Java知识大全

# Spring 验证框架（JSR-303）

## 空检查

@Null 验证对象是否为空；

@NotNull 验证对象不为空；

@NotBlank 验证字符串不为空或者不是空字符串，比如””、” “都会验证失败；

@NotEmpty 验证对象不能为null，或者集合不为空。

## 长度检查

@Size(min=,max=) 验证对象长度，可支持字符串、集合；

@Length 字符串大小

## 数值校验

@Min 验证数字是否大于等于指定的值；

@Max 验证数字是否小于等于指定的值；

@Digits 验证数字是否符合指定格式，如@Digits(Integer=9,fraction=2);

@Range 验证数字是否在指定的范围内，如@Range（min=1，max=1000）

## 其他

@Email 验证是否为邮件格式，为null则不做校验；

@Pattern 验证Spring对象是否符合正则表达式的规则。

# Beetl模板引擎

## 设置定界符号和占位符

Beetl默认使用<% %>作为定界符号；使用${ }作为占位符号。也支持自定义定界符号和占位符号。常用的有：

@ 和回车作为定界符号 如:

@ var salary = user.salary \* 2;

<span>${user.name}</span>

<? ?> 类似PHP的定界符号 如:

<? var salary = user.salary \* 2; ?>

<span>${user.name}</span>

## 全局变量

request 标准的HTTPServletRequest，可以在模板中引用request属性(getter)，如${request. requestURL}

parameter 读取用户提交的参数，如${ parameter.userId}

# Spring Data JPA

## 相关配置

#----------------------

#持久层配置

#----------------------

#jpa规范使用的数据源

spring.jpa.database=MYSQL

#InnoDB是一种MySQL数据库引擎。MySQL5.5及之后使用它做为默认引擎。它提供了ACID兼容的事务（Transaction）功能，并提供外键支持

spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect

#是否打印SQL

spring.jpa.show-sql=true

#是否自动建库

spring.jpa.hibernate.ddl-auto=update

### ddl-auto

 create：每次运行程序时，都会重新创建表，故而数据会丢失

 create-drop：每次运行程序时会先创建表结构，然后待程序结束时清空表

 upadte：每次运行程序，没有表时会创建表，如果对象发生改变会更新表结构，原有数据不会清空，只会更新（推荐使用）

 validate：运行程序会校验数据与数据库的字段类型是否相同，字段不同会报错

 none: 禁用DDL处理

### ddl-auto=update

当ddl-auto为update模式时，数据库会根据实体类中新增的属性在表中添加相应的字段。实体类中增加属性，如：

* @Column(name = "create\_time",columnDefinition = "default CURRENT\_TIMESTAMP ",insertable=false,updatable=false)

**private** Date createTime;

* @Column(name="count", length=11, nullable=**false**,columnDefinition="int default '0'")

**private** Integer count;

* @Column(name="str", length=50, columnDefinition="varchar(50) default '空'")

**private** String str;

name可选,字段名(默认值是属性名)

unique可选,是否在该字段上设置唯一约束(默认值false)

nullable可选,是否设置该字段的值可以为空(默认值false)

insertable 可选,该字段是否作为生成的insert语句中的一个字段(默认值true)

updatable 可选,该字段是否作为生成的update语句中的一个字段(默认值true)

columnDefinition 可选: 为这个特定字段覆盖sql DDL片段 （这可能导致无法在不同数据库间移植）

table 可选,定义对应的表(默认为主表)

length 可选,字段长度(默认值255)

precision 可选,字段数字精精度(默认值0)

scale 可选,如果字段数字刻度可用,在此设置(默认值0)

## Repository

**Repository**是Spring Data的和信息概念，抽象了对数据库和NoSQL的操作，提供了如下接口供开发者使用：

* **CrudRepository** 提供了基本的增删改查，批量操作接口；
* **PagingAndSortingRepository** 集成CrudRepository，提供了夫家的分页查询功能；
* **JpaRepository** 专门用于JPA，提供了更多丰富的数据库访问接口，比如根据Example来查询。

**CrudRepository、PagingAndSortingRepository**了解即可，这里略过。

### **JpaRepository**

JpaRepository提供的内置查询：

|  |  |
| --- | --- |
| **方法** | **描述** |
| List findAll(); | 返回所有实体 |
| List findAll(Iterable ids); | 返回指定id的所有实体 |
| T getOne(ID id) | 根据id返回对应的实体，如果未找到，则返回空 |
| List findAll(Sort sort); | 返回所有实体，按指定顺序排序返回 |
| Page findAll(Pageable pageable); | 返回实体列表，实体的offset和limit通过pageable来指定 |

#### Sort

Sort对象用来指示排序，默认升序排列。

例：

Sort sort = new Sort(“id”);

Return userDao.findAll(sort);

Sort其他的构造方法还包含：

// 按指定的属性列表升序排列。

* public Sort(String… properties) ;

// 按指定的属性列表排序，排序由Direction指定，Direction是一个枚举类，有Direction.ASC和Direction.DESC两类。

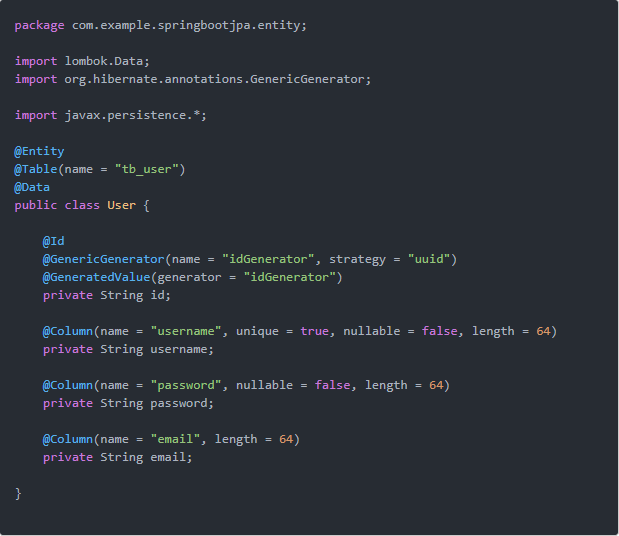
* public Sort(Direction direction,String… properties)

// 可通过Order静态方法来创建。

* public Sort(Order… orders) ;
* public static Order asc(String propertyName);
* public static Order desc(String propertyName);

## Demo实例

### 模型层（Model）



主键采用UUID策略  
@GenericGenerator是Hibernate提供的主键生成策略注解，注意下面的@GeneratedValue（JPA注解）使用generator = "idGenerator"引用了上面的name = "idGenerator"主键生成策略

#### JPA自带的几种主键生成策略

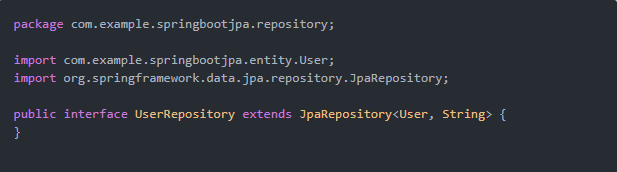
 TABLE： 使用一个特定的数据库表格来保存主键

 SEQUENCE： 根据底层数据库的序列来生成主键，条件是数据库支持序列。这个值要与generator一起使用，generator 指定生成主键使用的生成器（可能是orcale中自己编写的序列）

 IDENTITY： 主键由数据库自动生成（主要是支持自动增长的数据库，如mysql）

 AUTO： 主键由程序控制，也是GenerationType的默认值

### 持久层（Dao）



### 业务层（Service）

这里简单起见省略Service层

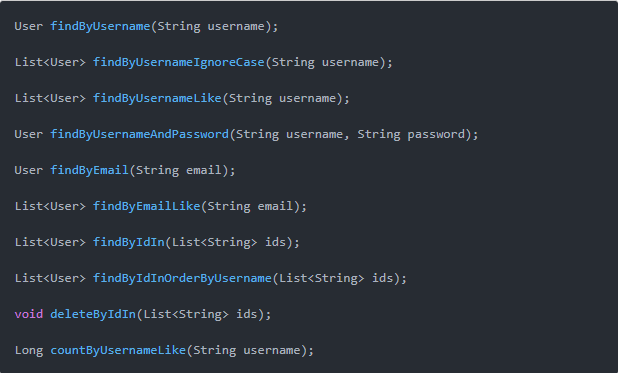
### Web控制器（Controller）



### 查询方法

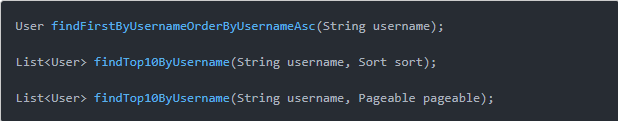
#### 查询创建Query Creation

Spring Data Jpa通过解析方法名创建查询，框架在进行方法名解析时，会先把方法名多余的前缀find…By, read…By, query…By, count…By以及get…By截取掉，然后对剩下部分进行解析，第一个By会被用作分隔符来指示实际查询条件的开始。 我们可以在实体属性上定义条件，并将它们与And和Or连接起来，从而创建大量查询：



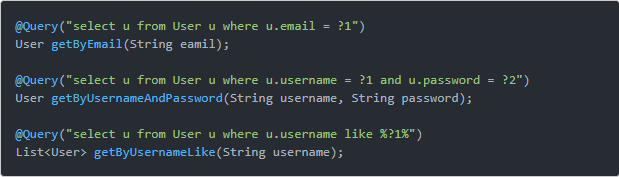
#### 限制查询结果

Spring Data Jpa支持使用first、top以及Distinct 关键字来限制查询结果，如：



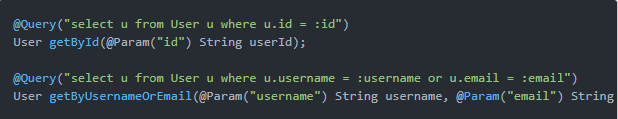
#### 自定义查询Using @Query

@Query 注解的使用非常简单，只需在声明的方法上面标注该注解，同时提供一个 JPQL 查询语句即可



#### 使用命名参数Using Named Parameters

默认情况下，Spring Data JPA使用基于位置的参数绑定，如前面所有示例中所述。 这使得查询方法在重构参数位置时容易出错。 要解决此问题，可以使用@Param注解为方法参数指定具体名称并在查询中绑定名称，如以下示例所示：



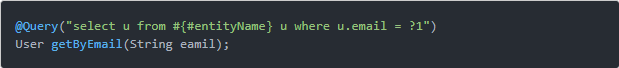
#### 用List<Object[]> 接收查询结果集

@Query(value=”select user\_id,count(0) from user group by sex”,nativeQuery=true)

public List< Object[]> queryUserCount();

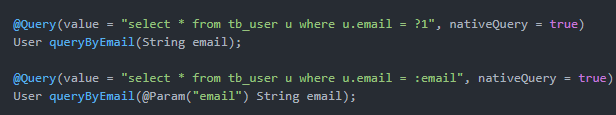
#### Using SpEL Expressions

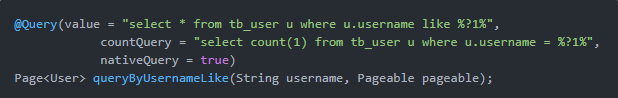
从Spring Data JPA release 1.4开始，Spring Data JPA支持名为entityName的变量。 它的用法是select x from #{#entityName} x。 entityName的解析方式如下：如果实体类在@Entity注解上设置了name属性，则使用它。 否则，使用实体类的简单类名。为避免在@Query注解使用实际的实体类名，就可以使用#{#entityName}进行代替。如以上示例中，@Query注解的查询字符串里的User都可替换为#{#entityName}



#### 原生查询Native Queries

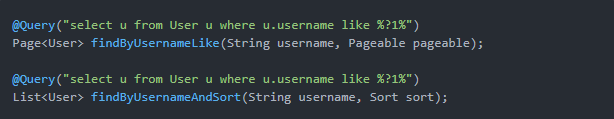
@Query注解还支持通过将nativeQuery标志设置为true来执行原生查询，同样支持基于位置的参数绑定及命名参数，如：



注意：Spring Data Jpa目前不支持对原生查询进行动态排序，但可以通过自己指定计数查询countQuery来使用原生查询进行分页、排序，如：

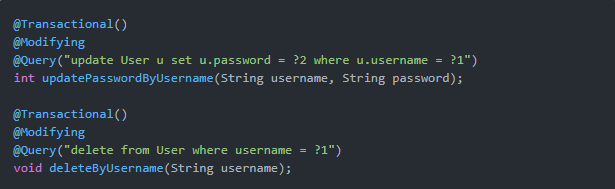
#### 分页查询及排序

Spring Data Jpa可以在方法参数中直接传入Pageable或Sort来完成动态分页或排序，通常Pageable或Sort会是方法的最后一个参数，如：



#### 自定义修改、删除 Modifying Queries

单独使用@Query注解只是查询，如涉及到修改、删除则需要再加上@Modifying注解，如：



注意：编辑查询只能使用void或int / Integer作为返回类型！

# ****开发规范****

**良好的开发规范，能提高开发效率，且看起来使人赏心悦目。**

* 双斜线注释时，作用在被描述体的上方，且在描述体前加一个空格。*例*：

// 是否删除

boolean isDel = false;

* 操作符号和代码之间要有空格相隔。例：

× int age=0; if(1==0){…}else if(1 == 1){…}else{…}

√iint age = 0; if (1 == 0) {…} else if (1 == 1) {…} else {…}

* 属性命名以驼峰的格式命名。*例*：

String redTheBed = “红色的床”;

* 持久层、业务层、Web控制层的命名前缀，依次以：

findxxx、queryxxx、getxxx

* 日志记录规范：

log.error() 一般是需要if()的；

log.info()一般是在try  catch 里面

log.debug() 做记录一般标志着方法的开始和结束。

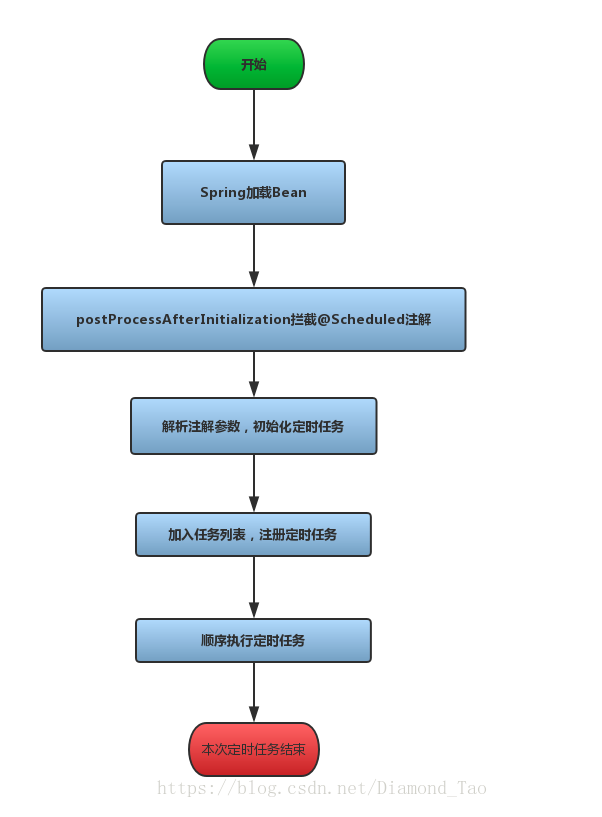
# SpringBoot/Maven聚合项目

## 注意事项

在有springboot启动类的聚合项目中，子项目的package目录要保持一致，使springboot能扫描到。

# Spring 定时器

## 运作流程



# @Component和@Configuration配置类的区别

1. 后者功效大于前者
2. 后者基于CGLIB代理，会直接从上下文调用并返回bean（单例），而不是再次执行方法（new对象）
3. 后者在项目启动后，一切已初始化就绪，且只会初始化一次；前者在项目启动后，每次调用都会被初始化
4. 后者处理逻辑较为复杂的业务场景，比如：注入bean（@bean）;前者“简单粗暴”，类型与@Controller的作用。

# 其他

## @Deprecated

划线注解：该注解作用在方法上来声明——该方法后期不在进行更新，且随着jar包的升级，该方法将不在兼容，直至废除。

## 研究方向

缓存、并发、事务、创建实例的方式、设计模式

## 大企业缓存机制流程

**Redis → ORM缓存 → DB**

## **ES6**

### 获取url参数（最简写法）

q={};location.search.replace(/([^?&=]+)=([^&]+)/g,(\_,k,v)=>q[k]=v);q;

# 相关链接

## 玩转spring boot——properties配置

<https://www.cnblogs.com/GoodHelper/p/6260708.html>

## SpringBoot学习笔记九：Spring Data Jpa的使用

<https://www.jianshu.com/p/c23c82a8fcfc>

## Spring Data JPA @Column说明

<https://blog.csdn.net/xufei512/article/details/81633447>

## abstract class和interface的区别

<https://blog.csdn.net/u012092924/article/details/78342193>

## 拦截器（Interceptor）和过滤器（Filter）的执行顺序和区别

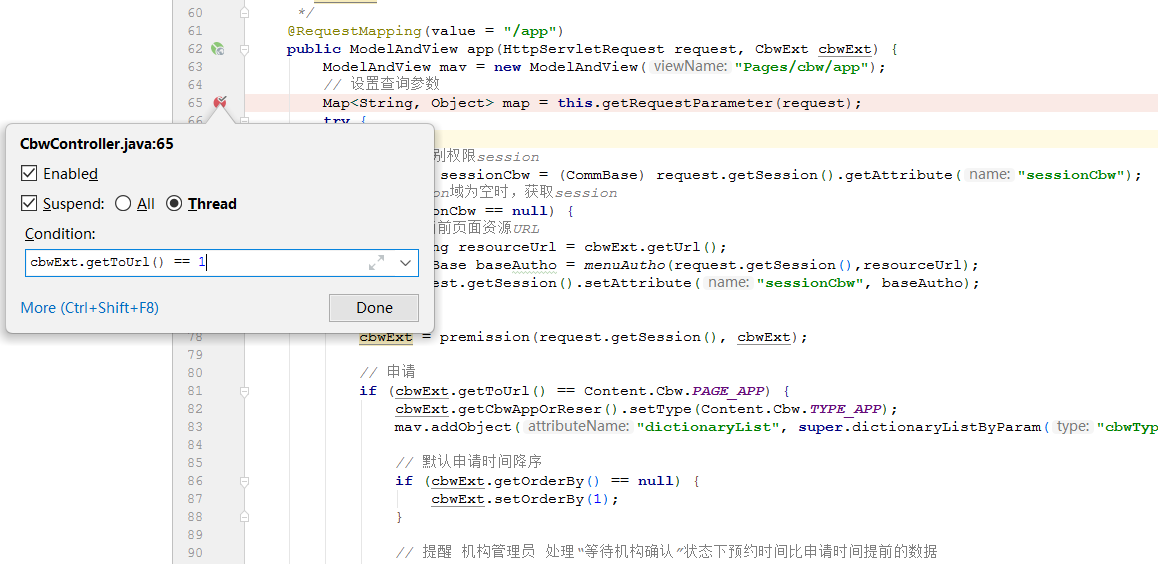
<https://blog.csdn.net/xyajia/article/details/85128924>

# 冥想

Java是一种语言，做一个真正意义上的项目，肯定是多语言共存，各种优良三方工具相结合，协同合作完成的。

# IDEA Debug技能

## FOR 循环插入式判断

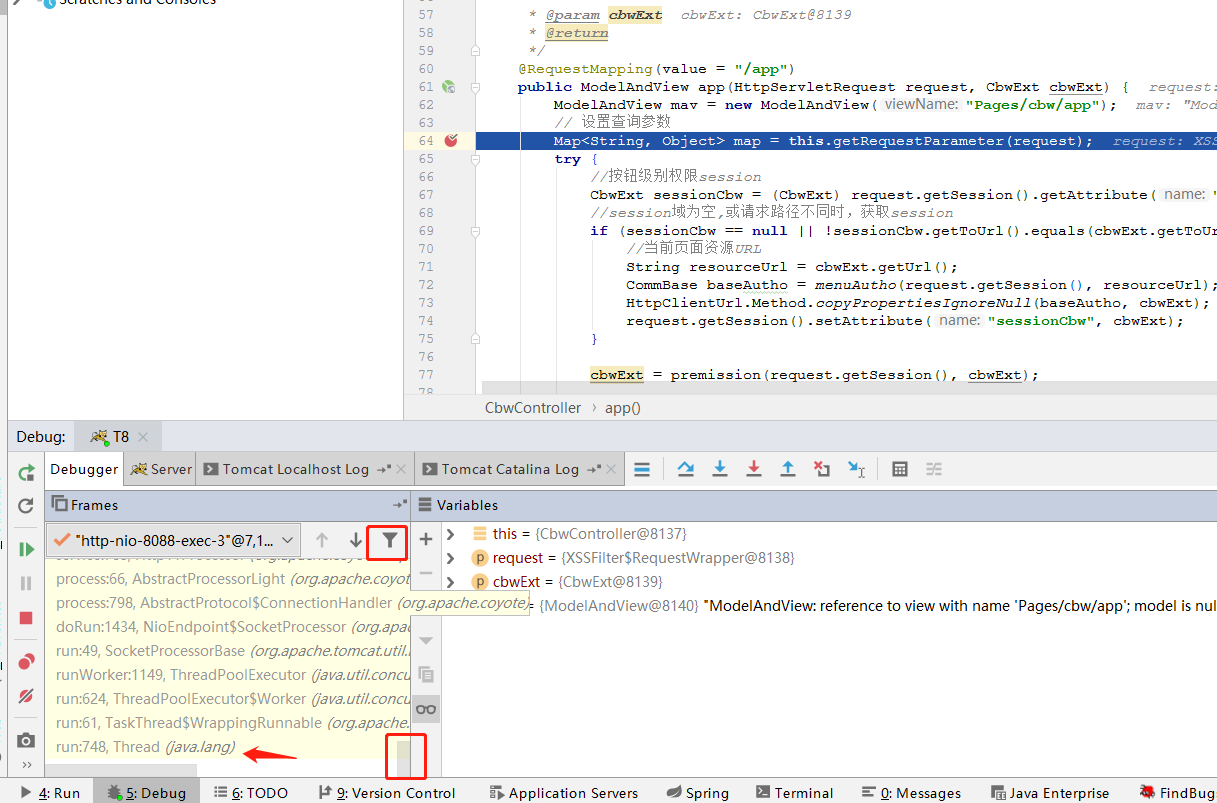


如上图，在debug for循环时：

老的办法是在里面加一个if判断，才能找到自己想要debug的地方。

高效的办法是在for循环debug的地方，右键输入自己的条件，来找到想要debug的地方。

## 蜈蚣式查找问题



如上图，debug走的每一步，进行的操作：

在红框标记的状态下，是走每一步时，要走的内部方法，如果错误藏得很深，那么可以打断点到底层，一步一步排查。

比如：一个请求的访问执行，都是从java底层的：

创建线程 》 加载线程池（如有） 》 运行程序 》 处理程序 》 处理底层业务 》 调用堆栈 》 筛查过滤器（如有） 》 拦截器（如有） 》 自己编写的程序

### 过滤器和拦截器的区别和优先级

过滤器(Filter)：它依赖于servlet容器。在实现上，基于函数回调，它可以对几乎所有请求进行过滤，但是缺点是一个过滤器实例只能在容器初始化时调用一次。使用过滤器的目的，是用来做一些过滤操作，获取我们想要获取的数据，比如：在Javaweb中，对传入的request、response提前过滤掉一些信息，或者提前设置一些参数，然后再传入servlet或者Controller进行业务逻辑操作。通常用的场景是：在过滤器中修改字符编码（CharacterEncodingFilter）、在过滤器中修改HttpServletRequest的一些参数（XSSFilter(自定义过滤器)），如：过滤低俗文字、危险字符等。

拦截器（Interceptor）：它依赖于web框架，在SpringMVC中就是依赖于SpringMVC框架。在实现上,基于Java的反射机制，属于面向切面编程（AOP）的一种运用，就是在service或者一个方法前，调用一个方法，或者在方法后，调用一个方法，比如动态代理就是拦截器的简单实现，在调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在调用方法后打印出字符串，甚至在抛出异常的时候做业务逻辑的操作。由于拦截器是基于web框架的调用，因此可以使用Spring的依赖注入（DI）进行一些业务操作，同时一个拦截器实例在一个controller生命周期之内可以多次调用。但是缺点是只能对controller请求进行拦截，对其他的一些比如直接访问静态资源的请求则没办法进行拦截处理。

Filter的执行顺序在Interceptor之前