웹 서버?

다른 말로 HTTP Server라고도 한다. 웹 브라우저의 파트너로서 서버의 정보를 제공하는 소프트웨어라고 할 수 있다.
대표적으로 Nginx, Apache, Microsoft IIS Server가 있다.
웹 서버는 인터넷 네트워크 위에서 HTTP 프로토콜을 이용해 HTML, CSS, Javascript, image/mediafile과 같은 정적인 정보(즉시 응답 가능한 컨텐츠)들을 웹 브라우저에 전송한다.
이때 웹 서버가 정적 컨텐츠가 아닌 동적 컨텐츠를 요청받으면 WAS에게 해당 요청을 넘겨주고, WAS에서 처리한 결과를 클라이언트에게 전달하는 역할도 해준다.

WAS(Web Application Server)

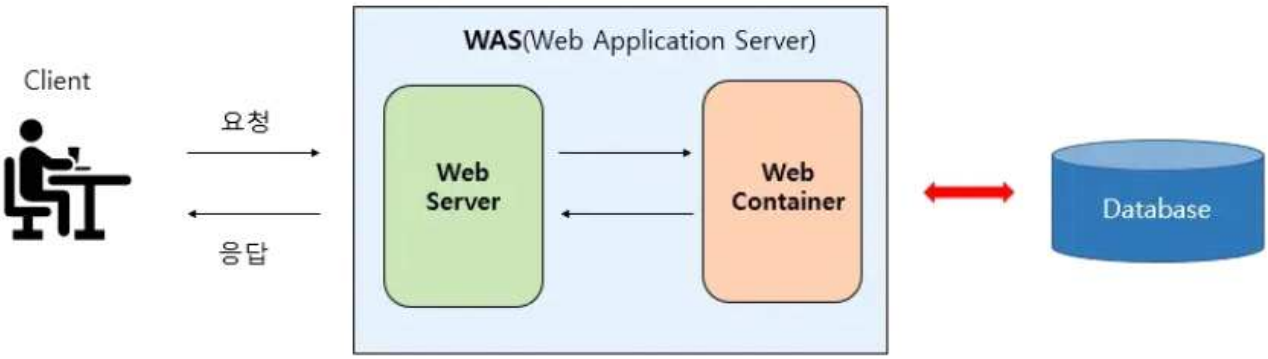
WAS란 DB 조회 혹은 다양한 로직 처리를 요구하는 동적 컨텐츠를 제공하기 위해 만들어진 Application 서버이다. HTTP 프로토콜을 기반으로 사용자 컴퓨터나 장치에 애플리케이션을 수행해주는 미들웨어로서, 주로 데이터베이스 서버와 같이 수행된다.

WAS는 Thymeleaf, JSP, Servlet 구동환경을 제공해주기 때문에 서블릿 컨테이너 혹은 웹 컨테이너로 불린다.

이러한 WAS는 웹 서버의 기능들을 구조적으로 분리하여 처리하고자 하는 목적으로 제시되었다. 분산 트랜잭션, 보안, 메시징, 쓰레드 처리 등의 기능을 처리하는 분산 환경에서 사용된다. WAS는 프로그램 실행 환경과 DB 접속 기능을 제공하고, 여러 개의 트랜잭션을 관리 가능하다. 또한 비즈니스 로직을 수행할 수 있다.

이러한 WAS에는 Tomcat, JBoss, WebSphere 등이 있다.

웹 서버와 WAS



[그림 1] WAS 구조

WAS는 Web Server와 Web Container의 역할을 모두 할 수 있다. 여기서 컨테이너는 JSP, Servlet을 실행시킬 수 있는 소프트웨어를 말한다. 현재 WAS의 웹 서버도 정적인 컨텐츠를 처리하는 데 성능상 큰 차이가 없다.

그렇다면 WAS가 웹 서버의 기능까지 모두 수행하면 되는 것일까?

웹 서버와 WAS를 분리해야 한다. 그 이유는 다음과 같다.

1. 서버 부하 방지
WAS와 웹 서버는 분리하여 서버의 부하를 방지해야 한다. WAS는 DB 조회나 다양한 로직을 처리하고, 단순한 정적 컨텐츠는 웹 서버에서 처리해줘야 한다. 만약 정적 컨텐츠까지 WAS가 처리한다면 부하가 커지게 되고, 수행 속도가 느려질 것이다.

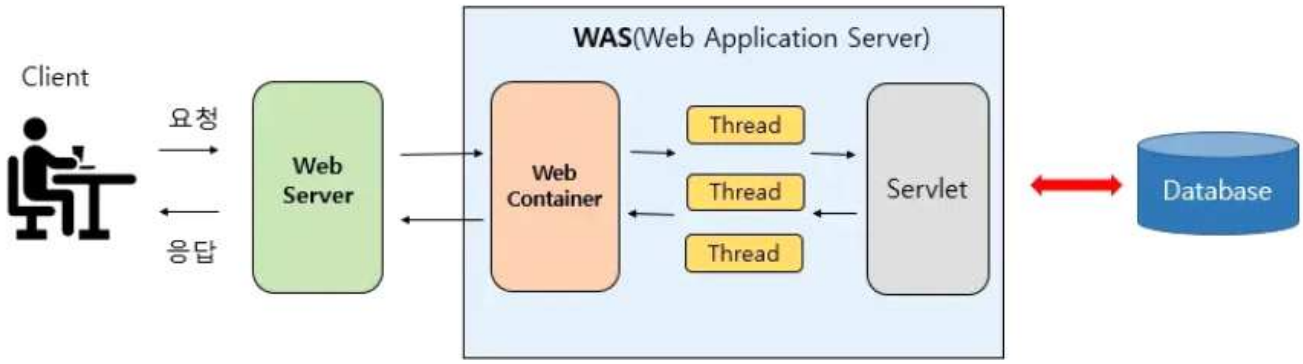
2. 보안 강화
SSL에 대한 암호화, 복호화 처리에 웹 서버를 사용 가능

3. 여러 대의 WAS 연결 가능
로드 밸런싱을 위해 웹 서버를 사용할 수 있다. 여러 개의 서버를 사용하는 대용량 웹 어플리케이션의 경우 웹 서버와 WAS를 분리하여 무중단 운영을 위한 장애 극복에 쉽게 대응할 수 있다.

4. 여러 웹 어플리케이션 서비스 가능
하나의 서버에서 PHP, JAVA 애플리케이션을 함께 사용할 수 있다.

이러한 이유로 웹 서버를 WAS 앞에 두고 필요한 WAS들을 웹 서버에 플러그인 형태로 설정하면 효율적인 분산 처리가 가능하다.

Web Service Architecture



[그림 2] 웹 서비스 구조

웹 서비스는 아래와 같이 다양한 구조를 가질 수 있다.

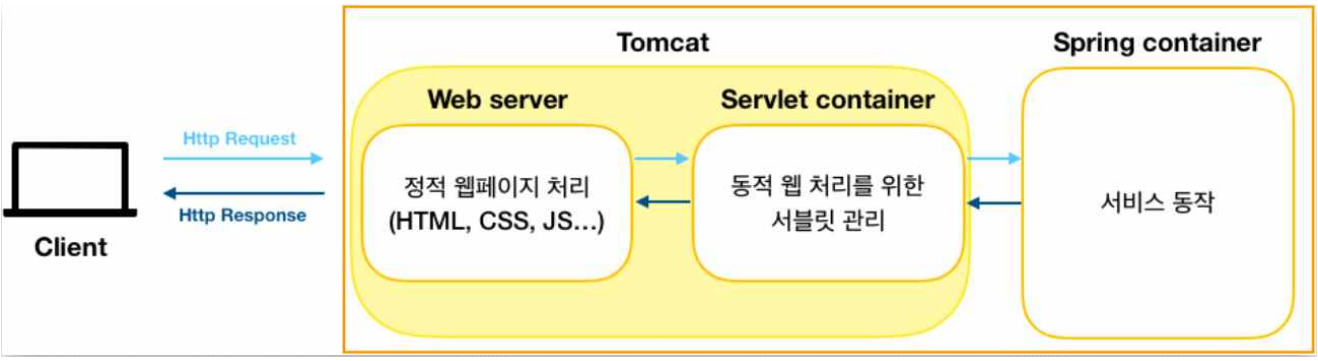
1. Client → 웹 서버 → DB
2. Client → WAS → DB
3. Client → 웹 서버 → WAS → DB

[그림 2]는 3번 구조를 나타낸다. 클라이언트가 웹 서버에 HTTP 요청을 보내면 웹 서버는 정적인 컨텐츠 요청은 바로 응답하고, 동적인 컨텐츠 요청은 WAS에게 넘겨서 처리하고 결과를 WAS에게 받아서 클라이언트에게 넘겨준다.

Tomcat

일반적으로 Tomcat은 'WAS(Web Application Server)'의 대표적인 미들웨어이다. 하지만 Tomcat은 일반적으로 Apache Tomcat이라 불리며 회사명이자 웹서버의 대표적인 미들웨어인 아파치(Apache)의 기능 일부분을 가져와 함께 사용되면서 웹서버(Web Server)의 기능과 웹 애플리케이션 서버(Web Application Server) 모두를 포함하고 있다.

개발자가 Spring으로 작성한 웹 프로그램이 Apache Tomcat을 이용해 웹 서비스로 등록되게 되면 어떤 프로세스로 Client의 요청을 처리하는가.

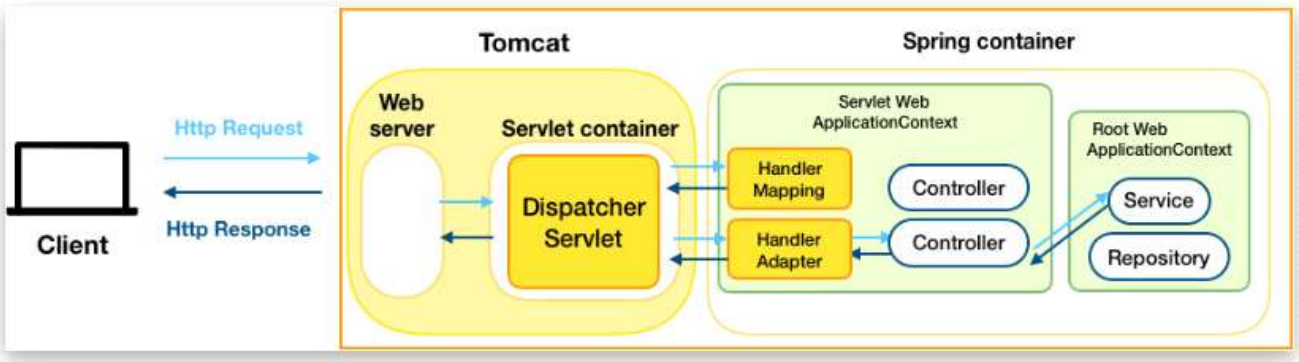


일반적으로 자바 웹프로그래밍을 할 때 사용하는 Spring + Tomcat 조합으로 서비스를 올리게 되면 위와 같은 구조를 통해 클라이언트와 통신 하게 된다.

Apache tomcat 5.5 이후부터 Web server의 기능인 httpd(웹서비스 데몬) native 모듈을 가지고와서 정적파일을 처리하기 때문에 별도의 Web server 기능에 뒤쳐지지 않는 정적 파일 처리를 할 수 있게 되었다.

스프링MVC 에서는 Dispatcher Servlet이라는 모든 요청을 담당하는 서블릿을 두고 컨트롤러에 위임을 하여 요청을 처리하게 된다.

이와 같이 프론트 컨트롤러 디자인 패턴이 적용된 SpringMVC를 통해 개발자는 별도의 서블릿 개발 없이, Controller의 구현만으로도 동적인 response를 클라이언트에게 줄 수 있게 되었다.



웹 브라우저?

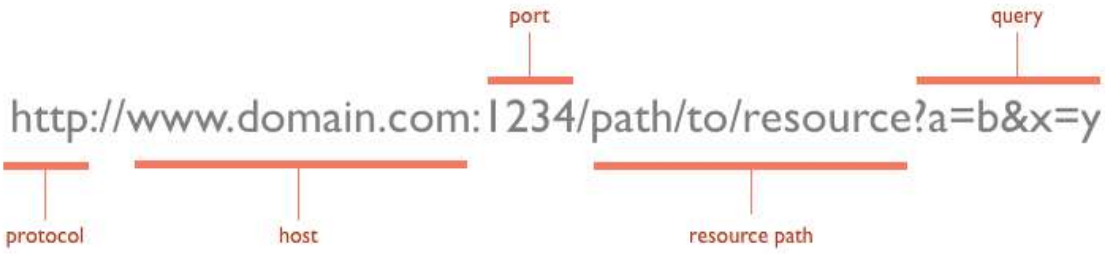
크롬, 사파리, 네이버 웨일, 파이어폭스, ~~인터넷 익스플로러?~~ 등을 웹 브라우저라고 한다.
웹 서버와 통신하며 html문서나 파일을 출력해준다.

HTTP 프로토콜?

웹에서 데이터를 주고받기 위한 일종의 규약이다.
HTTP 프로토콜은 상태를 가지고 있지 않다. 따라서 이전 데이터 요청과 다음 데이터 요청이 완전히 분리되어 독립적으로 관리할 수 있다.
이와 같은 이점으로 당장 받은 요청 이외의 정보를 추가적으로 관리할 필요가 없어지고, 다수의 요청 및 서버의 부하를 줄일 수 있는 성능의 향상을 가져왔다.
HTTP 프로토콜은 일반적으로 TCP/IP 통신 위에서 동작하며 기본 포트는 80번이다.

URL (Uniform Resource Locators)

서버에 자원을 요청하기 위해서 입력하는 영문 주소.



출처 https://joshua1988.github.io/web-development/http-part1/

HTTP의 주요 메서드

html form에서 method를 지정해줄 때 사용하는 것들

- **GET** : 존재하는 자원에 대한 **요청**
- **POST** : 새로운 자원을 **생성**
- **PUT** : 존재하는 자원에 대한 **변경**
- **DELETE** : 존재하는 자원에 대한 **삭제**

HTTP 상태코드

2xx - 성공

200번 대의 상태 코드는 대부분 성공을 의미한다.

- 200 : GET 요청에 대한 성공
- 204 : No Content. 성공했으나 응답 본문에 데이터가 없음
- 205 : Reset Content. 성공했으나 클라이언트의 화면을 새로 고침 하도록 권고
- 206 : Partial Content. 성공했으나 일부 범위의 데이터만 반환

3xx - 리다이렉션

300번대 상태 코드는 대부분 클라이언트가 이전 주소로 데이터를 요청하여 서버에서 새로운 URL로 리다이렉트(redirect)를 유도하는 경우이다.

- 301 : Moved Permanently, 요청한 자원이 새 URL에 존재
- 303 : See Other, 요청한 자원이 임시 주소에 존재
- 304 : Not Modified, 요청한 자원이 변경되지 않았으므로 클라이언트에서 캐싱된 자원을 사용하도록 권고. ETag와 같은 정보를 활용하여 변경 여부를 확인

4xx - 클라이언트 에러

400번대 상태 코드는 대부분 클라이언트의 코드가 잘못된 경우이다. 없는 자원을 요청했거나 요청 권한 등이 잘못된 경우에 발생한다.

- 400 : Bad Request, 잘못된 요청
- 401 : Unauthorized, 권한 없이 요청. Authorization 헤더가 잘못된 경우
- 403 : Forbidden, 서버에서 해당 자원에 대해 접근 금지
- 404 : Not found, 요청한 자원이 서버에 없음
- 405 : Method Not Allowed, 허용되지 않은 요청 메서드

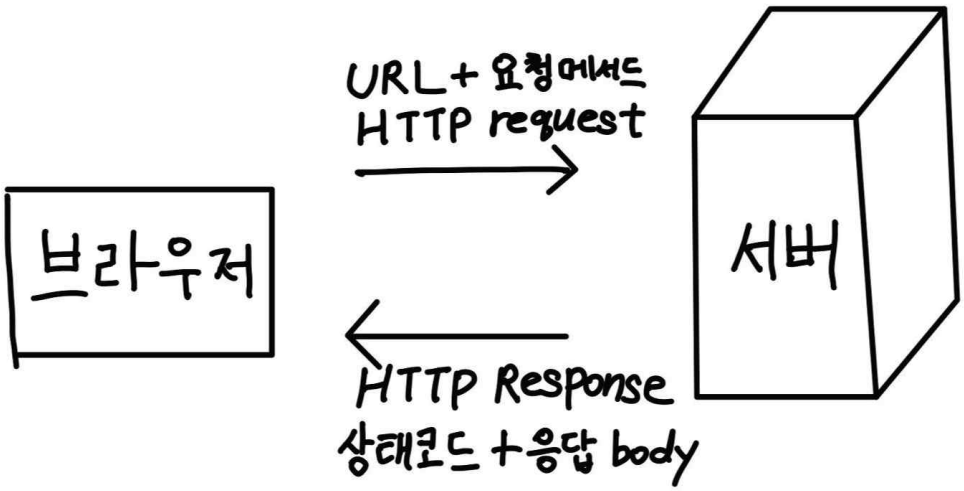
- 409 : Conflict, 최신 자원이 아닌데 업데이트하는 경우. ex) 파일 업로드 시 버전 충돌

5xx - 서버 에러

500번대 상태 코드는 서버 쪽에서 오류가 난 경우이다.

- 501 : Not Implemented, 요청한 동작에 대해 서버가 수행할 수 없는 경우
- 503 : Service Unavailable, 서버가 과부하 또는 유지 보수로 내려간 경우

앞에서 설명한 URL, Method, 상태 코드를 종합해서 그림으로 나타내 보면 다음과 같다.



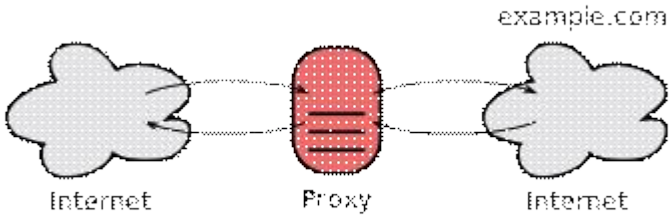
TCP/IP?

TCP/IP 는 두 개의 프로토콜(TCP, IP)을 사용하여 데이터 통신을 하는 것을 의미한다.
IP(Internet Protocol)는 복잡한 네트워크 망을 패킷 통신 방식(데이터를 작은 조각들로 나눠서 보내는 방식)을 사용해 데이터를 가장 효율적인 방법으로 보내는 작업을 한다.
TCP(Transmission Control Protocol)는 조각난 데이터를 받으면서 순서가 맞지 않다면 정렬하고, 누락된 데이터를 점검하여 다시 요청하는 작업을 한다.

Proxy?

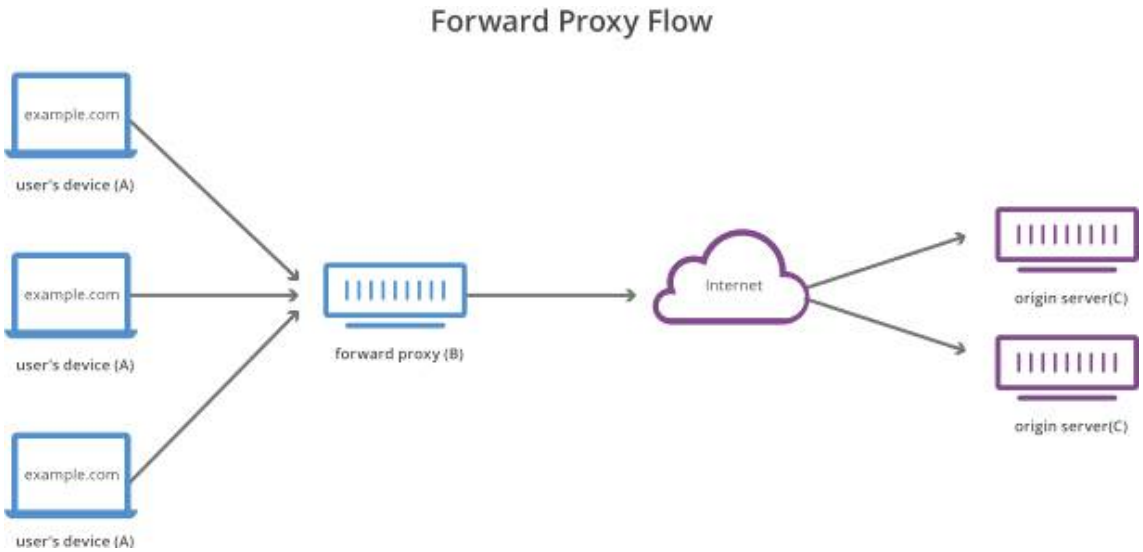
클라이언트가 데이터를 요청하면 서버가 요청한 리소스를 가져다줄 것이다. 하지만 중간에 클라이언트의 요청을 서버로 보내주는 Proxy라고 불리는 서버가 하나 존재하게 된다.
이처럼 프록시는 중계 서버라고 이해하면 편하다. 클라이언트와 서버가 서로 직접적으로

통신하지 않고, 프록시 서버를 이용해서 리소스를 전달하며 보안, 트래픽 분산, 캐시 사용(속도 향상) 등 여러 장점을 가지는 중요한 서버이다.



출처 https://ko.wikipedia.org/wiki/%ED%94%84%EB%A1%9D%EC%8B%9C_%EC%84%9C%EB%B2%84

Forword Proxy

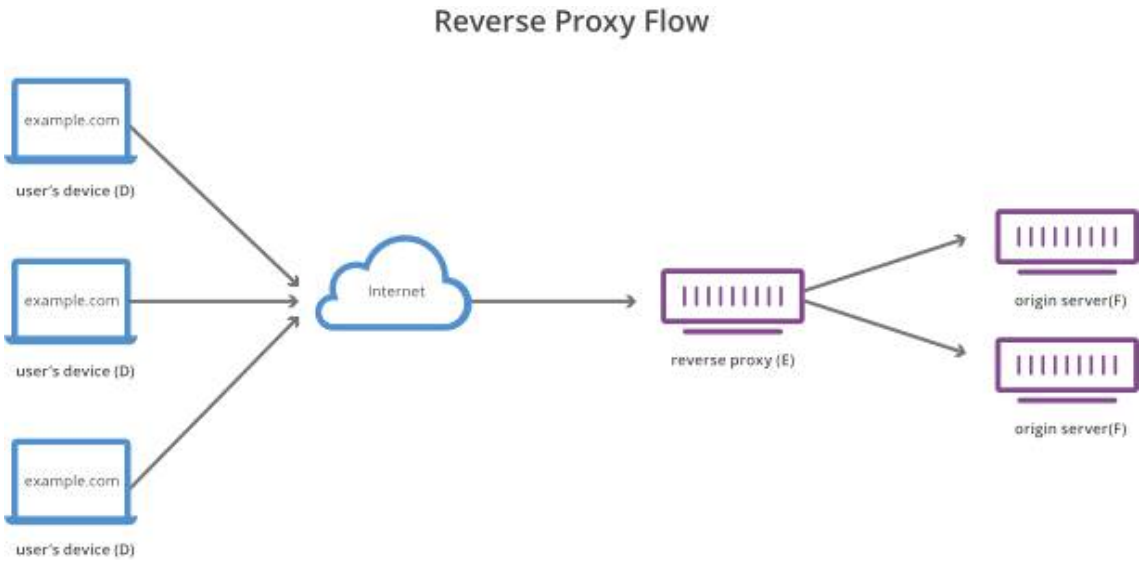


일반적인 프록시 서버를 말하며, 클라이언트와 웹 서버의 중개역할로 클라이언트가 요청 시 Proxy서버는 해당 요청을 웹 서버로 중계해 자원을 가져오는 개념이다.
프록시 서버는 클라이언트가 요청하기 전까지 웹 서버의 주소를 알 수 없다는 게 특징이다.

예를 들어 회사 내부망에서 인터넷의 www.google.com 으로 연결해 달라고 서버에 요청을 보내면 이때 프록시 서버를 호출하게 되는데 이러한 방식이 포워드 프록시이다.

포워드 프록시 방식은 정해진 사이트에만 연결이 가능하게끔 설정이 가능해서 주로 회사 내부의 인트라넷 등 보안이 중요한 환경에서 주로 사용된다.

Reverse Proxy



클라이언트와 서버 사이에(서버 앞에) 위치하여 보안, 로드밸런싱 역할을 한다. 그래서 클라이언트가 특정 리소스를 요청을 하면 프록시 서버가 WAS(Web Application Server)에 요청 후 응답 받은 리소스를 클라이언트에게 전달해주는 개념이다.
예를 들자면 이 방식은 특정 ip주소에서 제공하는 api서버를 호출하기 위해서 인터넷에 있는 클라이언트가 리버스 프록시 서버에 api를 요청하여 응답을 받는 방식이다.

리버스 프록시 서버는 실제 서버가 어디서 동작하는지 감추는 역할을 한다. 클라이언트는 리버스 프록시를 통해서 리소스를 요청하기 때문에 서버의 IP주소를 알 수 없다. 그리고 리버스 프록시 서버는 실제 서버들에 대한 주소를 매핑하고 있어야 한다.

포워드 프록시는 제한에 걸리는 것들 빼고 모든 심부름을 다 수행해주는 심부름꾼 같고, 리버스 프록시는 특정 가게와 연결되어 있으면서 손님이 A라는 물건을 요청하면 연결된 가게의 A 물건만 전문적으로 갖다 주는 역할을 수행하는 것 같다.

로드밸런싱?

서버의 부하(Load)를 분산(Balancing)시키는 기술이다.
많은 클라이언트들(이용자들)이 하나의 웹 사이트에 접속해서 해당 서버의 트래픽이 늘어날 때 여러 대의 서버를 이용해서 요청을 처리하도록 한다. 각 서버의 부하량, 속도 저

하 등을 고려해서 서버의 가중치, 스케줄링 등을 결정해 트래픽을 적절히 분산시키는 과정을 로드밸런싱이라고 한다.

로드밸런싱의 장점?

한 서버의 부하를 분산시킨다는 게 당연히 장점이 된다.
값싼 비용으로 다수의 서버를 증설하여 관리할 수 있기 때문에 고가의 서버를 구매할 필요가 없다.
1대의 서버가 멈춰도 서비스가 중단되지 않는다. 마찬가지로 서버를 늘리는 과정에서도 서비스가 중단되지 않는다.

ex> Nginx에서 제공하는 로드밸런싱 방법에 3가지가 있다.

1. Round Robbin : 운영체제 스케줄링 방법에도 있는 방식이다. 주어진 서버들을 순차적으로 돌아가면서 선택하는 방법이다.
2. Least-connected : 현재 접속자 수가 가장 적은 서버를 선택하는 방법이다.
3. ip-hash : hash function을 사용해서 클라이언트 ip를 hash 한 결과에 따라서 다른 서버를 할당해주는 방법이다.

API 란?

API(Application Programming Interface)는 특정 서비스나 응용 프로그램에서 소스 코드를 공개하지 않고 요청한 기능을 제공하기 위해서 이용한다.
서버 API는 web 서버에 클라이언트가 어떠한 요청을 하였을 때 그에 맞는 응답을 줄 수 있는 endpoint를 Web을 통하여 노출한 것이다.
예를 들어

http://12.34.56.78:8000/api/company

라는 endpoint에 서버와 연결된 데이터베이스에 있는 모든 회사의 주가정보를 json으로 반환하는 api를 열어놓았다면 사용자는 해당 endpoint를 요청하여 데이터를 json으로 받아올 수 있다.

JSON 이란?

JSON은 JavaScript Object Notation 의 약자

직역하면 '자바 스크립트 객체 표기법'으로 데이터를 쉽게 '교환' 하고 '저장' 하기 위한 텍스트 기반의 데이터 교환 표준이다.

JSON은 텍스트 기반이기 때문에 다양한 프로그래밍 언어에서 데이터를 읽고 사용할 수 있으며, JSON의 기본적인 형태는 아래와 같습니다.

1. **{ key : value }** → JSON의 형태는 키(Key)와 값(value)의 쌍으로 이루어져 있는 구조이며, Key와 Value사이에는 콜론(:)이 들어간다.

2. **{ key1 : value, key2 : value2 }** → 여러 데이터를 나열할 경우 쉼표(,)를 사용

3. **{ key1 : { inKey : inValue }, key2 : [arr1, arr2 arr3] }**

{"판매자정보" : { "이름" : "남도일", "지역" : "서울" }, "판매품목" : ['사과','배','딸기'] }

→ 객체(Object)는 중괄호({ })로 묶어서 표현하고, 배열(Array)은 대괄호([ ])로 묶어서 표현

API의 역할

모든 사람들이 서비스들의 고유 데이터베이스에 접근해서는 안 된다.

API는 이를 방지하기 위해서 서버와 데이터베이스에 대한 출입문 역할을 하며, 허용된 클라이언트, 서버에 대해서만 데이터를 넘겨준다.

API는 모든 접속을 표준화하기 때문에 클라이언트/운영체제와 상관없이 누구나 동일한 데이터를 얻을 수 있게 한다.

REST API 란?

API의 endpoint 아키텍처 스타일이다.

HTTP 메서드를 사용하면 진정한 RESTful API라고 할 수 있다.

REST = REpresentational State Transfer 라고 한다.

- REST API는 리소스를 중심으로 디자인 된다. 클라이언트에서 요구하는 모든 종

류의 데이터, 서비스가 리소스에 포함된다.

- 리소스마다 해당 리소스를 고유하게 식별할 수 있는 URI인 식별자가 존재한다.

  아래는 특정 유저에 대한 URI이다.

  https://localhost:8000/api/v1/user/1 ← End Point

- 클라이언트는 리소스를 위와 같은 URI를 통해 서버에 요청하면서 서버와 상호작용한다. 일반적으로 많은 Web API가 데이터 교환 형식으로 JSON을 사용한다. 예를 들어 위 URI에 대한 GET 요청은 다음과 같은 JSON을 반환할 수 있다.

  {"username": "admin", "email": "aaa@google.com"}

- REST API는 균일한 인터페이스를 사용하므로 클라이언트와 서버의 분리에 많은 도움이 된다. 가장 일반적인 작업은 GET, POST, PUT, PATCH, DELETE이다.

- REST API를 사용하는 HTTP 요청은 독립적이어야 하고 임의의 순서로 발생할 수 있기 때문에, 요청 사이에 일시적으로 상태 정보를 유지할 수 없다. 이러한 조건 때문에 웹 서비스의 확장성이 우수하다. 클라이언트와 특정 서버 사이의 연결 및 선호도를 유지할 필요가 없기 때문이다. 따라서 모든 서버는 모든 클라이언트의 모든 요청을 처리할 수 있다는게 REST의 특징이다.

REST API는 리소스를 중심으로 API 디자인을 구성해야 한다.

https://myrestapi.com/user	// Good
https://myrestapi.com/create-user	// Avoid

리소스 URI는 동사(리소스에 대한 작업)가 아닌 명사(리소스)를 기반으로 해야한다.

예를 통해 REST 형식을 가지는 URI에 대해 감을 잡아보자

https://myrestapi.com/user

위 URI에 HTTP GET 요청을 보낸다는 것은 어떤 것을 의미하는 것일까?
① 특정 user의 정보를 불러온다? - ❌
우선 특정 유저를 지칭하는 리소스 번호가 포함되지 않았기 때문에 특정 유저에 대한 동작은 아닐 것이다.
② user를 생성한다? - ❌
그리고 파라미터가 없는 GET요청을 보냈기 때문에 user를 생성하는 동작도 아닐 것이다.
일반적으로 리소스 생성을 위한 메소드는 POST이다.
③ 특정 user를 수정한다? - ❌
일반적으로 리소스 생성을 위한 메소드는 PATCH이다.
④ 특정 user를 삭제한다? - ❌
일반적으로 리소스 생성을 위한 메소드는 DELETE이다.

따라서 위 URI는 모든 user의 정보를 리스트로 받아오는 API가 아닐까 추측해 볼 수 있다.

URI에는 일관적인 명명 규칙을 적용한다. 그리고 리소스에 대한 URI를 계층 구조로 구성하는 것이 좋다.

예를 들어
/board는 현재 생성되어 있는 게시판 종류 리스트에 대한 경로이고,
/board/1는 ID가 1인 게시판의 경로이다.

이러한 접근 방식을 사용하면 웹 API를 직관적으로 유지할 수 있다.
또한 많은 Web API는 매개 변수가 있는 URI 경로를 기반으로 요청을 라우팅할 수 있으므로 개발자는 /board/<int:board_id> 에 대한 경로를 정의할 수 있다.

[GET] https://myrestapi.com/board/1 : 1번 게시판에 속한 정보를 요청하는 URI

[GET] https://myrestapi.com/board/1/post : 1번 게시판에 속한 모든 post 정보를 요청하는 URI

RESTful API에서 사용하는 일반적인 HTTP 메서드는 다음과 같다.

- GET : 지정된 URI에서 리소스의 정보를 요청한다.
- POST : 지정된 URI에 새로운 리소스를 생성한다. 요청 시 보내는 data는 리소스의 정보를 담고있다. 리소스를 만들지 않는 작업을 할 수 도 있다.
- PUT : 지정된 URI에 리소스를 만들거나 대체한다. 요청 시 보내는 data는 리소스를 만들 또는 업데이트할 정보를 담고 있다.
- PATCH : 리소스의 부분 업데이트를 수행한다. 요청 시 보내는 data는 리소스에 적용할 변경 내용을 담고 있다.
- DELETE : 지정된 URI의 리소스를 제거한다.

리소스	POST	GET	PUT	DELETE
/board	새 게시판 만들기	모든 게시판 검색	게시판 다수 업데이트	모든 게시판 제거
/board/1	1번 게시판 즐겨찾기	1번 게시판에 대한 세부정보 검색	게시판 1의 세부 정보 업데이트	1번 게시판 제거
/board/1/post	1번 게시판에 대한 새 post 만들기	1번 게시판의 모든 post 검색	1번 게시판의 post 다수 업데이트	1번 게시판의 모든 post 제거