



01. 웹 서버, 웹 애플리케이션 서버

| | |
|---------|-----------------------|
| 👤 생성자 | 👤 박지민 |
| 🕒 생성 일시 | @2025년 3월 19일 오전 9:21 |

◆ HTTP의 역할

HTTP는 클라이언트와 서버 간의 데이터 전송을 위한 기본 프로토콜.

클라이언트(예: 웹 브라우저)와 서버(웹 서버, WAS)가 소통하는 방식은 HTTP 메시지를 주고받는 것.

📌 HTTP로 전송할 수 있는 데이터 종류

- **HTML, TEXT** → 웹 페이지 문서
- **IMAGE, 음성, 영상, 파일** → 멀티미디어 콘텐츠
- **JSON, XML** → API 데이터 형식
- **서버 간 데이터 교환** → HTTP를 사용하여 시스템 간 연동 가능

결론:

과거에는 다양한 프로토콜이 사용되었으나, 현재는 대부분 HTTP 기반.

"지금은 HTTP 시대"라는 표현이 나오는 이유.

◆ 웹 서버(Web Server)와 웹 애플리케이션 서버(WAS)의 차이

웹 서버와 WAS는 유사한 개념처럼 보이지만, 역할이 다름.

1 웹 서버(Web Server)

- HTTP 요청을 받아 **정적 리소스**(파일) 제공
- HTML, CSS, JS, 이미지, 영상 등의 콘텐츠 응답
- 추가 기능으로 로드 밸런싱, 캐싱, 압축 제공
- 예) **NGINX, Apache**

2 웹 애플리케이션 서버(WAS - Web Application Server)

- 웹 서버의 기능을 포함하면서, **애플리케이션 로직 실행**
- HTTP API, 동적 HTML 페이지 생성
- 서블릿, JSP, 스프링 MVC 등 웹 애플리케이션 동작
- 예) **Tomcat, Jetty, Undertow**

3 웹 서버와 WAS의 차이

- 웹 서버는 **정적 리소스** 제공, WAS는 **애플리케이션 로직** 처리
- 웹 서버도 일부 프로그램 실행 가능하지만 한계 존재
- WAS도 웹 서버 기능을 포함할 수 있으나, 주 역할은 애플리케이션 로직 실행

◆ 웹 시스템 구성 방식

📌 1. WAS + DB 구성

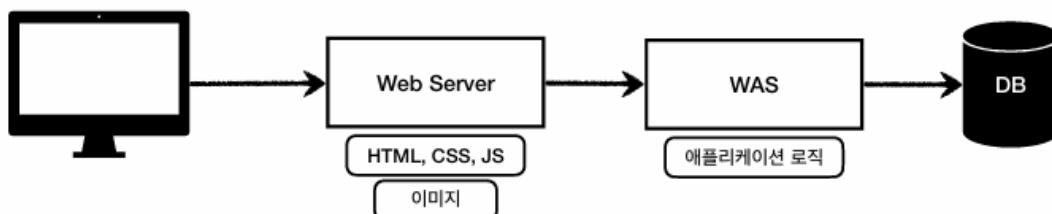
- WAS가 정적 리소스와 애플리케이션 로직 모두 제공
- 단점: WAS의 부담이 커지고, 장애 발생 시 전체 시스템 영향

📌 2. 웹 서버 + WAS + DB 구성

- 정적 리소스는 웹 서버 처리
- 애플리케이션 로직은 WAS 처리
- 트래픽 증가 시 웹 서버(WEB)와 WAS 개별 증설 가능

장점:

- 정적 리소스 요청이 많을 경우, **웹 서버만 추가 증설** 가능
- 애플리케이션 로직이 복잡할 경우, **WAS만 추가 증설** 가능
- WAS나 DB 장애 발생 시, 웹 서버가 기본 오류 페이지 제공 가능
-



◆ 효율적인 리소스 관리

- 웹 서버는 잘 죽지 않음, 단순한 정적 리소스 제공
 - WAS는 장애 발생 가능성 높음, 애플리케이션 로직 실행 부담
 - 정적 리소스 요청 증가 → 웹 서버 증설
 - 애플리케이션 로직 부하 증가 → WAS 증설
-

◆ HTML Form 데이터 전송과 서블릿의 역할

1 기본적인 HTTP 요청 처리 흐름

POST 방식으로 데이터를 전송하면, 서버는 다음과 같은 일련의 작업을 수행해야 함.

📌 서버가 수행하는 작업 흐름

1. TCP/IP 연결 대기, 소켓 연결
2. HTTP 요청 메시지 파싱
3. HTTP 메서드 확인 (POST 방식인지 확인)
4. 요청 URL 확인 (예: `/save` 인지 확인)
5. Content-Type 확인 (예: `application/x-www-form-urlencoded` 등)
6. HTTP 메시지 바디 내용 파싱 (폼 데이터 읽기: `username`, `age` 등)
7. 비즈니스 로직 실행 (데이터 검증, 저장 처리 등)
8. 데이터베이스 저장 요청
9. HTTP 응답 메시지 생성
 - HTTP 시작 라인 생성 (`200 OK` 등)
 - Header 생성
 - 응답 바디에 HTML 또는 JSON 데이터 입력
10. TCP/IP 응답 전달 및 소켓 종료

이 모든 과정을 직접 구현하면 매우 복잡하고 번거로움.

서블릿(Servlet)은 이러한 복잡한 작업을 대신 수행하고, **비즈니스 로직 처리만** 담당할 수 있도록 도와줌.

2 서블릿(Servlet)의 특징

서블릿은 Java 기반의 웹 애플리케이션 개발에서 **HTTP 요청을 쉽게 처리**할 수 있도록 도와주는 객체.

서블릿 기본 코드 예제

```
@WebServlet(name = "helloServlet", urlPatterns = "/hello")
public class HelloServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) {
        // 애플리케이션 로직 처리
    }
}
```

서블릿 특징

- `@WebServlet` 어노테이션을 사용하여 특정 URL 요청을 서블릿과 매핑 가능
- `HttpServletRequest` 객체를 사용하여 HTTP 요청 데이터를 쉽게 가져올 수 있음
- `HttpServletResponse` 객체를 사용하여 HTTP 응답을 쉽게 반환 가능
- 개발자가 HTTP 요청과 응답을 더 편리하게 다룰 수 있도록 도와줌

3 서블릿의 HTTP 요청/응답 처리 흐름

1. 클라이언트가 요청을 보냄 (예: `POST /hello`)
2. WAS (예: 톰캣)가 새로운 `Request` 와 `Response` 객체를 생성
3. 서블릿 컨테이너가 서블릿 객체를 호출하여 요청 처리
4. 개발자는 `HttpServletRequest` 객체에서 요청 데이터를 가져옴
5. 개발자는 `HttpServletResponse` 객체를 통해 응답 데이터를 설정
6. WAS가 `Response` 객체를 사용하여 최종 HTTP 응답을 생성 후 클라이언트에게 전송

4 서블릿 컨테이너(Servlet Container)

서블릿을 실행하는 환경을 **서블릿 컨테이너**라고 하며, 대표적인 예로 **Tomcat**이 있음.

서블릿 컨테이너의 역할

- 서블릿 객체의 **생성, 초기화, 호출, 종료** 생명주기 관리
- 서블릿 객체를 **싱글톤(Singleton)** 으로 관리하여 효율적인 요청 처리
- **멀티 쓰레드 처리 지원**, 동시에 여러 요청을 처리 가능
- JSP 파일도 내부적으로 서블릿으로 변환되어 실행

싱글톤 방식의 장점

- 매 요청마다 객체를 생성하면 성능 저하
- 초기 로딩 시점에 서블릿 객체를 한 번만 생성
- 모든 요청이 같은 서블릿 인스턴스를 사용하여 처리
- 공유 변수 사용 주의 필요 (멀티쓰레드 환경이므로 동기화 고려 필요)

◆ 동시 요청과 멀티 쓰레드의 작동 원리

웹 애플리케이션은 많은 사용자가 동시에 접속하여 요청을 보냄.

이 요청을 효율적으로 처리하려면 **멀티 쓰레드**가 필요함.

멀티 쓰레드를 사용하지 않으면, 한 번에 하나의 요청만 처리 가능하여 성능이 크게 저하됨.

1 기본 개념: 쓰레드(Thread)란?

- 프로그램 실행의 기본 단위
- 프로그램이 실행되면 기본적으로 "메인 쓰레드"에서 코드가 실행됨.
- 멀티 쓰레드 환경에서는 여러 개의 쓰레드가 동시에 실행 가능.

쓰레드의 특징

1. **순차 실행**: 기본적으로 쓰레드는 하나의 코드 라인만 실행 가능.
2. **멀티 쓰레드**: 동시에 여러 개의 쓰레드가 실행되어 여러 작업을 동시에 처리 가능.
3. **CPU 활용**: CPU가 여러 개의 쓰레드를 빠르게 전환하며 실행(컨텍스트 스위칭).
4. **메모리 공유**: 여러 쓰레드는 하나의 프로세스 내에서 같은 메모리를 공유하여 실행됨.

2 WAS(Web Application Server)에서 쓰레드 사용 방식

웹 애플리케이션 서버(WAS)는 클라이언트의 요청을 받고 **서블릿(Servlet)** 을 실행하여 응답을 반환하는 역할을 함.

이 과정에서 **멀티 쓰레드**가 핵심적으로 사용됨.

✓ 기본 요청 처리 흐름 (싱글 쓰레드)

1. 클라이언트가 요청을 보냄.
 2. **WAS가 요청을 수락하고 새로운 쓰레드(Worker Thread)를 생성.**
 3. 생성된 쓰레드가 서블릿(Servlet)을 실행.
 4. 서블릿에서 비즈니스 로직을 수행하고 응답을 생성.
 5. **응답을 클라이언트에게 반환 후 쓰레드 종료.**
-

3 단일 쓰레드 환경 vs 멀티 쓰레드 환경

✓ 1. 단일 쓰레드 환경

- 하나의 요청이 끝날 때까지 다음 요청은 **대기해야 함**.
- 예를 들어, 사용자가 A, B, C가 순서대로 요청을 보내면:
 - 1 A 요청 → 처리
 - 2 A 요청 완료 후 → B 요청 처리
 - 3 B 요청 완료 후 → C 요청 처리
- 많은 사용자가 동시 요청하면 **병목 현상 발생 (성능 저하, 응답 지연)**.

✓ 2. 멀티 쓰레드 환경

- 요청마다 새로운 쓰레드를 생성하여 동시에 여러 요청을 처리.
 - 예를 들어, A, B, C 사용자가 동시 요청 시:
 - 1 A 요청 →
쓰레드1 처리
 - 2 B 요청 →
쓰레드2 처리
 - 3 C 요청 →
쓰레드3 처리
 - 동시에 여러 요청을 처리하여 응답 속도 향상.
-

4 WAS의 멀티 쓰레드 처리 방식

✓ 1. 요청마다 스레드 생성

- 클라이언트 요청마다 새로운 스레드 생성하여 처리.
- 동시 요청을 처리할 수 있으므로 속도가 빠름.
- 하지만 문제점 발생:



스레드 생성 비용이 큼 (CPU, 메모리 소모).



너무 많은 요청이 발생하면
서버 과부하로 다운될 위험.

✓ 2. 스레드 풀(Thread Pool) 사용 (최적화)

스레드 풀(Thread Pool)은 미리 일정 개수의 스레드를 생성하여 관리하는 방식.

WAS(예: 톰캣, 제티, 언더토우)는 기본적으로 스레드 풀을 사용함.

5 스레드 풀(Thread Pool)의 원리

📌 스레드 풀이 어떻게 동작하는지 이해하려면, 요청이 들어올 때의 과정을 보자.

✓ 스레드 풀의 동작 과정

- 1 클라이언트가 요청을 보냄.
- 2 WAS는 스레드 풀에서 기존에 생성된 스레드 중 하나를 할당.
- 3 할당된 스레드는 요청을 처리하고 응답을 반환.
- 4 처리 후, 스레드는 다시 스레드 풀로 반환되어 재사용 가능.
- 5 만약 모든 스레드가 사용 중이면?
 - 새로운 요청은 대기 또는 거절될 수 있음.

6 스레드 풀의 장단점

✓ 장점

- 성능 향상: 새로운 스레드를 계속 생성하는 비용 절감.
- 안정성 확보: 과부하 방지 (최대 스레드 개수 제한).
- 빠른 응답 속도: 미리 생성된 스레드를 사용하여 요청을 빠르게 처리 가능.

✗ 단점

- 스레드 개수가 적으면? 요청이 대기하는 시간이 길어짐.
 - 스레드 개수가 많으면? CPU, 메모리 사용량 증가 → 서버 다운 위험.
 - 적절한 스레드 개수 설정이 중요
-

7 스레드 풀 최적화 (실무 적용)

✓ WAS의 주요 튜닝 포인트: 최대 스레드 개수 설정

- 최대 스레드 개수를 너무 낮게 설정하면?
 - 동시 요청이 많을 경우 **응답이 느려짐**.
 - 서버 자원은 여유롭지만, 요청이 처리되지 않아 **사용자 경험이 저하됨**.
- 최대 스레드 개수를 너무 높게 설정하면?
 - 동시 요청 처리량은 증가하지만, **CPU, 메모리 사용량 급증**.
 - 과부하로 서버 다운 가능.

✓ 최적의 스레드 개수 찾는 방법

1. 애플리케이션 로직의 복잡도 분석

- CPU 연산이 많은가?
- DB I/O가 많은가?
- 네트워크 요청이 많은가?

2. 성능 테스트 진행

- 최대한 실제 서비스 환경과 유사한 테스트 진행.
- 사용 도구: **Apache ab, JMeter, nGrinder**

3. CPU & 메모리 리소스 고려하여 조정

- CPU 코어 수, 메모리 사용량 분석 후 적정 값 설정.
-

8 실제 WAS(톰캣)의 스레드 풀 설정 예시


✓ 기본 설정 (최대 200개의 스레드)

톰캣 설정 파일(`server.xml`)에서 스레드 풀 크기 변경 가능.


```
<Connector port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000"
    maxThreads="200"/>
```

설정 설명

- `maxThreads="200"` → 최대 200개의 스레드만 사용.
- `connectionTimeout="20000"` → 클라이언트가 20초 동안 응답을 기다림.

 실무에서는 서비스 환경에 따라 최적의 `maxThreads` 값을 설정해야 함.

◆ 정적 리소스와 동적 HTML, HTTP API 개념 정리

웹 애플리케이션에서 서버는 다양한 방식으로 클라이언트에 데이터를 제공함.

제공 방식에 따라 정적 리소스, 동적 HTML 페이지, HTTP API로 나뉨.

1 정적 리소스 (Static Resources)

특징

- 미리 저장된 고정된 파일을 제공.
- HTML, CSS, JS, 이미지, 영상 등 포함.
- 주로 웹 브라우저에서 요청하여 사용됨.
- WAS에서 제공 가능하지만, 웹 서버(NGINX, Apache)에서 처리하는 것이 성능상 유리.

예제

```
<!-- 정적 HTML 페이지 -->
<!DOCTYPE html>
<html>
<head>
  <title>Static Page</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>정적 페이지</h1>
```

```
</body>
</html>
```

```
/* CSS 파일 */
body {
  background-color: lightgray;
}
```

```
// 정적 JavaScript 파일
console.log("정적 페이지에서 실행되는 JS 코드");
```

💡 정적 리소스는 서버에서 가공하지 않고, 그대로 클라이언트에 전달됨.

2 동적 HTML 페이지 (서버 사이드 렌더링, SSR)

📌 특징

- 요청이 올 때마다 서버에서 **동적으로 HTML**을 생성하여 응답.
- **JSP, Thymeleaf(타임리프)** 같은 템플릿 엔진을 사용.
- 백엔드 개발자가 주로 사용하는 방식.

📌 SSR 작동 방식

1. 클라이언트가 `/orders` 요청.
2. 서버에서 `orders.html` 을 동적으로 생성하여 반환.
3. 웹 브라우저가 받은 HTML을 화면에 렌더링.

📌 예제 (Thymeleaf - 스프링에서 주로 사용)

```
<!-- Thymeleaf 템플릿 -->
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>주문 목록</title>
</head>
<body>
  <h1>주문 목록</h1>
  <ul>
```

```
<li th:each="order : ${orders}" th:text="${order.name}"></li>
</ul>
</body>
</html>
```

📌 SSR의 장점



SEO(Search Engine Optimization) 최적화 가능 (검색 엔진이 HTML을 직접 읽음).



서버에서 HTML을 완성해 전달하므로, 브라우저가 추가 작업 없이 빠르게 렌더링 가능.

📌 SSR의 단점



서버 부하 증가 (매 요청마다 HTML 생성).



페이지 전체를 다시 로드해야 하므로 인터랙티브한 UI 구현이 어려움.

3 HTTP API

📌 특징

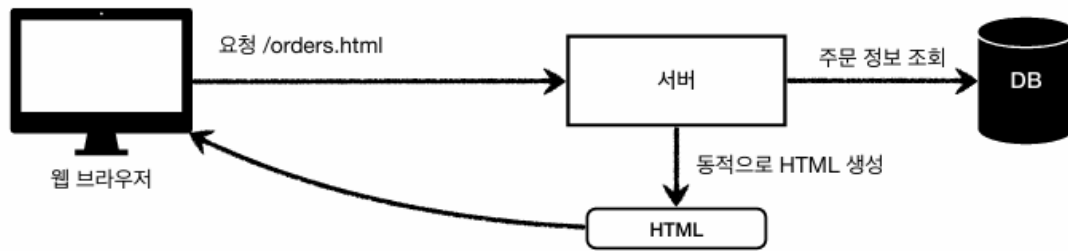
- HTML이 아니라 데이터를 **JSON** 형태로 제공.
- 클라이언트가 요청하면 서버가 데이터를 전달하고, 클라이언트가 UI를 직접 처리.
- 웹 애플리케이션뿐만 아니라 **모바일 앱, 서버 간 통신**에도 사용됨.
- 다양한 시스템에서 호출
- 앱, 웹 클라이언트, 서버 to 서버

4 서버 사이드 렌더링(SSR) vs 클라이언트 사이드 렌더링(CSR)



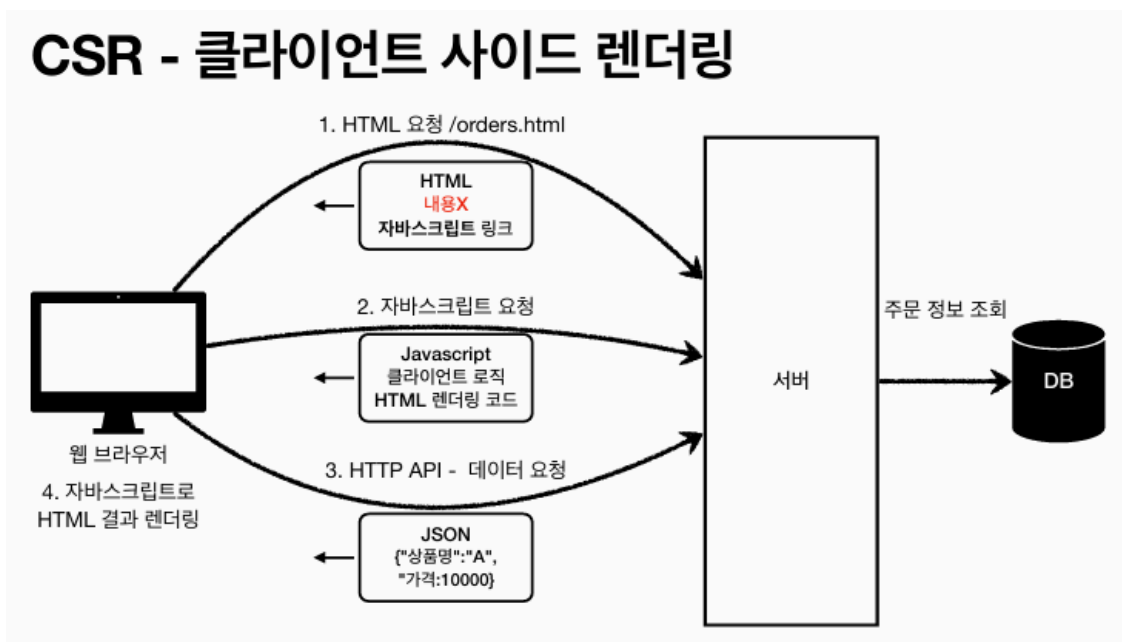
1. SSR (서버 사이드 렌더링)

- 서버에서 **HTML**을 완성한 후 클라이언트에 제공.
- 브라우저는 **받은 HTML**을 바로 렌더링.
- **정적인 페이지**에 적합 (예: 블로그, 뉴스 사이트).
- 관련 기술: **JSP, Thymeleaf**



✓ 2. CSR (클라이언트 사이드 렌더링)

- 브라우저에서 자바스크립트로 HTML을 동적으로 생성.
- **React, Vue.js** 같은 프론트엔드 라이브러리 사용.
- 앱처럼 동적인 UI 개발 가능.
- **SPA(Single Page Application)** 방식에서 많이 사용.



5 백엔드 개발자가 알아야 할 UI 기술

✓ 필수 (백엔드 개발자)

- **SSR (서버 사이드 렌더링)**
 - JSP, Thymeleaf 같은 서버 기반 템플릿 엔진.
 - 단순한 정적 화면을 만들 때 유용.

✓ 선택 (웹 프론트엔드 개발자)

- **CSR (클라이언트 사이드 렌더링)**
 - React, Vue.js 같은 프론트엔드 프레임워크 사용.
 - 동적인 UI 개발이 필요한 경우 적용.

✓ 최신 트렌드

- SSR + CSR 혼합 사용 가능 (Next.js, Nuxt.js 등).
- React, Vue.js에서도 **서버 사이드 렌더링을 지원**하여 SEO 문제 해결.

◆ 자바 백엔드 웹 기술 역사 정리

자바 기반의 웹 기술은 ****서블릿(Servlet)****부터 시작해 **MVC 프레임워크**, 그리고 현재의 **스프링 부트(Spring Boot)**, ****웹 플럭스(WebFlux)****까지 발전해옴.

각 시기의 기술들을 차례로 정리해보자.

1 과거 기술 (1997 ~ 2010년 초)

✓ 서블릿(Servlet) - 1997년 등장

- Java로 웹 애플리케이션을 개발하기 위한 첫 번째 기술.
- 클라이언트의 요청을 받고, 직접 **HTML을 생성하여 응답**.
- **문제점**: HTML을 문자열로 직접 생성해야 해서 개발이 어려움.
- **한계**: 비즈니스 로직과 UI 코드가 섞여 유지보수 어려움.

✓ JSP(Java Server Pages) - 1999년 등장

- 서블릿의 단점을 보완하여 HTML 안에서 Java 코드를 사용할 수 있도록 함.
- HTML을 쉽게 만들 수 있지만, **비즈니스 로직과 뷰 코드가 혼재됨**.
- 유지보수 어려움.

✓ 서블릿 + JSP 조합: MVC 패턴 사용

- 서블릿: 비즈니스 로직 처리
- JSP: HTML 렌더링

- 역할을 분리하여 개발하는 **MVC 패턴** 도입.

✓ **JSP가 HTML을 담당하고, 서블릿이 로직을 처리하는 구조.**

2 MVC 프레임워크 등장 (2000년대 ~ 2010년 초)

✓ MVC 프레임워크의 필요성

- 서블릿 + JSP 조합의 문제점
 - ✗ 컨트롤러 코드가 많아지고 중복 증가.
 - ✗ 복잡한 로직이 많아질수록 유지보수가 어려움.

📌 **해결 방법: MVC 패턴을 자동화하는 프레임워크 등장**

- **Struts (스트럿츠)**
- **WebWork**
- **Spring MVC (초기 버전)**

이 시기는 다양한 MVC 프레임워크가 경쟁하던 시기로, **MVC 춘추 전국 시대**라고 불림.

3 현재 사용 기술 (2010년 이후)

✓ 스프링 MVC (Spring MVC) - 애노테이션 기반 등장

- **Struts** 등 다른 MVC 프레임워크가 사라지고, 스프링 MVC가 표준으로 자리 잡음.
- XML 설정을 벗어나 ****애노테이션 기반 (@Controller, @RequestMapping 등)****으로 개발 편의성 증가.

📌 **스프링 MVC 컨트롤러 예제**

```
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String hello(Model model) {
        model.addAttribute("message", "Hello, Spring MVC!");
        return "hello"; // hello.html 템플릿 반환
    }
}
```

✓ @Controller, @GetMapping을 사용하여 간결한 코드 가능.

✓ 스프링 부트(Spring Boot) 등장

과거에는 WAS(Tomcat 등)를 따로 설치하고, War 파일을 배포해야 했음.

하지만, **스프링 부트(Spring Boot)**는 **내장 서버(WAS 포함)**로 실행할 수 있어 개발이 단순해짐.

🔧 Spring Boot 실행 방식

1. `jar` 파일을 실행하면 내장된 Tomcat이 함께 실행됨.
2. 별도의 WAS 설치 없이 애플리케이션 실행 가능.
3. 배포와 운영이 단순해짐.

```
java -jar myapp.jar
```

✓ 스프링 부트 등장으로 배포 과정 단순화됨.

4 최신 기술 (스프링 웹 기술의 분화)

✓ Spring MVC (Web Servlet)

- 전통적인 동기 방식 웹 개발 (서블릿 기반).
 - 대부분의 웹 애플리케이션에서 여전히 **Spring MVC** 사용 중.
-

✓ Spring WebFlux (비동기 웹)

- 비동기 넌블로킹 방식으로 동작.
- 최소한의 쓰레드로 고성능 처리 가능 (I/O 부하가 많은 환경에 적합).
- 기존 서블릿 방식 사용 안 함.

✓ 비동기 방식으로 높은 성능 가능.

✗ 하지만, 기술적 난이도가 높아 실무에서 많이 사용되지 않음.

✗ RDB 지원 부족으로 제한적 사용 (전체 1% 이하 사용률).

5 뷰 템플릿 엔진의 역사

✅ JSP → 프리마커(Freemarker) / 벨로시티(Velocity) → 타임리프(Thymeleaf)

- JSP: 속도 느림, 기능 부족.
- Freemarker, Velocity: 속도 문제 해결, 기능 강화.
- Thymeleaf: HTML 형태 유지 가능 (내추럴 템플릿), 스프링 MVC와 강력한 연동 가능.

📌 Thymeleaf 예제

```
<html xmlns:th="http://www.thymeleaf.org">
<body>
  <h1 th:text="${message}">기본 메시지</h1>
</body>
</html>
```

✅ HTML 구조를 유지하면서 동적 데이터를 주입할 수 있어 유지보수 용이.
