[Spring boot笔记](#_Toc515184136)

[Lombok](#_Toc515184137)

# JAVA语法

1. 开发REST接口常用注解

* java的访问控制是类级别，而不是对象级别，即任何一个类都可以访问同类的私有成员

# Spring boot笔记

1. 开发REST接口常用注解

|  |  |  |
| --- | --- | --- |
| @RestController | @ResponseBody和@Controller的组合注解  @Controller说明这是API的入口 | import org.springframework.web.bind.annotation.\*; |
| @RequestMapping | 可用在控制器的某个方法上，也可用在此控制器类上。当在控制器类级别上添加，这个注解会应用到控制器的所有处理器方法上；方法上的注解会对类级别上的的声明进行补充。 | import org.springframework.data.mongodb.core.mapping.Document |
| @GetMapping(“”) | 组合注解，是@RequestMapping  (method = RequestMethod.GET)的缩写 |  |
| @PostMapping(“”) | 同上 |  |
| @PutMapping(“”) | 同上 |  |
| @PatchMapping(“”) | 同上 |  |
| @DeleteMapping(“”) | 同上 |  |
| @PathVariable | 获取url中的数据  localhost：8080/hello/id  id通过@PathVariable获取 |  |
| @RequestParam | 获取请求参数的值  localhost:8080/hello?id=98  id通过@RequestParam获取 |  |
| @value("${键名}") | 获取application.properteis文件中的配置值 |  |
| @Document(collection = "") | 在实体类上的注解，指示该实体存储在MongoDB上的集合名 |  |
| @Id | 在实体类属性上进行注解，会把此属性当成主键，在查找时会根据此属性和MongoDB里的主键对比  （MongoDB默认主键是\_id） | import org.springframework.data.annotation.Id; |
| @Query | 在MongoRepository<T, TD>接口的方法上进行注解，可自定义mongodb原生查询语句 | import org.springframework.data.mongodb.repository.Query; |

2. 小提示

* 如果在实体类中不显式指定collection（在类前加上@Document(collection = "集合名")，Spring会根据实体类的名字去推测集合的名字
* URL约定
* URL中名词使用复数形式

3. MongoRepository

* 要使用Repository的功能，先继承MongoRepository<T, TD>接口，
* T为仓库保存的bean类，TD为该bean的唯一标识的类型，一般为ObjectId。
* 之后在service中注入该接口就可以使用，无需实现里面的方法，spring会根据定义的规则自动生成。
* MongoRepositor实现的只是最基本的增删改查，要想增加额外，可以按照以下规则定义接口方法：
* 自定义查询方法，格式为“findBy+字段名+方法后缀”，方法传进的参数即字段的值

|  |  |  |
| --- | --- | --- |
| **方法后缀** | **方法例子** | **mongodb原生查询语句** |
| GreaterThan | findByAgeGreaterThan(int age) | {"age" : {"$gt" : age}} |
| LessThan | findByAgeLessThan(int age) | {"age" : {"$lt" : age}} |
| Between | findByAgeBetween(int from, int to) | {"age" : {"$gt" : from, "$lt" : to}} |
| IsNotNull, NotNull | findByFirstnameNotNull() | {"firstname" : {"$ne" : null}} |
| IsNull, Null | findByFirstnameNull() | {" firstname " : null} |
| Like（模糊查询） | findByFirstnameLike(String name) | {"firstname" : name} ( name as regex) |
| (No keyword) | findByFirstname(String name) | {" firstname " : name} |
| Not（不包含） | findByFirstnameNot(String name) | {" firstname " : {"$ne" : name}} |
| Near（查询地理位置相近的） | findByLocationNear(Point point) | {"location" : {"$near" : [x,y]}} |
| Within（在地理位置范围内的） | findByLocationWithin(Circle circle) | {"location":{"$within": {"$center" : [ [x, y], distance]}}} |
| Within（在地理位置范围内的） | findByLocationWithin(Box box) | {"location":{"$within" : {"$box" : [ [x1, y1], x2, y2]}}} |

* 此外还支持分页查询，通过传进一个Pageable对象，返回Page集合。

eg: public Page<Product> findByAgeGreaterThan(int age,Pageable page)

* 还可基于mongodb原本查询语句的查询方式进行自定义，如

@Query("{ 'name':{'$regex':?2,'$options':'i'}, sales': {'$gte':?1,'$lte':?2}}")

public Page<Product> findByNameAndAgeRange(String name,double ageFrom,double ageTo,Pageable page);

* 还可以在后面指定要返回的数据字段，如上面的例子修改如下，则只通过person表里面的name和age字段构建person对象。

@Query(value="{ 'name':{'$regex':?2,'$options':'i'}, sales':{'$gte':?1,'$lte':?2}}",fields="{ 'name' : 1, 'age' : 1}")

public Page<Product> findByNameAndAgeRange(String name,double ageFrom,double ageTo,Pageable page);

Note：@Query 的value必须要有，value="{}"表示查询所有

* MongoRepository接口save
* mongodb 有两种添加数据的方式 save，insert

1. insert:当主键"\_id"在集合中存在时，不做任何处理，抛异常

　　 2. save:当主键"\_id"在集合中存在时，进行更新，数据整体都会更新，新数据会替换掉原数据 ID 以外的所有数据。如ID 不存在就新增一条数据

3. save 方法需要遍历列表，一个个插入，而insert方法 是直接批量插入

* Springboot-mongodb 的MongoRepository接口，并未提供insert方法，只提供了save方法

技术参考

文章链接：<http://www.jb51.net/article/132724.htm>

Spring Boot 项目中使用Swagger2的示例，文章链接：<http://www.jb51.net/article/132727.htm>

# Lombok

1. 需导入的包

2. 注解

|  |  |  |
| --- | --- | --- |
| @Data | 类注解，生成get、set方法、空构造器、toString方法、equals方法、hashCode方法 |  |
| @Setter/@Getter | 自动生成set和get方法 | **import** lombok.Getter; **import** lombok.Setter; |
| @CleanUp | 自动资源管理，不用再在finally中添加资源的close方法 |  |
| @NonNull | 在方法或构造函数的参数上使用@NonNull注解，Lombok将生成一个空值检查语句 |  |
| @ToString | 自动生成toString方法 |  |
| @EqualsAndHashcode | 从对象的字段中生成hashCode和equals的实现 |  |
| @NoArgsConstructor  @RequiredArgsConstructor  @AllArgsConstructor | 自动生成构造方法 |  |
| @Value | 用于注解final类 |  |
| @Builder | 产生复杂的构建器api类 |  |
| @SneakyThrows | 异常处理（谨慎使用） |  |
| @Synchronized | 同步方法安全的转化 |  |
| @Getter(lazy=true) |  |  |
| @Log | 支持各种logger对象，使用时用对应的注解，如：@Log4j |  |