文档内容基于SpringDataJpa 2.2.3版本

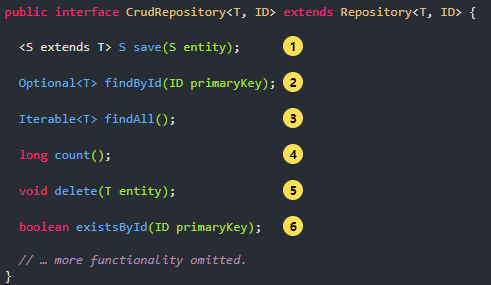
## 核心概念（Repository）：

SpringDataRepository的核心是Respository接口。它以<T,ID>中的T（实体类型）和T里面的id类型作为参数来管理，例如：

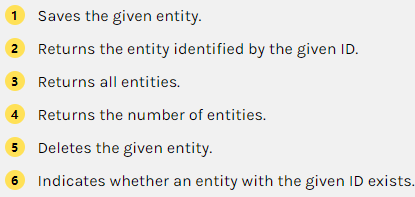
。此接口用作标记接口，主要是捕获要使用的类型和发现扩展此接口的接口。

### CrudRepository接口：

此接口继承了Repository接口，它为所管理的实体类提供了复杂的crud方法。



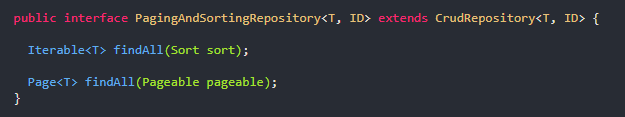
方法介绍：



#### 其他子接口

CrudRepository接口还有许多接口，比如JpaRepository和MongoRepository，PagingAndSortingRepository，它们不但具有CrudRepository接口的持久化功能，还扩展了其他功能。继承了CrudRepository的接口都会具有一些其他的派生功能，下面会讲到。

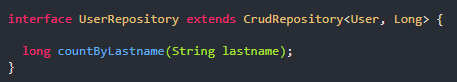
PagingAndSortingRepository：

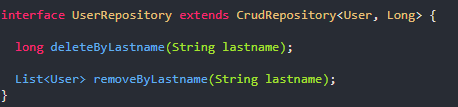


例子，查询第2页，然后每页显示20条数据：



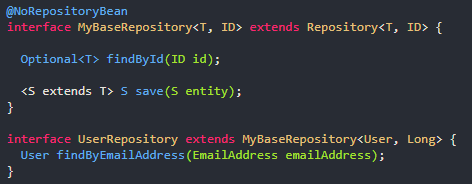
关于派生：





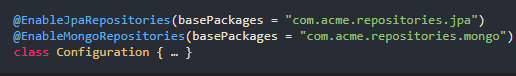
以上两个例子就是具备了count和delete的派生

#### 自定义接口



通过自定义接口来暴露需要的方法。@NoRepositoryBean注解告诉SpringData在运行时不要再创建对应的实例

#### 当配置了多个的SpringDataRepository





参考：<https://www.oschina.net/question/574036_2286640>

### 在接口中自定义查询方法的规则（根据方法名生成对应的sql）

Spring Data JPA框架在进行方法名解析时，会先把方法名多余的前缀截取掉，比如find，findBy，read，readBy，get，getBy，然后对剩下的部分进行解析。

假如创建如下的查询：findByUserDepUuid（），框架在解析该方法时，首先剔除findBy，然后对剩下的属性进行解析，假设查询实体为Doc，下面是查询解析步骤：

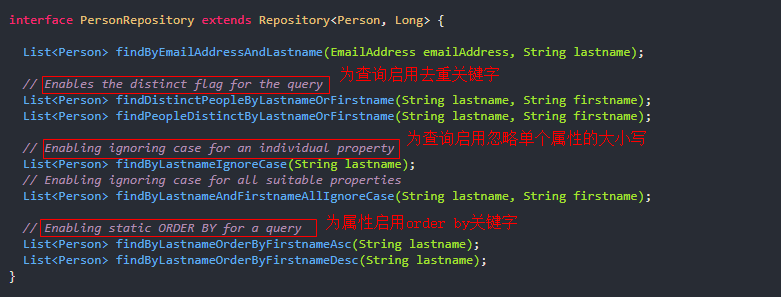
1、：先判断userDepUuid（根据POJO规范，首字母变为小写）是否为查询实体的一个属性，如果是，则表示根据该属性进行查询;如果没有该属性，继续第二步;

2、：从右往左截取第一个大写字母开头的字符串此处为UUID），然后检查剩下的字符串是否为查询实体的一个属性，如果是，则表示根据该属性进行查询;如果没有该属性，则重复第二步，继续从右往左截取;最后假设用户为查询实体的一个属性;

3、：接着处理剩下部分（DepUuid），先判断用户所对应的类型是否有depUuid属性，如果有，则表示该方法最终是根据“Doc.user.depUuid”的取值进行查询;否则继续按照步骤2的规则从右往左截取，最终表示根据“Doc.user.dep.uuid”的值进行查询。

4：可能会存在一种特殊情况，比如Doc包含一个用户的属性，也有一个userDep属性，此时会存在混合。可以明确在属性之间加上“\_”以显式表达意思，比如“findByUser\_DepUuid ）“或者”findByUserDep\_uuid（）“。可以使用@EnableJpaRepositories注解来自定义的方法解析的转义字符

还可以和**distinct、and、or、asc、desc、Between, LessThan, GreaterThan、Like**等关键字联合使用，然后会生成具有特定意义的sql。下面是一个例子：



定义的方法最后生成的sql取决于数据库。

自定义方法各种命名规则参考：

1. <https://blog.csdn.net/LHS19940203/article/details/80576867>
2. <https://blog.csdn.net/youngsend/article/details/51832581>
3. 官方文档：

<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#repositories.query-methods.query-creation>

<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#jpa.query-methods.query-creation>

### 关于自定义接口选择继承CrudRepository还是Repository或不进行任何继承

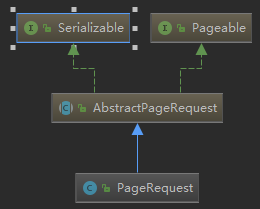
如果自定义的接口想要暴露crud方法，就继承CrudRepository而不是Repository。如果希望选择性的暴露一些方法，可以copy CrudRpository中的方法到自定义接口中去。如果不想继承任何Repository而是自己定义一个Repository也可以使用注解@RepositoryDefinition来实现

<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#repositories.create-instances.standalone>

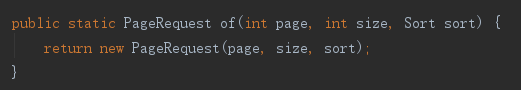
### 关于分页和排序

#### Pageable

Pageable是一个接口，主要用于分页，它可以通过PageRequest来创建。它是Pageable接口的孙子类，结构图如下：



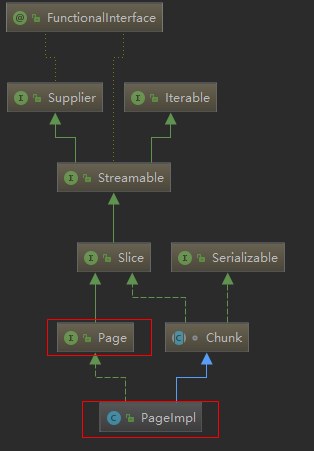
PageableRequest可通过of方法来创建pageable，方法如下：



#### Page

通过page可知道数据总条数，和多少页，它是通过触发一个count查询来计算总量的，所以当结果集的数据量很大时，它的开销是很大的，此时就可以考虑返回一个Slice。因为Slice是当前只知道下一个slice。

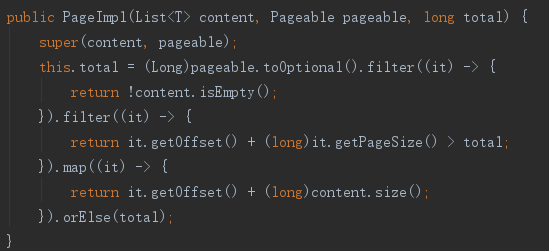
Page接口有PageImpl实现类，结构图如下：



PageImpl的分页实现，例子：



关于构造方法：

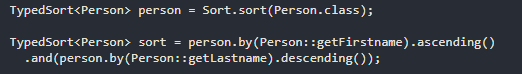


#### Slice

Page接口继承了Slice接口，在遍历较大的结果集时可以使用slice接口

#### Sort





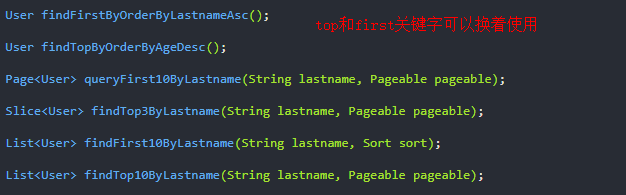
Querydsl的写法（需要数据库支持）



如果在定义排序时想要使用函数，可以采用JpaSort，如下：



#### 限制查询结果条数

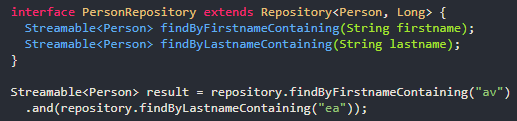


方法定义还支持distinct关键字，结果可以用Optional进行包装

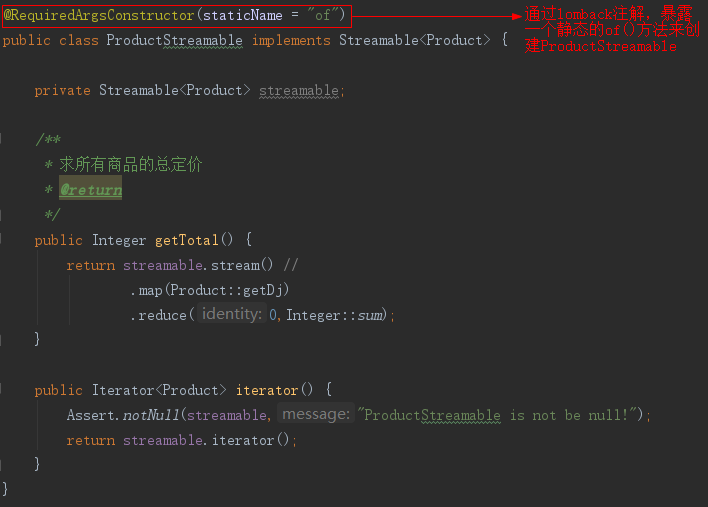
### 采用SpringData中的Streamable来包装查询结果

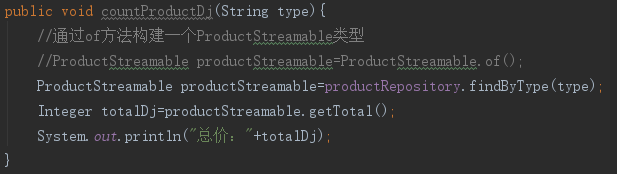
Streamable不需要手动调用close()方法来关闭

#### 直接用Streamable：



#### 扩展Streamable接口，实现具体操作：





### 对查询方法返回的结果进行null验证控制

1. [@NonNullApi](https://docs.spring.io/spring/docs/5.2.2.RELEASE/javadoc-api/org/springframework/lang/NonNullApi.html)--》对某个包下的类中所有方法返回的结果都进行null验证，此时就不需要再在方法或者属性上加@NonNull注解了



1. @NonNull---》规定返回结果不能为null，否则抛出异常



1. @Nullable--》允许结果可为空

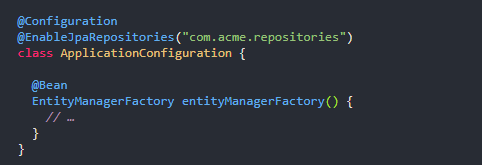
### 异步查询--@Async

<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#repositories.query-streaming>

### 使用repository时针对不同的数据库如何进行配置

在SpringBoot中默认情况下使用jpa repository要使用

@EnableJpaRepository注解来开启对它的支持，如下所示：



针对不同的数据库的repository可使用@Enable${store}Repositories注解来开启对应数据库的respository的支持，比如@EnableRedisRepositories、

@EnableMapRepositories等。

<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#repositories.create-instances.java-config>

### 独立使用Repository

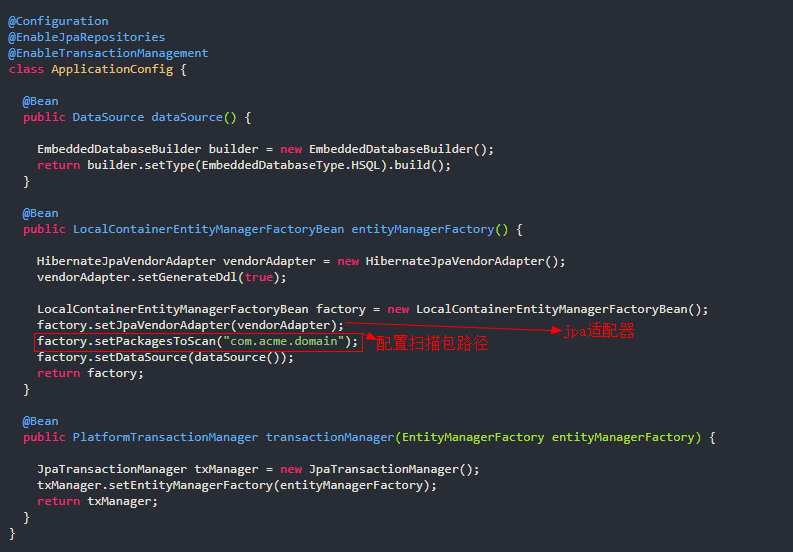
<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#repositories.create-instances.standalone>

### Custom SpringDataRepository

<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#repositories.customize-base-repository>

<https://www.jianshu.com/p/73f48095a7bf>

## SpringBoot中配置jpa（使用jpa做的配置）



注意，上面代码中的factory.setPackagesToScan是扫描路径下的实体需要注解@Entity。

前面的配置类使用spring-jdbc的EmbeddedDatabaseBuilder API来设置嵌入式HSQL数据库。然后，SpringData设置一个EntityManagerFactory并使用Hibernate作为示例里的持久性提供程序。这里声明的最后一个基础设施组件是JpaTransactionManager。最后，该示例通过使用@Enablejparepos注释激活Spring Data JPA存储库，该注释本质上携带与XML名称空间相同的属性。如果没有配置包扫描路径，则默认扫描当前配置类所在包。

事务管理配置：

<https://blog.csdn.net/weixin_42661074/article/details/86687761>

<https://www.jianshu.com/p/7c6d4dbbe8fc>

Spring提供了对编程式事务和声明式事务的支持，编程式事务允许用户在代码中精确定义事务的边界，而声明式事务（基于AOP）有助于用户将操作与事务规则进行解耦。简单地说，编程式事务侵入到了业务代码里面，但是提供了更加详细的事务管理；而声明式事务由于基于AOP，所以既能起到事务管理的作用，又可以不影响业务代码的具体实现。上面例子中就是声明式事务。

## SpringDataJpa中bean实例化时机控制

### 默认情况下jpa repositories的实例化时机

默认情况下，Spring data jpa repositories就是Spring的bean，它们是单例的。在启动期间，它们早已经和JPA EntityManager进行了交互，以便数据验证和元数据分析。Spring支持在后台初始化JPA EntityManager，因为在Spring应用程序中，这个过程通常会占用大量的时间，所以为了更好的利用后台初始化，我们需要保证JPA repositories的初始化是尽可能晚的。

### 手动控制jpa repositories的实例化时机

1. default：默认repositories都是直接就实例化了
2. Lazy：@Lazy，懒加载，只会在与存储库的第一次交互时进行实例化。如果只是在作为一个字段属性类型是不会实例化的。Lazy多用于测试场景或者本地开发。
3. DEFERRED ：

[https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#jpa.namespace](https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/" \l "jpa.namespace)

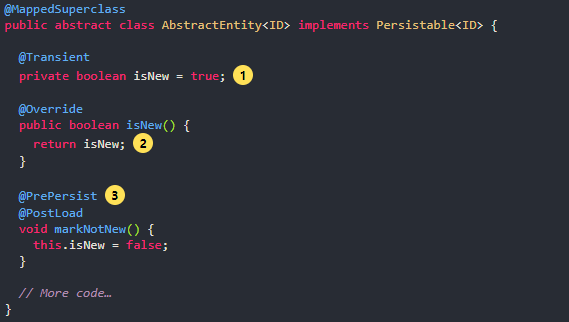
## （重点阅读）SpringDataJpa是作save还是update实体操作

可以使用CrudRepository的save方法来持久化（insert操作）或merge（update操作）。它是通过底层的JPA EntityManager来实现的。如果一个实体没有被持久化，那么将直接调用entityManager.persist()方法进行数据持久化（也就是insert）；否则，将调用entityManager.merge()方法进行合并（也就是update操作）。

SpringDataJpa通过以下策略来判断一个实体是否是新的（也就是如何判断当前操作的数据是在数据库存在还是不存在，然后对应的是作insert还是update操作）：

1. 版本类型（就是非原语言类型的属性）和id-property（默认）：默认情况下是先检查这个entity是否有原语言类型的版本属性。如果存在实体，且值为null，则认为这个实体是新的，就决定要进行的操作是insert；否则，将检查这个entity的标识符属性（也就是id-property），如果标识符属性是null，则决定将要的操作是insert，否则，是update
2. 实现Persistable接口，重写isNew()方法。true-》insert，false-》update
3. 实现EntityInfomation，这个基本上用不着，详情请参考：<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#jpa.namespace>

一个例子：



@Transient：表示isNew字段不会持久化到数据库中（就是表示这个字段和数据库没有联系）

@MappedSuperclass：

<https://blog.csdn.net/zty1317313805/article/details/80524900>

@PrePersist：就是表示在使用jpa时记录一些业务无关的字段

@PostLoad：表示markNotNew()在save到database后调用，相当于把此方法标记为了一个回调方法。

<https://blog.csdn.net/sunrainamazing/article/details/75041402>

**参考文献：**

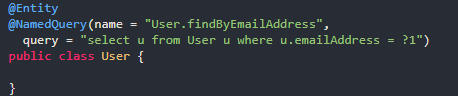
[https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#jpa.entity-persistence](https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/" \l "jpa.entity-persistence)

## Jpa的命名查询

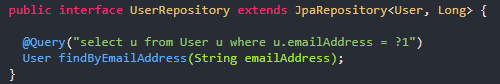
[可以在.xml文件定义<named-query/>标签或使用@NamedQuery注解来定义jpa语言查询。也可以在.xml文件中使用<named-native-query/>标签或使用@NamedNativeQuery注解来定义原生sql语句。](mailto:可以在.xml文件定义%3cnamed-query/%3e标签或使用@NamedQuery注解来定义jpa语言查询。也可以在.xml文件中使用%3cnamed-native-query/%3e标签或使用@NamedNativeQuery注解来定义原生sql语句。)

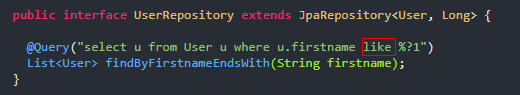
注意：使用标签的方式时，.xml文件时放在类路径下的META-INF文件夹中的。





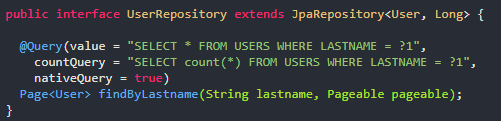
## 使用@Query注解（包含如何使用分页查询）





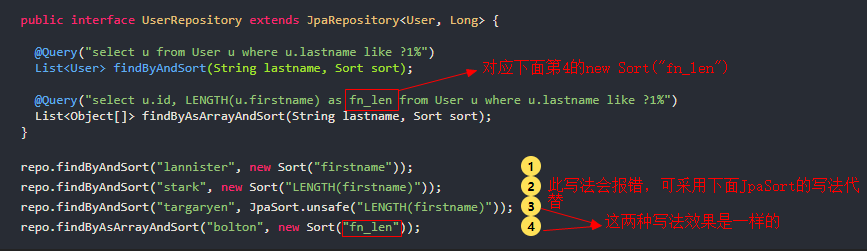
在前面的示例中，会识别类似的分隔符字符(%)，并将查询转换为有效的JPQL查询(删除%)。在执行查询时，传递给方法调用的参数将使用之前识别的LIKE模式进行扩充。

@Query注解的nativeQuery属性，使用原生sql：



<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#jpa.query-methods.at-query>

**注意：使用native时，不能动态排序（即不支持类似order By USERS.name），因为它操作的是实际声明的查询（也就是它是使用java的jdbc的数据库操作来操作sql语句，而java的jdbc操作sql是类似这样的 select \* from user where name = ?），（注意：**可以使用@EnableJpaRepositories注解来自定义方法解析的转义字符**）但是可以使用spel表达式来把要排序的字段传入sql中。也可以使用如下的方式进行排序：**



其它参考资料：

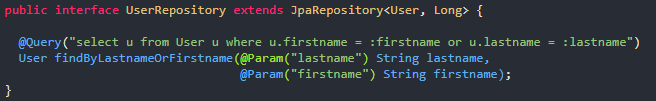
<https://blog.csdn.net/qq_20032995/article/details/79884093>

<https://blog.csdn.net/u010838785/article/details/101367432>

使用@Query注解传递参数的几种方式

<https://www.cnblogs.com/cnblog-long/p/7425923.html>

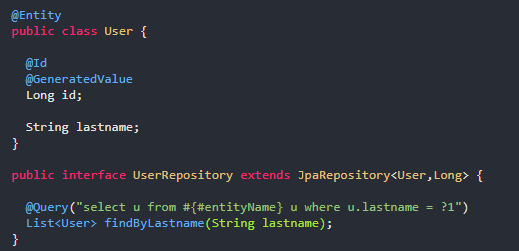
#### 使用命名参数传条件



#### 使用coalesce解决判断参数是否为空

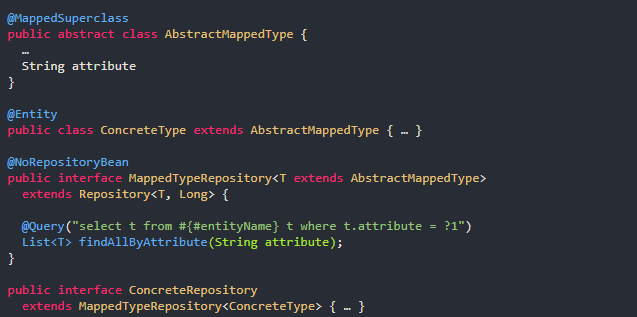
<https://blog.csdn.net/xb12369/article/details/81560225>

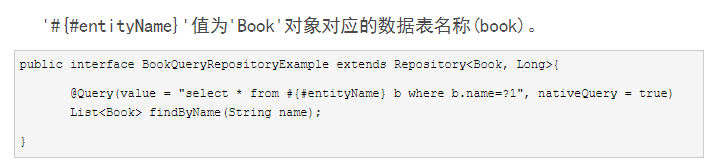
#### 采用SpEl表达式



上诉例子中的entityName是JPA提供的一个变量，代表实体名，在这个例子中就是指的User（@Entity默认将类的类名作为entity name）

使用@Query+spel表达式+@Entity定义一个通用的查询接口：





**Spel表达式参与查询**

//更新用户信息表的状态和描述（使用参数对象）

@Query(value = "update user\_info set state=:#{#userInfo.state},user\_info\_desc=:#{#userInfo.userInfoDesc} where oid =:#{#userInfo.oid}",nativeQuery = true)

@Modifying

int updateUserInfo(@Param("userInfo") UserInfo userInfo);

操作参数的SpEL表达式也可以用于操作方法参数。在这些SpEL表达式中，实体名称不可用，但是参数可用。可以通过名称或索引访问它们，如下面的示例所示：



使用类似like的关键字



对使用like进行模糊查询的查询做安全处理（使用JPA提供的escape()方法）：



## @Modifying和@Query注解配合使用完成修改、删除



上面的例子会发出两条sql，先查询再删除。在修改后，EntityManager可能包含过时了实体，这时可以配置@Modifying注解的clearAutomatically=true来从EntityManager中删除它



关于删除的更多细节，如生命周期调用的，请查看：

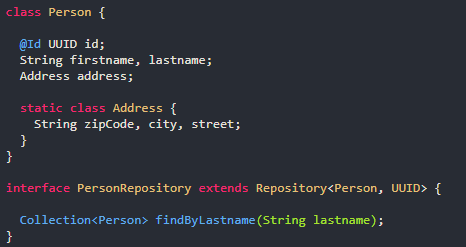
<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#jpa.modifying-queries.derived-delete>

## 实体图（2.1版本开始）

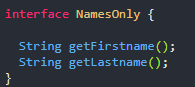
<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#jpa.entity-graph>

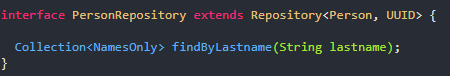
## 关于投影（只查询某些列）

一情况下，SpringData查询会投影相关实例的所有属性的对应的列，如下所示：

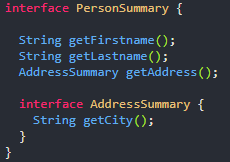


此例子中会在查询时投影Person中所有的属性对应的列，如果 只想要查询和name相关的列，可以先定义一个通用接口，然后把查询结果包装成这个接口，就可以了，如下所示：

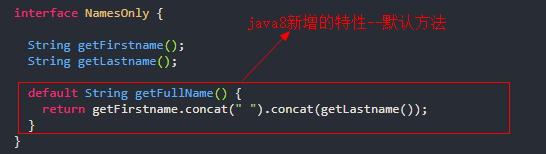




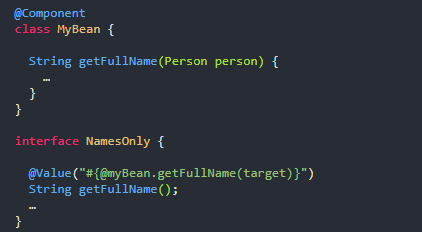
如果还想要投影address列，可以更改为：



还可以使用这种写法来定制得到的结果：



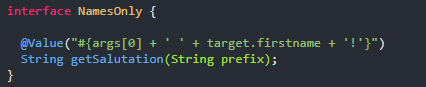
使用spel表达式的方式:



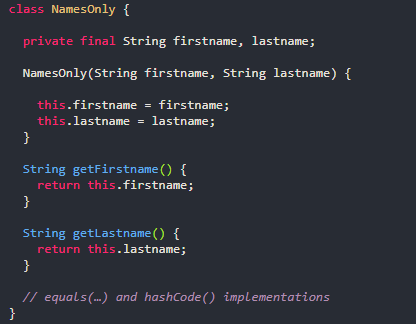
这个例子是在MyBean中的getFullName方法里定义了方法体：

，后面spel表达式就是直接调用这个方法（target代表目标对象）

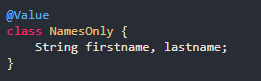
也可以写成如下的方式：



也可以使用dtd的方式（单独顶一个实体，里面只包含需要投影列的属性）：



也可以是使用lomback的@Value注解：



参考：

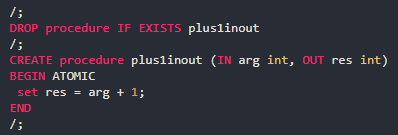
<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#projections.interfaces>

[https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#projections.dtos](https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/" \l "projections.dtos)

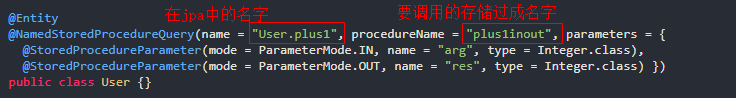
## 操作存储过程

从jpa2.1开始，支持了对存储过程的调用，下面将进行介绍。

有一个存储过程如下：



对于上面得存储过程，在Jpa中可以按如下方式写来进行调用：



其余的则是传入存储过程中的参数名

也可以使用@Procedure注解的value或proceduceName属性指定存储过程名来调用存储过程。这时候就会忽略@NamedStoredProcedureQuery注解。那上面的例子可以改写为如下方式：

或



通过使用方法名来调用存储过程



也可以通过@Procedure注解的name属性来调用@NamedStoredProcedureQuery注解已经定义好的存储过程调用。如下所示调用上面已经配置好的名为User.plusIO的存储过程调用：



如果要调用的存储过程有一个out参数，可将它作为java代码中方法的返回值。如果有多个out参数，且在@NamedStoredProcedureQuery中指定了多个out参数，那么java代码中方法可以返回一个Map来接收，其中key对应注解中out参数名字。

## JPA Criteria API（建议阅读）

JPA2开始可使用criteria api构建where子句查询

### 为什么使用Criteria

**JPQL的缺点：**

先看以下语句：

String jpql = “select s from Student where s.mark > 60”;

这是一个错误示范，虽然这行代码能够通过编译，但是不能正确运行。正确的语法应该是：

String jpql = “select s from Student s where s.mark > 60”;

可见，在以往的JPQL语句中，由于语法是以string类型写入，所以编译器并不发现错误在哪，影响了开发效率。随着项目的逐渐扩大，JPQL没有错误提示的缺点将拖慢整个开发进度。

**Criteria API的优点：**

将JPQL中使用的select, from, where等语句以函数的形式表达，所以编译器能够验证，从而避免错误。它使得Java编译器可以在运行时正确的检验JPQL语句的正确性

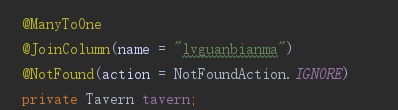
关于详细介绍Criteria构建查询的文章（建议阅读）：

<http://labs.supinfochina.com/criteria-api%E5%85%A5%E9%97%A8%E8%AE%B2%E8%A7%A3/>

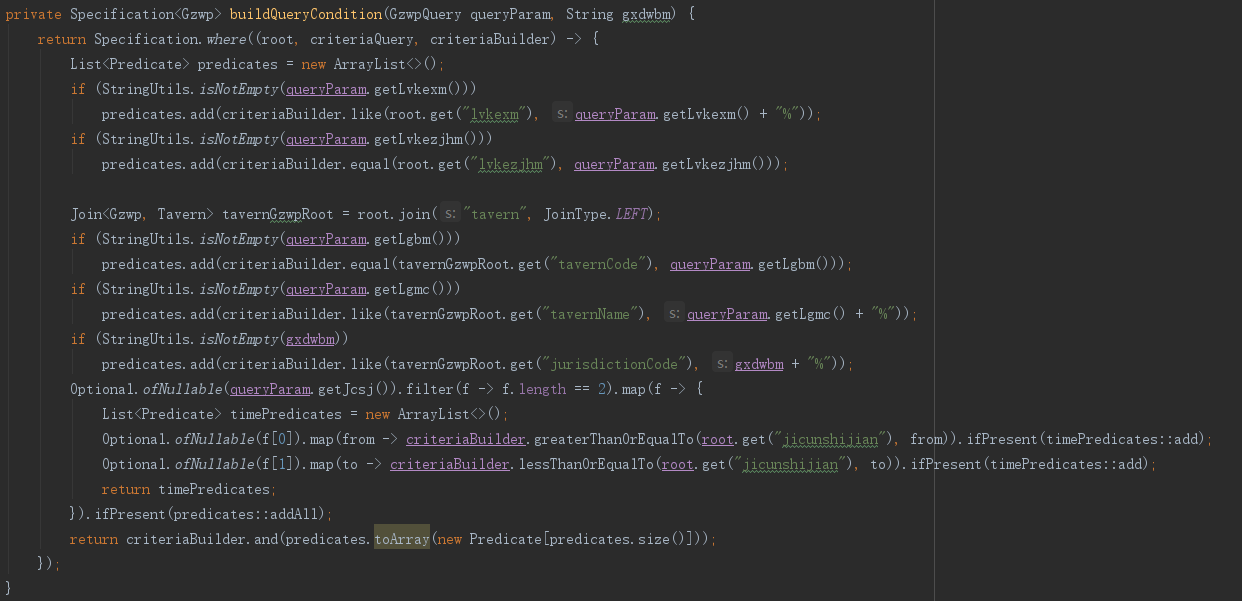
### 使用Jpa的criteria api构建查询的例子：

定义实体关系：

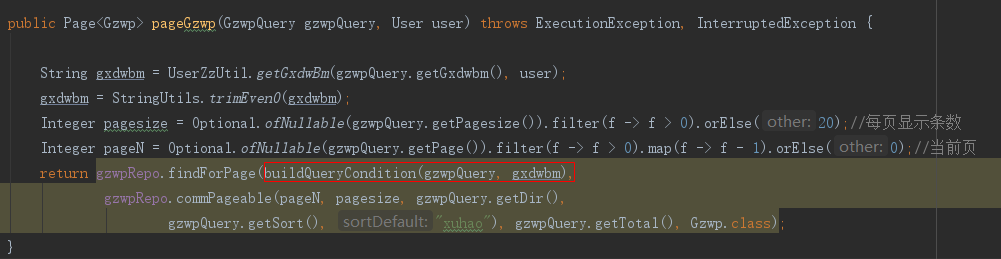
Gzwp实体关系定义部分（由于是单向的Tavern实体中不需要定义关系）：



构造where子句：



使用上面构造的where子句条件：



### 使用Metamodel API来加强Criteria api构建查询的类型验证

**介绍：**

Criteria API的优点在于避免string类型的SQL语句，从而让编译器能够检查错误。但是如果你较真一些的话，会发现并不是所有的属性都避免使用string，依旧有一部分是string类型。这时候就可以使用Metamodel API。

在JPQL语句中，由于SQL是string类型书写的，在编译器中无法自动发现错误在哪，会影响开发效率。Criteria API被设计用来改正这个问题，可以把select，from，where，update等等表示出来，避免了string，可以让编译器检查语法，从而避免错误。但是仍有一些不足，因为像属性名这种依旧是string，不是非常彻底。

Metamodel API需要在相同的package下创建新的实体来生成元模型，之后就可以直接调用属性名，而不是用string。这样，就完全避免了string的产生。

Metamodel API的使用与介绍（建议阅读）：

<http://labs.supinfochina.com/metamodel-api%E5%85%A5%E9%97%A8%E8%AE%B2%E8%A7%A3/>

### 参考资料：

关于Criteria api：

<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#specifications>

关于Metamodel api：

<https://docs.jboss.org/hibernate/jpamodelgen/1.0/reference/en-US/html_single/#whatisit>

## 使用Example Matchers 完成查询

<https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#query-by-example.matchers>

## 其他

<https://www.cnblogs.com/520playboy/p/7552141.html>

<https://blog.csdn.net/XU906722/article/details/84778788>