<http://shensuqiao.iteye.com/blog/2096114>

# ****1.    概述****

Spring JPA通过为用户统一创建和销毁EntityManager，进行事务管理，简化JPA的配置等使用户的开发更加简便。

Spring Data JPA是在Spring JPA的基础上，对持久层做了简化。用户只需声明持久层的接口，不需要实现该接口。Spring Data JPA内部会根据不同的策略、通过不同的方法创建Query操作数据库。

# ****2.    Spring Data JPA jar文件****

到目前为止，社区提供的最新的Spring Data JPA的jar文件如下：

-       spring-data-commons-1.8.0.RELEASE.jar：提供Spring Data共享的基础框架，适合各个子项目使用，支持跨数据库持久化。

-       spring-data-jpa-1.6.0.RELEASE.jar：简化创建 JPA 数据访问层

# ****3.    Spring Data JPA 依赖的jar文件****

现在主流的JPA实装包括Eclipselink、Toplink、OpenJPA和Hibernate。

SpringFramework的JPA机能(org.springframework.orm.jpa 中提供)对这四种实装都能支持。

但Spring JPA Data只支持JPA2.0，不支持JPA1.0。所以在TopLink上使用时可能会出错。

# ****4.    使用方法****

Spring Data JPA 简化持久层开发大致需要如下三个步骤。

       1)声明持久层接口，该接口继承Repository <T,ID>或其子接口，

           T是领域实体，ID是领域实体的主键类型。

      例：

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

       2)在持久层的接口中声明需要的业务方法，Spring Data JPA将会根据指定的策略（请

  参照4.3章节）为该方法生成实现代码。用户不需要实现该接口。

      例：

|  |
| --- |
| List<User> findByLastname(String lastname); |

        3)在Spring的配置文件中添加配置，为声明的接口设定代理对象。

      例：

|  |
| --- |
| <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.jpa.data.sample"  entity-manager-factory-ref="entityManagerFactory"   transaction-manager-ref="transactionManager"/> |

  注：

          ①Spring会在” base-package”中定义的package和其子package中搜寻继承了Repository的接口。

          ②entityManagerFactory和transactionManager的定义，请参照Spring JPA的使用方法。

       4)获得并使用repository的实例。

           ①在Spring Container中使用

|  |
| --- |
| public class SomeClient {  @Autowired  private UserRepository repository;    public void doSomething() {      User u = new User();  User user = repository. save (u);  }  } |

            ② 在Spring Container外使用

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

## ****4.1 Repository 接口****

Repository是SpringData的核心接口，它并不提供任何方法，用户需要自己定义需要的方法。

**4.1.1 继承Repository接口的两种方法**

 ①直接继承

|  |
| --- |
| public interface UserDao extends Repository<User, Long> { …… } |

 ②使用@RepositoryDefinition注解

|  |
| --- |
| @RepositoryDefinition(domainClass = User.class, idClass = Long.class)   public interface UserDao { …… } |

**4.1.2 其他Repository接口**

|  |  |
| --- | --- |
| CrudRepository  (Spring Data提供) | 继承Repository，提供增删改查方法，可以直接调用。 |
| PagingAndSortingRepository  (Spring Data提供) | 继承CrudRepository，增加了分页查询和排序两个方法 |
| JpaRepository  (Spring Data JPA提供) | 继承PagingAndSortingRepository，是针对JPA技术的接口，提供flush()，saveAndFlush()，deleteInBatch()等方法 |

**4.1.3 用户组定义实现(Spring Data提供的机能)**

Spring Data的repository允许用户自定义操作数据库的方法。用户可以与原有的repository结合起来使用。

**4.1.3.1 为单个的repository添加用户行为**

   实现步骤如下：

      ①定义一个接口，在此接口中声明自定义的操作数据库的方法

|  |
| --- |
| interface UserRepositoryCustom {  public void someCustomMethod(User user);  } |

      ② ①中定义接口的实现

|  |
| --- |
| class UserRepositoryImpl implements UserRepositoryCustom {  public void someCustomMethod(User user) {  //操作数据库  }  } |

      ③定义一个接口同时继承Spring原有Repository和①中定义接口

|  |
| --- |
| public interface UserRepository extends CrudRepository<User, Long>, UserRepositoryCustom {  ……  } |

      ④在Spring的配置文件中添加配置

|  |
| --- |
| <!--搜寻名为userRepositoryImpl的 class作为自定义repository的实现-->  <repositories base-package=" com.jpa.data.sample ">  <repository id="userRepository" />  </repositories>  <!--搜寻名为userRepositoryFooBar的 class作为自定义repository的实现-->  <repositories base-package=" com.jpa.data.sample " repository-impl-postfix="FooBar">  <repository id="userRepository" />  </repositories> |

**4.1.3.2为所有的repository添加用户行**为

   实现步骤如下：

      ①定义一个接口，声明自定义的操作数据库的方法

|  |
| --- |
| public interface MyRepository<T, ID extends Serializable> extends JpaRepository<T, ID> {  void sharedCustomMethod(ID id);  } |

      ②①中定义接口的实现，并继承SimpleJpaRepository类

|  |
| --- |
| public class MyRepositoryImpl<T, ID extends Serializable>  extends SimpleJpaRepository<T, ID> implements MyRepository<T, ID> {  public void sharedCustomMethod(ID id) {  // implementation goes here  }  } |

      ③取代RepositoryFactoryBean成为Spring Data repositores的base class，用于生成

          MyRepositoryImpl实例

|  |
| --- |
| public class MyRepositoryFactoryBean<T extends JpaRepository<?, ?>  extends JpaRepositoryFactoryBean<T> {  protected RepositoryFactorySupport getRepositoryFactory(…) {  return new MyRepositoryFactory(…);  }  private static class MyRepositoryFactory extends JpaRepositoryFactory{  public MyRepositoryImpl getTargetRepository(…) {  return new MyRepositoryImpl(…);  }  public Class<? extends RepositorySupport> getRepositoryClass() {  return MyRepositoryImpl.class;  }  }  } |

      ④在Spring的配置文件中添加配置

|  |
| --- |
| <repositories base-package=" com.jpa.data.sample "  factory-class="com.acme.MyRepositoryFactoryBean" /> |

## ****4.2 创建Query的三种方式****

Spring Data JPA中除了提供通过解析方法名的方式来创建Query之外，还提供了@Query和JPA NamedQueries两种方法。

**解析方法名**

Spring Data JPA会通过解析用户在持久层接口中定义的方法名来生成相应的query语句。（详细的解析方法名的规则请参照Spring Data JPA官方文档）

例：

持久层接口中定义如下：

|  |
| --- |
| public interface UserRepository extends Repository<User, Long> {  List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);  } |

将会解析为如下的query

|  |
| --- |
| select u from User u where u.emailAddress = ?1 and u.lastname = ?2 |

解析时能被识别的keyword和包含这些keyword的方法会被解析成什么样的Query，如下表所示。

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Sample** | **JPQL snippet** |
| And | findByLastnameAndFirstname | … where x.lastname = ?1 and x.firstname = ?2 |
| Or | findByLastnameOrFirstname | … where x.lastname = ?1 or x.firstname = ?2 |
| Between | findByStartDateBetween | … where x.startDate between 1? and ?2 |
| LessThan | findByAgeLessThan | … where x.age < ?1 |
| GreaterThan | findByAgeGreaterThan | … where x.age > ?1 |
| IsNull | findByAgeIsNull | … where x.age is null |
| IsNotNull,NotNull | findByAge(Is)NotNull | … where x.age not null |
| Like | findByFirstnameLike | … where x.firstname like ?1 |
| NotLike | findByFirstnameNotLike | … where x.firstname not like ?1 |
| OrderBy | findByAgeOrderByLastnameDesc | … where x.age = ?1 order by x.lastname desc |
| Not | findByLastnameNot | … where x.lastname <> ?1 |
| In | findByAgeIn(Collection<Age> ages) | … where x.age in ?1 |
| NotIn | findByAgeNotIn(Collection<Age> age) | … where x.age not in ?1 |

**使用@Query**

可以在自定义的查询方法上使用@Query来指定该方法要执行的查询语句，比如：

|  |
| --- |
| public interface UserRepository extends JpaRepository<User, Long> {    @Query("select u from User u where u.emailAddress = ?1")    User findByEmailAddress(String emailAddress);  } |

注意：

1：方法的参数个数必须和@Query里面需要的参数个数一致

2：如果是like，后面的参数需要前面或者后面加“%”

使用@Param可以用命名参数来代替位置编号，将方法参数与 JPQL 中的命名参数对应。JPQL 语句中通过": 变量"的格式来指定参数

例：

|  |
| --- |
| public interface UserRepository extends JpaRepository<User, Long> {   @Query("select u from User u where u.firstname = :firstname or u.lastname = :lastname")   User findByLastnameOrFirstname(@Param("lastname") String lastname,                                     @Param("firstname") String firstname);  } |

如果要生成更新类的Query语句，在@Query之前添加@Modifying即可。

例：

|  |
| --- |
| @Modifying  @Query("update User u set u.firstname = ?1 where u.lastname = ?2")  int setFixedFirstnameFor(String firstname, String lastname); |

注意：

1：方法的返回值应该是int，表示更新语句所影响的行数。

2：在调用的地方必须加事务，没有事务不能正常执行。

**使用JPA NamedQueries**

#### 在JPA配置文件中定义

在META-INF文件下的JPA的配置文件orm.xml中，通过<named-query/>元素进行定义。

例：

|  |
| --- |
| <named-query name="User.findByLastname">    <query>select u from User u where u.lastname = ?1</query>  </named-query> |

#### 通过Annotation配置

在Entity Bean中使用@NamedQuery(或@NamedNativeQuery)进行配置。

例：

|  |
| --- |
| @Entity  @NamedQuery(name = "User.findByEmailAddress",    query = "select u from User u where u.emailAddress = ?1")  public class User {  } |

#### 注意

①  上述两种方法都需要满足”DomainClass.methodName()”的命名规则。

②  无论是在JPA配置文件中使用<named-query/>定义还是在Entity Bean中使用@NamedQuery进行配置，

   在持久层的接口中必须声明对应的方法。

例：

|  |
| --- |
| public interface UserRepository extends JpaRepository<User, Long> {  List<User> findByLastname(String lastname);  User findByEmailAddress(String emailAddress);  } |

## ****4.3 创建Query的策略****

创建Query的策略有如下3种：

|  |
| --- |
| create：只通过解析方法名来创建Query。忽略@Query和NamedQuery方法。  use-declared-query：如果方法通过 @Query 指定了查询语句，则使用该语句创建Query；如果没有，则查找是否定义了符合条件的Named Query，如果找到，则使用该命名查询；如果两者都没有找到，则抛出异常。  create-if-not-found (default)：如果方法通过 @Query 指定了查询语句，则使用该语句创建查询；如果没有，则查找是否定义了符合条件的Named Query，如果找到，则使用该Named Query；如果两者都没有找到，则通过解析方法名字来创建Query。 |

<jpa:repositories>中提供的query-lookup-strategy 属性可以用来定义查找Query的顺序。定义方法如下：

|  |
| --- |
| <jpa:repositories base-package=" com.jpa.data.sample" query-lookup-strategy="create"/> |

# ****5.    Specifications(非重要机能)****

JPA2.0提供了Criteria API(具体的适用方法请参照JPA2.0的官方文档)，可以用于动态的生成query，并且在运行时检证其正确性。Spring Data JPA支持Criteria查询。使用方法如下

①     自定义Repository接口并继承JpaSpecificationExecutor

|  |
| --- |
| public interface UserRepository extends CrudRepository<User, Long>, JpaSpecificationExecutor {  … |

②     自定义Specification，实现Specification接口

|  |
| --- |
| public Specification<User> isYoungFemaleUser() {  return new Specification<User>() {  public Predicate toPredicate(Root<User> root, CriteriaQuery<?> query,CriteriaBuilder cb) {                Predicate isFemaleUser = cb.equal(root.get("sex").as(String.class), "girl");                Predicate isYoungUser = cb.lessThan(root.get("age").as(Integer.class), 18);    // 2个检索条件同时使用  query.where(cb.and(isFemaleUser,isYoungUser));                return query.getRestriction();  }  }; |

③     调用自定义的Repository中的方法(可由JpaSpecificationExecutor提供)，例如findAll

|  |
| --- |
| List<User> users = userRepository.findAll(isYoungFemaleUser ()); |

# ****6.    事务处理****

除了将查询的方法设为只读事务(@Transactional(readOnly=true))外，其他事务属性均采用默认值(@Transactional)。

用户可以在接口方法上使用 @Transactional 显式指定事务属性，该值覆盖 Spring Data JPA 提供的默认值。同时，也可以在业务层方法上使用 @Transactional 指定事务属性，这主要针对一个业务层方法多次调用持久层方法的情况。持久层的事务会根据设置的事务传播行为来决定是挂起业务层事务还是加入业务层的事务。@Transactional的详细用法请参照Spring的官方文档。

# ****7.    Auditing(非重要机能)****

Spring Data JPA也提供了对auditing的支持。在实体创建或更新的时候可以把操作时间或操作人一并更新到数据库里去。使用方法如下

①     Entity类拓展Auditable接口，或者继承AbstractPersistable或AbstractAuditable类

|  |
| --- |
| @Entity  public class Conference extends AbstractAuditable<User, Long>{……} |

Auditable接口、AbstractPersistable或AbstractAuditable类中提供了获得创建或更新Entity的操作时间或操作人的方法，比如

|  |
| --- |
| U getCreatedBy();  void setCreatedBy(U createdBy);  DateTime getCreatedDate();  void setCreated(Date creationDate);  U getLastModifiedBy();  void setLastModifiedBy(U lastModifiedBy);  DateTime getLastModifiedDate();  void setLastModified(Date lastModifiedDate); |

②     在orm.xml中进行配置

|  |
| --- |
| <persistence-unit-metadata>  <persistence-unit-defaults>  <entity-listeners>  <entity-listener  class="org.springframework.data.jpa.domain.support.AuditingEntityListener " />  </entity-listeners>  </persistence-unit-defaults>  </persistence-unit-metadata> |

③     声明AuditorAware的拓展类，在此类中用户可以取到当前的用户，在运行时将auditor注入到AuditingEntityListener

|  |
| --- |
| public class AuditorAwareImpl implements AuditorAware<Object> {  } |

④     在配置文件中配置配置③中的拓展类

|  |
| --- |
| <jpa:auditing auditor-aware-ref=" AuditorAwareImpl " /> |

# 8.QueryDSL

下次有空在另一篇文章里面细述。