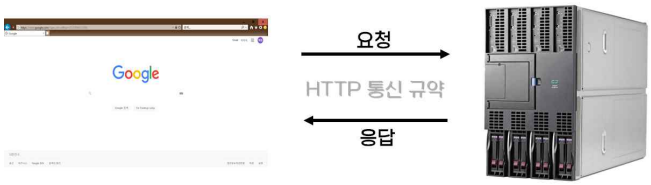
④ 메세지는 GET method이고 /google.png를 요청하는 메세지이다.
⑤ Web Browser는 Server와 TCP 3Way Handshaking 방식으로 커넥션을 생성한다.
⑥ Web Browser는 Server에 HTTP request를 보낸다.
⑦ Server는 메세지를 받고 HTTP Message를 해석한다. GET 이라는 method이고 /google.png라는 파일을 요청 했다는 것을 인지한다.
⑧ Server는 해당 리소스가 있는지 찾는다.
⑨ 찾으면 상태코드가 200인 메세지와 함께 response message를 작성 후 Client에게 전송한다.
⑩ Server는 Client와 TCP 4Way Handshaking 방식으로 커넥션을 종료한다.
⑪ 이미지를 받은 Client는 Web Browser에 이미지를 띄우고 사용자에게 보여준다.

여기서 중요 포인트는 **TCP 커넥션 과정**과 **HTTP Request/Response**를 보내는 부분이다.



HTTP Protocol 과 데이터 전달 방식

HTTP(HyperText Transfer Protocol)은 Web Browser와 Web Server가 서로 소통하기 위한 프로토콜, 즉 통신 규약이다. 우리가 말하는 Web도 HTTP가 제공하는 서비스를 바탕으로 이루어진다. Web Page에서의 **폼(form)** 양식 처리를 이해하려면 먼저 HTTP의 동작원리를 이해해야 한다.

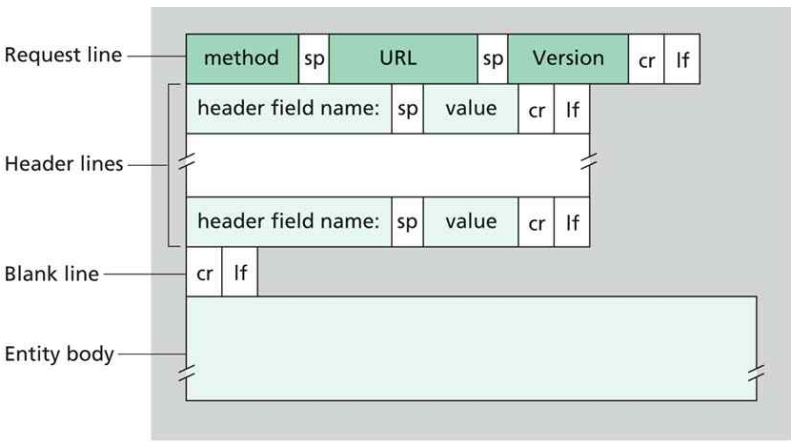


☑ HTTP 통신 규약에 의한 데이터 흐름

Web Browser가 HTTP 통신 규약에 의해 Web Server에 데이터를 요청하면 Web Server는 이를 처리하여 결과를 Web Browser에 돌려준다. 이러한 데이터 흐름은 아래와 같다.

① 사용자가 Web Browser 주소 창에 URL을 입력하거나 Web 페이지의 링크를 클릭한다.
② HTTP 통신 규약에 의해 사용자의 요청이 Web Server로 전달된다.
③ Web Server는 사용자의 요청을 처리하여 그 결과를 HTTP 통신 규약에 의해 사용자의 Web Browser로 전송한다.
④ Web Browser는 Web Server가 보낸 데이터를 분석하여 화면에 출력한다.
⑤ Web Browser 화면에 텍스트, 글자, 동영상 등의 형태로 나타난 요청 결과를 사용자가 확인한다.

☑ 데이터 전달 방식



General format of a request message

Web Browser에서 Web Server로 데이터를 전달하는 데에는 method 방식과 dynamic parameter 방식이 있다. dynamic parameter 방식의 경우 method 방식과 혼용해서 사용이 가능하다.

1. method 방식

⊙ POST

- ☞ HTTP 프로토콜 요청 메시지의 **Body**를 타고 데이터 전달된다(body에 숨겨져서 보내지며 암호화 개념은 아니다).
- ☞ POST 방식은 **보안성**이 우수하여 **대량**으로 보낼 때 주로 사용한다.
- ☞ **회원가입** 양식이나 **게시판**에 글을 쓸 때처럼 **사용자**가 폼 양식에 **입력한 데이터**를 Web Server로 전달할 때 사용한다. 이미지와 같은 바이너리 파일도 Web Server로 전달할 수 있다. POST는 Server로 데이터를 **전송**하여 Server의 상태 등을 바꾸기 위해서 사용된다.

⊙ GET

- ☞ URL 뒤에 '?' 를 붙여서 문자열의 시작을 알리고, 그 뒤에 '키 = 값 & 키 = 값 & ...' 형태의 query string을 붙여서 전송하는 방식이다. 이는 단지 문자열이기 때문에 route와는 관계가 없다. 여기서 말하는 'header' 와 'body' 는 HTML에서 쓰는 태그와 전혀 다른 개념이니 혼동하지 않아야 한다.
- ☞ HTTP 프로토콜 요청 메시지의 **Header**를 타고 데이터 전달된다. 데이터가 직접적으로 노출되기 때문에 보안성이 떨어지는 편이다.
- ☞ GET 방식은 POST와 반대의 특성을 가져서 **소량**으로 보낼 때 사용한다. GET은 **URL 주소 뒤**에 데이터를 입력하여 Web Server로 전달하기 때문에 많은 양의 데이터를 보내기에는 적합하지 않다.
- ☞ GET 방식은 POST 방식과 달리 **캐싱**이 가능하기 때문에 더 **빠르다**. 캐싱이란 한 번 접근한 사이트를 다시 재요청할 때 빠르게 접근하기 위해서 데이터를 저장해두는 것을 말한다.
- ☞ 흔히 GET 방식은 어떤 데이터를 **가져와서** 보여주는 경우, 예를 들면 특정 Web Page를 띄우는 요청하

는 경우에 적합하다.

GET이 URL로 노출되기 때문에 보안성이 약하다고 하였는데, POST 역시 message body를 타고 올 뿐이지 안전한 방식은 아니다. 애초에 보안이 강화된 HTTPS를 사용하거나, 보안에 민감한 데이터는 암호화를 거쳐서 전송하는 것이 옳다.

GET 방식은 주로 데이터를 가져와서 보여줄 때(페이지 요청), POST 방식은 주로 Server의 상태를 변경, 데이터를 수정할 때 등등에 사용된다.

2. 동적 파라미터 방식

☞ 데이터(파라미터)를 URL에 실어서 전송하는 방식이다. 동적 파라미터 방식은 보안에 민감하지 않은 데이터를 보낼 때 사용한다.

☞ '주소에 데이터가 있다' = '간단하다' = '보안에 취약하다' = '보안에 민감하지 않은 데이터 보낼 때 사용한다'

HTTP 응답코드 method 정리(GET, POST, PUT, PATCH, DELETE, TRACE, OPTIONS)

☑ HTTP method

HTTP method는 Client가 Web Server에게 사용자 request(요청)의 목적이나 종류를 알리는 수단이다. 최초의 HTTP에서는 GET method 하나 밖에 없었지만 이후 다양한 method들이 생겨났다.

☑ HTTP method 종류와 특징

HTTP method의 종류는 총 9가지가 있다. 이 중 주로 쓰이는 method는 5가지가 있다. 이제 각각의 이름과 특징에 대해 알아보도록 하자.

❖ 주요 method 5가지

- ☞ GET : resource 조회
- ☞ POST : 요청 데이터 처리, 주로 데이터 등록에 사용
- ☞ PUT : resource를 대체, 해당 resource가 없으면 생성
- ☞ PATCH : resource를 일부만 변경
- ☞ DELETE : resource 삭제

기타 method 4가지

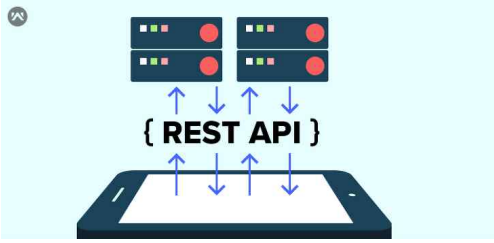
- ☞ HEAD : GET과 동일하지만 메시지 부분을 제외하고, 상태 줄과 헤더만 반환
- ☞ OPTIONS : 대상 resource에 대한 통신 가능 옵션을 설명(주로 CORS에서 사용)
- ☞ CONNECT : 대상 resource로 식별되는 Server에 대한 터널을 설정
- ☞ TRACE : 대상 resource에 대한 경로를 따라 message loop back test를 수행

여기서 주요 메서드 5가지에 대해 좀더 자세하게 알아보도록 하자.

- ☞ GET은 보통 리소스를 조회할 때 사용하며, Server에 전달하고 싶은 데이터는 query를 통해서 전달한다. message body를 사용해서 데이터를 전달할 수는 있지만, 지원하지 않는 곳이 많아서 권장하지 않는다.
- ☞ POST는 데이터 요청을 처리하고, message body를 통해 Server로 데이터를 전달한다. 주로 신규 리소스를 등록하거나 프로세스 처리에 사용된다.
- ☞ PUT은 리소스가 있으면 대체하고 리소스가 없으면 생성한다. 쉽게 말해 데이터를 덮어쓴다.
- ☞ PATCH는 PUT과 마찬가지로 리소스를 수정할 때 사용하지만, PATCH는 리소스를 일부분만 변경할 수 있다.
- ☞ DELETE는 리소스를 제거할때 사용한다.

☑ HTTP status code

http 상태 코드는 Client가 보낸 request(요청)의 처리 상태를 response(응답)에서 알려주는 기능이다.



☑ HTTP Request 정보

GET /index.html HTTP/1.1	요청 URL정보 (Mehotd /URI HTTP버젼)
user-agent: MSIE 6.0; Window NT 5.0	사용자 Web 브라우져 종류
accept: test/html; */*	요청 데이터 타입 (응답의 Content-type과 유사)
cookie:name=value	쿠키(인증 정보)
refere: http://abc.com	경유지 URL
host: www.abc.com	요청 도메인

☑ HTTP Response 정보

HTTP/1.1 200 OK	프로토콜 버젼 및 응답코드
Server: Apache	Web Server 정보
Content-type: text/html	MIME 타입
Content-length : 1593	HTTP BODY 사이즈
<html><head>.....	HTTP BODY 컨텐츠

⊙ HTTP Response Code

응답대역	응답코드	설명
정보전송 임시응답	100	Continue (Client로 부터 일부 요청을 받았으며 나머지 정보를 계속 요청함)
	101	Switching protocols
성공	200	OK(요청이 성공적으로 수행되었음
	201	Created (PUT method에 의해 원격지 Server에 파일 생성됨)
	202	Accepted(Web Server가 명령 수신함)
	203	Non-authoritative information (Server가 Client 요구 중 일부만 전송)
	204	No content, (PUT, POST, DELETE 요청의 경우 성공은 했지만 전송할 데이터가 없는 경우)
redirection	301	Moved permanently (요구한 데이터를 변경된 타 URL에 요청함 / Redirect된 경우)
	302	Not temporarily
	304	Not modified (컴퓨터 로컬의 캐시 정보를 이용함, 대개 gif 등은 Web Server에 요청하지 않음)
Client 요청에러	400	Bad Request (사용자의 잘못된 요청을 처리할 수 없음)
	401	Unauthorized (인증이 필요한 페이지를 요청한 경우)

	402	Payment required(예약됨)
	403	Forbidden (접근 금지, 디렉터리 리스팅 요청 및 관리자 페이지 접근 등을 차단)
	404	Not found, (요청한 페이지 없음)
	405	Method not allowed (혀용되지 않는 http method 사용함)
	407	Proxy authentication required (프락시 인증 요구됨)
	408	Request timeout (요청 시간 초과)
	410	Gone (영구적으로 사용 금지)
	412	Precondition failed (전체 조건 실패)
	414	Request-URI too long (요청 URL 길이가 긴 경우임)
Server 에러	500	Internal server error (내부 Server 오류)
	501	Not implemented (Web Server가 처리할 수 없음)
	503	Service unnailable (서비스 제공 불가)
	504	Gateway timeout (게이트웨이 시간 초과)
	505	HTTP version not supported (해당 http 버전 지원되지 않음)

⊙ HTTP method 정리

HTTP Method	전송형태	설명
GET	GET [request-uri]?query_string HTTP/1.1 Host:[Hostname] 혹은 [IP]	요청받은 URI의 정보를 검색하여 응답한다.
HEAD	HEAD [request-uri] HTTP/1.1 Host:[Hostname] 혹은 [IP]	GET방식과 동일하지만, 응답에 BODY가 없고 응답코드와 HEAD만 응답한다. WebServer 정보확인, 헬스체크, 버젼확인, 최종 수정일자 확인등의 용도로 사용된다.
POST	POST [request-uri] HTTP/1.1 Host:[Hostname] 혹은 [IP] Content-Lenght:[Length in Bytes] Content-Type:[Content Type] [데이터]	요청된 자원을 생성(CREATE)한다. 새로 작성된 리소스인 경우 HTTP헤더 항목 Location : URI주소를 포함하여 응답.
PUT	PUT [request-uri] HTTP/1.1 Host:[Hostname] 혹은 [IP] Content-Lenght:[Length in Bytes] Content-Type:[Content Type] [데이터]	요청된 자원을 수정(UPDATE)한다. 내용 갱신을 위주로 Location : URI를 보내지 않아도 된다. Client측은 요청된 URI를 그대로 사용하는 것으로 간주함.
PATCH	PATCH [request-uri] HTTP/1.1 Host:[Hostname] 혹은 [IP] Content-Lenght:[Length in Bytes] Content-Type:[Content Type] [데이터]	PUT과 유사하게 요청된 자원을 수정(UPDATE)할 때 사용한다. PUT의 경우 자원 전체를 갱신하는 의미지만, PATCH는 해당자원의 일부를 교체하는 의미로 사용.
DELETE	DELETE [request-uri] HTTP/1.1 Host:[Hostname] 혹은 [IP]	요청된 자원을 삭제할 것을 요청함. (안전성 문제로 대부분의 Server에서 비활성)
CONNECT	CONNECT [request-uri] HTTP/1.1 Host:[Hostname] 혹은 [IP]	동적으로 터널 모드를 교환, 프락시 기능을 요청시 사용.
TRACE	TRACE [request-uri] HTTP/ 1.1 Host: [Hostname] 혹은 [IP]	원격지 Server에 루프백 메시지 호출하기 위해 테스트용으로 사용.
OPTIONS	OPTIONS [request-uri] HTTP/ 1.1 Host: [Hostname] 혹은 [IP]	WebServer에서 지원되는 method의 종류를 확인할 경우 사용.

☑ HTTP POST과 PUT의 차이

- ☞ **POST는 보통 INSERT의 개념**으로 사용되고, **PUT은 UPDATE 개념**으로 생각하면 이해하기 쉽다. 또한 POST는 멱등(연산을 여러 번 적용하더라도 결과가 달라지지 않는 성질)하지 않고 PUT은 멱등하다. 즉 동일한 자원을 여러번 POST 하면 Server자원에는 변화가 생기지만, 여러번 PUT하는 경우는 변화가 생기지 않는다.
- ☞ 예를들어 POST의 경우 Client가 리소스의 위치를 지정하지 않는 경우 사용된다. (/dogs) 따라서, 아래와 같은 요청이 여러번 수행되는 경우 매번 새로운 dog가 생성되어 dogs/3, dogs/4 등 매번 새로운 자원이 생성된다. 멱등하지 않다는 말이다.

```
POST /dogs HTTP/1.1
{ "name": "blue", "age": 5 }
HTTP/1.1 201 Created
```

- ☞ 반면 PUT의 경우는 Client가 명확하게 리소스의 위치를 지정한다. (/dogs/3) 따라서, 아무리 많이 수행되더라도 리소스의 위치가 지정되어 새로운 자원이 생성되지 않으며 동일한 리소스(/dogs/3)를 수정하기 때문에 여러번 요청하더라도 멱등하다.

```
PUT /dogs/3 HTTP/1.1
{ "name": "blue", "age": 5 }
```

☑ HTTP PUT과 PATCH의 차이

- ☞ **PUT이 해당 자원의 전체를 교체**하는 의미를 지니는 대신, **PATCH는 일부를 변경한다는 의미**를 지니기 때문에 최근 update 이벤트에서 PUT보다 더 의미적으로 적합하다고 평가받고 있다.
- ☞ 또한 PUT의 경우는 멱등하지만, PATCH의 경우는 멱등하지 않다.
- ☞ PUT은 전체 자원을 업데이트 하기 때문에 동일 자원에 대해서 동일하게 PUT을 처리하는 경우 멱등하게 처리된다. 반면 PATCH로 처리되는 경우 자원의 일부가 변경되기 때문에 멱등성을 보장할 수 없다.

☑ HTTP 멱등성

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

- ☞ 멱등(idempotent)의 의미는 같은 작업을 계속 반복해도 같은 결과가 나오는 경우를 의미한다.
- ☞ 동일한 자원에 대한 GET 요청이라면 Client에 반환되는 모든 응답은 동일해야 한다.
- ☞ 특정 자원에 대한 DELETE의 경우도 자원은 더이상 이용하 수 없어야 하며, DELETE 요청을 다시 호출한 경우도 자원은 여전히 사용할 수 없는 상태야여 한다.