WAS(Web Application Server)

- 비즈니스 로직을 넣을 수 있음
- Tomcat, PHP, ASP, .NET 등

WS(Web Server)

- 비즈니스 로직을 넣을 수 없음
- Nginx, Apache 등

스프링 프레임워크는 자바 개발을 편리하게 해주는 오픈소스 프레임워크 입니다.

- 경량 컨테이너로서 자바 객체를 직접 관리
  - 각각의 객체 생성, 소멸과 같은 라이프 사이클을 관리하며 스프링으로부터 필요한 객체를 얻어올 수 있다.
- 제어의 역전(IoC)이라는 기술을 통해 어플리케이션의 느슨한 결합을 도모
  - 컨트롤의 제어권이 사용자가 아닌 프레임워크에 있어서 필요에 다라 스프링에서 사용자의 코드를 호출한다.
- 의존성 주입(DI, Dependency Injection)을 지원
  - 각각의 계층이나 서비스들 간에 의존성이 존재할 경우 프레임워크가 서로 연결시켜준다.
- 관점 지향 프로그래밍(AOP, Aspect-Oriented Programming)을 지원
  - 트랜잭션이나 로깅, 보안과 같이 여러 모듈에서 공통적으로 사용하는 기능의 경우 해당 기능을 분리하여 관리할 수 있다.

@RequestBody 는 클라이언트가 전송하는 JSON 형태의 HTTP Body 내용을 MessageConverter를 통해 Java Object로 변환시켜주는 역할을 합니다.
값을 주입하지 않고 값을 변환 시키므로(Reflection을 사용해 할당), 변수들의 생성자, Getter,Setter가 없어도 정상적으로 할당된다.

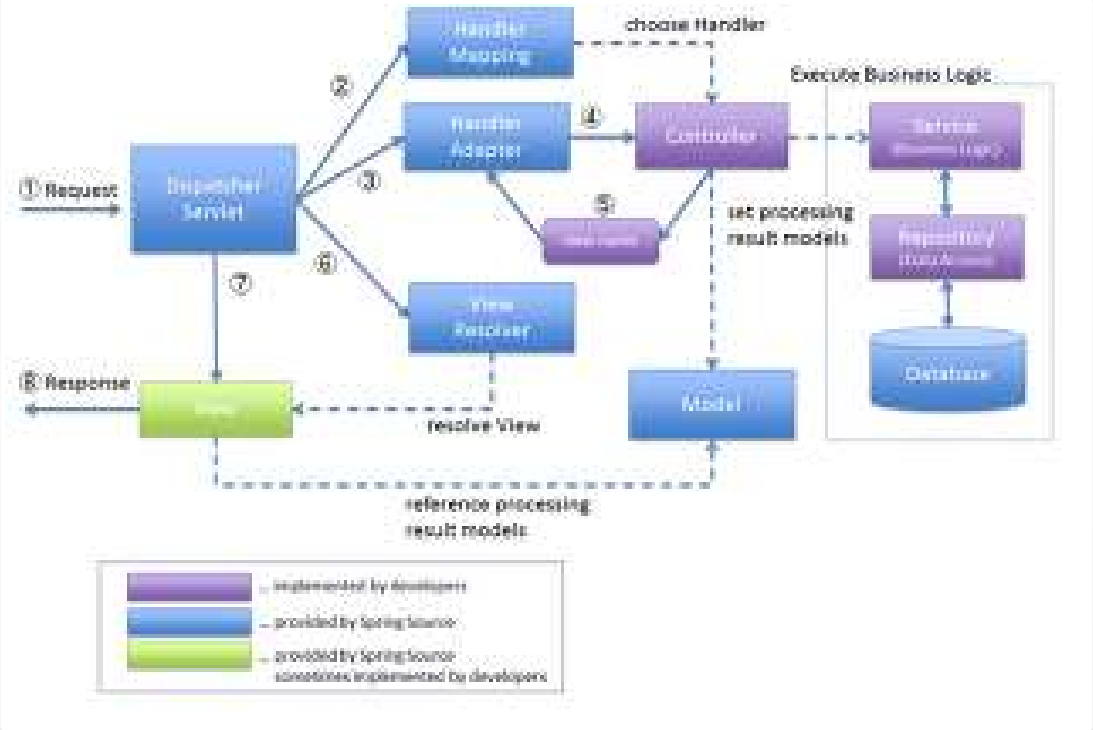
@RequestParam 은 1개의 HTTP 요청 파라미터를 받기 위해 사용합니다. @RequestParam은 필수 여부가 true이기 때문에,
기본적으로 반드시 해당 파라미터가 전송되어야 합니다. 전송되지 않으면 400Error를 유발할 수 있으며,
반드시 필요한 변수가 아니라면 required의 값을 false로 설정해줘야 합니다.

@ModelAttribute 는 HTTP Body 내용과 HTTP 파라미터의 값들을 생성자,Getter,Setter를 통해 주입하기 위해 사용합니다.
값 변환이 아닌 값을 주입시키므로 변수들의 생성자나 Getter,Setter가 없으면 변수들이 저장되지 않는다.

가장 큰 차이점은 Auto Configuration의 차이인 것 같습니다. Spring은 프로젝트 초기에 다양한 환경설정을 해야 하지만,
Spring Boot는 설정의 많은 부분을 자동화하여 사용자가 편하게 스프링을 활용할 수 있도록 돕습니다.
spring boot starter dependency만 추가해주면 설정은 끝나고, 내장된 톰캣을 제공해 서버를 바로 실행할 수 있습니다.

- MVC는 Model, View, Controller의 약자이며, 각 레이어간 기능을 구분하는데 중점을 둔 디자인 패턴입니다.
- Model은 데이터 관리 및 비즈니스 로직을 처리하는 부분이며, (DAO, DTO, Service 등)

- View는 비즈니스 로직의 처리 결과를 통해 유저 인터페이스가 표현되는 구간입니다. (html, jsp, tymeleaf, mustache 등 화면을 구성하기도 하고, Rest API로 서버가 구현된다면 json 응답으로 구성되기도 한다.)
- Controller는 사용자의 요청을 처리하고 Model과 View를 중개하는 역할을 합니다. Model과 View는 서로 연결되어 있지 않기 때문에 Controller가 사이에서 통신 매체가 되어줍니다.



DispatcherServlet : 클라이언트에게 요청을 받아 응답까지의 MVC 처리과정을 통제한다.
HandlerMapping : 클라이언트의 요청 URL을 어떤 Controller가 처리할지 결정한다.
HandlerAdapter : HandlerMapping에서 결정된 핸들러 정보로 해당 메소드를 직접 호출해주는 역할을 한다.
ViewResolver : Controller의 처리 결과(데이터)를 생성할 view를 결정한다.

1. 클라이언트는 URL을 통해 요청을 전송한다.
2. 디스패처 서블릿은 핸들러 매핑을 통해 해당 요청이 어느 컨트롤러에게 온 요청인지 찾는다.
3. 디스패처 서블릿은 핸들러 어댑터에게 요청의 전달을 맡긴다.
4. 핸들러 어댑터는 해당 컨트롤러에 요청을 전달한다.
5. 컨트롤러는 비즈니스 로직을 처리한 후에 반환할 뷰의 이름을 반환한다.
6. 디스패처 서블릿은 뷰 리졸버를 통해 반환할 뷰를 찾는다.
7. 디스패처 서블릿은 컨트롤러에서 뷰에 전달할 데이터를 추가한다.
8. 데이터가 추가된 뷰를 반환한다.

제어의 역전(IoC)란 모든 객체에 대한(생성, 라이프사이클 등) 제어권을 개발자가 아닌 IoC 컨테이너에게 넘긴 것을 말합니다.
스프링에서는 IoC 컨테이너에 객체들을 생성하면 객체끼리 의존성을 주입(DI, Dependency Injection)하는 역할을 하고 컨테이너에 등록한 객체들을 '빈'이라고 합니다.
[Spring] IoC 컨테이너 (Inversion of Control) 란?

빈을 등록하는 방법은 기본적으로 2 가지가 있습니다.

1. 우선 가장 쉬운 방법으로 @Component 어노테이션을 사용하는 것입니다.
@Controller, @Service, @Repository는 모두 @Component를 포함하고 있습니다.

2. 설정 클래스를 따로 만들어 @Configuration 어노테이션을 붙이고,
해당 클래스 안에서 빈으로 등록할 메소드를 만들어 @Bean 어노테이션을 붙여주면 자동으로 해당 타입의 빈 객체가 생성됩니다.
[Spring] 스프링 빈을 등록하는 두 가지 방법(@Component, @Bean)

의존성 주입은 필요한 객체를 직접 생성하는 것이 아닌 외부로부터 객체를 받아서 사용하는 것입니다.
이를 통해 객체간의 결합도를 줄이고 코드의 재사용성을 높일 수 있습니다.

의존성 주입은 생성자 주입, 필드 주입, 세터 주입의 3 가지 방법이 있습니다.
이 중 Spring에서 가장 권장하는 의존성 주입 방법은 생성자를 통한 주입 방법입니다.
그 이유는 1. 순환 참조를 방지 2. 불변성을 가짐 3. 테스트에 용이하기 때문입니다.

[Spring] 의존성 주입 3가지 방법 - (생성자 주입, Field 주입, Setter 주입)

먼저 스프링 Bean의 LifeCycle은 다음과 같습니다.

> *스프링 IoC 컨테이너 생성 → 스프링 빈 생성 → 의존관계 주입 → 초기화 콜백 메소드 호출 → 사용 → 소멸 전 콜백 메소드 호출 → 스프링 종료*

스프링은 크게 3가지 방법으로 빈 생명주기 콜백을 관리합니다.

1. 인터페이스(InitializingBean, DisposableBean)
2. 설정 정보에 초기화 메소드, 종료 메소드 지정
3. @PostConstruct, @PreDestroy 어노테이션 지원

[Spring] 빈 생명주기(Bean LifeCycle) 콜백 자세히 알아보기

필터는 말 그대로 요청과 응답을 거른뒤 정제하는 역할을 합니다.
스프링 컨테이너가 아닌 톰캣과 같은 웹 컨테이너에 의해 관리가 되는 것이고, 스프링 범위 밖에서 처리됩니다.
Dispatcher Servlet에 요청이 전달되기 전 / 후에 url 패턴에 맞는 모든 요청에 대해 부가 작업을 처리할 수 있는 기능을 제공합니다.
사용 사례 :

- 보안 및 인증/인가 관련 작업
- 모든 요청에 대한 로깅 또는 검사
- 이미지/데이터 압축 및 문자열 인코딩
- Spring과 분리되어야 하는 기능

인터셉터는 요청에 대한 작업 전 / 후로 가로채 요청과 응답을 참조하거나 가공하는 역할을 합니다.
웹 컨테이너에서 동작하는 필터와 달리 인터셉터는 스프링 컨텍스트에서 동작합니다.
Dispatcher Servlet이 Controller를 호출하기 전 / 후에 인터셉터가 끼어들어 요청과 응답을 참조하거나 가공할 수 있는 기능을 제공
사용 사례 :

- 세부적인 보안 및 인증/인가 공통 작업
- API 호출에 대한 로깅 또는 검사
- Controller로 넘겨주는 정보(데이터)의 가공

AOP는 핵심 비즈니스 로직에 있는 공통 관심사항을 분리하여 각각을 모듈화 하는 것을 의미하며
공통 모듈인 인증, 로깅, 트랜잭션 처리에 용이합니다.

핵심 비즈니스 로직에 부가기능을 하는 모듈이 중복되어 분포되어 있을 경우 사용할 수 있습니다.

AOP의 가장 큰 특징이자 장점은 중복 코드 제거, 재활용성의 극대화, 변화수용의 용이성이 좋다는 점입니다.

[Spring] AOP(Aspect Oriented Programming)란?

Lombok은 메소드를 컴파일 하는 과정에 개입해서 추가적인 코드를 만들어냅니다. 이것을 어노테이션 프로세싱이라고 하는데,
어노테이션 프로세싱은 자바 컴파일러가 컴파일 단계에서 어노테이션을 분석하고 처리하는 기법을 말합니다.
(Lombok 라이브러리를 추가할 때 CompileOnly, AnnotationProcessor를 추가하는 이유도 된다.)

클라이언트의 요청을 처리하고, 그 결과를 반환하는 Servlet 클래스의 구현 규칙을 지킨 자바 웹 프로그래밍 기술입니다.
Spring MVC에서 Controller로 이용되며, 사용자의 요청을 받아 처리한 후에 결과를 반환합니다.
간단히 - 자바를 사용해 웹을 만들기 위해 필요한 기술

더보기

- DAO(Data Access Object) DB의 데이터에 접근을 위한 객체를 말합니다. (Repository 또는 Mapper에 해당)
- BO(Business Object) 여러 DAO를 활용해 비즈니스 로직을 처리하는 객체를 말합니다. (Service에 해당)
- DTO(Data Transfer Object) 각 계층간의 데이터 교환을 위한 객체를 말합니다. (여기서 말하는 계층은 Controller, View, Business Layer, Persistent Layer)
- VO (Value Object) 실제 데이터만을 저장하는 객체를 말합니다.

DTO와 VO 그리고 Entity 차이점 알아보기

스케일 업을 통해 하드웨어 스펙을 향상 / 스케일 아웃을 통해 서버를 여러대 추가해 시스템을 증가
스케일 업(Scale-Up)과 스케일 아웃(Scale-Out)이란?

스프링에서 bean 생성시 별다른 설정이 없으면 default로 싱글톤이 적용됩니다.
스프링은 컨테이너를 통해 직접 싱글톤 객체를 생성하고 관리하는데,
요청이 들어올 때마다 매번 객체를 생성하지 않고, 이미 만들어진 객체를 공유하기 때문에 효율적인 사용이 가능합니다.
이를 통해 다음과 같은 장점을 얻을 수 있습니다.

- static 메소드나 private 생성자 등을 사용하지 않아 객체지향적 개발을 할 수 있다.
- 테스트하기 편리하다.

프로토타입 빈은 싱글톤(default bean) 빈과는 달리 컨테이너에게 빈을 요청할 때마다 매번 새로운 객체를 생성하여 반환해줍니다.
이렇게 빈의 scope를 간단하게 관리해줄 수 있는 것이 spring의 장점입니다.
빈의 scope 설정은 @Scope 어노테이션으로 설정하며, 프로토타입 scope로 설정하려면 @Scope("prototype")와 같이 문자열로 지정해줍니다.

@Transactional을 메소드 또는 클래스에 명시하면, AOP를 통해 Target이 상속하고 있는 인터페이스 또는 Target 객체를 상속한 Proxy 객체가 생성되며, Proxy 객체의 메소드를 호출하면 Target 메소드 전 후로 트랜잭션 처리를 수행합니다.

프록시는 클라이언트가 타겟 객체를 호출하는 과정에만 동작하며, 타겟 객체의 메소드가 자기 자신의 다른 메소드를 호출할 때는 프록시가 동작하지 않습니다.
즉, A 메소드는 프록시로 감싸진 메소드가 아니므로 트랜잭션이 적용되지 않은 일반 코드가 수행됩니다.

트랜잭션 전파 수준에 따라 달라지는데, 만약 기본 옵션인 Required를 가져간다면 로컬 트랜잭션 3개가 모두 부모 트랜잭션인 A에 합류하여 수행됩니다.
그래서 부모 트랜잭션이나 로컬 트랜잭션 3개나 모두 같은 트랜잭션이므로 어느 하나의 로직에서 문제가 발생하면 전부 롤백이 됩니다.

Transaction 참고 자료

트랜잭션 안에서 수정/삭제 작업이 아닌 ReadOnly 목적인 경우에 주로 사용하며,
영속성 컨텍스트에서 엔티티를 관리 할 필요가 없기 때문에 readOnly를 추가하는 것으로 메모리 성능을 높일 수 있고,
데이터 변경 불가능 로직임을 코드로 표시할 수 있어 가독성이 높아진다는 장점이 있습니다.
readOnly 속성이 없는 보통의 트랜잭션은 데이터 조회 결과 엔티티가 영속성 컨텍스트에 관리되며,
이는 1차 캐싱부터 변경 감지(Dirty Checking)까지 가능하게 된다.
하지만, 조회시 스냅샷 인스턴스를 생성해 보관하기 때문에 메모리 사용량이 증가한다.

N+1이란 1번의 쿼리를 날렸을 때 의도하지 않은 N번의 쿼리가 추가적으로 실행되는 것을 의미합니다.

해결 방법에는 여러 방법이 있지만 가장 많이 사용되는 방법은 Fetch Join을 사용해 해결하는 방법입니다.

N+1 문제가 발생하는 이유는 연관관계를 가진 엔티티를 조회할 때 한 쪽 테이블만 조회하고 연결된 다른 테이블은 따로 조회하기 때문인데,
Fetch Join을 사용하면 미리 두 테이블을 Join하여 한 번에 모든 데이터를 가져오기 때문에 N+1문제를 애초에 막을 수 있습니다.

[JPA] N+1 문제 원인 및 해결방법 자세히 알아보기

일단 JPA 자체는 정적인 상황에서 사용하는걸 권장하기 때문에 복잡한 쿼리와 동적인 쿼리에 대한 문제가 발생하게 되는데,
그럴때는 JPQL과 QuerydsI을 사용할 것을 권장하고 있습니다.