<http://www.cnblogs.com/WangJinYang/p/4257383.html>

# [了解 Spring Data JPA](http://www.cnblogs.com/WangJinYang/p/4257383.html)

**前言**

自 JPA 伴随 Java EE 5 发布以来，受到了各大厂商及开源社区的追捧，各种商用的和开源的 JPA 框架如雨后春笋般出现，为开发者提供了丰富的选择。它一改之前 EJB 2.x 中实体 Bean 笨重且难以使用的形象，充分吸收了在开源社区已经相对成熟的 ORM 思想。另外，它并不依赖于 EJB 容器，可以作为一个独立的持久层技术而存在。目前比较成熟的 JPA 框架主要包括 Jboss 的 Hibernate EntityManager、Oracle 捐献给 Eclipse 社区的 EclipseLink、Apache 的 OpenJPA 等。

Java持久化规范，是从EJB2.x以前的实体Bean(Entity bean)分离出来的，EJB3以后不再有实体bean，而是将实体bean放到JPA中实现。JPA是sun提出的一个对象持久化规范，各JavaEE应用服务器自主选择具体实现，JPA的设计者是Hibernate框架的作者，因此Hibernate作为Jboss服务器中JPA的默认实现，Oracle的Weblogic使用EclipseLink(以前叫TopLink)作为默认的JPA实现，IBM的Websphere和Sun的Glassfish默认使用OpenJPA(Apache的一个开源项目)作为其默认的JPA实现。  
JPA的底层实现是一些流行的开源ORM(对象关系映射)框架，因此JPA其实也就是java实体对象和关系型数据库建立起映射关系，通过面向对象编程的思想操作关系型数据库的规范。

**Spring 框架对 JPA 的支持**

Spring 框架对 JPA 提供的支持主要体现在如下几个方面：

* 首先，它使得 JPA 配置变得更加灵活。JPA 规范要求，配置文件必须命名为 persistence.xml，并存在于类路径下的 META-INF 目录中。该文件通常包含了初始化 JPA 引擎所需的全部信息。Spring 提供的 LocalContainerEntityManagerFactoryBean 提供了非常灵活的配置，persistence.xml 中的信息都可以在此以属性注入的方式提供。
* 其次，Spring 实现了部分在 EJB 容器环境下才具有的功能，比如对 @PersistenceContext、@PersistenceUnit 的容器注入支持。
* 第三，也是最具意义的，Spring 将 EntityManager 的创建与销毁、事务管理等代码抽取出来，并由其统一管理，开发者不需要关心这些，业务方法中只剩下操作领域对象的代码，事务管理和 EntityManager 创建、销毁的代码都不再需要开发者关心了。

**Spring Data JPA 更简洁**

Spring Data JPA 框架，主要针对的就是 Spring 唯一没有简化到的业务逻辑代码，至此，开发者连仅剩的实现持久层业务逻辑的工作都省了，唯一要做的，就只是声明持久层的接口，其他都交给 Spring Data JPA 来帮你完成！

下面就来了解Spring Data JPA。

**1.下载需要的包。**

需要先 下载Spring Data JPA 的发布包（需要同时下载 Spring Data Commons 和 Spring Data JPA 两个发布包，Commons 是 Spring Data 的公共基础包），并把相关的依赖 JAR 文件加入到 CLASSPATH 中。

**2.让持久层接口 Dao（以UserDao）  继承 Repository 接口**。

该接口使用了泛型，需要为其提供两个类型：第一个为该接口处理的域对象类型，第二个为该域对象的主键类型。 如下：

Spring Data JPA 风格的持久层接口：

public interface UserDao extends Repository<AccountInfo, Long> {

public AccountInfo save(AccountInfo accountInfo);

}

不需要***UserDao***的实现类，框架会为我们完成业务逻辑。

**3.在 Spring 配置文件中启用扫描并自动创建代理的功能。**

<-- 需要在 <beans> 标签中增加对 jpa 命名空间的引用 -->

<jpa:repositories base-package="footmark.springdata.jpa.dao"

entity-manager-factory-ref="entityManagerFactory"

transaction-manager-ref="transactionManager"/>

**4.测试代码。**

[复制代码](javascript:void(0);)

public interface UserDao extends Repository<AccountInfo, Long> {

public AccountInfo save(AccountInfo accountInfo);

// 你需要做的，仅仅是新增如下一行方法声明

public AccountInfo findByAccountId(Long accountId);

}

[复制代码](javascript:void(0);)

**5.总结**

使用 Spring Data JPA 进行持久层开发大致需要的三个步骤：

1.声明持久层的接口，该接口继承 Repository，Repository 是一个标记型接口，它不包含任何方法，当然如果有需要，Spring Data 也提供了若干 Repository 子接口，其中定义了一些常用的增删改查，以及分页相关的方法。

2.在接口中声明需要的业务方法。Spring Data 将根据给定的策略来为其生成实现代码。

3.在 Spring 配置文件中增加一行声明，让 Spring 为声明的接口创建代理对象。配置了 <jpa:repositories> 后，Spring 初始化容器时将会扫描 base-package 指定的包目录及其子目录，为继承 Repository 或其子接口的接口创建代理对象，并将代理对象注册为 Spring Bean，业务层便可以通过 Spring 自动封装的特性来直接使用该对象。

此外，<jpa:repository> 还提供了一些属性和子标签，便于做更细粒度的控制。可以在 <jpa:repository> 内部使用 <context:include-filter>、<context:exclude-filter> 来过滤掉一些不希望被扫描到的接口。

**接口继承**

持久层接口继承 Repository 并不是唯一选择。Repository 接口是 Spring Data 的一个核心接口，它不提供任何方法，开发者需要在自己定义的接口中声明需要的方法。与继承 Repository 等价的一种方式，就是在持久层接口上使用 @RepositoryDefinition 注解，并为其指定 domainClass 和 idClass 属性。如下两种方式是完全等价的：

**两种等价的继承接口方式示例：**

public interface UserDao extends Repository<AccountInfo, Long> { …… }

@RepositoryDefinition(domainClass = AccountInfo.class, idClass = Long.class)

public interface UserDao { …… }

1.如果持久层接口较多，且每一个接口都需要声明相似的增删改查方法，直接继承 Repository 就显得有些啰嗦，这时可以继承 CrudRepository，它会自动为域对象创建增删改查方法，供业务层直接使用。开发者只是多写了 "Crud" 四个字母，即刻便为域对象提供了开箱即用的十个增删改查方法。

2.使用 CrudRepository 也有副作用，它可能暴露了你不希望暴露给业务层的方法。比如某些接口你只希望提供增加的操作而不希望提供删除的方法。针对这种情况，开发者只能退回到 Repository 接口，然后到 CrudRepository 中把希望保留的方法声明复制到自定义的接口中即可.

3.分页查询和排序是持久层常用的功能，Spring Data 为此提供了 PagingAndSortingRepository 接口，它继承自 CrudRepository 接口，在 CrudRepository 基础上新增了两个与分页有关的方法。但是，我们很少会将自定义的持久层接口直接继承自 PagingAndSortingRepository，而是在继承 Repository 或 CrudRepository 的基础上，在自己声明的方法参数列表最后增加一个 Pageable 或 Sort 类型的参数，用于指定分页或排序信息即可，这比直接使用 PagingAndSortingRepository 提供了更大的灵活性。

4.JpaRepository 是继承自 PagingAndSortingRepository 的针对 JPA 技术提供的接口，它在父接口的基础上，提供了其他一些方法，比如 flush()，saveAndFlush()，deleteInBatch() 等。如果有这样的需求，则可以继承该接口。

**查询方式**

**1.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截取掉，比如 find、findBy、read、readBy、get、getBy，然后对剩下部分进行解析。并且如果方法的最后一个参数是 Sort 或者 Pageable 类型，也会提取相关的信息，以便按规则进行排序或者分页查询。

在创建查询时，我们通过在方法名中使用属性名称来表达，比如 findByUserAddressZip ()。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，详细规则如下（此处假设该方法针对的域对象为 AccountInfo 类型）：

* 先判断 userAddressZip （根据 POJO 规范，首字母变为小写，下同）是否为 AccountInfo 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；
* 从右往左截取第一个大写字母开头的字符串（此处为 Zip），然后检查剩下的字符串是否为 AccountInfo 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 user 为 AccountInfo 的一个属性；
* 接着处理剩下部分（ AddressZip ），先判断 user 所对应的类型是否有 addressZip 属性，如果有，则表示该方法最终是根据 "AccountInfo.user.addressZip" 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 "AccountInfo.user.address.zip" 的值进行查询。

在查询时，通常需要同时根据多个属性进行查询，且查询的条件也格式各样（大于某个值、在某个范围等等），Spring Data JPA 为此提供了一些表达条件查询的关键字，大致如下：

* 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<String> 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<String> 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；

**2.使用 @Query 创建查询**

@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 JP QL 查询语句即可，如下所示：

[复制代码](javascript:void(0);)

public interface UserDao extends Repository<AccountInfo, Long> {

@Query("select a from AccountInfo a where a.accountId = ?1")

public AccountInfo findByAccountId(Long accountId);

@Query("select a from AccountInfo a where a.balance > ?1")

public Page<AccountInfo> findByBalanceGreaterThan(

Integer balance,Pageable pageable);

}

[复制代码](javascript:void(0);)

很多开发者在创建 JP QL 时喜欢使用命名参数来代替位置编号，@Query 也对此提供了支持。JP QL 语句中通过": 变量"的格式来指定参数，同时在方法的参数前面使用 @Param 将方法参数与 JP QL 中的命名参数对应，示例如下：

[复制代码](javascript:void(0);)

public interface UserDao extends Repository<AccountInfo, Long> {

public AccountInfo save(AccountInfo accountInfo);

@Query("from AccountInfo a where a.accountId = :id")

public AccountInfo findByAccountId(@Param("id")Long accountId);

@Query("from AccountInfo a where a.balance > :balance")

public Page<AccountInfo> findByBalanceGreaterThan(

@Param("balance")Integer balance,Pageable pageable);

}

[复制代码](javascript:void(0);)

此外，开发者也可以通过使用 @Query 来执行一个更新操作，为此，我们需要在使用 @Query 的同时，用 @Modifying 来将该操作标识为修改查询，这样框架最终会生成一个更新的操作，而非查询。如下所示：

@Modifying

@Query("update AccountInfo a set a.salary = ?1 where a.salary < ?2")

public int increaseSalary(int after, int before);

**3.通过调用 JPA 命名查询语句创建查询**

命名查询是 JPA 提供的一种将查询语句从方法体中独立出来，以供多个方法共用的功能。Spring Data JPA 对命名查询也提供了很好的支持。用户只需要按照 JPA 规范在 orm.xml 文件或者在代码中使用 @NamedQuery（或 @NamedNativeQuery）定义好查询语句，唯一要做的就是为该语句命名时，需要满足”DomainClass.methodName()”的命名规则。假设定义了如下接口：

public interface UserDao extends Repository<AccountInfo, Long> {

......

public List<AccountInfo> findTop5();

}

如果希望为 findTop5() 创建命名查询，并与之关联，我们只需要在适当的位置定义命名查询语句，并将其命名为 "AccountInfo.findTop5"，框架在创建代理类的过程中，解析到该方法时，优先查找名为 "AccountInfo.findTop5" 的命名查询定义，如果没有找到，则尝试解析方法名，根据方法名字创建查询。

**Spring Data JPA 对事务的支持**

默认情况下，Spring Data JPA 实现的方法都是使用事务的。针对查询类型的方法，其等价于 @Transactional(readOnly=true)；增删改类型的方法，等价于 @Transactional。可以看出，除了将查询的方法设为只读事务外，其他事务属性均采用默认值。

如果用户觉得有必要，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，开发者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。具体 @Transactional 的使用可以参考Spring的参考文档。