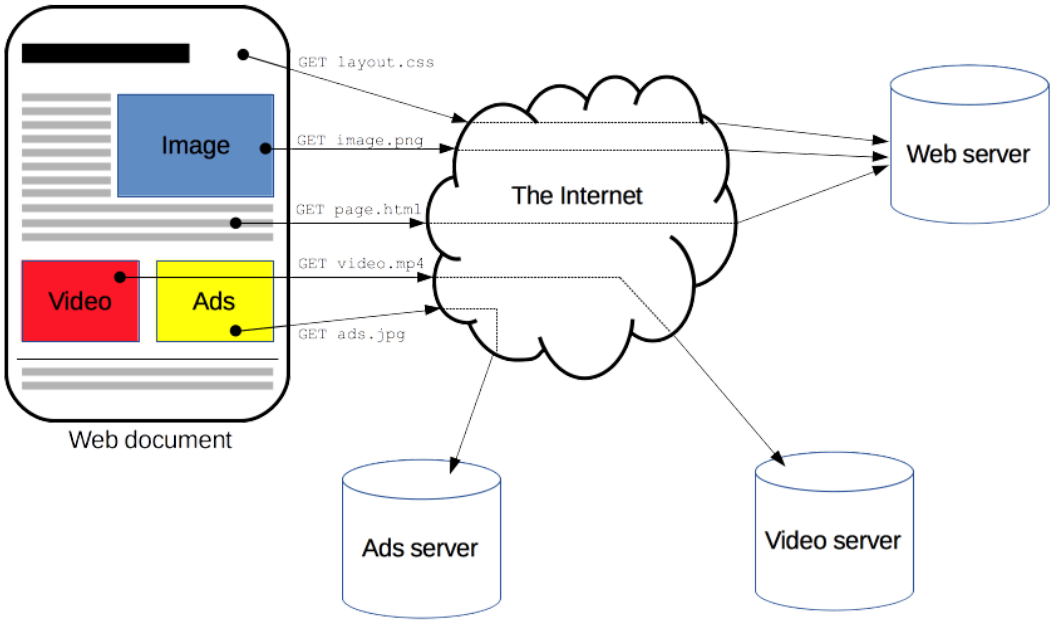
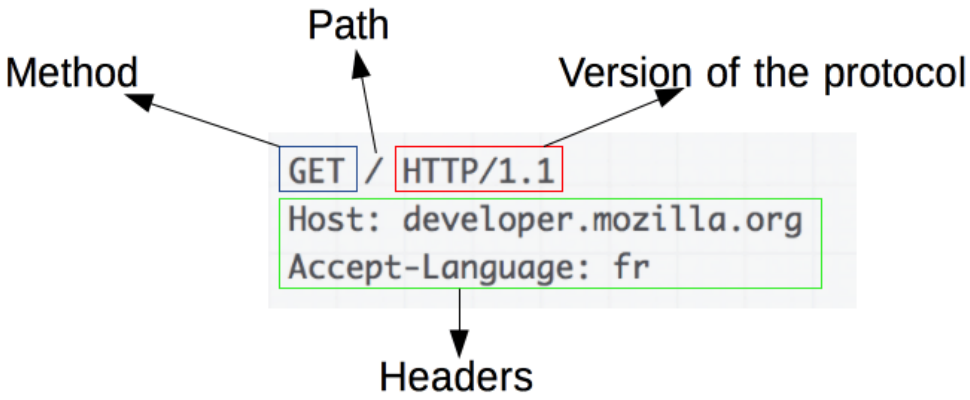
2023.03.26 HTTP, HTTPS, HTTP1.1, HTTP2.0

HTTP(Hyper Text Transfer Protocol)

1. HTTP란 **하이퍼 미디어 문서(HTML 등)를 전송하기 위한 애플리케이션 계층(7계층) 프로토콜**입니다.
    a. 기본적으로는 웹 브라우저와 웹 서버간의 통신을 위해 설계되었습니다. 이는 **수신자(웹 브라우저, 또는 프록시) 측에 의해 요청이 초기화**되는 프로토콜을 의미합니다.

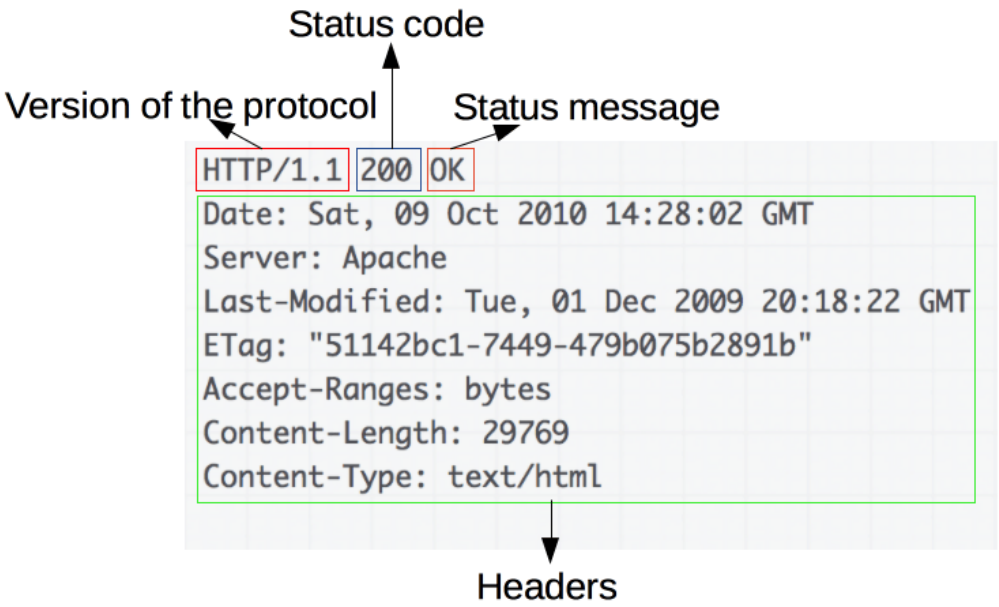


    b. 클라이언트와 서버는 **개별적인 메시지 교환에 의해 통신**합니다. 즉, **클라이언트가 요청(request)하면 서버가 응답(response)**하는 것으로, 대조적인 예로 데이터 스트림을 생각하면 조금 더 이해가 쉽습니다.
    c. 구체적인 흐름은 다음과 같습니다.
        i. TCP 커넥션을 열어 HTTP 메시지(요청, 응답)를 주고받을 준비를 합니다.
        ii. 클라이언트의 요청을 보냅니다. 요청은 아래와 같이 구성되어 있습니다.



1. Method: **클라이언트가 수행하고자 하는 동작**으로, **GET, POST**, OPTIONS, HEAD 등이 있습니다.
2. Path: 가져오려는 **리소스의 경로**입니다.
3. Version of the protocol: **HTTP 프로토콜의 버전**입니다.
4. Headers: HTTP 헤더로 **서버에 대한 추가적인 정보를 전달**합니다.
5. POST 등의 Method를 사용하는 경우에는 전송된 리소스를 포함하여 응답과 유사한 본문을 포함하기도 합니다.

iii. 서버에 의해 전송된 응답을 읽습니다. 응답은 아래와 같이 구성되어 있습니다.



1. Version of the protocol: **HTTP 프로토콜의 버전**입니다.

2. Status Code: **요청에 대한 성공 여부와 그 이유**를 알려 주는 상태 코드입니다.
3. Status Message: **상태 코드의 짧은 설명**을 나타내는 메시지로, 추가적인 영향력은 없습니다.
4. Headers: 요청 헤더와 마찬가지로 **추가적인 정보를 전달**합니다.
5. 요청과 마찬가지로 가져온 리소스가 포함된 본문을 포함하기도 합니다.

iv. 커넥션을 닫거나 다른 요청을 위해 재사용합니다.

d. 이러한 데이터 교환을 위해 가장 일반적으로 사용되는 API는 `XMLHttpRequest API`, `Fetch API`가 있습니다.

2. HTTP 기반 시스템의 구성 요소는 대표적으로 클라이언트, 웹 서버, 프록시가 있습니다. 실제로는 브라우저와 서버 사이에 라우터나 모뎀 등이 있지만, 웹의 계층적 설계로 인해 이들은 숨겨지게 됩니다.

a. 클라이언트(사용자 에이전트)

i. 클라이언트는 **사용자를 대신하여 동작하는 모든 도구**를 의미하며, 대부분 **브라우저**에 의해 수행됩니다.

ii. 브라우저는 항상 요청을 보내는 개체입니다. 웹 페이지를 표시할 때, 브라우저는 HTML 문서를 가져오기 위한 요청을 전송하고, 전송 받은 파일을 분석합니다. 이를 통해 스크립트를 실행하고, 이미지 등의 하위 리소스를 표시하기 위한 추가적인 요청을 가져옵니다. 이후 리소스를 혼합하여 완전한 문서를 표시합니다.

iii. 즉, HTTP 기반 시스템 내에서, 브라우저는 사용자의 **행동을 HTTP 요청으로 변환**하고, **응답을 해석하여 사용자에게 표시**해 주는 역할을 한다고 볼 수 있겠습니다.

b. 웹 서버

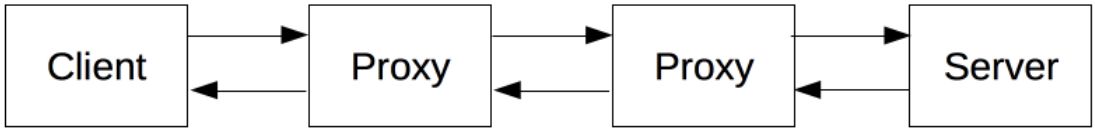
i. 서버는 클라이언트의 요청에 대한 문서를 제공합니다.

c. 프록시

i. **웹 브라우저와 서버 사이**에는 많은 컴퓨터와 기계가 HTTP 메시지를 이어 받고 전달하는데, 프록시는 이러한 컴퓨터 중 **애플리케이션 계층에서 동작하는 것들**을 가리킵니다. (대부분 전송 계층, 네트워크 계층, 물리 계층에서 동작하므로 HTTP 시스템에서는 눈에 보이지 않습니다.)

ii. 프록시는 대표적으로 캐싱, 필터링, 로드 밸런싱, 인증, 로깅 등의 역할을 수행합니다.

1. 캐싱: 공개 또는 비공개 캐시(브라우저 캐시 등)
2. 필터링: 바이러스 백신 스캔, 유해 컨텐츠 차단 등
3. 로드 밸런싱: 여러 서버들이 서로 다른 요청을 처리하도록 허용
4. 인증: 다양한 리소스에 대한 접근 제어
5. 로깅: 이력 정보 저장



3. HTTP는 다음과 같은 특징이 있습니다.
    a. HTTP 메시지는 사람이 읽을 수 있으며 **간단**하게 고안되었습니다.
    b. HTTP 1.0에서 소개된 HTTP 헤더는, 클라이언트와 서버가 헤더의 시멘틱에 대해 합의하면 새로운 기능을 추가할 수 있도록 하여 HTTP의 **확장성**을 높여 주었습니다.
    c. **무상태 프로토콜(Stateless Protocal)**로, 어떠한 이전 요청과도 무관한 각각의 요청을 독립적인 트랜잭션으로 취급하는 통신 프로토콜입니다. 기본적으로는 각 요청들 사이에 연결고리가 존재하지 않지만, HTTP 쿠키를 통해 각각의 요청들에 대한 **세션**을 만듦으로써 동일한 상태나 컨텍스트를 공유할 수 있습니다.
    d. 커넥션은 전송 계층에서 제어되므로 근본적으로는 HTTP 영역 밖이지만, HTTP는 신뢰할 수 있거나 메시지 손실이 없는 커넥션을 요구합니다. 따라서 일반적으로 사용되는 신뢰할 수 있는 전송 프로토콜인 TCP 표준에 의존합니다.
4. HTTP로 제어 가능한 일반적인 기능은 다음과 같습니다.
    a. **문서가 캐시되는 방식을 제어**할 수 있습니다. 서버는 캐시 대상과 기간을 프록시와 클라이언트에 지시할 수 있고, 클라이언트는 저장된 문서를 무시하라고 캐시 프록시에 지시할 수 있습니다.
    b. 스누핑을 포함한 프라이버시 침해를 막기 위해, 브라우저는 기본적으로 웹 사이트 간의 엄격한 분리를 강제합니다. 이는 동일한 origin으로부터 온 페이지만이 웹 페이지의 전체 정보에 접근할 수 있다는 의미입니다. 이러한 제약 사항은 서버에 부담이 되므로, HTTP 헤더를 통해 이러한 **origin 제약 사항을 완화**시킬 수 있습니다. 즉, 다른 도메인으로부터 전달된 정보를 패치할 수 있다는 것입니다.
    c. 특정 사용자만이 페이지에 접근할 수 있도록 **인증**을 강제할 수 있습니다.

d. 서버나 클라이언트는 실제 주소를 숨기는 경우가 있는데, 이러한 경우 HTTP 요청은 **프록시**를 통과하여 전송됩니다. 이러한 과정, 즉 다른 네트워크를 거쳐 요청을 전송하는 것을 **터널링**이라고 합니다.

e. HTTP 쿠키를 통해 서버의 상태를 클라이언트의 요청과 연결할 수 있으며, 이를 통해 무상태 프로토콜임에도 **세션**을 만들 수 있습니다.

HTTP의 진화

1. HTTP/0.9

   a. 1989년, 스위스의 제네바의 실험실에서 파일 교환을 위해 팀 버너스리에 의해 고안되었습니다. 기존의 TCP/IP 프로토콜 위에서, (a)하이퍼텍스트 문서를 표현하기 위한 마크업 언어 **HTML**, (b)문서 교환을 위한 프로토콜인 **HTTP**, (c)문서를 표시하기 위한 **브라우저**, (d)문서에 접근 가능하도록 하는 **HTTPD**로 구성된 월드 와이드 웹(WWW)이 탄생했습니다.

   b. 당시에는 버전 정보가 없었으나 추후 다른 버전과의 구분을 위해 0.9 버전이 부여되었습니다.

   c. 요청은 **단일 라인(One-line Protocol)**으로 구성되었으며, **GET Method만이 존재**했습니다.

   ```
   GET /mypage.html
   ```

   d. 응답은 **파일 내용 자체**로 구성되었습니다.

   ```
   <HTML>
   A very simple HTML page
   </HTML>
   ```

   e. HTML 파일만을 전송 가능했기 때문에, HTTP 메시지에 헤더가 존재하지 않았으며, 상태나 오류 코드가 존재하지 않았습니다. 오류가 생기는 경우에는 해당 파일 내부에 오류에 대한 설명을 덧붙여 전송했습니다.

2. HTTP/1.0

   a. 브라우저와 서버가 융통성을 가질 수 있도록 아래와 같은 기능들이 실험적으로 도입되었습니다.

      i. **버전 정보**가 추가되었습니다.

ii. 응답에 **상태 코드**가 추가되어, 브라우저가 요청의 결과에 따라 다른 동작을 할 수 있게 되었습니다.

iii. **헤더**가 도입되어 메타데이터의 전송이 허용되고, HTTP 프로토콜이 보다 유연하고 확장성 있게 발전되었습니다.

iv. 특히, 헤더의 Content-Type은 HTML 파일 이외의 다른 파일들을 전송할 수 있게 해 주었습니다.

3. HTTP/1.1

a. HTTP/1.0에서 발견된 다양한 문제점들을 개선한 **표준 프로토콜**입니다.

i. **커넥션의 재사용**이 허용되었습니다.

ii. **파이프라이닝**이 추가되어 첫 요청에 대한 응답이 완전히 전송되기 이전에 다음 요청 전송이 가능해졌습니다.

iii. **Chunked Message**(청크 단위로 분리된 메시지)가 지원됩니다.

iv. 추가적인 **캐시 제어 매커니즘**이 도입되었습니다.

v. 클라이언트와 서버 간에 언어, 인코딩, 타입 등의 **컨텐츠에 대한 협상**이 가능해졌습니다.

vi. 동일 IP 주소에 다른 도메인을 호스팅할 수 있게 되었습니다.

b. 이후 HTTPS를 비롯한 다양한 확장이 이루어졌습니다.

i. HTTPS: 기존의 TCP/IP 프로토콜 대신, **암호화된 전송 계층인 SSL(추후 TLS로 발전) 위에서**, 제 3자가 정보를 볼 수 없도록 고안된 HTTP입니다.

1. 433번 포트를 사용합니다.

2. 대칭 키 암호화 방식과 비대칭 키 암호화 방식을 병용합니다.

a. 대칭 키 암호화 방식은 클라이언트와 서버가 동일한 키를 사용해 암호화 및 복호화를 진행하는 방식으로, 연산 속도가 빠르지만 키가 노출되면 위험할 수 있습니다.

b. 비대칭 키 암호화 방식은 1개의 쌍으로 구성된 공개 키와 개인 키를 암호화 및 복호화에 사용하는 방식으로, 키가 노출되어도 비교적 안전하지만 연산 속도가 느립니다.

3. HTTPS는 다음과 같은 흐름을 따라 동작합니다.

a. 클라이언트가 서버로 연결을 시도합니다.

b. 서버가 공개 키(인증서)를 클라이언트에게 전송합니다. (비대칭 키 방식)

c. 클라이언트가 인증서의 유효성을 검사하고 세션 키를 발급하여 보관합니다.

d. 클라이언트가 서버의 공개 키로 세션 키를 암호화하여 서버로 전송합니다.

e. 서버는 개인 키로 암호화된 세션 키를 복호화하여 세션 키를 얻습니다.

f. 이로써 클라이언트와 서버가 동일한 세션 키를 공유하게 되므로 이를 통해 데이터를 전송할 때 빠르게 암호화, 복호화를 진행할 수 있게 됩니다. (대칭 키 방식)

ii. 그 외에도 REST 모델, 웹 소켓, Cross-Origin Resource Sharing 허용 등의 확장이 이루어졌습니다.

4. HTTP/2.0

a. HTTP/1.1 커넥션은 요청이 순서대로 전송되므로, **많은 데이터가 많은 요청을 통해 전송**되는 현대의 웹 페이지에 적합하지 않게 되었습니다. 구글에서는 2010년 전반기에, SPDY 프로토콜을 구현하여 클라이언트와 서버 간의 데이터 교환을 대체할 수단에 대한 실마리를 제공했으며, 이는 HTTP/2.0의 기초로 기여하게 되었습니다.

b. 텍스트 프로토콜이 아닌 **이진 프로토콜**로, 읽기 어렵습니다.

c. **병렬 요청이 동일 커넥션** 상에서 이루어집니다.

d. 전송된 데이터의 **중복이나 불필요한 오버헤드를 제거**하여 연속된 요청 사이에 유사한 내용이 존재하는 헤더를 압축합니다.

e. 서버 푸쉬라는 매커니즘을 통해 **사전에 클라이언트 캐시를 필요한 정보**로 채워 둘 수 있습니다.

f. HTTP/2.0은 2015년에 공식적으로 **표준화**되었습니다.