Spring Data JPA

官方文档 中文翻译

Table of Contents

关于本书 1.1

介绍 1.2

    关于这本指南 1.2.1

前言 1.3

第一章 新增及注意点 1.4

第二章 项目依赖 1.5

第三章 使用Spring Data Repositories 1.6

    3.1 核心概念 1.6.1

    3.2 查询方法 1.6.2

    3.3 定义repository的接口 1.6.3

    3.4 定义查询方法 1.6.4

    3.5 创建repository实例 1.6.5

    3.6 Spring Data repositories的自定义实现 1.6.6

    3.7. 从聚合根处发布事件 1.6.7

    3.8 Spring Data 的拓展 1.6.8

Spring Data JPA 参考指南 中文版

阅读地址

- Spring Data JPA 参考指南 目前正在翻译当中, 为了方便理解, 我们也会加入自己的观点和例子, 并不会完全照搬翻译, 希望大家理解也欢迎大家一起加入和完善
- 如果发现不通顺或者有歧义的地方, 可以在评论里指出来, 我们会及时改正的.
- Github托管地址
- 原文地址
- 我们会开放权限给每一个加入的伙伴 (翻译或者校对)，请提前邮箱联系 ityouknow@126.com
- 欢迎大家加入JPA交流群，群号：**592638519**
- 欢迎大家加入JPA翻译社QQ群，群号是：**567323266**
- 建议使用GitBook Editor编辑

如何参与

任何问题都欢迎直接联系我 ityouknow@126.com

Gitbook 提供了非常棒的在线编辑功能, 所以想贡献的同学可以直接联系我申请权限！

许可证

贡献者列表

成员	联系方式	**Github**
ityouknow	ityouknow@126.com	https://github.com/ityouknow
北方素素	beifangsusu@189.cn	https://github.com/bfss
可燃冰	ken.kong@outlook.com	https://github.com/ken-kong
dzzxjl	dzzxjl@126.com	https://github.com/dzzxjl
xiangflight	xiangflight@foxmail.com	https://github.com/xiangflight
guoxifeng	13193755630@163.com	https://github.com/guoxifeng
我的天空你的城	cytxiamen@163.com	https://gitee.com/domainchen
Chuck	986022405@qq.com	https://github.com/qiankaiyu
7451	handersome992@163.com	https://github.com/huangliang992
Hyun A	gq_941223@126.com	https://github.com/Sacokzk

介绍

很高兴能向大家介绍 spring data jpa, 这是一个数据方便的标准封装, 我们认为它是 java (JVM) 世界中构建技术的一个飞跃.

spring data jpa 提供了:

- 像操作对象一样操作数据库
- 标准的封装

关于这本指南

这本用户指南还并不完善, 就像 spring data jpa一样还在不断升级中

在这本指南中, spring data jpa的一些功能并没有被完整的展示出来. 一些内容的解释也并不是十分的清楚, 或者假设关于 spring data jpa你知道得更多. 我们需要你的帮助来完善这本指南. 在 spring data jpa网站上你可以找到更多关于完善这本指南的信息.

前言

项目信息

- 版本控制 - http://github.com/spring-projects/spring-data-jpa
- Bugtracker - https://jira.spring.io/browse/DATAJPA
- 版本库 - https://repo.spring.io/libs-release
- 里程碑库 - https://repo.spring.io/libs-milestone
- 快照存储库 - https://repo.spring.io/libs-snapshot

第一章 新增及注意点

1.1. Spring Data JPA 1.11的新增功能点

- 提高了与Hibernate 5.2的兼容性
- 支持通过实例来查询的任意匹配模式
- 优化分页查询
- 支持在查询推导中使用 `exists` 映射

1.2. Spring Data JPA 1.10的新增功能点

- 支持在查询方法中使用 `Projections（映射）` ，可获取对象更加细化的信息
- 支持通过实例来查询
- 增加以下注解： `@EntityGraph` , `@Lock` , `@Modifying` , `@Query` , `@QueryHints` 和 `@Procedure`
- 集合表达式支持Contains关键词
- AttributeConverters for ZoneId of JSR-310 and ThreeTenBP.
- 升级到Querydsl 4, Hibernate 5, OpenJPA 2.4 and EclipseLink 2.6.1

第二章 项目依赖

由于spring data依赖于很多不同的组件,其中大部分都有不同的版本号，找到兼容的最简单方式就是利用我们定义的bom模版，在maven项目中，你可以在pom文件中定义这样的片段 `<dependencyManagement />`

例1. 在BOM中使用spring data发布的版本

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.data</groupId>
      <artifactId>spring-data-releasetrain</artifactId>
      <version>${release-train}</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
  </dependencies>
</dependencyManagement>
```

最新发布的版本是 `Kay-SR1` 。名字是按照字母顺序的升序来排列，最新可用的列表在这里。版本的命名格式为: `${name}-${release}` ,其中 `release` 是下列5种之一：

- BUILD-SNAPSHOT - 最新的快照
- M1, M2 etc. - 里程碑
- RC1, RC2 etc. - 新版本预发布
- RELEASE - 正式发布的版本
- SR1, SR2 etc. - 服务版本

我们可以在spring data using bom这个链接中查看如何使用BOM模版。

此时，你在对JPA的模块引用中不需要添加版本号，如下。

例2.声明一个JPA的模块引用

```
<dependencies>
  <dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
  </dependency>
<dependencies>
```

2.1 使用Spring Boot管理依赖

Spring boot 已经选择了最新的版本，如果你想更新到最新的版本，只需配置 `spring-data-releasetrain.version` 选择不同的版本来使用即可。

2.2 Spring 框架

当前的Spring Data模版需要依赖Spring框架5.0.1发布版或者更高，也可以使用旧版中修复了bug的版本，但是还是推荐使用最新的版本。

第三章 使用Spring Data Repositories

使用Spring Data Repository可极大地简化为了实现各种持久层的数据库访问而写的样板代码量。

> Spring数据存储库文件和你的module层
>
> 本章解释了Spring的核心概念和接口数据存储库。本章信息来自Spring数据通用模块，它使用配置和代码示例Java Persistence API(JPA)模块。您正在使用的模块等同于调整XML名称空间声明和拓展类型。名称空间引用涵盖了 XML配置（所有使用库API的Spring Data模块都支持），库查询关键字涵盖了查询关键词库支持的抽象方法。您可到本指南的相关章节去去查询module层具体特性的详细信息。

3.1. 核心概念

Spring Data库的核心接口是 `Repository` 。它使用domain类去管理，domain类中的id类型作为类型参数。这个接口主要作为一个标记接口，依靠具体的类型运作并帮助您发现接口， `CrudRepository` 提供丰富的CRUD功能去管理实体类。

例 3. CrudRepository 接口

```
public interface CrudRepository<T, ID extends Serializable>

 extends Repository<T, ID> {

 <S extends T> S save(S entity);     (1)

 T findOne(ID primaryKey);           (2)

 Iterable<T> findAll();              (3)

 Long count();                       (4)

 void delete(T entity);              (5)

 boolean exists(ID primaryKey);      (6)

 // … more functionality omitted.

}
```

(1) 保存给定的实体。

(2) 返回给定id的实体。

(3) 返回所有实体。

(4) 返回实体的数量。

(5) 删除给定的实体。

(6) 表明一个指定id的实体是否存在。

> 我们还提供持久性特定于技术的抽象如: `JpaRepository` 或 `MongoRepository` . 这些接口继承于 `CrudRepository` ，实现了特定的一些功能

`CrudRepository` 有一个 `PagingAndSortingRepository` 抽象,增加了额外的方法来简化对实体的分页访问:

例4： PagingAndSortingRepository

```
public interface PagingAndSortingRepository<T, ID extends Serializable>
  extends CrudRepository<T, ID> {

  Iterable<T> findAll(Sort sort);

  Page<T> findAll(Pageable pageable);
}
```

进入 `用户类别` 的第二页（每一页的条目是20），可以照下面这样来分页

```
PagingAndSortingRepository<User, Long> repository = // … get access to a bean
Page<User> users = repository.findAll(new PageRequest(1, 20));
```

除了查询方法外，还有统计查询和删除查询。

例5 查询并统计

```
public interface UserRepository extends CrudRepository<User, Long> {

  Long countByLastname(String lastname);
}
```

例6 查询并删除

```
public interface UserRepository extends CrudRepository<User, Long> {

  Long deleteByLastname(String lastname);

  List<User> removeByLastname(String lastname);

}
```

3.2. 查询方法

标准的CRUD功能存储库通常对底层数据存储查询。Spring Data把这些查询变成了四个步骤的过程:

1、声明一个接口继承Repository或其子类，输入实体类型和ID类型。

```
interface PersonRepository extends Repository<User, Long> { … }
```

2、在接口里声明查询方法。

```
interface PersonRepository extends Repository<Person, Long> {
  List<Person> findByLastname(String lastname);
}
```

3、为这些接口创建代理实例，也可通过 JavaConfig :

```
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableJpaRepositories
class Config {}
```

或通过xml配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:jpa="http://www.springframework.org/schema/data/jpa"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/data/jpa
     http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

   <jpa:repositories base-package="com.acme.repositories"/>

</beans>
```

本例中使用了JPA名称空间。如果您正在使用repository中的抽象为任何其他数据源,你需要改变这种适当的名称空间声明你的存储模块来与jpa支持,例如: `mongodb` 。

注意，不用通过Java变量来配置包，默认情况下回根据注解的类来自动声明。定制的包扫描可以使用 `basePackage` 属性，特定的库可以使用 `@Enable` 来注解。

4、获得repository 实例注入并使用它。

```
class SomeClient {

  @Autowired
  private final PersonRepository repository;

  SomeClient(PersonRepository repository) {
      this.repository = repository;
  }

  public void doSomething() {
    List<Person> persons = repository.findByLastname("Matthews");
  }
}
```

接下来的小节详细解释每一个步骤。

3.3. 定义repository的接口

首先需要定义实体类的接口，接口必须继承repository并且输入实体类型和ID类型，如果需要用到CRUD方法，可以使用 `CrudRepository` 来替代 `Repository` 。

3.3.1. 自定义接口

通常,您的存储库接口将会扩展 `Repository` , `CrudRepository` 或 `PagingAndSortingRepository` 。 另外,如果你不想继承Spring Data接口,还可以注释库接口 `@RepositoryDefinition` 。 扩展 `CrudRepository` 公开了一套完整的方法来操作您的实体。 如果你喜欢选择调用方法,简单地从 `CrudRepository` 中复制你想要的方法到你的repository。

> 这允许您在已有的Spring Data存储库功能的基础上弹性地定义自己的抽象。

例7.有选择地公开CRUD方法

```
@NoRepositoryBean
interface MyBaseRepository<T, ID extends Serializable> extends Repository<T, ID> {

  Optional<T> findById(ID id);

  <S extends T> S save(S entity);
}

interface UserRepository extends MyBaseRepository<User, Long> {
  User findByEmailAddress(EmailAddress emailAddress);
}
```

第一步你定义了一个公共基础的接口提供了 `findById(…)` 和 `save(...)` 方法,这些方法将会引入到你选择的spring Data的实现类中，例如JPA： `SimpleJpaRepository` ,因为他们匹配 `CrudRepository` 的方法签名，所以 `UserRepository` 将会具备save Users和根据ID查询的功能，当然也具备 `findByEmailAddress` 的功能。

> 注意，如果中间的repository接口添加了 `@NoRepositoryBean` 注解，确认你所有的repository都添加了这个注解，这时候spring Data在运行时将不会创建实例。

3.3.2. Repository方法对Null的处理

在Spring Data 2.0中，Repository的CRUD方法使用Java 8的Optional返回一个独立的合计实例，表明一个值可能缺失。此外，Spring Data还支持查询方法返回其他包装类:

- `com.google.common.base.Optional`
- `scala.Option`

- `io.vavr.control.Option`
- `javaslang.control.Option` (deprecated as Javaslang is deprecated)

查询方法也可不返回任何包装类，缺失的查询结果将返回null。返回集合，可选集合，包装类和流的Repository方法将返回相应的空表示而不返回null。详情请见Repository query return types

Nullability注解

你可以使用

3.3.3. 使用Spring Data多模块来创建Repositories

使用唯一的Spring Data模块在应用中是非常简单，但有时候我们需要多的Spring Data模块，比如：需要定义个Repository去区分两种不同的持久化技术，如果在class path中发现多个Repository时，spring data会进行严格的配置限制，确保每个repository或者实体决定绑定那个Spring Data模块：

1、如果 repository 定义继承特殊的Repository，他是一个特殊的Spring Data模块

2、如果实体注解了一个特殊的声明，它是一个特殊的spring Data模块，spring Data模块接收第三方的声明（例如：JPA's `@Entity`）或者提供来自 Spring Data MonggoDB/Spring Data Elasticsearch的 `@Document` 。

例8. 自定义特殊的Repostity

```
interface MyRepository extends JpaRepository<User, Long> { }

@NoRepositoryBean
interface MyBaseRepository<T, ID extends Serializable> extends JpaRepository<T, ID> {
  …
}

interface UserRepository extends MyBaseRepository<User, Long> {
  …
}
```

`MyRepository` and `UserRepository` 继承于 `JpaRepository`在这个层级中是对Spring Data JPA 模块的合法替代

例9. 使用一般的接口定义Repository

```
interface AmbiguousRepository extends Repository<User, Long> {
 …
}

@NoRepositoryBean
interface MyBaseRepository<T, ID extends Serializable> extends CrudRepository<T, ID> {
  …
}

interface AmbiguousUserRepository extends MyBaseRepository<User, Long> {
  …
}
```

`AmbiguousRepository` 和 `AmbiguousUserRepository` 仅继承于 `Repository` 和 `CrudRepostory` 在他们的层级。当它们使用一个spring data模块的时候是完美的，但是如果使用多模块spring data 是，spirng 无法区分每个Repository的范围。

例10. 使用实体类注解来定义Repository的使用范围

```
interface PersonRepository extends Repository<Person, Long> {
 …
}

@Entity
public class Person {
  …
}

interface UserRepository extends Repository<User, Long> {
 …
}

@Document
public class User {
  …
}
```

`Person` 使用了 `@Entity` 注解 `PersonRepository` 引用了它，所以这个仓库清晰的使用了Sping Data JPA。 `UserRepository` 引用的 `User` 声明了 `@Document` 表面这个仓库将使用Spring Data MongoDB 模块。

例11. 使用混合的注解来定义仓库

```
interface JpaPersonRepository extends Repository<Person, Long> {
 …
}

interface MongoDBPersonRepository extends Repository<Person, Long> {
 …
}

@Entity
@Document
public class Person {
  …
}
```

这个例子中实体类`Person···使用了两种注解，表明这个实体类既可以用于` JpaPersonRepository `也可以用于` MongoDBPersonRepository ```，Spring Data不能确定仓库类型导致未定义的行为。

通过Repository继承或者使用注解都是为了确定使用那个Spring Data模块。使用多个注解到同一个实体来达到多类型的持久化技术，Spring Data不在限制只能绑定到一个Repostitory中。

最后一种方法来区分不同的仓库类型，使用包路径来判断。不同的包路径下的仓库使用不同的仓库类型，通过在配置类`configuration`中声明注解来实现，也可以通过xml配置来定义。

例12： 通过注解来实现不同包路径，使用不同的仓库

```
@EnableJpaRepositories(basePackages = "com.acme.repositories.jpa")
@EnableMongoRepositories(basePackages = "com.acme.repositories.mongo")
interface Configuration { }
```

3.4. 定义查询方法

repository 代理有两种方法去查询。一种是根据方法名或者自定义查询，可用的选项取决于实际的商店。然而,根据相应的策略来决定实际SQL的创建，让我们看看选择项吧。

3.4.1. 查询查找策略

以下策略可供查询库基础设施来解决。您可以配置策略名称空间通过 `query-lookup-strategy` 属性的XML配置或通过 `queryLookupStrategy` 启用的属性 `${store}` 库注释的Java配置。一些策略可能不支持特定的数据存储。

- `create` 试图构建一个能找到查询的查询方法名称。 通常的做法是把给定的一组注明前缀的方法名和解析的方法。
- `USE_DECLARED_QUERY` 试图找到一个声明查询并将抛出一个异常情况。查询可以定义注释上。
- `CREATE_IF_NOT_FOUND` (默认)结合 `CREATE` 和 `USE_DECLARED_QUERY` 。 看起来一个声明查询第一,如果没有声明查询发现,它创建一个定制的基于名称的查询方法。这是默认查找策略,因此如果你不使用任何显式配置。 它允许快速查询定义的方法名,还custom-tuning这些查询通过引入需要查询。

3.4.2 创建查询

query builder机制内置为构建约束查询库的实体。 带前缀的机制 `findXXBy` , `readAXXBy` , `queryXXBy` , `countXXBy` , `getXXBy` 自动解析的其余部分。进一步引入子句可以包含表达式等 `Distinct` 设置不同的条件创建查询。 然而,第一个 `By` 作为分隔符来表示实际的标准的开始。 在一个非常基础的查询,可以定义条件 `And` 或者 `Or` 。

例 13. 根据方法名创建查询

```
public interface PersonRepository extends Repository<User, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastnam
e);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstna
me);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstna
me);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstna
me);

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

实际结果的解析方法取决于你的持久性存储创建查询。-然而,也有一些一般要注意的事情。

- 遍历表达式通常结合运算符连接。您可以把表达式 `And` 和 `Or` , `Between` , `LessThan` (不超过), `GreaterThan` , `Like` 等运算符，这些操作对不同的数据库可能有所不同，具体参考各参考文档
- 方法解析支持设置 `IgnoreCase` 在属性上面（如， `findByLastnameIgnoreCase(…)`),或者支持查询所有属性忽略大小写（如， `findByLastnameAndFirstnameAllIgnoreCase(…)`), 忽略大小写支持所有的数据库，其它的查询参考相关文档
- 您可以应用静态排序通过附加一个 `OrderBy` 基准进行排序,引用属性和方向提供了一个排序(`asc` 或 `Desc`)。 创建一个支持动态排序的查询方法,明白了特殊参数处理 。

3.4.3. 属性表达式

属性表达式只能引用的直接财产管理的实体,如前面的示例所示。 在创建查询时你已经确保解析房地产管理的域类的一个属性。 然而,您还可以定义约束通过遍历嵌套属性。 假设一个 `Person` 有一个 `Address` 与一个 `Zipcode` 。 在这种情况下一个方法的名称

```
List<Person> findByAddressZipCode(ZipCode zipCode);
```

创建属性遍历 `x.address.zipCode` 。方法执行首先解释整个部分(`AddressZipCode`)作为财产和检查的域类属性的名称(小写形式)。 分割源在驼峰式大小写部分从右侧头部和尾巴,试图找到对应的属性,在我们的例子中,分割为 `AddressZip` 和 `Code` 。 分裂不匹配,该算法分割点移动到

左(`Address` , `Zipcode`)然后继续,

在大多数情况下，这种算法有可能会出现错误，您可以使用来解决这种模糊性 _ 在方法名来手动定义遍历点。所以我们的方法名称最终将像这样:

```
List<Person> findByAddress_ZipCode(ZipCode zipCode);
```

如果你的属性名称包含下划线(如。 first_name 中下划线),建议使用驼峰的方式来避免。

3.4.4 特殊参数处理

处理参数查询只需方法参数定义为已经在上面的例子中。 除了基础查询将会认识到某些特定类型 `Pageable` 和 `Sort` 应用动态查询分页和排序

例 14. 使用 `Pageable` , `Slice` 和 `Sort` 来查询

```
Page<User> findByLastname(String lastname, Pageable pageable);

Slice<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

第一个方法允许在你的查询方法的静态定义查询中通过一个org.springframework.data.domain.Pageable实例来动态的添加分页。分页类知道元素的总数和可用页数。它通过基础库来触发一个统计查询计算所有的总数。由于这个查询可能对store消耗巨大，可以使用Slice来替代。Slice仅仅知道是否有下一个Slice可用，这对查询大数据已经足够了。

排序选项和分页的处理方式一样。如果你需要排序，简单的添加一个org.springframework.data.domain.Sort参数到你的方法即可。也正如你所见，简单的返回一个列表也是可以的，在这种情况下，生产的分页实例所需的附加元数据将不会被创建(这也意味着额外的计数查询可能需要但不一定被公布)。

> 要找到在你的查询中有多少页，你需要触发一个额外的计数查询。按照默认来说这个查询可以从你实际触发查询中衍生出来

3.4.5. 限制查询结果

查询方法的结果可以通过关键字first或者top来限制,它们可以交替使用。在top/firest后添加数字来表示返回最大的结果数。如果没有数字，则默认假定1作为结果大小。

示例15 用 `Top` 和 `First` 查询限制结果大小

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

限制表达式也支持Distinct关键字。对于限制查询的结果集定义到一个实例中包装这个结果到一个Optional中也是被支持的。

如果分页或者切片被应用到一个限制查询分页(计算多少页可用)则它也能应用于限制结果。

> 要注意结合通过Sort参数动态排序的限制结果容许表达查询的方法为“K”最小的，以及“K”最大的元素。

3.4.6. 流查询结果

查询方法能对以JAVA 8的Stream为返回的结果进行逐步处理。而不是简单地包装查询结果在被用来执行流的流数据存储特定的方法。

例16 以JAVA 8的Stream来进行查询的流处理结果

```
@Query("select u from User u")

Stream<User> findAllByCustomQueryAndStream();

Stream<User> readAllByFirstnameNotNull();

@Query("select u from User u")

Stream<User> streamAllPaged(Pageable pageable);
```

> 一个数据流可能包裹底层数据存储特定资源，因此在使用后必须关闭。 你也可以使用close()方法或者JAVA 7 try-with-resources区块手动关闭数据流。

例17 在try-with-resources块中操作一个Stream

```
try(Stream<User stream = repository.findAllByCustomQueryAndStream()){

stream.forEach(...);

}
```

> 当前不是所有的Spring Data模块都支持Stream作为返回类型

3.4.7. 异步查询结果

```
@Async
Future<User> findByFirstname(String firstname);               (1)

@Async
CompletableFuture<User> findOneByFirstname(String firstname);  (2)

@Async
ListenableFuture<User> findOneByLastname(String lastname);     (3)
```

- (1) 使用 `java.util.concurrent.Future` 作为返回类型
- (2) 使用 `Java 8` `java.util.concurrent.CompletableFuture` 作为返回类型
- (3) 使用 `org.springframework.util.concurrent.ListenableFuture` 作为返回类型

3.5. 创建repository实例

在这个部分，你创建实例和为repository接口定义的bean。这样做的一个方法是使用Spring的名称空间，这是与每个Spring Data模块，支持存储机制，虽然我们一般建议使用JAVA配置风格的配置。

3.5.1. XML配置

每一个Spring Data模块都包含repositories元素能够让你简单的基于base-package定义来进行Spring扫描。

例 21. 通过XML来开启Spring Data repositories

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.springframework.org/schema/data/jpa"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

  <repositories base-package="com.acme.repositories" />

</beans:beans>
```

在上面这个例子中，Spring会在 `com.acme.repositories` 和它的子包中扫描继承 `Repository` 或其子类的接口。框架会为这些找到的接口注册持久化 `FactoryBean` （技术特定），为处理查询方法的调用创建合适的代理。每个bean注册的名称都是来自于接口名，所以一个名为 `UserRepository` 的接口注册名称是 `userRepository` 。 `base-package` 属性允许使用通配符，所以你可以定义一个被扫描包名的模式。

使用**Filter**

默认情况下，框架会为在back-package定义的包中每个继承 `Repository` 或其子接口的接口创建一个bean实例。然而，你可能想要更精细地控制为哪个接口创建bean实例。为此，你可以在 `<repositories />` 内使用 `<include-filter />` 和 `<exclude-filter >` 。语义和Spring的上下文命名空间中的元素是完全等价的。详细信息请参阅Spring参考文档对这些元素的介绍。

例如，要将某些接口从作为Repository的实例中排除，可以使用如下配置：

例 22. 使用exclude-filter元素

```
<repositories base-package="com.acme.repositories">
  <context:exclude-filter type="regex" expression=".*SomeRepository" />
</repositories>
```

这个例子排除了以SomeRepository结尾的接口被实例化。

3.5.2. Java配置

也可以在一个Java配置的类使用 `@Enable${store}Repositories` 注解来触发repository框架。有关Spring容器的基于Java配置的介绍，请参阅参考文档1。

以下是一个启用Spring Data repositories的配置示例。

例 23. 基于repository配置的注解示例

```
@Configuration
@EnableJpaRepositories("com.acme.repositories")
class ApplicationConfiguration {

  @Bean
  EntityManagerFactory entityManagerFactory() {
    // …
  }
}
```

> 该示例使用特定于JPA的注解，您可以根据实际使用的store模块做相应改变。这同样适用于 `EntityManagerFactory` bean的定义。请参阅有关store特定配置的章节。

3.5.3. 独立使用

您还可以使用Spring容器之外的repository基础架构，例如CDI环境。你仍然需要在classpath中添加一些Spring库，但是通常来说你可以在代码中设置repositories。 提供repository支持的Spring Data模块提供了一个RepositoryFactory（持久化的技术特定），使用如下所示。

例 24. repository工厂的独立使用

```
RepositoryFactorySupport factory = … // Instantiate factory here
UserRepository repository = factory.getRepository(UserRepository.class);
```