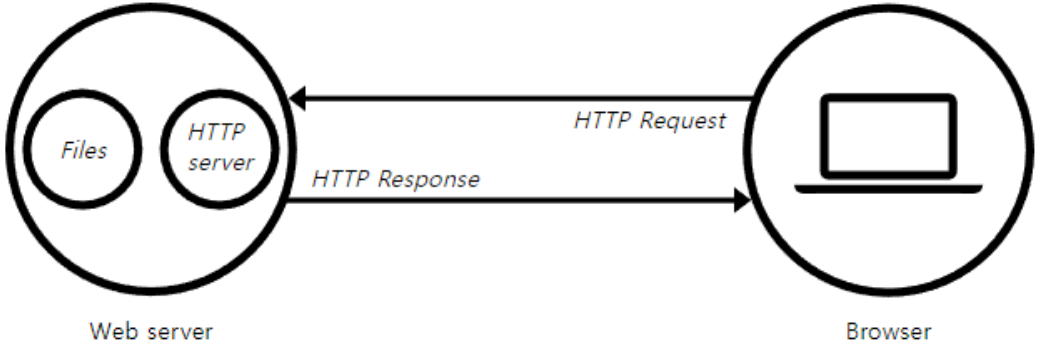
05 웹 서버

CodeDiary18(codediary18@gmail.com)

2022.03.01

목차

5.1 다채로운 웹 서버



✓ 웹 서버란?

- **HTTP 요청을 처리하고 응답을 제공**
  - 웹 서버라는 용어는 **소프트웨어**와 웹 페이지 제공에 **특화된 장비(컴퓨터)** 양쪽 모두를 가리킴
- 기능, 형태, 크기가 다양
- 모든 웹 서버는 리소스에 대한 HTTP 요청을 받아서 콘텐츠를 클라이언트에게 제공

5.1 다채로운 웹 서버

✓ 웹 서버 구현

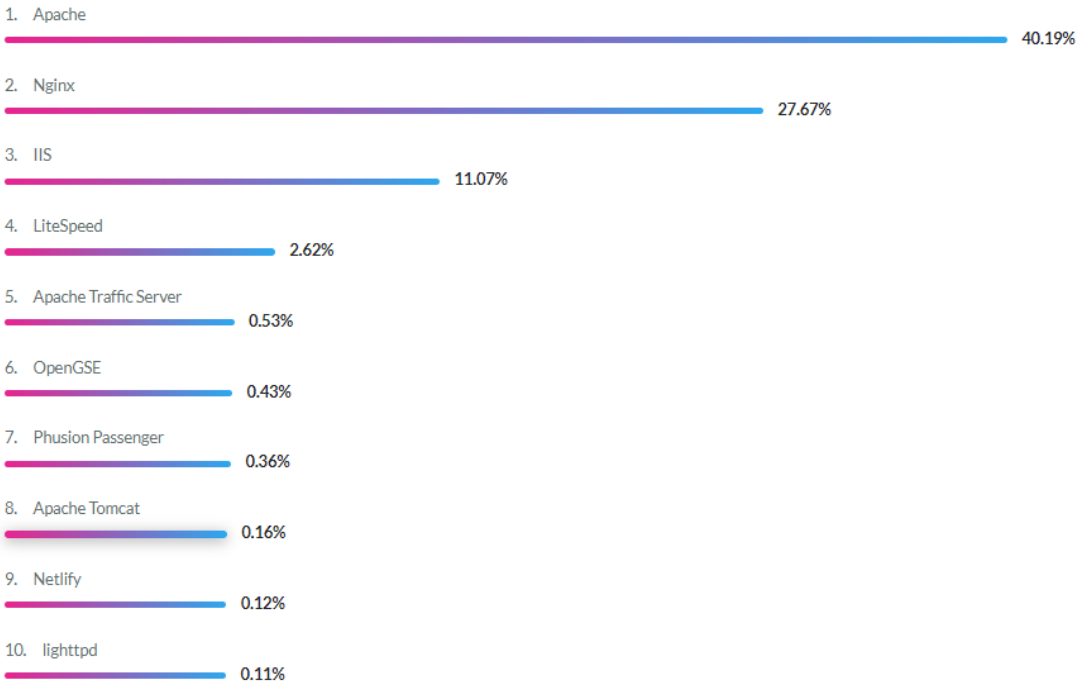
- HTTP 및 그와 관련된 TCP 처리를 구현
- 웹 리소스를 관리하고 웹 서버 관리 기능을 제공
- TCP 커넥션 관리의 책임을 운영체제와 나눠 가짐
  - 운영체제
    컴퓨터 시스템의 하드웨어를 관리하고 TCP/IP 네트워크 지원, 웹 리소스를 유지하기 위한 파일 시스템, 현재 연산 활동을 제어하기 위한 프로세스 관리를 제공

5.1 다채로운 웹 서버

- ✓ 다목적 소프트웨어 웹 서버
  - 네트워크에 연결된 표준 컴퓨터 시스템에서 동작
  - 거의 모든 컴퓨터와 운영체제에서 동작

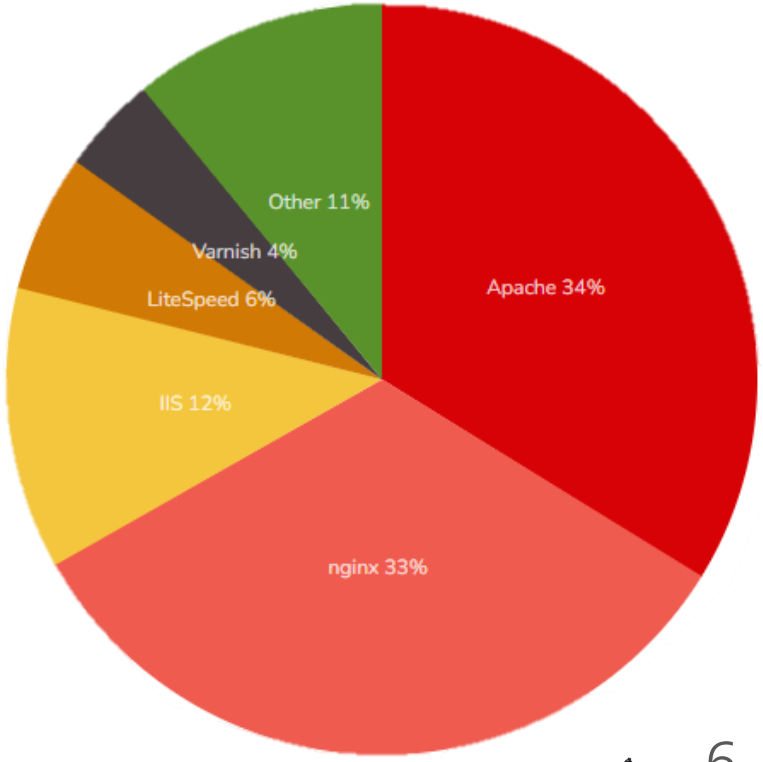
5.1 다채로운 웹 서버

<2022 글로벌 웹 서버 시장 점유율>



출처 : hostadvice

<상위 100만개 사이트의 웹 서버 사용 분포>



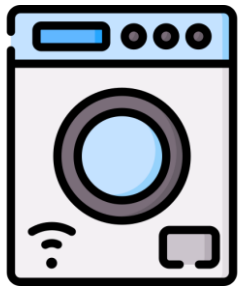
출처 : https://trends.builtwith.com/web-

5.1 다채로운 웹 서버

✓ 임베디드 웹 서버

- **일반 소비자용 제품에 내장**될 목적으로 만들어진 작은 웹 서버
  - ex) 프린터, 가전 제품

- 사용자가 그들의 일반 소비자용 기기를 간편한 웹 브라우저 인터페이스로 관리 가능

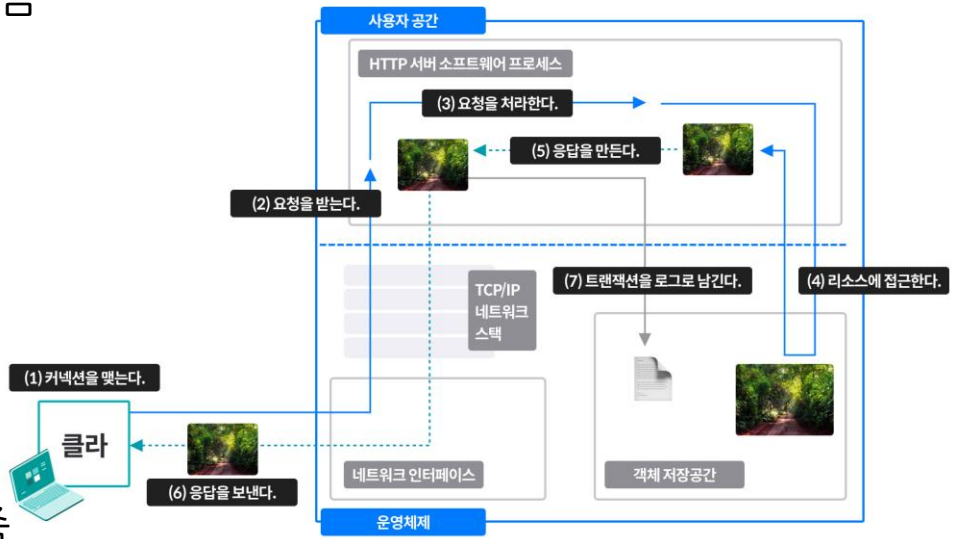


5.2 간단한 펄 웹서버

- ✓ 완전한 기능을 갖춘 HTTP 서버를 만들고자 한다면 해야 할 일이 많음
  - 아파치 웹 서버 코어는 50,000줄이 넘는 코드로 구성되어 있으며, 부가적인 처리 모듈들을 더하면 훨씬 더 커짐
- ✓ **HTTP/1.1**의 기능들을 지원하려면, **풍부한 리소스 지원, 가상 호스팅, 접근 제어, 로깅, 설정, 모니터링, 그 외** 성능을 위한 기능 필요
- ✓ 최소한으로 기능을 하는 HTTP 서버라면 30줄 이하의 펄(Perl) 코드로도 만들 수 있음

5.3 진짜 웹 서버가 하는 일

1) **커넥션을 맺기** : 클라이언트의 접속을 받아들이거나, 원치 않는 클라이언트라면 닫음
2) **요청 메시지 수신** : HTTP 요청 메시지를 네트워크로부터 읽어 들임
3) **요청 처리** : 요청 메시지를 해석하고 행동을 취함
4) **리소스의 매핑과 접근** : 메시지에서 지정한 리소스에 접근
5) **응답을 만듬** : 올바른 헤더를 포함한 HTTP 응답 메시지를 생성
6) **응답을 보냄** : 응답을 클라이언트에게 돌려줌
7) **트랜잭션을 로그에 남김** : 로그파일에 트랜잭션 완료에 대한 기록을 남김



5.4 단계 1:클라이언트 커넥션 수락

- ✓ 클라이언트가 이미 서버에 대해 **지속적 커넥션**을 갖고 있다면,
  클라이언트는 요청을 보내기 위해 **그 커넥션을 사용**할 수 있음

- ✓ 그렇지 않다면, 클라이언트는 서버에 대한 **새 커넥션**을 열어야 함

5.4 단계 1:클라이언트 커넥션 수락

1. 새 커넥션 다루기

- 커넥션 순서
  1. 클라이언트가 웹 서버에 TCP 커넥션을 요청
  2. 웹 서버는 커넥션을 맺음
  3. TCP 커넥션에서 IP 주소를 추출하여 어떤 클라이언트가 있는지 확인
- 커넥션이 맺어지면 커넥션을 목록에 추가, 데이터를 지켜보기 위한 준비
- 웹 서버는 어떤 커넥션이든 마음대로 거절하거나 즉시 닫을 수 있음
  - ex) 클라이언트의 IP 주소나 호스트명이 인가되지 않았거나 악의적이라고 알려진 경우

5.4 단계 1:클라이언트 커넥션 수락

2. 클라이언트 호스트 명 식별
    - 대부분의 웹 서버는 **역방향 DNS**로 클라이언트의 IP 주소를 클라이언트의 호스트 명으로 변환
    - 클라이언트의 호스트 명을 구체적인 접근 제어와 로깅을 위해 사용

5.4 단계 1:클라이언트 커넥션 수락

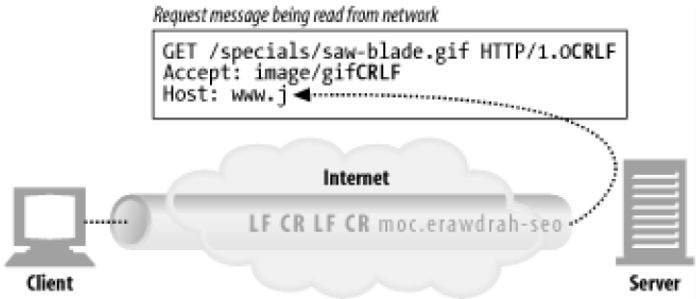
3. ident를 통해 클라이언트 사용자 알아내기

- ident 프로토콜을 이용해 서버에게 어떤 사용자 이름이 HTTP 커넥션을 초기화했는지 찾을 수 있음
- 공공 인터넷에서는 잘 동작 하지 않음
  - 많은 클라이언트 PC는 ident 신원확인 프로토콜 데몬 소프트웨어를 실행X
  - ident 프로토콜은 HTTP 트랜잭션을 유의미하게 지연
  - 방화벽이 ident 트래픽이 들어오는 것을 막는 경우가 많음
  - ident 프로토콜은 안전하지 않고 조작하기 쉬움
  - Ident 프로토콜은 가상 IP 주소를 잘 지원하지 않음
  - 클라이언트 사용자 이름의 노출로 인한 프라이버시 침해의 우려
- 웹 서버 로깅에서 유용
  - 일반 로그 포맷의 두 번째 필드는 각 HTTP 요청의 사용자 이름을 담음

5.5 단계 2:요청 메시지 수신

✓ 웹 서버가 요청 메시지를 파싱할 때 하는 일

- 요청줄을 파싱하여 요청 **메서드**, 지정된 리소스의 식별자(**URI**), **버전 번호**를 찾으며, 각 값은 스페이스 하나로 분리, 요청줄은 CRLF 문자열로 끝남
- 메시지 헤더를 읽으며, 각 메시지 헤더는 CRLF로 끝남
- 헤더의 끝을 의미하는 CRLF로 끝나는 빈 줄을 찾아냄
- 요청 본문이 있다면 읽어 들임(길이는 Content-Length 헤더로 정의)



5.5 단계 2:요청 메시지 수신

- ✓ 요청 메시지를 파싱할 때, 웹 서버는 입력 데이터를 네트워크로부터 불규칙적으로 받으며, 커넥션은 언제든 무효화 가능

- ✓ 웹 서버는 파싱해서 이해하는게 가능한 수준의 분량을 확보할 때까지 데이터를 읽어서 메시지 일부분을 메모리에 임시로 저장해 둘 필요가 있음

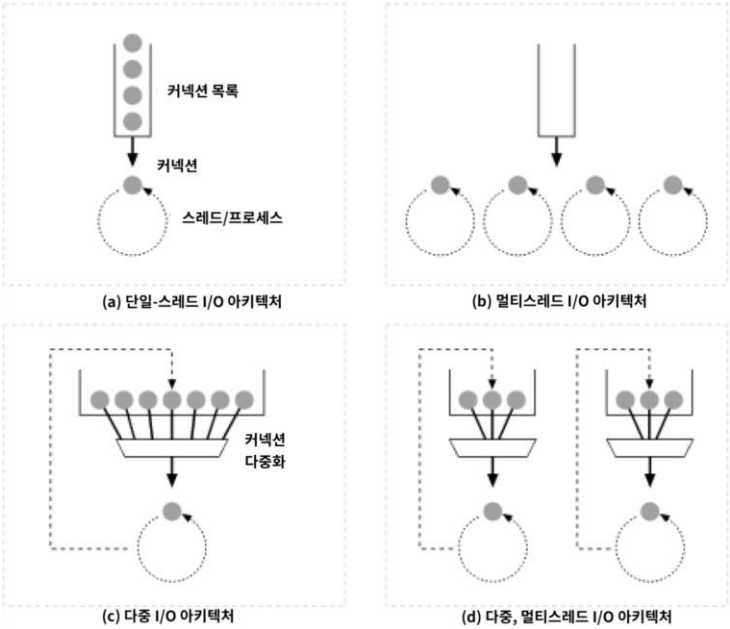
5.5 단계 2:요청 메시지 수신

- ✓ 메시지의 내부 표현
  - 요청 메시지를 쉽게 다룰 수 있도록 내부의 자료 구조에 저장
    - 요청 메시지의 각 조각에 대한 포인터와 길이
    - 헤더는 속도가 빠른 룩업 테이블에 저장되어 각 필드에 신속하게 접근

5.5 단계 2:요청 메시지 수신

✓ 커넥션 입력/출력 처리 아키텍처

- 고성능의 웹 서버는 수천 개의 커넥션을 동시에 지원
- 어떤 커넥션의 요청은 느리거나 드물게 들어오고, 또 어떤 커넥션들은 조용히 대기하거나 급속히 웹 서버로 요청을 보낼 수 있음
- 웹 서버의 아키텍처의 차이에 따라 요청을 처리하는 방식도 다양
  a. 단일 스레드 웹 서버
  b. 멀티프로세스와 멀티스레드 웹 서버
  c. 다중 I/O 서버
  d. 다중 멀티스레드 웹 서버

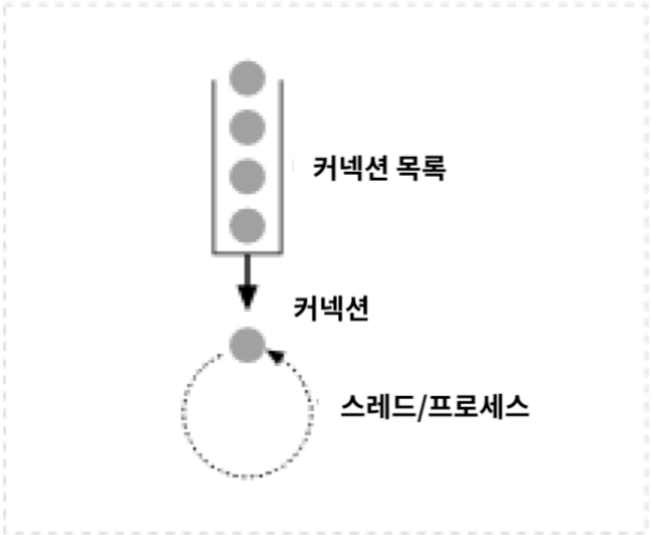


5.5 단계 2:요청 메시지 수신

✓ 단일 스레드 웹 서버

- 한 번에 하나씩 요청을 처리하며, 트랜잭션이 완료되면 다음 커넥션이 처리

- 구현은 간단하지만 처리 도중에 다른 커넥션은 무시됨 → 이는 심각한 성능 문제를 야기하기때문에 부하가 적은 서버에 적당



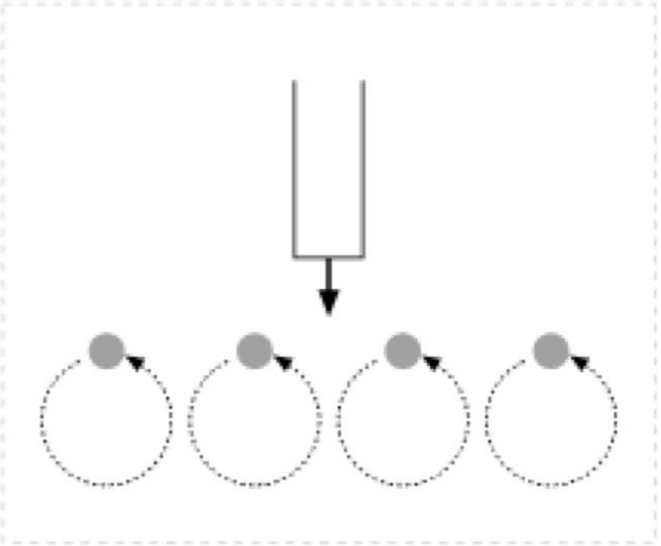
(a) 단일-스레드 I/O 아키텍처

5.5 단계 2:요청 메시지 수신

✓ 멀티프로세스와 멀티스레드 웹 서버

- 여러 요청을 동시에 처리하기 위해 여러 개의 프로세스 혹은 고효율 스레드를 할당

- 서버가 수백, 수천 개의 커넥션을 처리하면 많은 메모리나 시스템 리소스를 소비
  → 대부분의 멀티프로세스와 멀티스레드 웹 서버는 최대 개수에 제한을 둔다

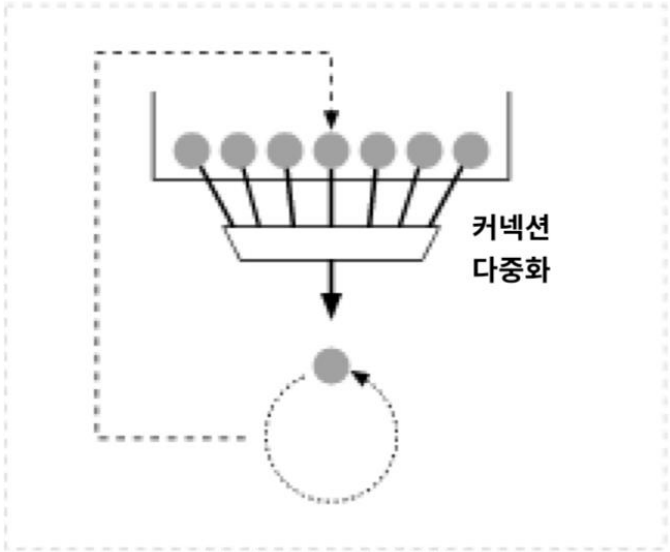


(b) 멀티스레드 I/O 아키텍처

5.5 단계 2:요청 메시지 수신

✓ 다중 I/O 서버

- 대량의 커넥션을 지원하기 위해서 다중 아키텍쳐를 채택
- 모든 커넥션은 동시에 그 활동을 감시 당함
- 커넥션의 상태가 바뀌면 그 커넥션에 대해 작은 양의 처리가 수행되며 처리가 완료되면, 다음 상태 변경을 위해 열린 커넥션 목록으로 이동
- 스레드와 프로세스는 유휴 상태의 커넥션에 매여 기다리느라 리소스를 낭비하지 않음



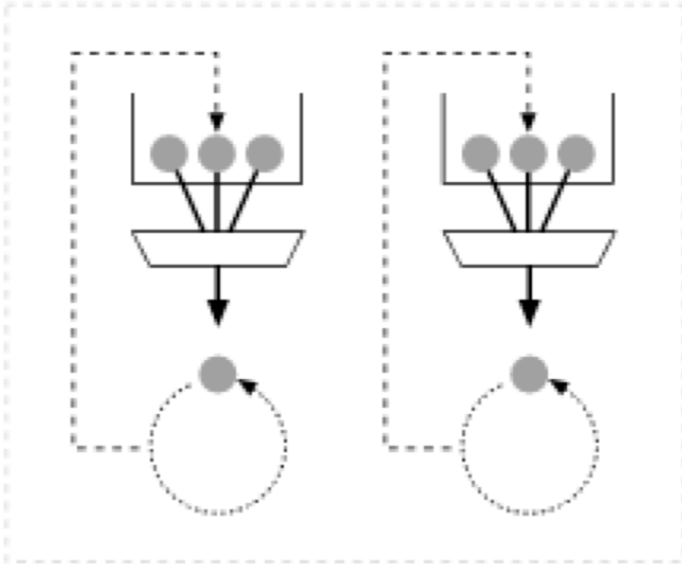
(c) 다중 I/O 아키텍처

5.5 단계 2:요청 메시지 수신

✓ 다중 멀티스레드 웹서버

- CPU 여러 개의 이점을 살리기 위해 시스템에서 멀티스레딩과 다중화를 결합

- 여러 개의 스레드는 각각 열려있는 커넥션을 감시하고 각 커넥션에 대해 조금씩 작업을 수행



(d) 다중, 멀티스레드 I/O 아키텍처

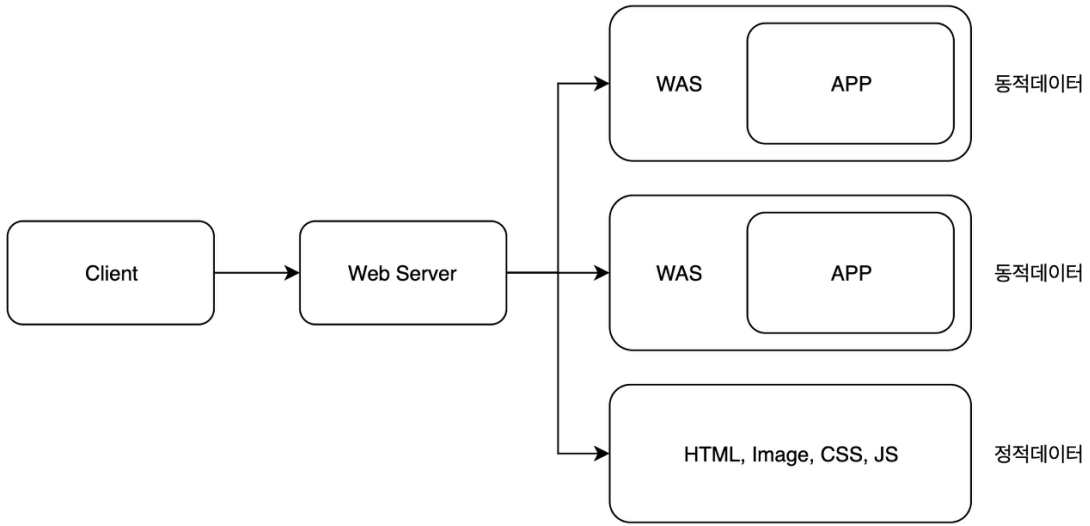
5.6 단계 3:요청 처리

- ✓ 웹 서버는 요청으로부터 메서드, 리소스, 헤더, 본문을 얻어내어 처리
  - 메서드에 따라 본문이 없을 수 있음
  - POST를 비롯한 몇몇 메서드는 요청 메시지에 엔터티 본문이 있을 것을 요구
  - 그 외 OPTIONS를 비롯한 다수의 메서드는 요청에 본문이 있는 것을 허용하되 요구하진 않음
  - 많지는 않지만 GET과 같이 요청 메시지에 엔터티 본문이 있는 것을 금지하는 메서드 존재

5.7 단계 4:리소스의 매핑과 접근

✓ 웹 서버는 리소스 서버

- 정적이거나 동적인 컨텐츠를 제공하며, 클라이언트에게 전달하기 전에는 요청 메시지의 URI에 대응하는 컨텐츠를 찾아서 식별해야 함



5.7 단계 4:리소스의 매핑과 접근

✓ Docroot

- 리소스 매핑의 가장 단순한 형태는 요청 URI를 웹 서버의 파일 시스템 안에 있는 파일 이름으로서 사용
- 일반적으로 웹 서버 파일 시스템의 특별한 폴더를 웹 컨텐츠를 위해 예약하며, 이 폴더를 문서 루트 혹은 docroot라고 함
- 웹 서버는 상대적인 URL이 docroot를 벗어나 docroot 이외 부분이 노출되는 일이 생기지 않도록 주의해야 함
  - http://www.joes-hardware.com/../
    문서 루트 위의 파일을 보려고 하는 URI 허용 막아야 함

5.7 단계 4:리소스의 매핑과 접근

✓ 가상 호스팅 된 docroot

- 각 사이트에 분리된 문서 루트를 주는 방법으로 하나의 웹 서버에서 여러 개의 웹 사이트를 호스팅

- 가상 호스팅 웹 서버는 URI나 Host 헤더에서 얻은 IP주소나 호스트 명을 이용해 올바른 문서 루트를 식별
  → 하나의 웹 서버 위에서 두 개의 사이트가 완전히 분리된 콘텐츠를 갖고 호스팅 되도록 가능

5.7 단계 4:리소스의 매핑과 접근

- ✓ 사용자 홈 디렉터리 docroot
  - 사용자들이 한 대의 웹 서버에서 각자의 개인 웹 사이트를 만들 수 있도록 해주는 것
  - 보통 빗금(/)과 물결표(~) 다음에 사용자 이름이 오는 것을 시작으로 URI는 그 사용자의 개인 문서 루트를 가리킨다

5.7 단계 4:리소스의 매핑과 접근

✓ 디렉터리 목록

- 클라이언트가 웹 서버에 디렉터리 URL을 요청할 때 웹 서버는 아래와 같은 몇가지 행동을 취할 수 있음
  - 에러를 반환
  - 디렉터리 대신 특별한 '색인 파일 ' 을 반환
  - 디렉터리를 탐색해서 내용을 담은 HTML 페이지를 반환

5.7 단계 4:리소스의 매핑과 접근

✓ 동적 콘텐츠 리소스 매핑

- 웹 서버는 URI를 동적 리소스에 매핑할 수 있음
  → 즉, 요청에 맞게 콘텐츠를 생성하는 프로그램에 URI를 매핑하는 것

- 웹 서버들 중에서 애플리케이션 서버라 불리는 것들은 웹 서버를 복잡한 백엔드 애플리케이션과 연결

- 오늘날의 애플리케이션 서버는, 마이크로소프트의 액티브 서버 페이지와 자바 서블릿과 같은 한층 더 강력하고 효과적인 서버사이드 동적 콘텐츠를 제공

5.7 단계 4:리소스의 매핑과 접근

- ✓ 서버사이드 인클루드(Server-Side Include, SSI)
  - 어떤 리소스가 서버사이드 인클루드를 포함하도록 설정되어 있다면 서버는 그 리소스의 콘텐츠를 클라이언트에 보내기전에 처리
  - 서버는 콘텐츠에 변수 이름이나 내장된 스크립트가 될 수 있는 어떤 특별한 패턴이 있는지 검사하고 특별한 패턴은 변수 값이나 실행가능한 스크립트의 출력 값으로 치환

5.7 단계 4:리소스의 매핑과 접근

- ✓ 접근제어
  - 웹 서버는 각각의 리소스에 접근 제어를 할당할 수 있음
  - 제어되는 리소스에 대해 요청이 도착하면, 웹 서버는 클라이언트의 IP주소에 근거하여 접근을 제어하거나 비밀번호를 요구할 수 있음

5.8 단계 5:응답 만들기

- ✓ 서버가 리소스를 식별하면, 요청 메서드로 서술되는 동작을 수행한 뒤 응답 메시지를 반환

- ✓ 응답 메시지는 **응답 상태 코드, 응답 헤더, 응답 본문**을 포함

5.8 단계 5:응답 만들기

✓ 응답 엔터티

- 트랜잭션에 따라 응답 본문을 생성한다면, 내용을 응답 메시지와 함께 돌려보내며, 본문이 있다면 주로 다음을 포함
  - 본문의 MIME 타입을 서술하는 Content-Type 헤더
  - 응답 본문의 길이를 서술하는 Content-Length 헤더
  - 실제 응답 본문의 내용

5.8 단계 5:응답 만들기

✓ MIME 타입 결정하기

- 웹 서버에게는 응답 본문의 MIME 타입을 결정해야하는 책임이 있음

  - mime.types
    웹 서버는 MIME 타입을 나타내기 위해 파일의 확장자를 사용할 수 있으며, 각 리소스의 MIME 타입을 계산하기 위해 확장자별 MIME 타입이 담겨 있는 파일을 탐색
    가장 흔한 방법

  - 매직 타이핑(Magic typing)
    아파치 웹 서버는 파일의 MIME 타입을 알아내기 위해 파일의 내용을 검사해서 알려진 패턴에 대한 테이블(매직 파일)에 해당하는 패턴이 있는지 찾아 볼 수 있음
    느리긴 하지만 파일이 표준 확장자 없이 이름이 지어진 경우에 편리

5.8 단계 5:응답 만들기

✓ MIME 타입 결정하기

- 유형 명시(Explicit typing)
  특정 파일이나 디렉터리 안의 파일을 확장자나 내용에 상관없이 어떠한 MIME 타입을 갖도록 웹 서버를 설정할 수 있음

- 유형 협상(Type negotiation)
  어떤 웹 서버는 하나의 리소스가 여러 종류의 문서 형식에 속하도록 설정할 수 있음
  이때 웹 서버가 사용자와의 협상을 통해 사용하기 가장 좋은 형식을 판별할 것 인지의 여부도 설정할 수 있음

5.8 단계 5:응답 만들기

✓ 리다이렉션

- 웹 서버는 요청을 수행하기 위해 브라우저가 다른 곳으로 가도록 리다이렉트 할 수 있음
- 리다이렉션 응답은 3XX 상태 코드로 지칭
- Location 응답 헤더는 컨텐츠의 새롭거나 선호하는 위치에 대한 URI를 포함
- 리다이렉트는 다음의 경우 유용
  - 영구히 리소스가 옮겨진 경우
  - 임시로 리소스가 옮겨진 경우
  - URL 증강
  - 부하 균형
  - 친밀한 다른 서버가 있을 때
  - 디렉터리 이름 정규화

5.8 단계 5:응답 만들기

- ✓ 영구히 리소스가 옮겨진 경우 - 301 Moved Permanently
  - 리소스는 새 URL이 부여되어 위치가 옮겨졌거나 이름이 바뀜
  - 웹 서버는 클라이언트에게 북마크를 갱신할 수 있다고 말해줄 수 있음

- ✓ 임시로 리소스가 옮겨진 경우 - 303 See Other, 307 Temporary Redirect
  - 리소스가 임시로 옮겨지거나 이름이 변경된 경우에는 서버는 클라이언트를 새 위치로 리다이렉트하길 원할 것
  - 그러나 이름 변경이 임시적이기 때문에, 서버는 클라이언트가 나중에는 원래의 URL로 찾아오고 북마크도 갱신하지 않기를 원함

5.8 단계 5:응답 만들기

- ✓ URL 증강 - 303 See Other, 307 Temporary Redirect
  - 웹 서버는 종종 문맥 정보를 포함시키기 위해 재 작성된 URL로 리다이렉트
  - 요청이 도착하면 서버는 상태 정보를 포함한 새 URL을 생성하고 사용자를 이 URL로 리다이렉트
  - 클라이언트는 리다이렉트를 따라가서, 상태정보가 추가된 완전한 URL을 포함한 요청을 다시 보냄
- ✓ 부하 균형 - 303 See Other, 307 Temporary Redirect
  - 만약 과부하된 서버가 요청을 받으면, 덜 부하가 걸린 서버로 리다이렉트

5.8 단계 5:응답 만들기

- ✓ 친밀한 다른 서버가 있을 때 - 303 See Other, 307 Temporary Redirect
  - 웹 서버는 어떤 사용자 정보를 가질 수 있으며, 서버는 클라이언트를 클라이언트의 정보를 가지고 있는 다른 서버로 리다이렉트할 수 있음

- ✓ 디렉터리 이름 정규화
  - 클라이언트가 디렉터리 이름에 대한 URI 요청을 하는데 빗금(/) 을 빠트렸다면, 웹 서버는 이것이 잘 작동하도록 추가한 URI로 리다이렉트

5.9 단계 6:응답 보내기

- ✓ 웹 서버는 요청들 받을 때와 마찬가지로 커넥션 너머로 데이터를 보낼 때도 비슷한 이슈에 직면
- ✓ 웹 서버는 커넥션 상태를 추적
- ✓ 비지속적인 커넥션이라면, 모든 메시지를 전송하고 자신의 커넥션을 닫음
- ✓ 지속적인 커넥션이라면, Content-Length 헤더를 바르게 계산하기 위해 특별한 주의를 필요로 하는 경우나, 클라이언트가 응답이 언제 끝나는지 알 수 없는 경우에 커넥션은 열린 상태로 유지

5.10 단계 7:로깅

- ✓ 트랜잭션이 완료되었을 때 트랜잭션이 어떻게 수행되었는지에 대한 로그를 로그파일에 기록
- ✓ 대부분의 웹 서버는 로깅에 대한 여러 가지 설정 양식을 제공

참고

- HTTP 완벽 가이드
- https://velog.io/@yejineee/HTTP-%EC%99%84%EB%B2%BD%EA%B0%80%EC%9D%B4%EB%93%9C-5%EC%9E%A5-%EC%9B%B9-%EC%84%9C%EB%B2%84-%EB%B8%94%EB%A1%9C%EA%B7%B8%EC%9A%A9
- https://suuntree.tistory.com/347
- https://docs.google.com/presentation/d/12vrK3FRtLD8PMpFvnPa5w_2WnMOd1tAN_kCFPXTifpg/edit#slide=id.g6103154412_0_89
- https://feel5ny.github.io/2019/09/07/HTTP_005/

- 이미지 아이콘 : https://www.flaticon.com/

QnA