2023년 2월 25일
HTTP
보고서

# HTTP

임원호

-목차-



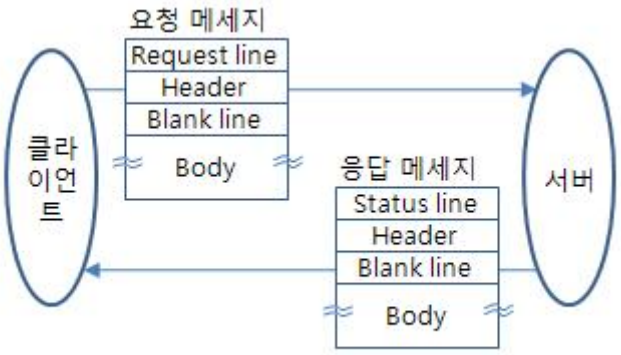
[참고 문헌]
---

# Ⅰ. HTTP

## 1. HTTP 정의

HTTP는 HyperText Transfer Protocol의 준말로 웹에서 클라이언트와 서버의 정보가 교환되는 방법을 관리하는 규칙 및 표준의 집합이다. Web의 기본 프로토콜로 HTTP 프로토콜을 사용하여 원하는 리소스와 전송 방법을 지정한다. 그런 다음, 웹서버는 HTTP 프로토콜에 따라 요청된 리소스로 응답한다. HTTP는 클라이언트-서버 아키텍처를 사용하는데, 이는 클라이언트가 웹서버에 요청을 시작하고, 요청된 리소스로 응답한다는 것을 의미하게 된다. HTTP는 또한 Stateless, Connectionless이며, 서버는 요청 간 클라이언트에 대한 정보를 기억하지 않는다. 다만 웹 애플리케이션에서 쿠키 및 기타 기술을 사용해 사용자의 상태나 정보를 저장하고, 유지하며 사용자에게 서버가 통신을 유지하는 듯한 대화형 환경을 제공할 수 있다.

## 2. HTTP 요청 메소드




웹 통신에서 클라이언트와 서버가 서로 상호작용을 하기위한 HTTP의 Request Method는 많이있지만, 그 중 가장 일반적으로 사용하는 요청 메소드는 GET, POST, PUT, DELETE, HEAD, OPTIONS, PATCH가 존재한다.

1. GET : GET 메소드는 웹 서버에서 웹 페이지 또는 이미지와 같은 리소스를 검색하는 데 사용된다. 요청은 일반적으로 요청되는 리소스를 식별하는 URL를 포함한다.

2. POST : POST는 일반적으로 양식 제출과 같이 사용자의 입력의 형태로 웹 서버에 데이터를 제출하는데 사용된다. 요청에는 일반적으로 KEY-VALUE 쌍의 형태로 제출되는 데이터가 포함된다.

3. PUT : PUT 메소드는 웹 서버의 리소스를 업데이트하는 데 사용된다. 요청에는 업데이트 중인 리소스에 대한 업데이트된 데이터가 포함된다.

4. DELETE : DELETE는 웹 서버에서 리소스를 삭제하는 데 사용이된다.

5. HEAD : HEAD 메소드는 GET 메소드와 유사하지만 실제로 리소스 자체를 검색하지 않고 컨텐츠 유형 및 컨텐츠 길이와 같은 리소스의 헤더 정보만 요청한다.

6. OPTIONS : OPTIONS 메소드는 웹 서버의 리소스에 사용할 수 있는 통신 옵션에 대한 정보를 검색하는 데 사용된다.

7. PATCH : PATCH는 웹 서버의 리소스를 부분적으로 업데이트하는 데 사용된다. 요청에는 리소스에 대해 수행해야 하는 특정 변경 사항이 포함된다.

모든 HTTP 요청 메소드는 보안 문제에 잠재적으로 취약할 수 있지만, 일부 메소드는 다른 메소드보다 보안에 취약하다.

1. GET : GET은 웹 서버에서 리소스를 검색하는데 사용이 되므로, XSS 공격 및 SQL 주입 공격과 같은 다양한 유형의 공격에 취약할 수 있다. 악의적인 사용자가 URL에 자신의 코드를 삽입할 수 있는 경우 발생하게 된다.

2. POST : POST는 CSRF 또는 MITM 공격과 같은 다양한 유형의 공격에 취약할 수 있다. 이런 공격은 악의적인 사용자가 POST 요청에서 전송되는 데이터를 가로채거나, 조작할 수 있는 경우 발생할 수 있다.

3. PUT, DELETE : PUT과 DELETE 메소드는 웹 서버의 리소스를 수정하거나, 삭제하는데 사용되어서 보안이 제대로 설정이 되어있지 않다면, 공격에 취약하다. PUT 메소드를 사용해 악의적인 코드를 서버에 업로드 하거나 DELETE 메소드를 이용해 서버의 중요한 리소스를 삭제할 수 있다.

오늘날의 HTTP의 발전으로 웹 서버와 웹 클라이언트의 절대적인 통신 규약이 생기게 되었다. 만약 HTTP가 존재하지않았다면, 각 통신 프로토콜을 따로 만들었을 것이고, 지금보다 훨씬 복잡해 졌을 것이다.

## 3. HTTP 응답 코드

HTTP 응답 코드는 요청된 리소스 또는 페이지의 상태를 나타내기 위해 웹 서버에서 반환되는 세자리 숫자이다. 응답 코드는 HTTP 응답의 응답 헤더에 포함되어있으며, HTTP 응답 메시지의 일부로 클라이언트에게 전송이된다. HTTP 응답 코드에는, 5개의 범주가 있으며, 응답 코드의 첫 번째 숫자로 표시가 된다. 정보(1xx), 성공(2xx), 리디렉션(3xx), 클라이언트 오류(4xx), 서버 오류(5xx)가 존재한다.

1. 정보(1xx) : 서버가 요청을 수신하고, 처리하고 있지만, 응답이 완료되지 않았을 때 표시가 된다.

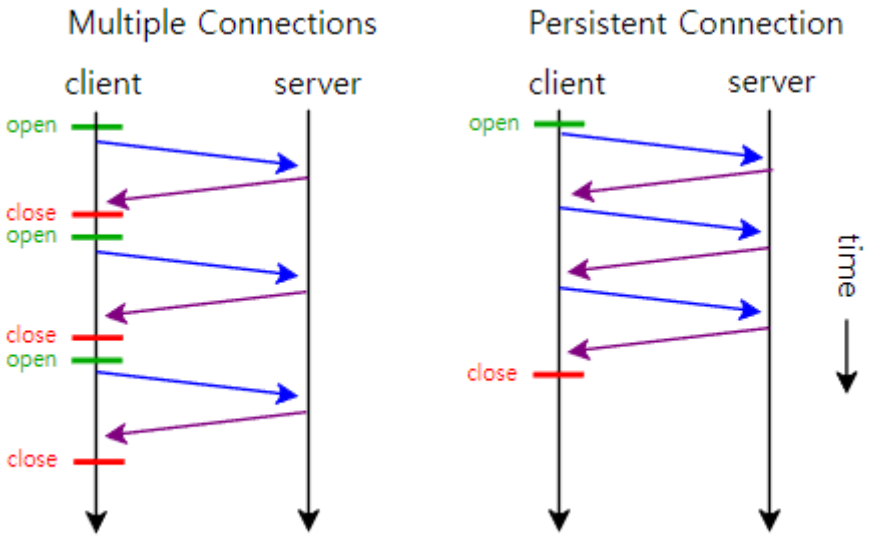
2. 성공(2xx) : 서버가 요청을 성공적으로 처리하고 요청된 리소스를 반환했음을 나타낸다.

3. 리디렉션(3xx) : 요청된 리소스가 다른 위치로 이동 또는 리디렉션 되었음을 나타낸다.

4. 클라이언트 오류(4xx) : 잘못된 URL 또는 필수 매개 변수 누락과 같은 요청에 오류가 있었음을 나타낸다.

5. 서버 오류(5xx) : 서버 측에서 시간 초과 또는 서버 구성 문제와 같은 오류가 있었음을 나타낸다.

# 4. Keep - Alive




Http는 Stateless(3way-handshake를 통해 요청을 보낸다. 하지만 이게 비효율적이라 생각해, perisistent connection을 사용한다. 여기서 keep-alive를 사용한다. 하지만 keep - alive 기능을 사용하게 되면, 쓰레드를 유지해야되기 때문에 자원이 낭비될 수 있다.)이며, Connection(서버가 클라이언트 요청에 대해 응답을 마치면 맺었던 연결을 끊는다. 해결방법으로 쿠키, 세션, JWT가 존재한다.) less이다. 즉 클라이언트와 서버가 계속해서 연결되고 있는것이 아닌, 받을 것 받고, 줄건 주고나서 연결을 끊는다. 하지만, 이런 비연결성은 단점 또한 가지고 있게 된다. 인터넷에서 모든 데이터를 보내는데, 생각보다 많은 비용을 지불한다는 것이다. 그래서 생기게된 기능이 Keep - Alive이다.

Keep - Alive는 단일 TCP 연결을 통해 여러 요청과 응답을 전송할 수 있는 HTTP 헤더이다. 기본적으로 HTTP는 위와 같이 비연결성, 무상태이다. 이 경우 계속해서 클라이언트와 서버가 모든 요청과 응답에 대해 새로운 TCP 연결을 설정해야하고, 단일 사용자 작업에 대해 여러 요청을 수행해야 한다. 이는 상당한 오버헤드를 추가하고 웹 애플리케이션의 성능을 저하시킬 수 있게 된다. Keep - Alive를 사용하게 된다면, 클라이언트와 서버 간의 TCP 연결은 첫 번째 요청 및 응답 후 에도 열려 있으며, 이후 요청 및 응답은 동일한 연결속에서 진행하게 된다. 이 때 헤더는 서버가 연결을 닫기 전에 연결을 열어 두어야 하는 최대 시간을 지정해야하며, 이 시간 동안 요청이 없다면, 연결이 닫히게 된다. 또한 헤더는 연결이 닫히기 전에 연결을 통해 전송할 수 있는 최대 요청 수를 지정할 수 있습니다.

Keep - Alive는 새로운 TCP연결과 관련된 오버 헤드를 줄이면서 성능을 높을 수 있지만, 잠재적인 단점 또한 존재한다. 서버 리소스 사용량 증가, 연결 시간 초과, 서비스 거부 공격 위험 증가, 복잡성 증가, 일부 네트워크 인프라와 호환되지 않음이 있다.

1. 서버 리소스 사용량 증가 : TCP 연결을 장시간 열어 두면 메모리 및 CPU 사용률을 비롯한 서버의 자원 사용량이 증가하게 된다. 이는 한 번에 여러 개의 활성 연결이 있는 트래픽이 많은 웹 응용 프로그램에서 문제가 발생할 수 있다.

2. 연결 시간 초과 : 경우에 따라 네트워크 문제 또는 서버 시간 초과로 인해 Keep - Alive 연결이 조기에 종료될 수 있다. 이로 인해 연결이 끊어지고, 사용자의 응답 시간이 느려질 수 있다.

3. 서비스 거부 공격 위험 증가 : TCP 연결을 장시간 열어 두면 Dos 공격 위험이 증가하며, 서버에 트래픽이 증가할 경우 응답하지 못할 수 있다.

4. 복잡성 증가 : Keep - Alive를 사용하려면, 더 복잡한 서버 및 클라이언트 측 구현이 필요하므로 구성 오류 및 기타 성능 및 보안에 영향을 미칠 수 있는 문제의 위험이 증가 할 수 있다.

5. 일부 네트워크 인프라와 호환되지 않음 : 경우에 따라 네트워크 인프라에서 Keep - Alive 연결을 지원하지 않을 수 있으며, 이로 인해 연결 오류 및 기타문제가 발생할 수 있다.