spring_boot_annotation

- description : spring boot annotation
- author : 도봉산핵주먹
- email : hylee@repia.com
- lastupdate : 2020-08-21

Annotation

A

B

C

> @ComponentScan

- @Component와 @Service, @Repository, @Controller, @Configuration이 붙은 클래스 Bean들을 찾아서 Context에 bean등록을 해주는 Annotation
- @Component로 다 쓰지 왜 굳이 @Repository, @Service, @Controller등을 사용하냐면 예를들어 @Repository는 DAO의 메소드에서 발생할 수 있는 unchecked exception들을 스프링의 DataAccessException으로 처리할 수 있기 때문이다. 또한 가독성에서도 해당 애노테이션을 갖는 클래스가 무엇을 하는지 단 번에 알 수 있다.

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

@EnableAutoConfiguration

- 스프링 애플리케이션 컨텍스트를 만들 때 자동으로 설정하는 기능을 켠다.

classpath의 내용에 기반해서 자동 생성해준다. 만약 tomcat-embed-core.jar가 존재하면 톰캣 서버가 setting된다.

@Configuration

- Configuration을 클래스에 적용하고 @Bean을 해당 클래스의 메소드에 적용하면 @Autowired로 빈을 부를 수 있다.

@Required

- setter 메서드에 적용해주면 빈 생성시 필수 프로퍼티 임을 알린다.

@Qualifier(“id123”)

- @Autowired와 같이 쓰이며, 같은 타입의 빈 객체가 있을 때 해당 아이디를 적어 원하는 빈이 주입될 수 있도록 하는 애노테이션

(같은 타입이 존재하는 경우 ex) 동물, 원숭이, 닭, 개, 돼지)

@Resource

- @Autowired와 마찬가지로 빈 객체를 주입해주는데 차이점은 Autowired는 타입으로, Resource는 이름으로 연결해준다.

@PostConstruct, @PreConstruct

- 의존하는 객체를 생성한 이후 초기화 작업을 위해 객체 생성 전/후에(pre/post) 실행해야 할 메소드 앞에 붙인다.

@PreDestroy

- 객체를 제거하기 전(pre)에 해야할 작업을 수행하기 위해 사용한다.

@PropertySource

- 해당 프로퍼티 파일을 Environment로 로딩하게 해준다.

클래스에 @PropertySource(“classpath:/settings.properties”)라고 적고 클래스 내부에 @Resource를 Environment타입의 멤버변수앞에 적으면 매핑된다.

@ConfigurationProperties

- yaml파일 읽는다. default로 classpath:application.properties 파일이 조회된다.

속성 클래스를 따로 만들어두고 그 위에 (prefix=“mail”)을 써서 프로퍼티의 접두사를 사용할 수 있음

mail.host = mailserver@mail.com

mail.port = 9000

mail.defaultRecipients[0] = admin@mail.com

mail.defaultRecipients[1] = customer@mail.com

@Lazy

- 지연로딩을 지원한다.

@Component나 @Bean 애노티에션과 같이 쓰는데 클래스가 로드될 때 스프링에서 바로 bean등록을 마치는 것이 아니라 실제로 사용될 때 로딩이 이뤄지게 하는 방법이다.

@Value

- properties에서 값을 가져와 적용할 때 사용한다.

@Value(“${value.from.file}”)

private String valueFromFile; 이라고 구성되어 있으면 value.from.file의 값을 가져와서 해당 변수에 주입해준다.

spEL을 이용해서 조금 더 고급스럽게 쓸 수 있다.

@Value(#{systemProperties['priority'] ?: 'some default'})

@SpringBootApplication

- @Configuration, @EnableAutoConfiguration, @ComponentScan 3가지를 하나의 애노테이션으로 합친 것이다.

@RequestMapping

- 요청 URL을 어떤 메서드가 처리할지 mapping해주는 애노테이션이다.

컨트롤러나 컨트롤러의 메서드에 적용한다.

@RequestMapping(“/list”), @RequestMapping(“/home, /about”);

@RequestMapping(“/admin”, method=RequestMethod.GET)

@CookieValue

- 쿠기 값을 파라미터로 전달 받을 수 있는 방법

해당 쿠기가 존재하지 않으면 500 에러를 발생시킨다.

속성으로 required가 있는데 default는 true다. false를 적용하면 해당 쿠키 값이 없을 때 null로 받고 에러를 발생시키지 않는다.

public String view(@CookieValue(value=“auth”)String auth){...}; *쿠키의 key가 auth에 해당하는 값을 가져옴 @CrossOrigin - CORS 보안상의 문제로 브라우저에서 리소스를 현재 origin에서 다른 곳으로의 AJAX요청을 방지하는 것이다. @RequestMapping이 있는 곳에 사용하면 해당 요청은 타 도메인에서 온 ajax요청을 처리해준다. @CrossOrigin(origins = “http://jeong-pro.tistory.com”, maxAge = 3600) → 기본 도메인이 http://jeong-pro.tistory.com 인 곳에서 온 ajax요청만 받아주겠다. @ModelAttribute - view에서 전달해주는 파라미터를 클래스(VO/DTO)의 멤버 변수로 binding 해주는 애노테이션 바인딩 기준은 <input name=“id” /> 처럼 어떤 태그의 name값이 해당 클래스의 멤버 변수명과 일치해야하고 set메서드명도 일치해야한다. class Person{ String id; public void setId(String id){ this.id = id;} } @GetMapping - @RequestMapping(Method=RequestMethod.GET)과 같음 @PostMapping, @PutMapping, @PatchMapping, @DeleteMapping은 유추 가능함. @SessionAttributes - 세션에 데이터를 넣을 때 쓰는 애노테이션 @SessionAttributes(“name”) 이라고 하면 Model에 key값이 “name”으로 있는 값은 자동으로 세션에도 저장되게 한다. @Valid - 유효성 검증이 필요한 객체임을 지정한다. @InitBinder - @Valid 애노테이션으로 유효성 검증이 필요하다고 한 객체를 가져오기전에 수행해야할 메서드를 지정한다. @RequestAttribute - Request에 설정되어 있는 속성 값을 가져올 수 있다. @RequestBody - 요청이 온 데이터(JSON이나 XML형식)를 바로 클래스나 model로 매핑하기 위한 애노테이션 @RequestHeader - Request의 header값을 가져올 수 있다. 메소드의 파라미터에 사용 @RequestHeader(value=“Accept-Language”)String acceptLanguage 로 사용* ko-KR,ko;q=0.8,en-US;q=0.6

@RequestParam

- @PathVariable과 비슷하다. request의 parameter에서 가져오는 것이다. 메소드의 파라미터에 사용됨

@RequestPart

- Request로 온 MultipartFile을 바인딩해준다.

@RequestPart(“file”)MultipartFile file로 받아올 수 있음.

@ResponseBody

- view가 아닌 JSON 형식의 값을 응답할 때 사용하는 애노테이션으로 문자열을 리턴하면 그 값을 http response header가 아닌 response body에 들어간다.

만약 객체를 return하는 경우 JACKSON 라이브러리에 의해 문자열로 변환되어 전송된다.

context에 설정된 resolver를 무시한다고 보면된다. (viewResolver)

@PathVariable

- 메서드 파라미터 앞에 사용하면서 해당 URL에서 {특정값}을 변수로 받아 올 수 있다.

@ExceptionHandler(ExceptionClassName.class)

- 해당 클래스의 예외를 캐치하여 처리한다.

@ControllerAdvice

- 클래스 위에 ControllerAdvice를 붙이고 어떤 예외를 잡아낼 것인지는 각 메소드 상단에 @ExceptionHandler(에외클래스명.class)를 붙여서 기술한다.

@RestControllerAdvice

- 문법적 설탕으로 @ControllerAdvice + @ResponseBody다.

@ResponseStatus

- 사용자에게 원하는 response code와 reason을 리턴해주는 애노테이션

@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = “my page URL changed..”) ⇒ 예외처리 함수 앞에 사용한다.

@EnableEurekaServer

- Eureka 서버로 만들어준다.

@EnableDiscoveryClient

- Eureka 서버에서 관리될 수 있는 클라이언트 임을 알려주기위한 애노테이션

@Transactional

- 데이터베이스 트랜잭션을 설정하고 싶은 메서드에 애노테이션을 적용하면 메서드 내부에서 일어나는 데이터베이스 로직이 전부 성공하게되거나 데이터베이스 접근중 하나라도 실패하면 다시 롤백할 수 있게 해주는 애노테이션

@Transaction(readOnly=true, rollbackFor=Exception.class) : readOnly는 읽기전용임을 알리고 rollbackFor는 해당 Exception이 생기면 롤백하라는 뜻

@Transaction(noRollbackFor=Exception.class)는 해당 Exception이 나타나도 롤백하지 말라는 뜻

@Transaction(timeout = 10)은 10초안에 해당 로직을 수행하지 못하면 롤백하라는 뜻.

@Cacheable

- 메서드 앞에 지정하면 해당 메서드를 최초에 호출하면 캐시에 적재하고 다음부터는 동일한 메서드 호출이 있을 때 캐시에서 결과를 가져와서 리턴하므로 메서드호출 횟수를 줄여주는 애노테이션

주의할 점은 입력이 같으면 항상 출력이 같은 메서드(=순수 함수)에 적용해야한다.

그런데 또 항상 같은 값만 뱉어주는 메서드에 적용하려면 조금 아쉬울 수 있다. 따라서 메서드 호출에 사용되는 자원이 많고 자주 변경되지 않을 때 사용하고 나중에 수정되면 캐시를 없애는 방법을 선택할 수 있다.

@Cacheable(value=“cacheKey”), @Cacheable(key=“cacheKey”)

@CachePut

- 캐시를 업데이트하기 위해서 메서드를 항상 실행하게 강제하는 애노테이션

해당 애노테이션이 있으면 메서드호출을 항상한다. 그러므로 @Cacheable과 상충되어 같이 사용하면 안된다.

@CacheEvict

- 캐시에서 데이터를 제거하는 트리거로 동작하는 메소드

캐시된 데이터는 언제가는 지워져야한다. 그러지 않으면 결과값이 변경이 일어났는데도 기존의 데이터(캐시된 데이터)를 불러와서 오류가 발생할 수 있다.

물론 캐시 설정에서 캐시 만료시간을 줄 수도 있다.

@CacheEvict(value=“cacheKey”), @CacheEvict(value=“cacheKey”, allEntries=true)

allEntries는 전체 캐시를 지울지 여부를 선택하는 것이다.

@CacheConfig

- 클래스 레벨에서 공통의 캐시설정을 공유하는 기능이다.

@Scheduled

- Linux의 crontab처럼 뭔가 정해진 시간에 실행해야하는 경우에 사용한다.

@Scheduled(cron = “0 0 07 * * ?”) “초 분 시 일 월 요일 년(선택)에 해당 메서드 호출

도봉산핵주먹, Springboot, annotation

From:
http://rwiki.repia.com/ - **문제를 잘 정의하는 것은 문제를 절반 해결한 것이다. - 2023.12**

Permanent link:
http://rwiki.repia.com/doku.php?id=wiki:springboot:spring_boot_annotation:spring_boot_annotation&rev=1597983298

Last update: **2022/03/10 19:52**