<https://blog.csdn.net/zhongzh86/article/details/54313706>

**Spring Data JPA: 为所有Repository添加自定义方法**

2017年01月10日 14:38:53 [天蓝Skyline](https://me.csdn.net/zhongzh86) 阅读数 7929

Spring Data JPA: 为所有Repository添加自定义方法

Spring Data JPA中的Repository是接口，是JPA根据方法名帮我们自动生成的。但很多时候，我们需要为Repository提供一些自定义的实现。今天我们看看如何为Repository添加自定义的方法。

自定义Repository接口

首先我们来添加一个自定义的接口：

* 添加BaseRepository接口
* BaseRepository继承了PagingAndSortingRepository，这样可以保证所有Repository都有基本的增删改查以及分页等方法。
* 在BaseRepository上添加@NoRepositoryBean标注，这样Spring Data Jpa在启动时就不会去实例化BaseRepository这个接口
* 添加support(String modelType)方法，表示该Repository的领域对象是否为modelType类型

1. @NoRepositoryBean
2. public interface BaseRepository<T, ID extends Serializable>
3. extends PagingAndSortingRepository<T, ID> {
5. boolean support(String modelType);
7. }

然后，使所有Repository接口都继承BaseRepository

实现BaseRepository接口

定义好自定义的方法后，我们现在通过一个基本的Repository类来实现该方法：

首先添加BaseRepositoryImpl类，继承SimpleJpaRepository类，使其拥有Jpa Repository的基本方法。

我们发现Repository有两个构造函数：

* SimpleJpaRepository(JpaEntityInformation entityInformation, EntityManager entityManager)
* SimpleJpaRepository(Class domainClass, EntityManager em)

这里我们实现第二个构造函数，拿到domainClass和EntityManager两个对象。因为我们要实现的是知道某个Repository是否支持某个领域对象的类型，因此在实现构造函数时我们将domainClass的信息保留下来。

最后实现support方法，其参数是领域对象的类型，将其和domainClass对比，如果相等，则该Repository支持该类型的领域对象：

1. public class BaseRepositoryImpl<T, ID extends Serializable>
2. extends SimpleJpaRepository<T, ID>
3. implements BaseRepository<T, ID> {
5. private final Class<T> domainClass;
7. public BaseRepositoryImpl(Class<T> domainClass, EntityManager entityManager) {
8. super(domainClass, entityManager);
9. this.domainClass = domainClass;
10. }
12. @Override
13. public boolean support(String modelType) {
14. return domainClass.getName().equals(modelType);
15. }
17. }

创建自定义RepositoryFactoryBean

接下来我们来创建一个自定义的RepositoryFactoryBean来代替默认的RepositoryFactoryBean。RepositoryFactoryBean负责返回一个RepositoryFactory，Spring Data Jpa 将使用RepositoryFactory来创建Repository具体实现，这里我们用BaseRepositoryImpl代替SimpleJpaRepository作为Repository接口的实现。这样我们就能够达到为所有Repository添加自定义方法的目的。

1. public class BaseRepositoryFactoryBean<R extends JpaRepository<T, I>, T, I extends Serializable>
2. extends JpaRepositoryFactoryBean<R, T, I> {
4. @Override
5. protected RepositoryFactorySupport createRepositoryFactory(EntityManager em) {
6. return new MyRepositoryFactory(em);
7. }
9. private static class MyRepositoryFactory<T, I extends Serializable> extends JpaRepositoryFactory {
11. private final EntityManager em;
13. public MyRepositoryFactory(EntityManager em) {
14. super(em);
15. this.em = em;
16. }
18. @Override
19. protected Object getTargetRepository(RepositoryMetadata metadata) {
20. return new BaseRepositoryImpl<T, I>((Class<T>) metadata.getDomainType(), em);
21. }
23. @Override
24. protected Class<?> getRepositoryBaseClass(RepositoryMetadata metadata) {
25. return BaseRepositoryImpl.class;
26. }
27. }
28. }

配置Jpa factory class

最后，我们需要配置Jpa使用我们自定义的BaseRepositoryFactoryBean。Spring支持使用标注进行配置，我们在com.tmy.App中添加标注@EnableJpaRepositories(repositoryFactoryBeanClass = BaseRepositoryFactoryBean.class)：

1. @SpringBootApplication
2. @EnableJpaRepositories(repositoryFactoryBeanClass = BaseRepositoryFactoryBean.class)
3. public class App {
5. public static void main( String[] args ){
6. SpringApplication.run(App.class, args);
7. }
8. }

这样我们就为所有Repository添加了自定义的方法。

测试

我们添加了一个TestController进行测试。进入根目录，执行mvn spring-boot:run可以运行我们的应用。

应用启动后，分别访问<http://localhost:8080/test?type=blog&id=1>和<http://localhost:8080/test?type=article&id=1>，我们将看到article和blog两个不同的对象。

在TestController中，我们通过依赖式注入获取到所有Repository的列表。当用户访问/test，系统将根据传进来的type遍历所有Repository，找到对应的Repository，再调用findOne(id)方法找到对应的对象。这样我们就不需要一个一个的去获取Repository实例，当领域对象越来越多时，通过这种方式是一种更加高效的对象管理方法。

1. @RestController
2. public class TestController {
4. @Autowired
5. private List<BaseRepository> repositories;
7. @RequestMapping(value = "/test", method=RequestMethod.GET)
8. public Object getEntry(@RequestParam(value="type", required = true) String type,
9. @RequestParam(value="id", required=true) Integer id){
10. if(type.equals("article")){
11. type = Article.class.getName();
12. }else if (type.equals("blog")) {
13. type = Blog.class.getName();
14. }
15. for (BaseRepository baseRepository : repositories) {
16. if(baseRepository.support(type)){
17. return baseRepository.findOne(id);
18. }
19. }
20. return null;
21. }
23. }