一、Spring Bean相关

#### @Autowired

自动导入对象到类中，被注入进的类同样要被Spring容器管理：

比如：Service类注入到Controller类中。

#### @Component、@Repository、@Service、@Controller

一般使用@Autowired让Spring容器帮我们自动装配bean。要想把类标识成可用于@Autowired自动装配的bean的类，可以采用以下注解：

@Component：可标注任意类为Spring组件。如果一个Bean不知道属于那个层，可以使用

@Repository：对应持久层即Dao层，主要用于数据库相关操作。

@Service：对应服务层，主要涉及一些复杂的逻辑，需要用到Dao层。

@Controller：对应Spring MVC控制层。

#### @RestController

@RestController注解是@Controller和@ResponseBody的合集，表示这是个控制器bean，并且是将函数的返回值直接填入HTTP响应体中，是REST风格的控制器。

单独使用@Controller不加@ResponseBody的话一般使用在要返回一个视图的情况，这种情况属于比较传统的Spring MVC的应用，对应于前后端不分离的情况。@Controller + @ResponseBody返回JSON或XML形式数据。

#### @Scope

声明Spring Bean的作用域，使用方法：

@Bean

@Scope("singleton")

public Person personSingleton() {

return new Person();

}

四种常见的Spring Bean的作用域：

* singleton : 唯一 bean 实例，Spring 中的 bean 默认都是单例的。
* prototype : 每次请求都会创建一个新的 bean 实例。
* request : 每一次 HTTP 请求都会产生一个新的 bean，该 bean 仅在当前 HTTP request 内有效。
* session : 每一次 HTTP 请求都会产生一个新的 bean，该 bean 仅在当前 HTTP session 内有效。

#### @Configuration

一般来声明配置类，可以使用@Component注解替代。

二、处理常见的HTTP请求类型

@GetMapping

@PostMapping

@PutMapping

@DeleteMapping

@PatchMapping

三、前后端传值

#### @PathVariable和@RequestParam

@PathVariable用于获取路径参数，@RequestParam用于获取查询参数。

@GetMapping("/klasses/{klassId}/teachers")

public List<Teacher> getKlassRelatedTeachers(

@PathVariable("klassId") Long klassId,

@RequestParam(value = "type", required = false) String type ) {

...

}

如果请求url为：/klasses/{123456}/teachers?type=web

那么我们服务获取到的数据就是：klassId=123456,type=web

#### @RequestBody

用于读取请求的body部分并且Content-Type为application/json格式的数据，接收到数据之后会自动将数据绑定到Java对象上去。系统会使用HttpMessageConverter或者自定义的HttpMessageConverter将请求的body中的json字符串转换为java对象。

注意：一个请求方法只可以有一个@RequestBody，但是可以有多个@RequestParam和@PathVariable。

四、读取配置信息

很多时候我们需要将一个常用的配置信息比如阿里云oss、发送短信、微信认证的相关配置信息等等放在配置文件中。

#### @Value（常用）

使用@Value(“${property}”)读取比较简单的配置信息。

#### @ConfigurationProperties（常用）

通过@ConfigurationProperties读取配置信息并与bean绑定。

@Component

@ConfigurationProperties(prefix = "library")

class LibraryProperties {

@NotEmpty

private String location;

private List<Book> books;

@Setter

@Getter

@ToString

static class Book {

String name;

String description;

}

省略getter/setter

......

}

可以像使用普通的Spring Bean一样，将其注入到类中使用。

#### @PropertySource

@PropertySource读取指定properties文件。

@Component

@PropertySource("classpath:website.properties")

class WebSite {

@Value("${url}")

private String url;

省略getter/setter

......

}

五、校验参数

数据的校验的重要性就不用说了，即使在前端对数据进行校验的情况下，我们还是要对传入后端的数据再进行一遍校验，避免用户绕过浏览器直接通过一些 HTTP 工具直接向后端请求一些违法数据。

校验的时候我们实际用的是Hibernate Validator框架。

SpringBoot 项目的 spring-boot-starter-web 依赖中已经有hibernate-validator包，不需要引用相关依赖。

注意：所有的注解，推荐使用JSR注解，即javax.validation.constraints，而不是org.hibernate.validator.constraints。

#### 常用的字段验证的注解

* @NotEmpty 被注释的字符串的不能为 null 也不能为空
* @NotBlank 被注释的字符串非 null，并且必须包含一个非空白字符
* @Null 被注释的元素必须为 null
* @NotNull 被注释的元素必须不为 null
* @AssertTrue 被注释的元素必须为 true
* @AssertFalse 被注释的元素必须为 false
* @Pattern(regex=,flag=)被注释的元素必须符合指定的正则表达式
* @Email 被注释的元素必须是 Email 格式。
* @Min(value)被注释的元素必须是一个数字，其值必须大于等于指定的最小值
* @Max(value)被注释的元素必须是一个数字，其值必须小于等于指定的最大值
* @DecimalMin(value)被注释的元素必须是一个数字，其值必须大于等于指定的最小值
* @DecimalMax(value) 被注释的元素必须是一个数字，其值必须小于等于指定的最大值
* @Size(max=, min=)被注释的元素的大小必须在指定的范围内
* @Digits (integer, fraction)被注释的元素必须是一个数字，其值必须在可接受的范围内
* @Past被注释的元素必须是一个过去的日期
* @Future 被注释的元素必须是一个将来的日期

#### 验证请求体

需要在参数上加上@Valid，如果验证失败，它将抛出MathodArgumentNotValidException。

@PostMapping("/person")

public ResponseEntity<Person> getPerson(@RequestBody @Valid Person person) {

return ResponseEntity.ok().body(person);

}

#### 验证请求参数（Path Variables和Request Parameters）

一定一定不要忘记在类上加上@Validated，这个注解可以告诉Spring去校验方法参数。

@RestController

@RequestMapping("/api")

@Validated

public class PersonController {

@GetMapping("/person/{id}")

public ResponseEntity<Integer> getPersonByID(@Valid @PathVariable("id")

@Max(value = 5,message = "超过 id 的范围了") Integer id) {

return ResponseEntity.ok().body(id);

}

}

六、全局处理Controller层异常

@ControllerAdvice：注解定义全局异常处理类

@ExceptionHandler：注解声明异常处理方法

如何使用呢？拿我们在第5节参数校验这块来举例子。如果方法参数不对的话就会抛出MethodArgumentNotValidException，我们来处理这个异常。

@ControllerAdvice

@ResponseBody

public class GlobalExceptionHandler {

/\*\*

\* 请求参数异常处理

\*/

@ExceptionHandler(MethodArgumentNotValidException.class)

public ResponseEntity<?> handleMethodArgumentNotValidException(

MethodArgumentNotValidException ex,

HttpServletRequest request) {

......

}

}

七、JPA相关

#### 创建表

@Entity 声明一个类对应一个数据库实体

@Table 设置声名

#### 设置主键

@Id 声明一个字段为主键。

@GeneratedValue 指定主键生成策略。

JPA内置了四种主键生成策略：

public enum GenerationType {

/\*\*

\* 使用一个特定的数据库表格来保存主键

\* 持久化引擎通过关系数据库的一张特定的表格来生成主键,

\*/

TABLE,

/\*\*

\*在某些数据库中,不支持主键自增长,比如Oracle、PostgreSQL其提供了一种叫做"序列(sequence)"的机制生成主键

\*/

SEQUENCE,

/\*\*

\* 主键自增长

\*/

IDENTITY,

/\*\*

\*把主键生成策略交给持久化引擎(persistence engine),

\*持久化引擎会根据数据库在以上三种主键生成 策略中选择其中一种

\*/

AUTO

}

一般使用MySQL数据库，使用GenerationType.IDENTITY策略比较普遍一点（分布式系统的话需要另外考虑使用分布式 ID）。

通过@GenericGenerator声明一个主键策略，然后@GenaratedValue使用这个策略。

@Id

@GeneratedValue(generator = "IdentityIdGenerator")

@GenericGenerator(name = "IdentityIdGenerator", strategy = "identity") private Long id;

JPA提供的主键生成策略有如下几种：

public class DefaultIdentifierGeneratorFactory

implements MutableIdentifierGeneratorFactory, Serializable, ServiceRegistryAwareService {

@SuppressWarnings("deprecation")

public DefaultIdentifierGeneratorFactory() {

register( "uuid2", UUIDGenerator.class );

register( "guid", GUIDGenerator.class ); // can be done with UUIDGenerator + strategy

register( "uuid", UUIDHexGenerator.class ); // "deprecated" for new use

register( "uuid.hex", UUIDHexGenerator.class ); // uuid.hex is deprecated

register( "assigned", Assigned.class );

register( "identity", IdentityGenerator.class );

register( "select", SelectGenerator.class );

register( "sequence", SequenceStyleGenerator.class );

register( "seqhilo", SequenceHiLoGenerator.class );

register( "increment", IncrementGenerator.class );

register( "foreign", ForeignGenerator.class );

register( "sequence-identity", SequenceIdentityGenerator.class );

register( "enhanced-sequence", SequenceStyleGenerator.class );

register( "enhanced-table", TableGenerator.class );

}

public void register(String strategy, Class generatorClass) {

LOG.debugf( "Registering IdentifierGenerator strategy [%s] -> [%s]", strategy, generatorClass.getName() );

final Class previous = generatorStrategyToClassNameMap.put( strategy, generatorClass );

if ( previous != null ) {

LOG.debugf( " - overriding [%s]", previous.getName() );

}

}

}

#### 设置字段类型

@Column 声明字段。

@Column(name = "user\_name", nullable = false, length=32)

private String userName;

#### 指定不持久化特定字段

@Transient 声明不需要与数据库映射的字段，在保存的时候不需要保存进数据库。

#### 声明大字段

@Lob 声明某个字段为大字段。

@Lob

//指定 Lob 类型数据的获取策略， FetchType.EAGER 表示非延迟 加载，而 FetchType. LAZY 表示延迟加载 ；

@Basic(fetch = FetchType.EAGER)

//columnDefinition 属性指定数据表对应的 Lob 字段类型

@Column(name = "content", columnDefinition = "LONGTEXT NOT NULL")

private String content;

#### 创建枚举类型的字段

可以使用枚举类型的字段，不过枚举字段要用@Enumerated注解修饰。

public enum Gender {

MALE("男性"),

FEMALE("女性");

private String value;

Gender(String str){

value=str;

}

}

@Entity

@Table(name = "role")

public class Role {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String description;

@Enumerated(EnumType.STRING)

private Gender gender;

省略getter/setter......

}

数据库里对应存储的是MALE/FEMALE

#### 增加审计功能

只要继承了AbstractAuditBase的类都会默认加上下面四个字段。

@Data

@AllArgsConstructor

@NoArgsConstructor

@MappedSuperclass

@EntityListeners(value = AuditingEntityListener.class)

public abstract class AbstractAuditBase {

@CreatedDate

@Column(updatable = false)

@JsonIgnore

private Instant createdAt;

@LastModifiedDate

@JsonIgnore

private Instant updatedAt;

@CreatedBy

@Column(updatable = false)

@JsonIgnore

private String createdBy;

@LastModifiedBy

@JsonIgnore

private String updatedBy;

}

对应的审计功能对应地配置类可能是下面这样的（Spring Security项目）：

@Configuration

@EnableJpaAuditing

public class AuditSecurityConfiguration {

@Bean

AuditorAware<String> auditorAware() {

return () -> Optional.ofNullable(SecurityContextHolder.getContext())

.map(SecurityContext::getAuthentication)

.filter(Authentication::isAuthenticated)

.map(Authentication::getName);

}

}

* @CreatedDate: 表示该字段为创建时间时间字段，在这个实体被 insert 的时候，会设置值
* @CreatedBy :表示该字段为创建人，在这个实体被 insert 的时候，会设置值
* @LastModifiedDate、@LastModifiedBy同理。
* @EnableJpaAuditing：开启 JPA 审计功能。

#### 删除/修改数据

@Modifying 提示JPA该操作是修改操作,注意还要配合@Transactional注解使用。

#### 关联关系

@OneToOne 声明一对一关系

@OneToMany 声明一对多关系

@ManyToOne声明多对一关系

MangToMang声明多对多关系

八、事务

在要开启事务的方法上使用@Transactional注解即可。

我们知道 Exception 分为运行时异常 RuntimeException 和非运行时异常。在@Transactional注解中如果不配置rollbackFor属性,那么事物只会在遇到RuntimeException的时候才会回滚,加上rollbackFor=Exception.class,可以让事物在遇到非运行时异常时也回滚。

@Transactional 注解一般用在可以作用在类或者方法上。

作用于类：当把@Transactional 注解放在类上时，表示所有该类的public 方法都配置相同的事务属性信息。

作用于方法：当类配置了@Transactional，方法也配置了@Transactional，方法的事务会覆盖类的事务配置信息。

九、json数据处理

#### 过滤json数据

@JsonIgnoreProperties 作用在类上用于过滤掉特定字段不返回或者不解析。

#### 格式化json数据

@JsonFormat一般用来格式化 json 数据。

@JsonFormat(shape=JsonFormat.Shape.STRING, pattern="yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", timezone="GMT")

private Date date;

#### 扁平化对象

@JsonUnwrapped 扁平对象

十、测试相关

@ActiveProfiles一般作用于测试类上， 用于声明生效的Spring配置文件。

@SpringBootTest(webEnvironment = RANDOM\_PORT)

@ActiveProfiles("test")

@Slf4j

public abstract class TestBase {

......

}

@Test声明一个方法为测试方法

@Transactional被声明的测试方法的数据会回滚，避免污染测试数据。

@WithMockUser Spring Security 提供的，用来模拟一个真实用户，并且可以赋予权限。