이론, 어노테이션 위주 그리고 GET, POST 등 메소드 방식들 알아두기

Build.gradle 알아두기

웹 서버와 웹 애플리케이션 서버의 차이점(정적 동적 처리의 차이이다.)

| 구분 | 웹 서버(Web Server) | 웹 애플리케이션 서버(WAS) |
|---|---|---|
| 역할 | 정적(Static) 콘텐츠 제공 | 동적(Dynamic) 콘텐츠 처리 |
| 처리 방식 | HTML, CSS, JS, 이미지 파일 전달 | 사용자 입력을 받아 데이터를 처리 후 응답 생성 |
| 데이터베이스 연동 | ✕ 없음 | ✅ 있음 (DB 연결 가능) |
| 주요 예제 | Apache, Nginx, IIS | Tomcat, WebLogic, JBoss |
| 속도 | 빠름 (파일만 전달) | 상대적으로 느림 (비즈니스 로직 처리) |
| 사용 목적 | 정적인 웹사이트 | 로그인, 쇼핑몰, 게시판 등 동적인 웹 서비스 |

두 가지의 서버는 밑 설명처럼 함께 사용하여 성능을 높일 수 있다.

사용자가 웹 페이지에 접속해서 요청을 보내면 웹 서버에서 요청을 받아서

정적인 요청은 바로 처리해주고 동적인 요청은 WAS(tomcat, spring boot)로 전달한다.

WAS – Tomcat, WebLogin, JBoss, WildFly, GlassFish

웹 프로그래밍

www 월드 와이드 웹 : 영국의 컴퓨터 과학자 팀 버너스리가 고안

네트워크 – 컴퓨터와 컴퓨터 연결 망이다

프로토콜 – 네트워크 구축을 위한 컴퓨터 간 연결 규격 보통 TCP/IP를 인터넷 프로토콜로 사용한다. 포트 및 프로토콜이 다르면 통신할 수 없다.

| 서비스 이름 | 기능 | 프로토콜 | 포트 |
|---|---|---|---|
| 웹www | 웹 서비스 | HTTP/HTTPS | 80/443 |
| 이메일Email | 이메일 서비스 | SMTP/POP3/IMAP | 25/110/143 |
| FTP | 파일 전송 서비스 | FTP | 21 |
| Telnet/SSH | 원격 로그인 서비스 | TELNET/SSH | 23/22 |
| DNS | 도메인 이름 변환 서비스 | DNS | 53 |

정적 페이지 : 변화 없음

동적 페이지 : 변화 있음(데이터를 삽입하여 동적으로 변화한다.) ASP, PHP, JSP 등의 언어가 있음

ASP

윈도우의 IIS 웹 서버에서 실행될 수 있는 웹 프로그래밍 언어를 말함

VBScript를 이용한 비교적 쉬운 스크립트 언어

과거에는 많은 인기를 얻었지만 최근에는 거의 사용하지 않으며, ASP를 만든 마이크로소프트사(MS)에서도 더 이상 지원하지 않음

PHP

C를 기반으로 만들어진 웹 프로그래밍 언어로 빠른 실행 속도와 입문자가 배우기 쉬움 대규모 시스템을 개발하기에 적합하지 않다는 단점이 있음

ASP와 달리 지속적인 지원이 이루어지고 있으며, 최근에는 PHP 8.x로 업그레이드됨

특히 소규모 온라인 쇼핑몰에 많이 적용되어 있음

JSP

자바 Java 기반으로 만들어졌으며 컴파일 이후 서블릿(Servlet)으로 변환되어 서버에서 동적으로 작동함

서블릿으로 변환된 JSP는 사용자의 요청을 스레드를 이용해 처리하기 때문에 서버 부 하 걱정 없이 효율적인 웹 서비스를 제공할 수 있음

Model1 : JSP와 자바빈즈를 이용해서 웹 애플리케이션 개발하는 방법

개발 속도가 비교적 빠르고 초기에는 스트레스를 줄일 수 있다. 하지만 컨트롤러, 뷰 코드가 JSP에 섞여있어 유지보수 어렵다.

Model2 : 컨트롤러와 뷰 분리해서 유지 보수가 쉽다. 각각의 기능을 모듈화하여 기능에 따른 코드를 명확하게 분리하기 때문이다. 요청과 프로그램 제어는 컨트롤러가, 비즈니스 로직은 빈이, 화면에 출력하기 위한 응답은 뷰가 담당한다. MVC 패턴 얘기가 나오면 model2다.

비즈니스 로직은 데이터를 가공하는 로직이라고 생각하면 된다. (내부 업무)

프레젠테이션 로직은 로직 결과를 보여주기 위한 화면 디자인 구성이다.

백엔드 중심 개발

서비스 연동에 필요한 다양한 서버 환경에 대응이 가능하고 검색 엔진 최적화 유리, 안정적이고 검증된 방식이고 오랫동안 유지된다.

모바일 네트워크에서 부적합하다. 새로고침 시 데이터를 다시 다 받아와야 하는 것이 문제다. REST API와 클라우드 인프라가 보편화되면서 기존의 대규모로 서버를 구축하는 모놀리식 아키텍처 방식보다는 소규모 서버를 연동하는 MSA 방식이 확산되는 중이다.

프론트엔드 중심 개발

필요한 부분의 데이터만 갱신이 가능하기 때문에 전체 화면을 받아올 필요가 없고, 실시간 데이터 갱신이 자유롭다. SPA, PWA 등의 구현에 적용할 수 있다.

React, vue 등 라이브러리, 프레임워크 사용 가능하다.

스프링 프레임워크는 자바 기반의 애플리케이션 개발을 위한 오픈소스 프레임워크이다 줄여서 스프링이라고 한다.

스프링 MVC는 스프링을 기반으로 하는 하위 프레임워크, 웹 애플리케이션 제작에 최적화되어 있다.

스프링은 방대하고 다양한 모듈이 있는데 필요한 것들만 가져와서 사용할 수 있는 경량 컨테이너이다. 모듈을 사용하려면 프로젝트에 모듈에 대한 의존 설정을 해야 한다.

대표적인 모듈은 아래 5가지가 있다.

Spring-core : 핵심 기능인 DI(Dependency Injection)과 IoC(제어 역전, Inversion of Control) 기능이 있다.

Spring jdbc : 데이터베이스를 쉽게 이용 할 수 있는 기능을 제공한다.

Spring webmvc : 스프링의 컨트롤러와 뷰를 이용해 스프링 mvc 구현 기능을 제공한다.

Spring tx : 트랜잭션 관련 기능을 제공한다.

Spring security : 보안을 담당하고 관련 기능을 제공한다.

DI : IoC를 구현하는 방법 중 하나이다. 의존성 주입을 외부에서 받는 것이다. 즉 객체 간의 의존성을 외부에서 주입해줌으로써 결합도를 낮출 수 있다.

3가지 방식이 있다

1. 생성자 주입(권장 방식) : 생성자 매개변수로 주입한다. 생성자는 하나만 있어야 한다. 클래스에서 사용하고 싶은 객체 private BookRepository bookRepository; 이렇게 생성한 후 this.bookrepository = bookRepository; 이렇게 사용
   그리고 생성자 주입은 클래스가 생성될 때 의존성 주입이 이루어지기 때문에 객체가 변경되지 않도록 final을 쓰는 것이 좋다.
2. 세터 주입 : set~ 세터처럼 사용하여 위에 Autowired 어노테이션을 사용한다.
   세터 주입에서는 final을 사용하지 않는다. 이유는 의존성 주입이 객체 생성 이후에 이루어지기 때문이다. 세터 특성상 생성자 이후에 주입되기 때문에 불변 변수가 아니어야 이후에 의존성을 주입 받을 수 있다.

| 항목 | 생성자 주입 | 세터 주입 |
| --- | --- | --- |
| **주입 시점** | 객체 생성 시점에 의존성이 주입됨. | 객체 생성 후, 세터 메서드를 호출하여 의존성이 주입됨. |
| `final` **키워드 사용** | `final`을 사용하여 의존성을 **불변** 상태로 설정. | `final`을 사용하지 않음. 의존성 값이 나중에 변경될 수 있음. |
| **불변성** | **불변성** 보장 (한 번 설정된 값은 변경되지 않음). | **불변성** 보장 안 됨 (나중에 값이 변경될 수 있음). |
| **유연성** | 의존성을 **강제적으로** 주입해야 하므로, 선택적 의존성에는 불편함. | **선택적 의존성**을 유연하게 주입할 수 있음. |
| **장점** | 객체가 **완전히 초기화된 후** 사용되므로 **안정성** 높음. | 의존성 주입 순서에 대해 유연함. **옵셔널한 의존성**에 적합. |

3. 필드 주입 : @Autowired를 private BookRepository bookRepository; 이 코드 위에 붙이는 방식이다. 이렇게 하면 BookRepository 객체를 Bean에서 주입해주면 해당 객체에 메서드를 사용할 수 있다.

IoC : 객체의 생성, 의존성 관리(누가 언제 어떤 객체를 생성할지)를 개발자가 직접 하는 것이 아니라 스프링 프레임워크에게 넘겨서 컨테이너가 대신 해주는 것이다. Bean에 등록해서 스프링 컨테이너가 관리해준다.

스프링 내에는 스프링 부트와 MVC가 있다. 위에서 말했듯이 스프링은 자바 기반 애플리케이션 개발을 위한 강력한 오픈소스 프레임워크이다. 부트와 MVC는 주요 프레임워크 요소에 포함되어 있다.

MVC는 model view controller가 있다. Model은 비즈니스 로직과 데이터를 담당하고

View는 사용자가 보는 화면을 담당한다. Jsp, thymeleaf, FreeMarker등의 템플릿 엔진이 있다.

이 템플릿 엔진은 HTML과 같은 정적인 템플릿에 데이터를 동적으로 삽입하여 최종 HTML 문서를 생성하는 도구이다. 즉 백엔드에서 데이터를 받아 동적으로 삽입한 후 최종 HTML 문서를 생성하여 사용자에게 보여주는 역할을 한다.

Controller는 요청을 받아서 처리(모델에 인계)하고 적절한 응답을 반환한다.
요청을 받아서 해당 데이터를 모델에 넘겨주거나 모델에 요청하여 데이터를 받아 사용자에게 돌려주는 역할을 주로 한다.

스프링 부트는 톰캣(WAS)이 내장되어 있어서 별도로 설치할 필요가 없고 복잡한 설정(XML 설정 파일)을 최소화하여 자동 설정을 할 수 있다. JAR 파일을 생성하여 자체 실행이 가능하다.

쉽게 정리하면 스프링 프레임워크 + 내장 웹 서버 – XML 설정파일(이 파일을 제거함으로써 설정이 편리해졌다) 주된 목표는 애플리케이션 개발, 단위 테스트, 통합 테스트에 들어가는 시간을 줄이기 위해서다. 다음에 나오는 주요 기능들이 목표들을 이루어준다.

환경 설정 자동화 : 필요한 라이브러리를 추가하면 부트가 관련된 설정을 자동 처리한다.

종속성 관리 자동화 : 스타터 종속성으로 특정 기능에 필요한 라이브러리 의존성을 간단히 처리한다. 의존성 버전도 권장 버전으로 자동 설정되므로 충돌 없이 쉽게 설정 가능하다.

설정 파일 외부화 : propertires 파일 YAML 파일, 환경 변수, 명령줄 인수 등을 외부화할 수 있어서 애플리케이션을 다양한 환경에서 작동할 수 있다.

라이브러리 버전 관리 자동화 : build.gradle에 스프링 부트 버전을 설정하면 해당 라이브러리와 의존 관계 라이브러리도 호환 버전으로 자동 다운로드 및 관리해준다.

독립형 애플리케이션 생성 : 내장된 톰캣이 실행 가능한 JAR로 애플리케이션을 패키징하여 단독 실행이 가능한 스프링 애플리케이션을 생성할 수 있다.

프로덕션 : 모니터링, 통계, 상태 점검을 제공한다.

Spring Boot Starter

pring-boot-starter-web에 스프링 MVC와 내장 톰캣이 포함되어 있다.

Spring Boot Actuator : 애플리케이션 상태 모니터링 및 관리를 위한 기능을 제공한다.

! build.gradle : 빌드 설정과 의존성 관리를 담당하는 핵심 구성 파일이다.

build.gradle 파일은 Groovy 또는 Kotlin DSL(Domain Specific Language)로 작성되며, 다음과 같은 주요 섹션으로 구성됩니다:

1. **플러그인 설정 (plugins)** 프로젝트에 필요한 기능을 추가합니다. 예를 들어:

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '2.7.11'
    id 'io.spring.dependency-management' version '1.0.15.RELEASE'
}
```

2. **저장소 설정 (repositories)** 의존성 라이브러리를 다운로드할 저장소를 지정합니다. 예:

```
repositories {
    mavenCentral()
}
```

3. **의존성 설정 (dependencies)** 프로젝트에서 사용할 라이브러리들을 선언합니다. 예:

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

4. **태스크 설정 (tasks)** 빌드 과정에서 수행할 작업들을 정의합니다. 예:

```
tasks.register('hello') {
    doLast {
        println 'Hello, Gradle!'
```

```
    }
}
```

### 스프링 부트(Spring Boot)와 스프링 MVC(Spring MVC)의 차이점

| 비교 항목 | 스프링 MVC | 스프링 부트 |
|---|---|---|
| 사용 목적 | 웹 애플리케이션 개발을 위한 프레임워크 | 스프링 애플리케이션을 쉽게 개발할 수 있도록 도와주는 프레임워크 |
| 설정 방식 | 수동 설정 (XML 또는 Java Config 필요) | 자동 설정 (Spring Boot Starter 사용) |
| 내장 웹 서버 | 없음 (Tomcat, Jetty 등 별도 설정 필요) | 있음 (Tomcat 내장) |
| 실행 방식 | WAR 파일로 패키징하여 배포 | JAR 파일로 실행 가능 (독립 실행) |
| 개발 속도 | 상대적으로 설정이 많아 시간이 걸림 | 자동 설정 덕분에 빠르게 개발 가능 |
| 대표적인 스타터 | spring-webmvc | spring-boot-starter-web |

**WAR(Web Application Archive)** 파일은 웹 애플리케이션을 패키징하여 배포하는 파일 형식입니다. WAR 파일은 JAR 파일과 유사하지만, 웹 애플리케이션 실행을 위해 특정한 구조를 갖춘 파일입니다.
**JAR(Java Archive)** 파일은 여러 개의 .class 파일(컴파일된 자바 코드), 리소스 파일(예: HTML, CSS, 이미지), 그리고 META-INF 등의 설정 파일을 포함하는 압축 파일입니다.

MVC 흐름

### 스프링 MVC의 흐름 (1단계)

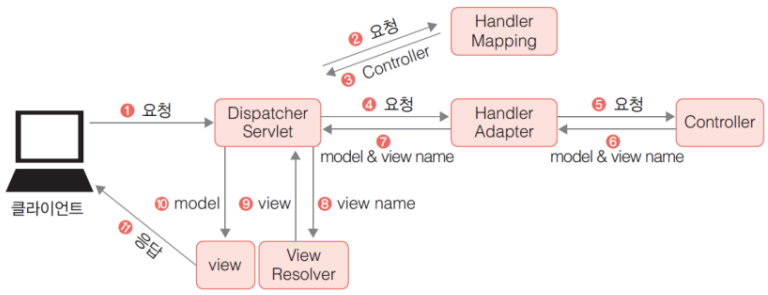


그림 1-11 스프링 MVC 프레임워크의 주요 모듈

클라이언트로부터 요청이 들어오면 DispatcherServlet, HandlerMapping, HandlerAdapter를 이용해 클라이언트 요청에 적합한 컨트롤러를 찾고 해당 컨트롤러의 메서드를 실행합니다.

**클라이언트의 요청**

클라이언트의 요청이 최초 발생(①)되면, 스프링 MVC 프레임워크의 DispacherServlet이 요청을 받습니다.

1단계

클라이언트로부터 요청이 들어오면 DispatcherServlet, HandlerMapping, HandlerAdapter를 이용해 클라이언트 요청에 적합한 컨트롤러를 찾고 해당 컨트롤러 의 메서드를 실행

함

2단계

DispatcherServlet이 클라이언트의 요청을 처리한 결과를 HandlerAdapter로부터 받으 면, 처리 결과를 클라이언트에 응답하기 위해 뷰를 찾는 단계

3단계

클라이언트의 요청에 응답하기 위한 마지막 단계

DispatcherServlet 객체는 ViewResolver 객체가 보내준 뷰 정보를 이용해 뷰 객체를 준비함

개발자가 실제로 코딩하는 객체는 컨트롤러와 뷰 그리고 비즈니스 로직이다.

DispatcherServlet, HandlerMapping, HandlerAdapter 객체는 스프 링 MVC 프레임워크에 이미 만들어져 있다.

뷰 객체는 일반적으로 JSP 파일이며, JSP 파일은 WAS를 통해 클라이언트의 브라우저에 응답 결과를 출력하고 모든 작업을 종료함

<3장> 스프링 부트 구조

src/main/java : 서블릿 관련 폴더, 클래스, 인터페이스 등의 자바 파일 저장. 예외 클래스나 기타 유틸리티 클래스 파일도 여기에 저장된다.

src/main/resource : 웹 관련 폴더, HTML, CSS, Javascript, xml, 환경 설정 파일 등이 저장된다.

static : CSS, js, jpg, png 파일 등 저장된다.

application.properties : 프로젝트 구성하는 요소의 속성을 설정하는 환경 설정 파일, 프로젝트의 환경 변수, 데이터베이스 등을 설정한다.

build.gradle : 빌드 정보를 정의하는 파일이다. 프로젝트의 기본 설정, 라이브러리 의존성, 플러그인, 라이브러리 저장소 등을 설정한다. 관련 의존성, 라이브러리 버전을 호환성 좋은 버전으로 자동 설정 해준다.

Gradle은 groovy를 기반으로 한 오픈소스 빌드 도구이다. 문법이 간결하여 가독성이 높다. 그루비는 Gradle 빌드 스크립트를 작성하는 데 사용하는 스크립트 언어이다.

@SpringBootApplication : 메인 클래스에 반드시 적용되는 어노테이션이다. 스프링 부트 실행에 필요한 어노테이션.

EnableAutoConfiguration : 프로젝트 외부 라이브러리를 빈으로 등록

CompnenetScan : Component, Configuration, Service, Repository, Contoller, RestController 가 적용된 클래스를 탐색하여 빈으로 자동 등록한다.

Configuration(@SpringBootConfiguration) : 빈이나 설정 클래스를 사용자가 추가로 등록한다.

라이브러리 의존성 주입의 유형

implementation: 컴파일할 때 의존(사용)하는 라이브러리를 지정

testCompile: 테스트 컴파일(단위 테스팅)에 사용하는 라이브러리를 지정

classpath: 컴파일부터 실행까지 의존하는 라이브러리를 지정

스프링 부트 계층적 구조

계층적 구조는 코드 복잡성 증가, 유지보수 문제, 유연성, 결합력, 중복 코드, 확장성 문제를 감소시켜주는 구조이다.

---

## ■ 계층적 구조

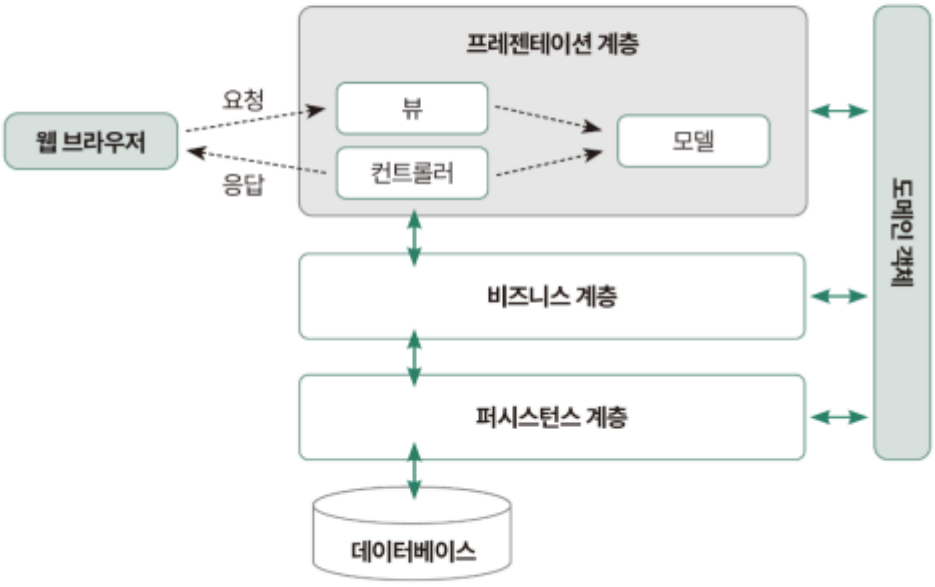


그림 3-2 스프링 부트의 계층적 구조

데이터 베이스와 통신하는 퍼시스턴트는 우리가 사용한 Repository이다.(DAO)

비즈니스 계층은 우리가 사용한 Service 비즈니스 로직 처리 클래스였다.

도메인 객체는 필드 정보 하지만 민감 정보를 처리하기 위해 DTO 거치는 형태를 많이 사용한다.

프레젠테이션 계층은 사용자와 접촉이 이루어지는 곳, 요청을 받아서 처리하고 결과를 클라이언트에 다시 반환한다.

<4장> 컨트롤러 구현(Controller, RequestMapping) 뷰, 모델

<5장> 요청 처리 메서드의 파라미터 유형(RequestParam, PathVariable, MatrixVariable)

<6장> 폼 태그(ModelAttribute, InitBinder)