HTTP 조사 보고서

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 나의 양심과 명예를 지킬 것을 약속합니다.

학　　과 : 융합소프트웨어학부 데이터테크놀로지 전공
과　　목 : 기초 웹 프로그래밍
담당교수 : 권동섭
시　　간 : 화목 12:00 - 13:15
학　　번 : 60181660
이　　름 :　　이지현　　(서명)

HTTP 프로토콜은 TCP와 UDP 프로토콜을 기반으로 하여 웹에서 사용되는 것으로서 클라이언트와 서버 사이에 이뤄지는 요청과 응답 데이터를 전송하는 방법을 말한다.

(1) HTTP 프로토콜의 Request Method 종류와 설명

HTTP 메시지 중 하나인 Request 메시지는 client-server 구조에서 클라이언트가 웹 서버에 파일을 요청할 때 보내는 HTTP 메시지이다. 이 메시지의 구조는 다음과 같다.

<table>
<tr><td>POST/HTTP/1.1</td><td colspan="2">Start-line</td></tr>
<tr><td>Host : localhost : 8000
User-Agent: Mozilla/5.0 (Macintosh; ...) ... Firefox/51.0
Accept: text/html, application/xhtml+xml, ... ,*/*;q=0.8
Accept-Language: en_US, en;q=0.5
Accept-Encoding: gzip, deflate</td><td>Request headers</td><td rowspan="3">Headers</td></tr>
<tr><td>Connection: keep-alive
Upgrade-Insecure-Requests: 1</td><td>General headers</td></tr>
<tr><td>Content-Type: multipart/form-date; boundary=-12656974
Content-Length: 345</td><td>Entity headers</td></tr>
<tr><td>CRLF</td><td colspan="2">Empty</td></tr>
<tr><td>-12656974
(more data)</td><td colspan="2">Body</td></tr>
</table>

Request Method의 종류

- GET

클라이언트가 웹 서버에게 URL에 해당하는 서버의 자료를 요청한다. GET 메소드 사용 시 본문(body)에 해당하는 부분은 비어 있다. URI를 통한 Query String을 다른 페이지로 전송할 때 주로 사용된다. 이때 모든 파라미터는 URL을 통해 전달되며 ?(구분자) 뒤에 오는 값이 파라미터 값으로 여러 개가 올 수 있다. 단, URL을 통해서 정보가 전달되므로 URL형식에 맞지 않거나 길이를 초과할 경우 전달되지 않는다. 이처럼 GET 메소드의 파라미터는 URL에 묻어나오기 때문에 비밀번호나 계좌번호 등 개인정보와 관련된 것들을 다룰 때는 사용하면 안 된다. 예를 들어 게시판 목록을 보는 것은 GET 메소드를 통해서 가능하다.

- HEAD

GET method와 같은 response를 요구하지만, response body는 포함하지 않는다. 즉 HTTP Header 정보만 수신한다. HTTP Header는 클라이언트, 서버 또는 HTTP와 관련된 정보를 담고 있는 General Header, 요청 형식과 서버의 매개 변수인 Request Header, 응답을 보내는 서버에 대한 정보를 담고 있는 Response Header가 있다. 웹 서버의 다운 여부 점검(Health Check)이나 웹 서버의 버전 정보 등을 얻기 위해 사용될 수 있다.

- POST

클라이언트가 HTML Form에 입력한 데이터를 웹서버로 전달할 때 사용되는데, 요청하는 URL 주소에 입력한 정보가 묻어나오지 않아서 보안성이 보장되어야 하는 데이터를 다룰 때 주로 사용된다. 이때 입력한 정보는 글자 수 제한이 없어서 많은 정보를 담을 수 있지만 그만큼 GET 메소드에 비해 처리 속도가 느리다. 글자 수 제한이 없는 대신에 Time Out이 존재하는데 time out에 설정한 시간이 지나도록 아무런 응답이 없다면 에러 페이지가 뜬다. 예를 들어 게시판에 올릴 내용을 작성하여 게시판에 올리는 것은 POST 메소드를 사용하는 것이다.

- PUT

POST 메서드와 비슷한 전송 구조를 가지며 메시지 본문의 내용을 지정된 URL에 저장하는데 이때 저장되는 것은 데이터 전체가 된다. 그래서 PUT 메소드를 사용할 때마다 데이터 전체가 수정되는 것이다. POST와 PUT의 차이는 POST가 정보를 insert한다면 PUT은 update의 개념이다. 따라서 동일한 데이터를 여러 번 POST 메소드를 사용한다면 서버에 아무런 변화도 없지만, PUT을 사용한다면 같은 데이터가 여러 개 생기게 되는 셈이다.

- DELETE

지정된 URL의 리소스를 서버에서 삭제한다. PUT 메소드와 반대되는 개념이다. 안정성 문제로 대부분 서버에서 비활성 모드이다.

- CONNECT

터널링을 목적으로 사용하거나 클라이언트가 원하는 목적지와의 TCP 연결을 HTTP 프록시 서버에 요청할 때 사용된다. 프록시 서버에는 클라이언트를 대신하여 연결을 생성하는데 한 번 생성된 연결은 클라이언트와 서버 사이에 오가는 TCP 스트림을 계속 프록시한다.

터널링 : 가상의 링크를 형성하는 것으로, 하나의 프로토콜이 다른 프로토콜을 감싸는 캡슐화 기능을 통해 운반한다. 일반적으로 터널링은 보안 채널의 역학을 하므로 암호화 기법 적용이 일반적이다.

프록시 서버(proxy server) : 클라이언트가 자신을 통해서 다른 네트워크 서비스에 간접적으로 접속할 수 있게 해주는 컴퓨터 시스템이나 응용 프로그램을 뜻한다. 서버와 클라이언트 사이에 중계기로서 대신 통신을 수행하는 것을 프록시라고 하며 그 중계 기능을 하는 것을 프록시 서버라고 한다. 프록시 서버 중 일부는 요청된 내용을 캐시로 저장해두었다가 캐시 안에 있는 정보를 요구할 때 원격 서버에 접속할 필요없이 프록시 서버의 저장된 캐시를 사용하므로 불필요한 연결이나 전송 시간을 절약할 수 있다. 또한 외부와의 트래픽을 줄이게 됨으로 네트워크 병목 현상을 방지하는 효과도 얻을 수 있다.

- OPTIONS

목적 리소스에 해당하는 통신 옵션들을 설정하기 위해 사용된다. 이때 OPTIONS request 메소드는 entity-Body(Content-Length or Transfer-Encoding을 지정하는 부분)를 포함하는데 이때 반드시 Content-Type field에 대한 정보가 있어야 한다. 또한, URL 자리에 *(asterisk)가 있다면 클라이언트에게 목적지에 해당하는 리소스뿐만 아니라 전체 서버에 대해

서 OPTIONS 메소드를 수행한다.

- TRACE

메시지가 프록시를 거쳐서 최종 목적지에 도착할 때까지의 경로를 기록하는 loop-back 메시지를 호출하기 위해 테스트용으로 사용된다.

- PATCH

PUT 메소드와 유사하게 요청된 자원을 수정(Update의 개념)할 때 사용한다. PUT의 경우에는 데이터 전체를 수정하는 의미지만, PATCH는 해당 자원의 일부를 교체하는 역할을 한다.

(2) HTTP 프로토콜의 Response의 Status의 종류와 설명

Response Status Code에는 여러 종류가 있는데 다음과 같다.

WebDAV(Web Distributed Authoring and Versioning, 웹 분산 저작 및 버전 관리) : 하이퍼텍스트 전송 프로토콜인 HTTP의 확장판으로, WWW에 저장된 문서와 파일을 편집하고 관리하는 사용자들 사이에서의 협업을 용이하게 해준다.

1. 1XX

정보 응답에 대한 코드. 요청을 받았으며 프로세스를 계속함을 알려준다. 이 상태 코드는 상태 라인과 선택적 헤더만을 포함하는 임시의 응답을 나타내고 공백 한 줄에 의해서 끝나는 코드이다. 서버들은 실험적인 상태를 제외하고 클라이언트에게 보내면 안 된다.

1) 100 Continue.

이제까지 보낸 모든 것들이 성공적으로 요청처리가 되었으며, 클라이언트의 요청이 끝났다면 무시해도 되고 혹은 계속해서 요청할 수 있음을 알려준다.

2) 101 Switching Protocol.

이 코드는 클라이언트가 보낸 업그레이드 요청 헤더에 대한 응답이 들어가며 서버에서 프로토콜을 변경할 것임을 알려준다.

3) 102 Processing(WebDAV).

이 코드는 서버가 요청을 수신했고 이를 처리하고 있지만, 아직 제대로 응답을 알려줄 수 없음을 알려준다.

4) 103 Early Hints.

이 코드는 서버가 응답을 준비하는 동안 헤더만 먼저 보내고 응답은 나중에 보낸다. 서버에서 모든 처리가 끝나면 다음 응답을 보낸다.

2. 2XX

성공 응답에 대한 코드. 이 상태 코드들은 클라이언트가 요청한 동작을 수신하여 서버측에서 이해하고 승낙했으며 성공적으로 요청을 처리했음을 알려준다.

1) 200 OK.

요청이 성공적으로 수행되었음을 의미하는데 이는 메소드마다 조금씩 다르게 해석된다. GET 메소드의 경우 클라이언트가 요청한 리소스를 불러와서 메시지 본문에 전송되었다는 의미이고, HEAD 메소드의 경우 header 전체가 메시지 본문에 있다는 것이고, PUT 또는 POST 메소드는 수행 결과에 대한 리소스가 메시지 본문에 전송되었다는 의미이고, TRACE는 메시지 본문이 서버에서 수신한 요청 메시지를 포함하고 있다는 의미이다.

2) 201 Created.

요청이 성공적으로 수행되어 새로운 리소스가 생성되었다는 의미이다. 일반적으로 POST 혹은 PUT 메소드 사용에 대한 응답으로 보내진다.

3) 202 Accepted.

웹 서버가 요청을 수신했지만, 요청에 맞게 아직 처리가 안 된 상태를 의미한다. 이 응답은 요청처리 결과를 이후에 보낸다는 보장은 없다. 이 응답은 다른 프로세스에서 처리하고 있거나 서버가 요청을 다루고 있거나 배치 프로세스를 하고 있는 경우를 위해 만들어졌다.

4) 203 Non-Authoritative Information.

이 코드는 서버가 요청을 성공적으로 처리했으나, 해당 서버가 아닌 다른 리소스에서 받은 정보를 제공하고 있다는 의미이다. 이 코드는 HTTP 프록시에 의해 사용되는데 프록시가 클라이언트에게 원래 상태 코드에 대한 정보를 전달하기 이전에 해당 정보를 바꿀 수 있기에 정확한 응답 상태 코드가 뭔지 모르게 되므로 사용하는 것은 추천하지 않는다.

5) 204 No Content.

PUT, POST, DELETE 요청의 경우에 이 응답 상태 코드가 보내졌다면 요청에 대해서 처리는 성공했으나 전송할 데이터가 없음을 의미한다. 대부분의 API에서는 200 OK와 204 No Content는 같은 것으로 간주한다.

6) 205 Reset Content.

서버가 이 상태 코드를 보냈다면 브라우저는 그 폼(형식)을 리셋할 수 있으며 이는 HTML 폼에서 리셋 버튼과 같은 역할을 한다.

7) 206 Partial Content.

이 응답 코드는 HTTP 클라이언트가 범위 요청을 사용하여 리소스의 일부분만을 요청할 수 있다. 예를 들어 클라이언트가 비디오를 재생하고자 할 때, 전체 비디오 재생이 아닌, 일부분만을 보고자 할 때 클라이언트는 범위 요청을 하고 서버가 이를 근거로 컨텐츠-범위 헤더와 함께 이 코드를 보낸다.

8) 207 Multi-Status(WebDAV).
 이 코드는 다양한 동작(operation)이 발생했으며 각각의 동작은 응답 본문에서 찾을 수 있다. 일반적인 2XX 코드의 의미와는 달리, 이 코드는 필수적으로 동작이 성공했음을 의미하지 않는다.

9) 208 Already Reported(WebDAV).
 이 코드는 클라이언트가 서버에게 같은 바인딩으로 같은 리소스가 이전에 언급되었음을 말해준다. 이것은 무한 루프를 방지하고 데이터가 중복되지 않도록 한다.

10) 226 IM Used.
 이 코드는 특정한 HTTP 프로토콜의 확장에 의해 사용되며 이 확장은 서버가 리소스의 변화를 클라이언트에게 알려준다. 서버가 GET 요청에 대한 리소스의 의무를 다 했고, 그리고 응답이 하나 또는 그 이상의 인스턴스 조작이 현재 인스턴스에 적용되었음을 알려준다.

3. 3XX

재전송에 대한 상태 코드. 리다이렉션이 완료되었음을 의미하며 클라이언트는 요청을 완료하기 위해 추가 동작을 해야 한다.

1) 300 Multiple Choice.
서버가 요청에 대해서 하나 이상의 응답이 가능하다. 사용자는 그중에 반드시 하나를 선택해야 한다. 이때 선택방법에 대한 표준화된 방법은 존재하지 않는다.

2) 301 Moved permanently.
이 응답 코드는 요청한 리소스의 URI가 영구적으로 변경되었음을 의미한다. GET 또는 HEAD 요청에 대한 응답으로 이 코드를 표시하면 새로운 URI로 전달된다.

3) 302 Found.
이 응답 코드는 요청한 리소스의 URI가 일시적으로 변경되었음을 의미한다. 새롭게 변경된 URI는 나중에 만들어질 수 있다.

4) 303 See Other.
클라이언트가 다른 위치에 별도의 GET 요청을 하여 응답을 검색할 경우 서버가 이 코드를 표시한다. 서버가 즉각적으로 요청에 대한 응답을 보내지 않고 클라이언트가 결과를 얻을 수 있는 어떤 곳으로 갈 것으로 지시할 것을 서버 측에서 표시할 때 사용된다.

5) 304 Not modified.
이 코드는 마지막 요청 이후 요청한 페이지가 수정되지 않았을 때 표시된다. GET 또는 HEAD 요청의 조건적으로 응답할 때 사용된다.

6) 305 Use Proxy.
요청한 응답은 반드시 프록시를 통해서 접속해야 하는 것을 알려준다. 서버가 이 응답을 표시했다면 클라이언트가 사용할 프록시를 가리키는 것을 의미하기도 한다. 이 코드는 보안성의 문제로 사용하지 않는 것이 좋다.

7) 306 Switch Proxy.
이 응답 코드는 클라이언트가 이미 프록시를 사용했다면 새로운 프록시를 쓸 것을 알려주는 의미였다. 현재는 보안상의 이유로 사용되지 않고 있다.

8) 307 Temporary Redirect.
클라이언트가 요청한 리소스가 다른 URI에 있어서 일시적으로 요청된 리소스에 대한 다른 위치를 재전송(리다이렉트)한다. 이 코드는 302 Found 응답 코드와 유사하지만, 클라이언트가 반드시 사용된 HTTP 메소드를 변경하지 말아야 한다는 점에서 차이가 있다. 만약 첫 요청에 POST가 사용되었다면 그다음 요청에서도 반드시 POST를 사용해야 한다.

9) 308 Permanent Redirect.

이 응답 코드는 리소스가 영구적으로 다른 URI에 있음을 의미한다. 이것은 301 Moved Permanently 응답 코드와 같은 의미를 가지며 목표 위치에 대해서 정확히 같은 요청을 해야 하지만 301 Moved Permanently 응답 코드에서는 해도 되고 안 해도 된다.

4. 4XX

클라이언트 요청 에러에 대한 상태 코드이다. 에러의 원인이 클라이언트 측에 있음을 의미한다.

1) 400 Bad Request.
사용자의 잘못된 문법으로 인해 서버가 요청을 처리할 수 없을 때 표시된다.

2) 401 Unauthorized.
이 응답 상태 코드는 인증이 필요할 때 표시된다. 예를 들어 서버가 로그인이 필요한 페이지에 대해서 이 코드를 보낼 수 있고, 권한 없음을 의미하는 것이 아닌 인증이 안 되었음을 의미한다.

3) 402 Payment Required.
이 코드는 미래에 디지털 결제 시스템에 사용하기 위해 만들어졌고 결제가 필요할 때 표시된다.

4) 403 Forbidden.
서버가 요청을 거부하고 있을 때 표시된다. 클라이언트가 콘텐츠에 접근할 권리를 가지고 있지 않음을 알려준다. 401 Unauthorized와 다른 점은 서버가 클라이언트가 누구인지 알고 있다는 것이다.

5) 404 Not Found.
서버가 요청받은 리소스를 찾을 수 없을 때 표시된다.

6) 405 Method Not Allowed.
요청한 메소드를 사용할 수 없을 때 표시된다. 예를 들어서 POST 메소드를 사용해야 하는 서버에 GET 메소드를 사용할 경우, 이 코드가 표시된다.

7) 406 Not Acceptable.
이 코드는 클라이언트가 서버가 지원하지 않는 리소스의 구체적인 표시를 요청할 때 표시된다.

8) 407 Proxy Authentication Required.
401 Unauthorized와 비슷하지만 클라이언트가 프록시를 사용하여 인증해야 할 때 표시된다.

9) 408 Request Timeout.
이 응답 코드는 요청을 한지 오래된 연결(즉, 설정된 timeout에 해당하는 시간이 초과된 경우)에 표시된다.

10) 409 Conflict.
이 코드는 서버가 요청을 처리하는 데 충돌이 발생했을 때 표시된다. 이때 서버는 응답할 때 충돌에 대한 정보를 담아야 한다. 클라이언트나 서버는 이 정보를 바탕으로 충돌 문제를 해결할 수 있다.

11) 410 Gone.
서버가 요청한 리소스가 영구적으로 완전히 삭제되었을 때 이 코드를 표시한다. 404 Not Found와 비슷하며 이전에 존재했으나 현재는 없는 리소스에 대해 표시하기 위해 404 코드 대신에 사용되기도 한다. 이동의 의미가 아닌 삭제의 의미이므로 301 Moved permanently와 다르다.

12) 411 Length Required.
서버에서 필요로 하는 컨텐츠 길이 헤더 필드가 정의되지 않은 요청이 들어오면 서버가 이 코드를 보내서 요청을 거절함을 알려준다.

13) 412 Precondition Failed.
클라이언트의 헤더에 있는 전제조건은 서버의 전제조건에 적절하지 않을 때 표시된다.

14) 413 Payload Too Large.
클라이언트가 보낸 요청이 너무 커서 서버 측에서 처리할 수 없을 때 표시되는 코드이다.

15) 414 URI Too Long.
클라이언트가 요청한 URI가 서버에서 처리 가능한 것보다 길 때 표시된다.

16) 415 Unsupported Media Type.
클라이언트가 요청한 미디어 포맷이 서버에서 지원하지 않을 때 표시된다.

17) 416 Requested Range Not Satisfiable.
클라이언트가 요청한 페이지가 서버에서 처리할 수 있는 범위가 아닐 때 표시된다.

18) 417 Expectation Failed.
서버가 Expect 요청 헤더 필드로 요청한 값을 이해하지 못하거나 지원하지 않을 때 표시된다. 클라이언트는 Expect 헤더를 사용해서 서버로부터 특정한 행동을 요구할 수 있다.

19) 418 I`m a teapot.
1998년 IETF(국제 인터넷 표준화 기구, Internet Engineering Task Force)의 만우절 농담으로 시작되었으며 실제로 HTCPCP(Hyper Text Coffee Pot Control Protocol, 하이퍼텍스트 커피포트 제어 규약)으로 정의되었다. 이 규약은 실제로 HTTP를 사용하여 원격으로 커피포트의 상태를 서비스하는 서버의 구현을 염두해 두고 정의되었다. 이러한 개념은 오늘날 사물인터넷의 기반이 되었다. HTTP/1.1 표준 프로토콜에 정식으로 포함된 것은 아니지만 관련 문서는 존재한다.

20) 420 Enhance your calm.
이 상태 코드는 트위터 사에 의해서 만들어진 비공식 확장이다. 트위터 사는 이 코드를 사용함으로써 클라이언트에게 클라이언트의 트래픽 속도가 제한되고 있음을 알려준다. 이러한 새로운 비공식적 상태 코드를 사용하는 것은 일반적으로 피하는 것이 좋다.

21) 421 Misdirected Request.
해당 서버가 요청 리소스를 처리할 수 없을 때 표시된다. 이 코드는 HTTP/2가 도입되면서 포함되었으며 HTTP/1.1 서버에 적용 가능하다. 즉, HTTP/2에서 요청되어야 할 메시지가 HTTP/1.1 서버로 요청될 때 표시된다.

22) 422 Unprocessable Entity(WebDAV).
서버가 올바른 요청 문법에 따른 요청을 이해했으나 컨텐츠가 정확하지 않을 때 표시된다.

23) 423 Locked(WebDAV).
접근하려는 리소스가 잠겨있을 때 표시된다.

24) 424 Failed Dependency(WebDAV).
이전 요청이 실패했기 때문에 지금의 요청도 실패했다는 의미이다. HTTP 응답 상태 라인에는 표시되지 않으며 오직 HTTP 응답 본문에 207 Multi Status 코드를 포함할 때만 표시된다.

25) 426 Upgrade Required.
서버가 클라이언트에게 더 새로운 버전의 서버를 사용하거나 다른 프로토콜을 사용할 것을 말하기 위해 이 코드를 표시한다. 클라이언트는 업그레이드 헤더 필드에 주어진 프로토콜로 요청을 보내야 한다.

26) 428 Precondition Required.
다양한 사용자들이 같은 리소스에 쓰거나 각각 서로의 변화를 오버라이팅(새롭게 입력된 정보로 업데이트되는 것)하는 것을 피하기 위해서는 조건부 요청을 이용하는 것이 좋다. 이때 서버가 If-Match, If-Nome-Match, If-Modified-Since 그리고 If-Unmidified-Since 헤더들을 사용할 것을 클라이언트에게 강요할 때 이 코드를 표시한다.

27) 429 Too Many Requests.
 클라이언트가 제한된 양보다 많은 요청을 보냈을 때 표시된다.

28) 431 Request Header Fields Too Large.
 요청한 헤더 필드가 너무 크기 때문에 서버는 이 요청을 처리하지 못할 때 표시한다.

29) 451 Unavailable For Legal Reasons.
 클라이언트가 요청한 것이 합법적인 것이 아닐 때 서버에서는 이를 거절하고 이 코드를 표시한다.

5. 5XX

서버 오류에 대한 응답 코드. 서버 측에서 요청을 제대로 수행하지 못했을 때 표시된다.

1) 500 Internal Server Error.
서버에 오류가 발생하여 요청을 수행할 수 없을 때 표시된다.

2) 501 Not Implemented.
요청 메소드가 서버에서 지원되지 않으므로 수행할 수 없을 때 표시된다. 또는 서버가 요청 메소드를 인식하지 못할 때도 이 코드가 표시된다.

3) 502 Bad Gateway.
서버가 게이트웨이나 프록시 역할을 하는 중에 업스트림 서버로부터 잘못된 응답을 받았을 때 표시된다.

4) 503 Service Unavailable.
서버가 요청을 처리할 준비가 되지 않았을 때 표시된다. 일반적으로는 유지보수를 위해 작동이 중단되거나 과부하가 걸린 서버다. 이는 현재 서버를 사용할 수 없으나 대부분은 일시적인 상태이다.

5) 504 Gateway Timeout.
이 응답 코드는 서버가 게이트웨이나 프록시 역할을 하고 있으며, 요청을 수행해야 하는 업스트림 서버로부터 제때에 응답을 받지 못했을 때 표시된다.

6) 505 HTTP Version Not Supported.
요청에 사용된 http 버전이 서버에서 지원되지 않을 때 표시된다.

7) 507 Insufficient Storage(WebDAV).
서버에서 요청을 성공적으로 수행할 때 필요한 저장소가 충분하지 않을 때 표시된다.

8) 508 Loop Detected(WebDAV).
서버가 요청을 처리하는 동안 무한 루프를 감지했음을 알려준다.

9) 510 Not Extended.
서버가 요청을 이행하려면 요청에 대한 추가 확장이 필요할 때 표시된다. 예를 들면 클라이언트가 확장된 HTTP 요청 메소드를 사용했으나 서버가 확장된 상태가 아닐 때 이 코드가 표시된다. 엄밀하게는 클라이언트로 인한 에러이므로 4XX에 포함되어야 한다.

10) 511 Network Authentication Required.
이 응답 코드는 클라이언트가 네트워크에 접근하기 위해 인증할 필요가 있을 때 표시된다.

(3) HTTP/2의 특징은 무엇이며, HTTP/1.1과의 차이점은 무엇인가?

HTTP/1.1 는 연결 하나당 하나의 요청과 응답을 처리하기에 동시에 전송하는 것과 많은 리소스를 순차적으로 처리하기 때문에 동시에 처리하는 것이 어렵다. 이러한 특징 때문에 발생하는 문제점은 다음과 같다.

* HOL(Head-Of-Line) Blocking - TCP 프로토콜은 체인처럼 연결된 형태로 순차적으로 데이터를 전송하는데, 첫 번째 패킷이 손실되거나 분실된 경우 그 패킷을 다시 찾거나 서버로부터 재전송을 받기 전까지 다음 패킷이 기다려야 하는 현상을 말한다. 특정 응답의 지연 현상.
* RTT(Round Trip Time)의 증가 - TCP 프로토콜 위에서 동작하는 HTTP의 특성 상, 3-way Handshake가 반복적으로 발생하는데 이는 곧 불필요한 RTT 증가와 네트워크 지연을 초래하여 성능을 떨어뜨린다.
* 무거운 헤더 구조 - HTTP/1.1의 헤더에는 많은 메타 정보들이 저장되어 있는데, 요청시마다 중복되어 헤더값을 전송한다.

이러한 문제점들을 개선하기 위해 나온 것이 HTTP/2이다. HTTP/2에서는 HTTP/1.1과 달리, 한 커넥션에 여러 개의 메시지를 동시에 주고받을 수 있고(multiplexing streams), 서버 푸시를 하며, 헤더 압축을 하여 전송 지연 시간을 획기적으로 감소시켰다. 정보를 전달하는 과정을 음식에 비유하자면 HTTP/1.1이 코스 요리라면 HTTP/2는 한 상 차림인 셈이다. 그만큼 속도 측면에서 확실히 개선되었음을 알 수 있다. 단, 이러한 점은 표준 프로토콜인 HTTP/1.1를 대체하는 것이 아니라 확장한다는 것이다.

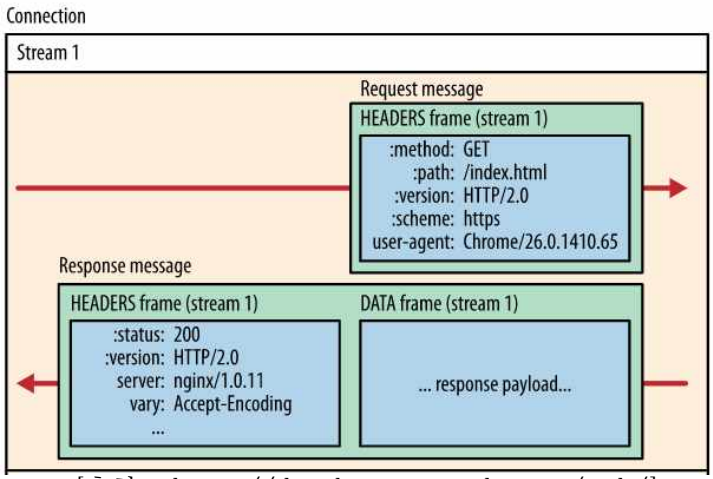
#TCP 3-way Handshake : TCP/IP 프로토콜을 이용해서 통신을 하는 응용프로그램이 데이터를 전송하기 전에 먼저 정확한 정보 전송을 보장하기 위해 상대방 컴퓨터와 사전에 세션을 수립하는 과정을 의미한다. 이러한 절차는 TCP 접속(연결)을 성공적으로 하기 위해 반드시 필요하며 그 과정은 다음과 같다.
  클라이언트 -> 서버 : TCP SYN
  서버 -> 클라이언트 : TCP SYN ACK
  클라이언트 -> 서버 : TCP ACK
handshake : 정보기술과 전기통신 및 관련 분야에서 채널에 대한 정상적인 통신이 시작되기 전에 두 개의 실체 간에 확립된 통신 채널의 변수를 동적으로 설정하는 자동화된 협상 과정이다. 정상적인 정보 전송 이전에 이뤄지며 채널의 물리적인 확립이 따른다.

HTTP/2의 각 특징에 대해 살펴보면 다음과 같다.

- 바이너리 프레이밍 계층(Binary Framing Layer)

HTTP/2에서 가장 중요한 특징으로, HTTP/1.1에서 줄바꿈으로 구분되던 일반 텍스트와 달리 프레임을 단위로 하여 분할되고 바이너리 형식으로 인코딩되어 전달된다. 즉, 정보를 캡슐화하여 전송하는 것이다. HTTP/2에서는 프레임(frame)이 가장 작은 단위가 되고 최소 하나의 프레임 헤더가 포함된다. 요청이나 응답 메시지에 대응되는 프레임의 전체 시퀀스를 메시지라고 하며 이 메시지는 스트림 위에 올라가서 전달된다. 이때 스트림은 양방향성 연결을 갖는다. 다음 그림을 참고하면 프레임, 메시지, 스트림, 커넥션 간의 관계를 파악할 수 있다.

이러한 기능은 HTTP/1.1에서 발생하는 HOL Blocking 문제를 해결할 수 있으며 하나의 커넥션으로도 여러 응답과 요청을 처리할 수 있다. 즉, 하나의 출처(origin)는 하나의 커넥션으로 충분하다. 그 결과 TCP 커넥션을 더욱 효율적으로 사용할 수 있으며 전체적으로 보면 사용되는 커넥션이 감소했으므로 전체 연결 경로에서 메모리와 처리량이 줄어드는 셈이다. 이는 곧 운영 비용의 절감으로 연결된다.



[출처 : https://developers.google.com/web/]

- 다중화(Multiplexing-멀티플렉싱)

바이너리로 인코딩된 프레임들은 특정한 스트림에 속하게 되고 이러한 스트림이 하나의 커넥션 안에서 멀티플렉싱하게 동작하므로 동시에 여러 개의 정보 전달을 수행할 수 있다. HTTP/1.1에서는 여러 병렬 요청을 처리하려며 그만큼 여러 개의 커넥션을 사용해야 했으나, HTTP/2의 멀티플렉싱은 연결 하나당 여러 병렬 요청을 처리하는 것이 가능하다. 이때 HTTP 메시지를 독립된 프레임으로 세분화하고 이 프레임을 인터리빙(교차되어 정리된 것)한 다음, 다른 쪽에 다시 재조립하는 기능은 여러 요청이나 응답을 병렬 처리 가능하며 하나의 커넥션으로 전달할 수 있다. 이러한 기능은 지연 시간을 축소하여 네트워크 용량의 활용성을 개선시킨다.

- **스트림의 우선순위 지정(Stream Prioritization)**

HTTP/1.1에서는 순차적으로 정보를 처리했다면 HTTP/2에서는 프레임들로 구성된 메시지를 담은 스트림의 가중치에 따라 처리된다. 각 스트림은 가중치와 다른 스트림에 대한 명시적 종속성을 가지며 클라이언트가 '우선순위 지정 트리'를 구성하고 통신할 수 있다. 서버는 클라이언트가 보낸 우선순위를 바탕으로 CPU, 메모리 및 기타 리소스의 할당을 제어함으로써 스트림 처리의 우선순위를 지정한다. 응답 메시지를 처리할 때도 해당 요청의 가중치에 따라 우선순위를 지정한다. 단, 클라이언트는 특정 순서로 스트림을 처리하도록 서버에게 강요할 수는 없다.

- **흐름 제어(Flow Control)**

흐름 제어 메커니즘을 사용하면 필요 이상으로 데이터가 크거나 처리 불가능한 데이터를 수신기가 받는 것을 막을 수 있다. 예를 들어서 프록시 서버의 다운스트림 커넥션은 빠르고 업스트림 커넥션이 느린 경우, 프록시 서버가 업스트림 속도에 맞게 다운스트림의 데이터 전달 속도를 조절하여 리소스 사용량을 제어한다. 흐름 제어는 양방향이고 비활성화될 수 없다. 흐름 제어 안의 기본값은 65,535바이트로 설정되지만 데이터가 수신될 때마다 수신기가 window-update frame을 전송하여 창의 최대 크기(2^31-1 바이트)를 설정하고 유지할 수 있다. 흐름 제어는 홉(Hop-by-Hop)방식을 따르는데 각 패킷이 매 노드(또는 라우터)를 건너뛰면서 전달되는 것을 뜻한다. 자체적으로 최적의 경로를 찾아내어 수신 패킷을 다음 라우터에 전달한다.

프록시 서버(proxy) : 클라이언트와 서버 간의 중간 매개체 역할을 한다. 프록시 서버에 요청된 내용들을 캐시를 이용하여 저장해두고 이러한 캐시 안에 담긴 내용을 요청받을 경우, 굳이 서버까지 가지 않더라도 캐시를 전달함으로써 전송 시간을 줄일 수 있고 불필요한 서버와의 연결을 할 필요가 없으므로 효율적이다. 이러한 특징은 원치 않은 사이트를 차단하거나 역으로 IP추적을 당하지 않게 하거나 전달받거나 전달할 데이터를 검사하는 등에 이용될 수 있다.

- **서버 푸시(Server Push)**

HTTP/2가 HTTP/1.1과 다른 점 중에 하나로 서버가 클라이언트 요청에 해당하는 응답뿐만 아니라 여러 응답을 보낼 수 있는 기능을 말한다. 서버 측에서는 클라이언트에게 어떤 리소스가 필요한지 알고 있기에, 클라이언트가 모든 리소스를 확인하여 서버에게 요청하는 과정을 덜어주는 격이다. 즉, 클라이언트가 할 일을 서버가 대신 해주는 셈이다. 서버 푸시 리소스에는 클라이언트에 의해 캐시된 것, 다른 페이지에서 재사용된 것, 다른 리소스와 함께 멀티플렉싱(다중화)된 것, 서버에서 우선 순위가 지정된 것, 클라이언트에 의해 거부된 것 등이 있다. HTTP/2에서 클라이언트는 서버 푸시의 사용 방식을 제어하는데 푸시되는 스트림의 수를 제한하거나 완전히 비활성화시킬 수 있다.

- **헤더 압축(Head Compression)**

HTTP/1.1에서 HTTP 전송 시, 헤더 세트를 함께 전달하는데 이 메타데이터는 항상 일반 텍스트로 전송되며 기본 500-800 바이트의 오버헤드가 추가되는데 쿠키를 사용할 경우 더 커진다. 이 오버헤드를 줄이기 위해 HTTP/2에서는 HPACK 압축 형식을 사용하여 HTTP 메시지의 헤더 부분을 압축한다. 이 방식은 이전에 전송했던 메시지의 헤더 부분과 중복되는 부분은 생략하고 차이가 있는 부분만 전송한다. 이러한 코딩은 Huffman 코딩이라 하며, 이전에 표시된 헤더 필드의 인덱스 리스트를 레퍼런스로 사용하여 이전에 전송된 값을 효율적으로 인코딩할 수 있다.

(4) HTTP/3의 특징은 무엇이며, 이전 버전들과의 차이점은 무엇인가?

HTTP/3은 QUIC 프로토콜을 기반으로 한다. 이전 버전들과의 가장 두드러지는 차이점은 바로 이 전송 프로토콜이다. 기존의 버전들은 모두 TCP 프로토콜 위에서 동작했는데, HTTP/3에서는 QUIC 프로토콜 위에서 동작한다. 그 이유는 기능을 확장하기 위해 TCP의 구조를 살펴봤더니 OPTION 필드의 제한으로 인해 한계가 생겼다. 또한, 이미 오랫동안 사용해 온 헤더 자체가 암호화되지 않은 상태로 전달되다 보니 외부에서 관찰하거나 패킷을 수정할 수 있어서 보안성에 취약해졌다. 라는 것을 이유로 들 수 있다. QUIC 프로토콜은 UDP 위에서 동작하는데 UDP란, User Datagram Protocol의 준말이다. UDP 프로토콜의 전송 방식은 매우 단순해서 정보 전달의 신뢰성이 떨어지는데, 순서가 섞이거나 중복 혹은 손실되기도 한다. 이러한 특징 때문에 일반적으로 오류의 검사나 수정이 필요 없는 곳에서 쓰이는데 그 예로는 DNS 서비스, IPTV, VoIP 등이 있다.

HTTP/3의 특징은 다음과 같다.

1) QUIC 프로토콜을 기반으로 하여 처리 속도가 빠르다.

QUIC 프로토콜의 가장 큰 특징이자 장점은 0-RTT(Round Trip Time)과 1-RTT인데, 이는 새로운 커넥션을 생성하기 위해서 요구되는 시간을 줄이기 위해서 클라이언트가 이전에 연결했던 서버의 커넥션 캐시를 바탕으로 특정한 파라미터를 가져오는 것이다. 이것 덕분에 클라이언트에게 즉시 정보를 보낼 수 있어서 지연 시간이 단축된다. 또한 이것은 UDP 프로토콜의 원칙이므로 UDP 위에서 동작하는 QUIC 프로토콜에서도 적용된다.

반면에, 이전 버전들이 TCP 위에서 동작하면서 커넥션을 새로 만들기 위해 여러 단계의 송수신 과정을 거쳐야 하는데 이 단계를 거치면서 실제 필요한 데이터를 전송하기도 전에 여러 단계에서 왕복이 발생하게 된다. 따라서 커넥션 생성을 위한 오버헤드가 증가하고 지연 시간이 증가한다. 반면에 0-RTT인 HTTP/3에서는 오버헤드가 감소하는 것을 알 수 있다.

오버헤드(overhead) : 어떤 처리를 하기 위해 들어가는 간접적인 처리 시간이나 메모리 등을 말한다.

2) 향상된 보안성.

QUIC은 TLS 1.3 버전의 암호화를 기본적으로 사용하고 있다. 또한, 불안전하거나 암호화되지 않은 곳에서 존재하지 않는다.

TLS(Transport Layer Security) : 컴퓨터 네트워크에 통신 보안을 제공하기 위해 설계된 암호 규약이다. TCP/IP를 사용하는 통신에 적용된다.

3) 이전보다 향상된 멀티플렉싱 및 오류 정정 방법.

멀티플렉싱은 HTTP/2의 가장 큰 특징 중 하나인데, TCP 프로토콜에서는 데이터가 체인처럼 연결되어 있기에 여러 개별 데이터를 하나로 전송할 때, 첫 번째 패킷이 전송 도중에 손실되거나 없어지면 그 패킷 다음에 오는 패킷들은 손실된 패킷이 서버 측에서 다시 보내지거나 다시 찾아질 때까지 기다려야 한다. 이를 Head-Of-Line Blocking이라 하는데, QUIC에서는 같은 상황에서 별다른 블로킹없이 지속적으로 데이터를 처리할 수 있다. 이렇게 데이터를 처리하는 것은 FEC(Forward Error Correction)이라 하며 이전 버전에 비해 향상되었다.

HTTP/3와 HTTP/2의 공통점은 스트림(stream), 서버 푸시, 헤더 압축(이때, HTTP/3은 QPACK를, HTTP/2는 HPACK를 사용한다. 둘다 비슷함.), 멀티플렉싱(multiplexing), 스트림 우선순위 설정 등등이 있다. 두 버전의 차이점을 살펴보면 다음과 같다. HTTP/3은 QUIC 프로토콜 기반이며 스트림을 자체적으로 처리 가능하다. 반면에 HTTP/2는 TCP 프로토콜 기반이며 HTTP 계층에서 스트림을 다룬다.