Spring MVC 01

[웹 서버, WAS]

스프링 MVC 1편 - 백엔드 웹 개발 핵심 기술

김영한

2022.08.18

웹 서버 **Web Server**

- HTTP 기반으로 동작
- 정적 리소스 제공, 기타 부가기능
- 정적(파일) HTML, CSS, JS, 이미지, 영상 등을 서빙함 ,,

- NGINX, APACHE 등

웹 애플리케이션 서버 **(WAS)**

- HTTP 기반으로 동작
- 웹 서버 기능 포함+ (정적 리소스 제공 가능)
- 프로그램 코드를 실행해서 애플리케이션 로직 수행
- 동적 HTML, HTTP API(JSON) , 서블릿, JSP, 스프링 MVC

- 톰캣(Tomcat) Jetty, Undertow 등

차이 **&** 정리

웹 서버는 정적 리소스 실행 - WAS는 애플리케이션 로직 실행

⇒ 하지만 WAS가 웹 서버 기능을 포함 하기 때문에 WAS, DB만으로 시스템 구성이 가능하다.

BUT WAS,DB만으로 구성하게 된다면**?!**

WAS가 너무 많은 역할을 담당하게 되면 서버 과부하가 우려되고 정적 리소스 때문에 비싼 애플리케이션 로직이 정적리소스 때문에 수행이 어려울 수 있다. 또한 **WAS** 장애시 오류 화면도 노출 불가능 → 큰 시스템 구축에 부담된다**.**

웹 시스템 구성 - **WEB, WAS, DB**

정적 리소스를 웹서버가 처리 → 웹 서버는 애플리케이션 로직등 동적 처리 요청을 WAS에게 보낸다. → WAS는 그 동적 처리를 담당한다.

리소스 관리가 용이하다**. Web Server** 는 정적이기 때문에 잘 죽지 않는다**.** (WAS는 잘 죽음)

잘 죽지 않는 Web Server 에서 오류 화면을 뿌려줄 수 있다.

서블릿

HTML Form 데이터 전송(POST) → HTTP 메시지를 만들어서 서버로 전송

만약 웹 애플리케이션을 직접 구현한다면?

직접구현

- 서버 TCP/IP 연결 대기, 소켓 연결
- HTTP 요청 메시지를 파싱해서 읽기
- POST 방식, /save URL 인지
- Content-Type 확인
- HTTP 메시지 바디 내용 피싱
  - username, age 데이터를 사용할 수 있게 파싱
- 저장 프로세스 실행
- 비즈니스 로직 실행
  - 데이터베이스에 저장 요청
- HTTP 응답 메시지 생성 시작
  - HTTP 시작 라인 생성
  - Header 생성
  - 메시지 바디에 HTML 생성에서 입력
- TCP/IP에 응답 전달, 소켓 종료

서블릿을 지원하는 **WAS** 사용

- ~~서버 TCP/IP 대기, 소켓 연결~~
- ~~HTTP 요청 메시지를 파싱해서 읽기~~
- ~~POST 방식, /save URL 인지~~
- ~~Content-Type 확인~~
- ~~HTTP 메시지 바디 내용 피싱~~
  - ~~username, age 데이터를 사용할 수 있게 파싱~~
- ~~저장 프로세스 실행~~
- 비즈니스 로직 실행
  - 데이터베이스에 저장 요청
- ~~HTTP 응답 메시지 생성 시작~~
  - ~~HTTP 시작 라인 생성~~
  - ~~Header 생성~~
  - ~~메시지 바디에 HTML 생성에서 입력~~
- ~~TCP/IP에 응답 전달, 소켓 종료~~

서블릿 컨테이너

- 서블릿 컨테이너에서 서블릿 객체를 생성해줘서 HTTP 요청, 응답을 받게 된다.
- 서블릿 객체는 싱글톤으로 관리 **(** 공유 변수 사용 주의 해야함 **)**
- 동시 요청을 위한 멀티 스레드 처리를 지원해준다**.**

동시 요청 **-** 멀티 쓰레드

WAS에서 서블릿 객체를 누가 호출하나? ⇒ 쓰레드

쓰레드

- 애플리 케이션 코드를 하나하나 순차적으로 실행하는 것은 쓰레드
- 쓰레드는 한번에 하나의 코드 라인만 수행
- 동시처리가 필요하면 쓰레드가 추가로 필요하다.

WAS 는 요청이오면 쓰레드를 할당하고 서블릿 객체 하고 요청을 처리한다.

만약 여러 요청이 오게된다면**?**

쓰레드가 사용되고 있다면 쓰레드를 대기시키면 문제가 생긴다.

그냥 하나 더 만들자**!** - 요청마다 쓰레드 만들어 보자

요청마다 만들게 된다면 …. 동시요청도 가능하고 하나가 문제 생겨도 정상 작동하지만 단점들이 존재한다. 쓰레드 생성 비용이 비싸고, 컨텍스트 스위칭 비용이 발생하고, 쓰레드 생성에 제한이 없다. –많아지면 서버가 죽어 버릴수도 있다.

쓰레드 풀을 만들자

쓰레드 풀에서 꺼내서 쓰고 사용 후 반납 개념으로 사용한다. 쓰레드 생성, 삭제 사용 비용이 없어진다. 응답시간도 빨라진다.

쓰레드 풀보다 많은 요청이 들어오면 대기 시키거나 거절 시킬수 있다.

실무 팁

- WAS의 주요 튜닝 포인트는 최대 쓰레드 (max thread) 수이다.
- 최대 쓰레드가 낮으면 **:** 서버 리소스는 여유롭지만 클라이언트는 늦게 응답 받는다.
- 최대 쓰레드가 높으면 **:** 요청이 많아지면 CPU 메모리 임계점 초과로 서버 다운
- 장애가 발생한다면**?** 서버를 늘리고, 이후에 튜닝

쓰레드 풀 적정 숫자

- 애플리케이션 로직의 복잡도,CPU 메모리 IO 리소스 상황에 따라 모두 다름
- 성능 테스트가 필요하다! - 실제 서비스와 유사하게 성능 테스트 시도

결과적으로 **WAS**의 멀티 쓰레드 지원을 해주기 때문에 멀티 쓰레드 관련 코드는 신경 안쓰고 싱글 쓰레드 프로그래밍 하듯이 소스 코드 개발하고 **(** 싱글톤 객체 주의 **)** 쓰레드 관련은 설정으로만 신경 쓰면 된다는 장점이 생긴다**.**

HTML, HTTP API, CSR, SSR

백앤드 개발자가 서비시를 제공할때 고민해야 할 3가지

정적 리소스**, HTML** 페이지**, HTTP API**

정적리소스 **:** 고정된 HTML 파일, CSS, JS, 이미지, 영상 등을 제공

HTML 페이지 **:** 동적으로 필요한 HTML 파일을 생성해서 전달

HTTP API : HTML이아니라 데이터를 전달 **[** 데이터만 주고 받음, UI화면이 필요하면 클라이언트가 별도 처리, 앱, 웹 클라이언트, 서버 to 서버 (주문 서버 → 결제 서버) **]**

- 주로 JSON 형태로 데이터 통신

SSR - 서버 사이드 렌더링

서버에서 최종 HTML을 생성해서 클라이언트에 전달

즉, HTML을 만드는 과정을 서버에서 끝내고 웹에 전달

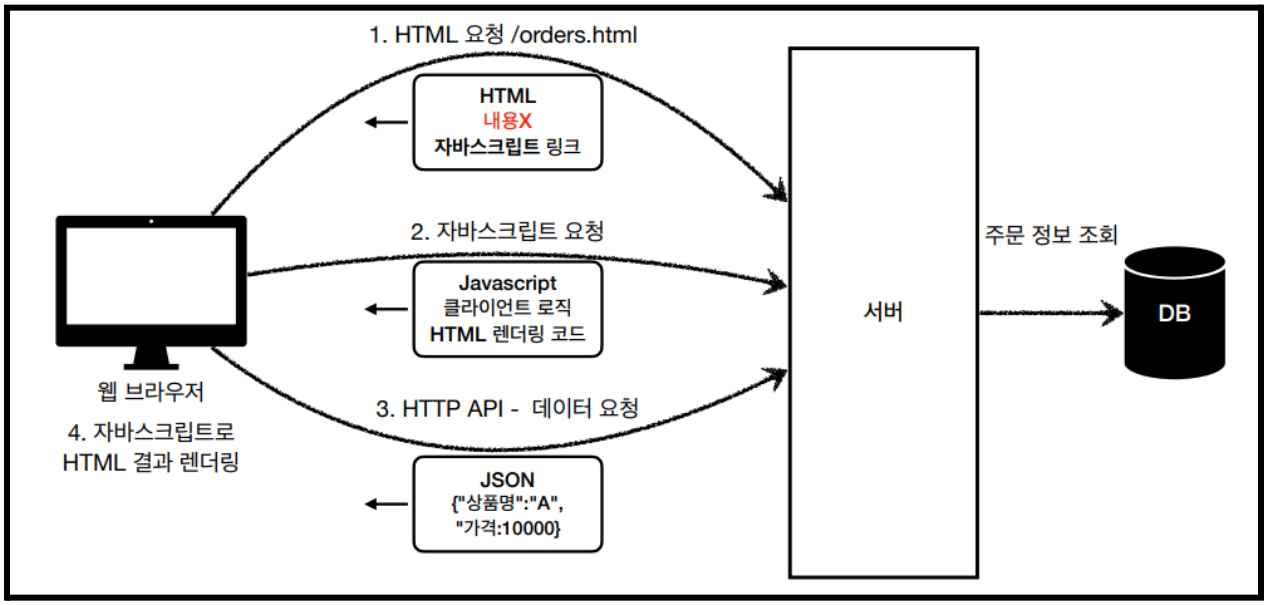
- JSP, 타임리프 등

CSR - 클라이언트 사이드 렌더링

HTML 결과를 자바스크립트를 사용해 웹 브라우저에서 동적으로 생성해서 적용

주로 동적인 화면에 사용, 웹 환경을 마치 앱처럼 필요한 부분부분 변경할 수 있게 하는 것.

- React, Vue.js 등



백엔드 개발자 입장에서 UI 기술

JSP, 타임리프 [화면이 정적이고, 복잡하지 않을 때 사용]

백엔드 개발자는 서버 사이드 렌더링 기술 학습 필수

React, Vue.js [복잡하고 동적인 UI 사용] → 웹 프론트엔드 개발자의 전문분야.

백엔드 개발자의 웹 프론트엔드 기술 학습은 옵션

백엔드 개발자는 서버, DB, 인프라 등등 수 많은 백엔드 기술을 공부해야 한다. 웹 프론트엔드도 깊이있게 잘 하려면 오랜 시간이 필요한다.