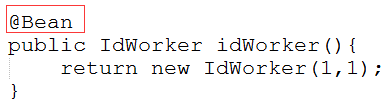
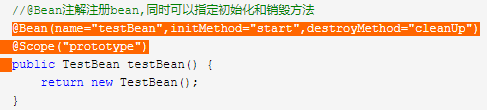
# @Bean

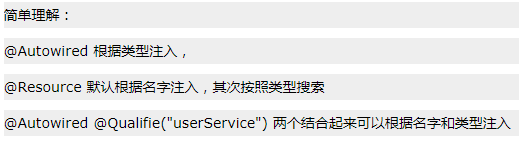
@Bean是一个方法级别上的注解,主要用在@Configuration注解的类里，也可以用在@Component注解的类里.

作用：把方法的返回对象装配到容器里





# @Resource @Autowired @Qualifier



参考文章：

<https://blog.csdn.net/working_hewei/article/details/85764793>

容器为我们创建bean时默认类名首字母小写。

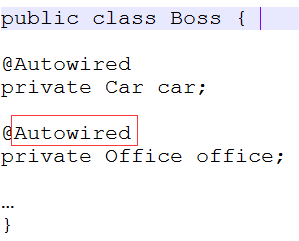
也就是说如果有多个相同类型的bean,byName就是类名首字母小写。文章：

<https://blog.csdn.net/wangzuojia001/article/details/54312074>

# @Autowired

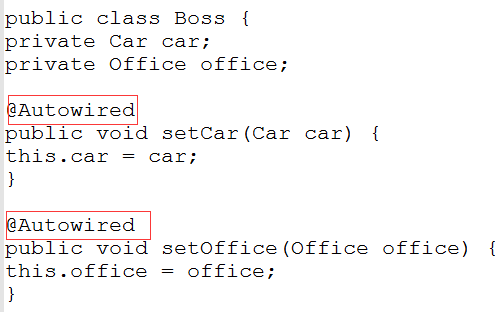
对类成员变量、方法及构造函数进行标注，完成自动装配的工作。消除 set ，get方法。

## 1.作用于类成员变量：



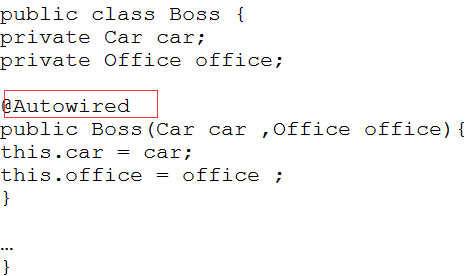
## 2.方法(setter方法)

将查找被标注的方法的入参类型的 Bean，并调用方法自动注入这些 Bean



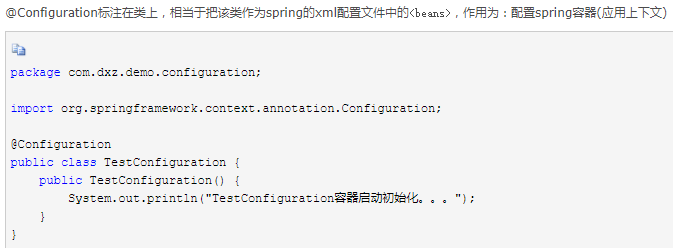
## 3.作用于构造函数

将查找被标注的方法的入参类型的 Bean，并调用方法自动注入这些 Bean



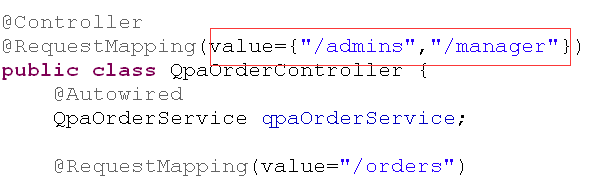
# @Configuration

用于定义配置类，可替换xml配置文件，被注解的类内部包含有一个或多个被@Bean注解的方法，初始化Spring容器



# @RequestMapping

可以映射多个路径



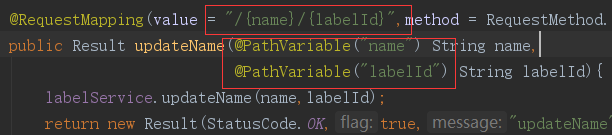
String[] value()

RequestMethod[] method()

两者都是数组，可以映射多个

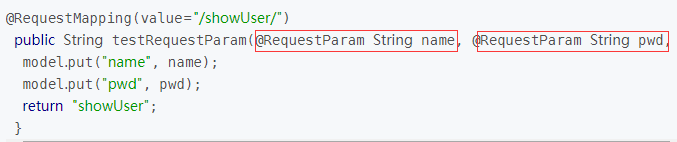
# @PathVariable

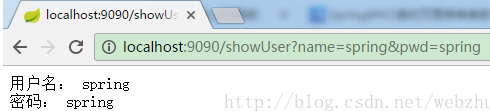
映射路径变量



# @RequestParam

映射请求参数





# @RequestBody

接收前端返回的json数据，并自动转换为相应的数据类型，比如String、JavaBean，Map、List

参考文章：

<https://blog.csdn.net/justry_deng/article/details/80972817>

# 0.@component

（把普通pojo实例化到spring容器中，相当于配置文件中的<bean id="" class=""/>）

# 1、@controller 控制器（注入服务）

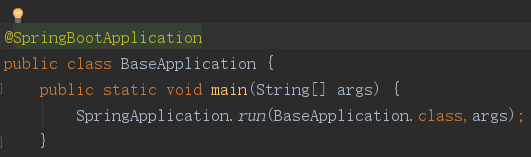
# 2、@service 服务（注入dao）

# 3、@repository dao（实现dao访问）

**@SpringBootApplication**

(默认属性)@Configuration + @EnableAutoConfiguration + @ComponentScan。

作用于启动类



# @EnableAutoConfiguration：

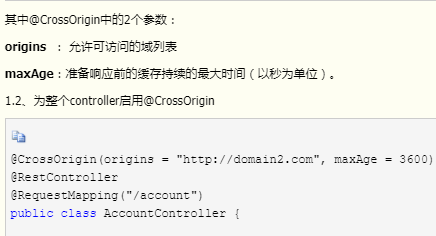
能够自动配置spring的上下文，试图猜测和配置你想要的bean类，通常会自动根据你的类路径和你的bean定义自动配置。

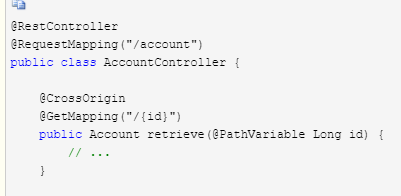
# @ComponentScan

会自动扫描指定包下的全部标有@Component的类，并注册成bean，当然包括@Component下的子注解@Service,@Repository,@Controller

# @CrossOrigin

controller方法的CORS配置，您可以向@RequestMapping注解处理程序方法添加一个@CrossOrigin注解，以便启用CORS



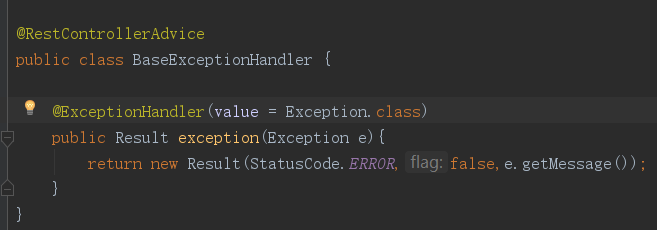


# @RestController

相当于 @Controller和@ResponseBody，直接将返回对象转换为json类型

**@RestControllerAdvice**

包含了@ControllerAdvice和@ResponseBody，拦截异常后返回异常类的json数据



## @ControllerAdvice

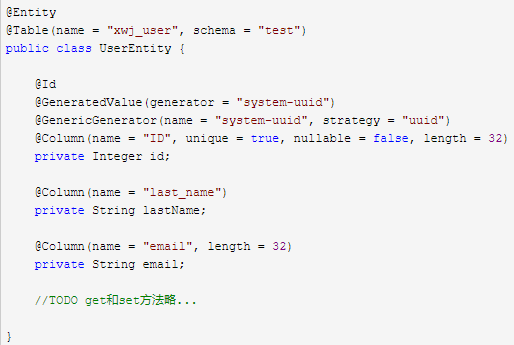
用于定义@ExceptionHandler、@InitBinder、@ModelAttribute，并应用到所有@RequestMapping中

# JPA常见注解

使用方法参考链接：

<https://blog.csdn.net/qq_22172133/article/details/81192040>

可以将常规的普通Java对象（有时被称作POJO）映射到数据库



## @Entity

表明该类 (UserEntity) 为一个实体类，它默认对应数据库中的表名是user\_entity。这里也可以写成：

@Entity(name = "xwj\_user")

或者

@Entity

@Table(name = "xwj\_user", schema = "test")

查看@Entity注解，发现其只有一个属性name，表示其所对应的数据库中的表名

## @Table

当实体类与其映射的数据库表名不同名时需要使用 @Table注解说明，该标注与 @Entity 注解并列使用，置于实体类声明语句之前，可写于单独句行，也可与声明语句同行

@Table注解的常用选项是 name，用于指明数据库的表名

@Table注解还有两个选项 catalog 和 schema 用于设置表所属的数据库目录或模式，通常为数据库名

## @Column

改变class中字段名与db中表的字段名的映射规则

@Column注释定义了将成员属性映射到关系表中的哪一列和该列的结构信息，属性如下：

　　1）name：映射的列名。如：映射tbl\_user表的name列，可以在name属性的上面或getName方法上面加入；

　　2）unique：是否唯一；

　　3）nullable：是否允许为空；

　　4）length：对于字符型列，length属性指定列的最大字符长度；

　　5）insertable：是否允许插入；

　　6）updatetable：是否允许更新；

　　7）columnDefinition：定义建表时创建此列的DDL；

8）secondaryTable：从表名。如果此列不建在主表上（默认是主表），该属性定义该列所在从表的名字

## @Id

@Id注释指定表的主键，它可以有多种生成方式：

　　1）TABLE：容器指定用底层的数据表确保唯一；

　　2）SEQUENCE：使用数据库德SEQUENCE列莱保证唯一（Oracle数据库通过序列来生成唯一ID）；

　　3）IDENTITY：使用数据库的IDENTITY列莱保证唯一；

　　4）AUTO：由容器挑选一个合适的方式来保证唯一；

　　5）NONE：容器不负责主键的生成，由程序来完成。

@Id一起使用的还有另外两个注解：@GeneratedValue、@GenericGenerator

## @GeneratedValue

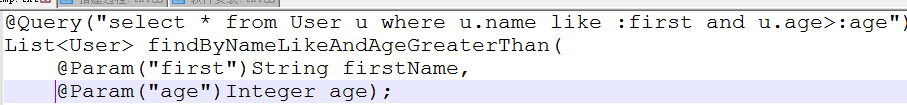
JPA通用策略生成器

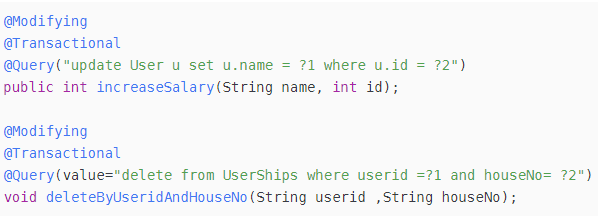
## @GenericGenerator

自定义主键生成策略

## @Query @Modifying @Transactional

可以通过@Query（若使用nativeQuery属性，则使用原生的sql语句）注解来创建查询（参数也可用 ?1 ?2 代替，则不需@Param）





更新、删除操作需要加上@Transactional @Modifying

* 索引参数与命名参数

　　　　1、索引参数如下所示，索引值从1开始，查询中**"?X"**个数需要与方法定义的参数个数相一致，并且顺序也要一致。

1 @Query("SELECT p FROM Person p WHERE p.lastName = **?1** AND p.email = **?2**")

2 List<Person> testQueryAnnotationParams1(String lastName, String email);

　　　　注释：上面代码中的?1,?2表示参数的占位符，需要和方法中所传递的参数顺序一致。X是从1开始。

　　　　2、命名参数(推荐使用此方式)：可以定义好参数名，赋值时使用@Param("参数名"),而不用管顺序。

1 // 为@Query注解传递参数的方式1：命名参数

2 @Query("SELECT p FROM Person p WHERE p.lastName = **:lastName** AND p.email = **:email**")

3 List<Person> testQueryAnnotationParams2(@Param("**email**") String email, @Param("**lastName**") String lastName);

　　　　注释：上面代码中:lastName ,:email 表示为参数命名，方法中所传递的参数使用@Param注解标识命名参数。这种方式不用管参数的顺序。

4、使用原生SQL进行查询

[复制代码](javascript:void(0);)

　　　\*\*

\* 设置nativeQuery=true 即可以使用原生的SQL进行查询

\* @return

\*/

@Query(value = "SELECT count(id) FROM jpa\_persons", nativeQuery = true)

long getTotalCount();

[复制代码](javascript:void(0);)

　　　　注释：当设置nativeQuery=true即可以使用原生SQL进行查询

* @Modifying注解

　　　　1、在@Query注解中编写JPQL实现DELETE和UPDATE操作的时候必须加上@modifying注解，以通知Spring Data 这是一个DELETE或UPDATE操作。

　　　　2、UPDATE或者DELETE操作需要使用事务，此时需要 定义Service层，在Service层的方法上添加事务操作。

　　　　3、注意JPQL不支持INSERT操作。

1 @Transactional

2 @Modifying

3 @Query("UPDATE Person p SET p.email = :email WHERE p.id = :id")

4 void updatePersonEmail(@Param("id") Integer id, @Param("email") String email);

## @Param

在dao层中用来注解方法中的参数，在@Query使用时，不用管参数的引用顺序。

2、命名参数(推荐使用此方式)：可以定义好参数名，赋值时使用@Param("参数名"),而不用管顺序。

1 // 为@Query注解传递参数的方式1：命名参数

2 @Query("SELECT p FROM Person p WHERE p.lastName = **:lastName** AND p.email = **:email**")

3 List<Person> testQueryAnnotationParams2(@Param("**email**") String email, @Param("**lastName**") String lastName);

1、索引参数如下所示，索引值从1开始，查询中**"?X"**个数需要与方法定义的参数个数相一致，并且顺序也要一致。

1 @Query("SELECT p FROM Person p WHERE p.lastName = **?1** AND p.email = **?2**")

2 List<Person> testQueryAnnotationParams1(String lastName, String email);

　　　　注释：上面代码中的?1,?2表示参数的占位符，需要和方法中所传递的参数顺序一致。X是从1开始。

# JPA解析方法名创建查询

find+全局修饰+By+实体的属性名称+限定词+连接词+ …(其它实体属性)+OrderBy+排序属性+排序方向

全局修饰： Distinct， Top， First

关键词： IsNull， IsNotNull， Like， NotLike， Containing， In， NotIn，

IgnoreCase， Between， Equals， LessThan， GreaterThan， After， Before…

排序方向： Asc， Desc

连接词： And， Or

And — 等价于 SQL 中的 and 关键字，比如 findByUsernameAndPassword(String user, Striang pwd)；

Or — 等价于 SQL 中的 or 关键字，比如 findByUsernameOrAddress(String user, String addr)；

Between — 等价于 SQL 中的 between 关键字，比如 findBySalaryBetween(int max, int min)；

LessThan — 等价于 SQL 中的 “<”，比如 findBySalaryLessThan(int max)；

GreaterThan — 等价于 SQL 中的”>”，比如 findBySalaryGreaterThan(int min)；

IsNull — 等价于 SQL 中的 “is null”，比如 findByUsernameIsNull()；

IsNotNull — 等价于 SQL 中的 “is not null”，比如 findByUsernameIsNotNull()；

NotNull — 与 IsNotNull 等价；

Like — 等价于 SQL 中的 “like”，比如 findByUsernameLike(String user)；

NotLike — 等价于 SQL 中的 “not like”，比如 findByUsernameNotLike(String user)；

OrderBy — 等价于 SQL 中的 “order by”，比如 findByUsernameOrderBySalaryAsc(String user)；

Not — 等价于 SQL 中的 “！ =”，比如 findByUsernameNot(String user)；

In — 等价于 SQL 中的 “in”，比如 findByUsernameIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；

NotIn — 等价于 SQL 中的 “not in”，比如 findByUsernameNotIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；

## 嵌套实体:

主实体中子实体的名称+ \_ +子实体的属性名称

List findByAddress\_ZipCode(ZipCode zipCode)

表示查询所有 Address（地址）的zipCode（邮编）为指定值的所有Person(人员)