第二章：JPA基础查询方法

1.Repository接口

简介

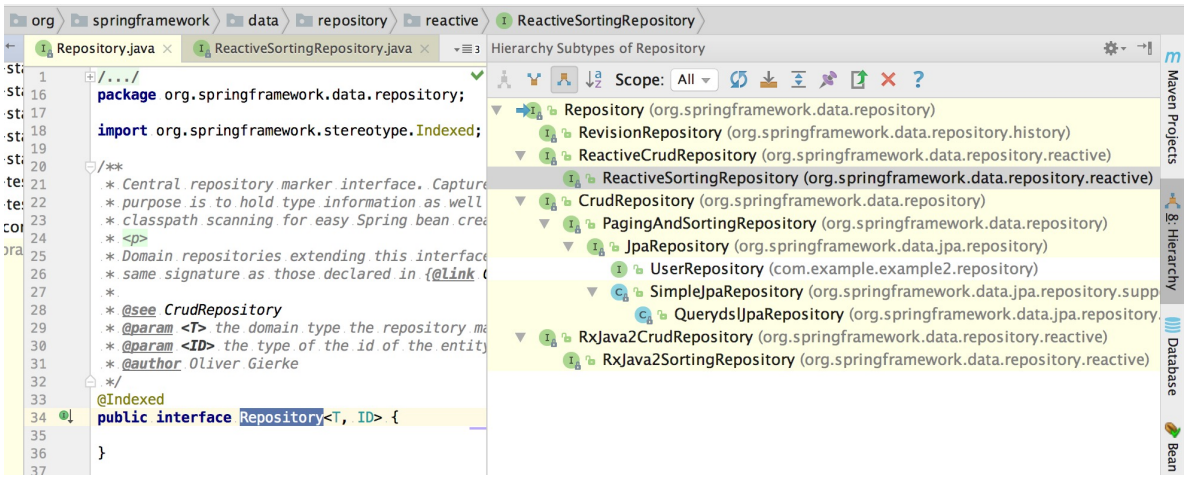
Repository 位于 Spring Data Common 的 lib 里面，是 Spring Data 里面做数据库操作的最底层的抽象接口，最顶级的父类，源码里面其实什么方法都没有，仅仅起到一个标识作用。管理域类以及域类的 ID 类型作为类型参数，此接口主要作为标记接口来捕获要使用的类型，并帮助用户发现扩展此接口的接口。Spring 底层做动态代理的时候发现只要是它的子类或者实现类，都代表储存库操作。

Repository 的源码如下：

```
package org.springframework.data.repository;
import org.springframework.stereotype.Indexed;
@Indexed
public interface Repository<T, ID> {
}
```

Repository接口的层次关系

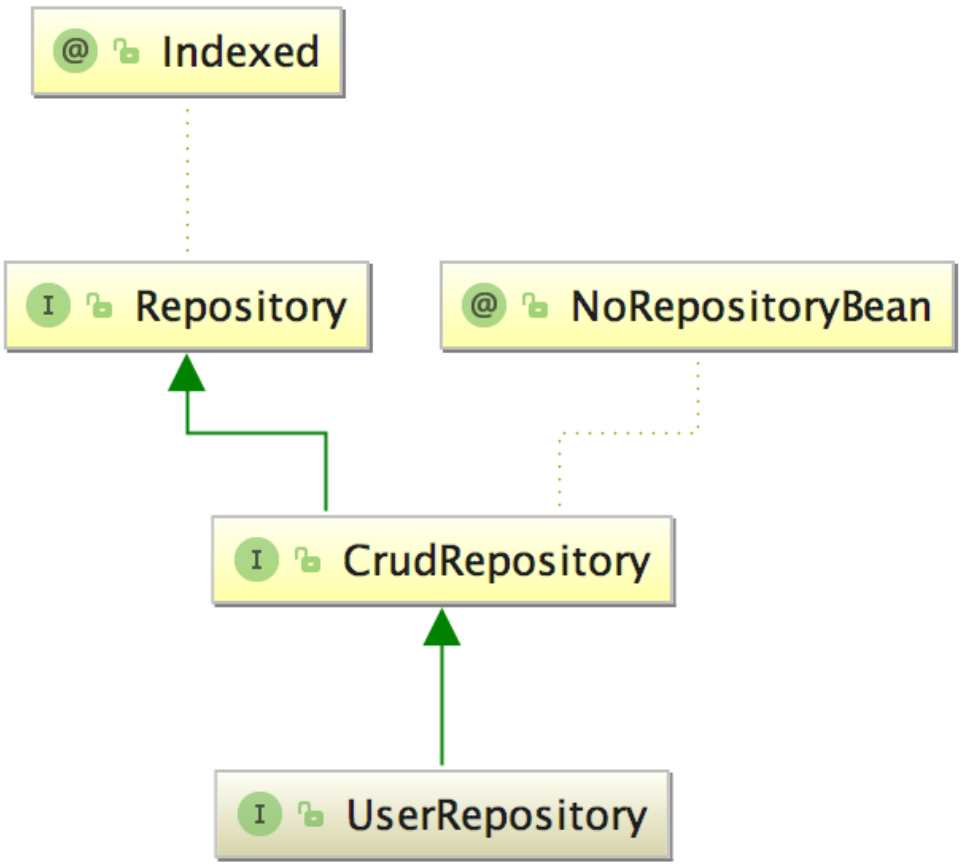
（1）我们用工具 Intellij Idea，打开类 Repository.class，然后单击 Navigate → Type Hierchy，会得到如下视图：



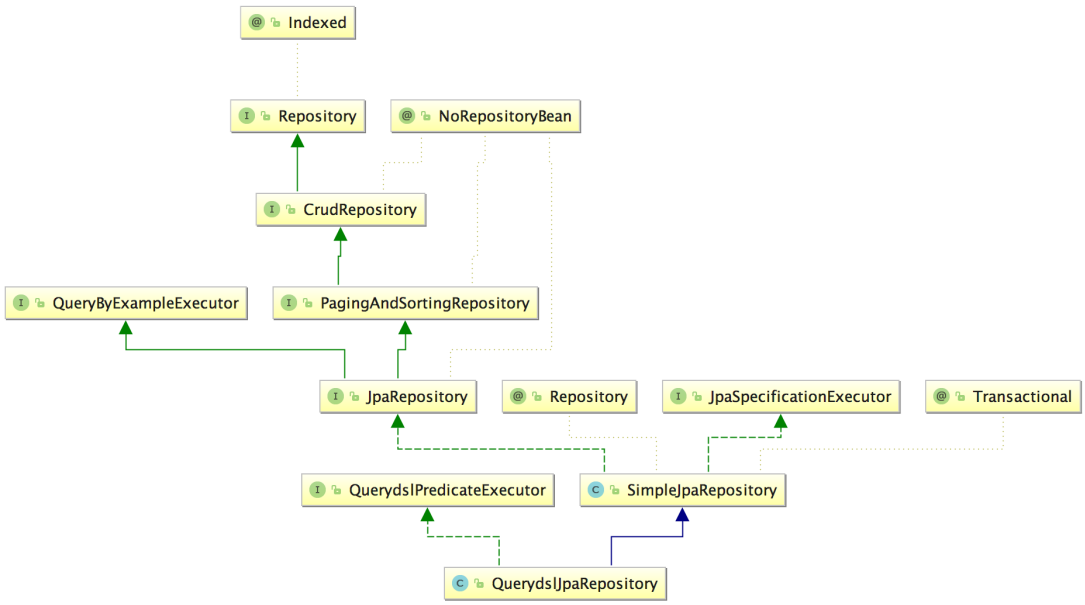
通过该层次结构视图，就会明白基类 Repository 的用意，需要对工程里面的所有 Repository 了如执掌，项目里面有哪些，Spring 的项目里面有哪些一目了然。我们通过上面的类的结构图，可以看得出来 Repository 可以分为三个部分：

- 即本篇要介绍的正常的 JpaRepository 这条线的操作。
- ReactiveRepository 这条线响应式编程，主要支持目前的 NoSQL 方面的操作，因为 NoSQL 大部分的操作都是分布式的，所以足可以看的出来 Spring Data 的野心，想提供关于所有 Data 方面的操，目前主要有 Cassandra、MongoDB 的实现，与 JPA 属于平级项目。
- RxJava2CrudRepository 这条线是为了支持 RxJava 2 做的标准的响应式编程的接口。

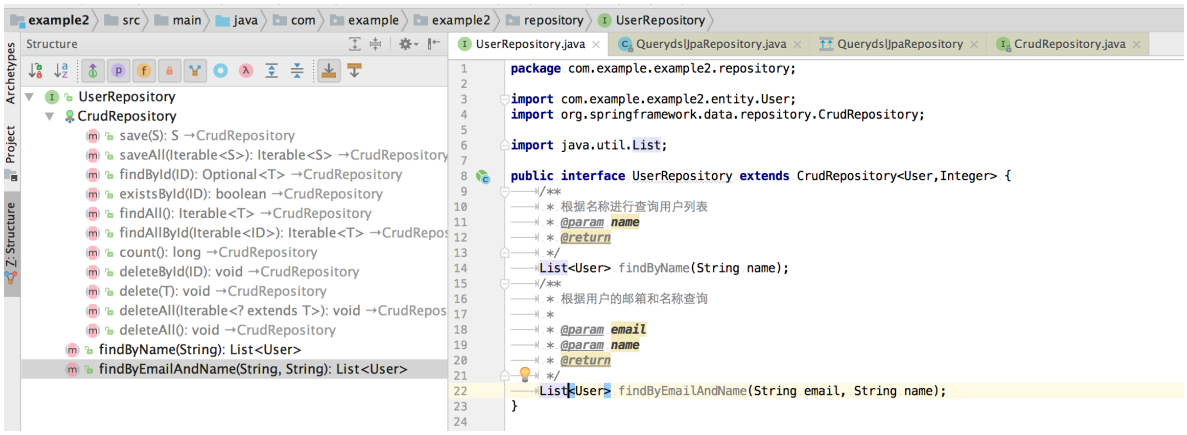
（2）通过 Intellij Idea，打开类上面 Example 1 里面的 UserRepository.java，单击鼠标右键 show diagrams 用图表的方式查看类的关系层次，打开如下图所示：



（3）通过 Intellij Idea，打开类 QueryDslJpaRepository，单击鼠标右键 show diagrams 用图表的方式查看类的关系层次，打开如下图所示：



（4）通过 Intellij Idea，打开类上面的 Example 1 里面的 UserRepository.java，单击 Navigate | File Structure 命令，可以查看此类的结构及有哪些方法，以此类推到其他类上，打开如下图所示：



2.CrudRepository接口

CrudRepository接口的方法

```
package org.springframework.data.repository;

import java.util.Optional;

@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    //保存或修改
    <S extends T> S save(S var1);//(1)
    //批量保存
    <S extends T> Iterable<S> saveAll(Iterable<S> var1);//(2)
    //根据主键查询实体
    Optional<T> findById(ID var1);//(3)
    //根据主键判断实体是否存在
    boolean existsById(ID var1);//(4)
    //查询实体的所有列表
    Iterable<T> findAll//(5)
    //根据主键列表查询实体列表
    Iterable<T> findAllById(Iterable<ID> var1);//(6)
    //查询总数
    long count();//(7)
    //根据主键删除
    void deleteById(ID var1);//(8)
    //根据实体对象删除
    void delete(T var1);//(9)
    //根据实体对象批量删除
    void deleteAll(Iterable<? extends T> var1);//(10)
    //删除所有数据
    void deleteAll();//(11)
}
```

方法分析：(对应上述源码编号)

提示：快速查看方法实现类快捷键ctrl + alt + B

1.保存实体方法

```
@Transactional
public <S extends T> S save(S entity) {
 if (this.entityInformation.isNew(entity)) {
     this.em.persist(entity);
     return entity;
 } else {
     return this.em.merge(entity);
 }
}
```

我们发现他是先出查一下传进去的实体是不是存在，然后判断是新增还是更新，是不是存在根据两种机制，一种是根据主键来判断，还有一种是根据 Version 来判断，后面介绍 Version 的时候详解，所以如果去看 JPA 的控制台打印出来的 SQL 最少会有两条，一条是查询，一条是 Insert 或者 Update。

2.批量保存，原理和上面的那一条相同，我们去看实现的话，就是 for 循环调用上面的 save 方法。

3.根据主键查询实体，返回 JDK 1.8 的 Optional，这可以避免 null exception。

4.根据主键判断实体是否存在。

5.查询实体的所有列表。

6.根据主键列表查询实体列表。

7.查询总数。

8.根据主键删除，查看源码会发现，其是先查询出来再进行删除。

```
@Transactional
public void deleteById(ID id) {
    Assert.notNull(id, "The given id must not be null!");
    this.delete(this.findById(id).orElseThrow(() -> {
        return new EmptyResultDataAccessException(String.format("No %s entity with id %s exists!",
this.entityInformation.getJavaType(), id), 1);
    }));
}
```

9.根据 entity 进行删除。

10.批量删除。

11.删除所有，原理：通过刚才的类关系查看其的实现类，SimpleJpaRepository 里面的 delete 实现方法如下，都是调用 delete 进行删除。

```
@Transactional
public void deleteById(ID id) {
    Assert.notNull(id, ID_MUST_NOT_BE_NULL);
    delete(findById(id).orElseThrow(() -> new EmptyResultDataAccessException(
        String.format("No %s entity with id %s exists!", entityInformation.getJavaType(), id), 1)));
}
@Transactional
public void delete(T entity) {
    Assert.notNull(entity, "The entity must not be null!");
    em.remove(em.contains(entity) ? entity : em.merge(entity));
}
@Transactional
public void deleteAll(Iterable<? extends T> entities) {
    Assert.notNull(entities, "The given Iterable of entities not be null!");
    for (T entity : entities) {
        delete(entity);
    }
}
```

CrudRepository接口的使用

具体环境与上一章案例环境一致，包括：数据库、实体类等

UserCrudRepository接口

```
package com.kazu.springdatajpa01.dao;

import com.kazu.springdatajpa01.entity.User;
import org.springframework.data.repository.CrudRepository;

/**
 * 用户接口
 */
public interface UserCrudRepository extends CrudRepository<User,Integer> {
}
```

测试类

新增测试类，命名为UserCrudRepositoryTest

```
package com.kazu.springdatajpa01;

import com.kazu.springdatajpa01.dao.UserCrudRepository;
import com.kazu.springdatajpa01.entity.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;
import java.util.Iterator;
import java.util.Optional;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserCrudRepositoryTest {

    @Resource
    private UserCrudRepository userCrudRepository;

    /**
     * 新增
     */
    @Test
    public void testInsert() {
        //创建用户
        User user = new User("王名",20,"广州市海珠区");
        userCrudRepository.save(user);
    }

    /**
     * 修改
     */
    @Test
    public void testUpdate() {
        //创建用户对象并指定主键
        User user = new User("王明",20,"广州市海珠区",4);
        userCrudRepository.save(user);
    }

    /**
     * 根据主键查询
     */
    @Test
    public void testFindById(){
        //根据主键查询
       Optional<User> user = userCrudRepository.findById(1);
       System.out.println(user);
    }

    /**
     * 判断实体是否存在
     */
    @Test
    public void testExistsById(){
        //判断实体是否存在
        boolean flag = userCrudRepository.existsById(4);
        System.out.println(flag);
    }

    /**
     * 查询所有数据
     */
    @Test
    public void testFindAll(){
        //调用查询所有的方法
       Iterable<User> iterable =  userCrudRepository.findAll();
       //获取迭代器
       Iterator<User> it = iterable.iterator();
```

```
        //循环遍历
        while (it.hasNext()){
            //获取每一个user对象
            User user = it.next();
            System.out.println(user);
        }
    }

    /**
     * 统计数量
     */
    @Test
    public void testCount(){
        long count = userCrudRepository.count();
         System.out.println(count);
    }

    /**
     * 根据主键删除
     */
    @Test
    public void testDeleteById(){
        userCrudRepository.deleteById(4);
    }
}
```

3.PagingAndSortingRepository接口

PagingAndSortingRepository接口的方法

```
package org.springframework.data.repository;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;

@NoRepositoryBean
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {
    //查询列表，支持排序
    Iterable<T> findAll(Sort var1);
    //分页查询
    Page<T> findAll(Pageable var1);
}
```

PagingAndSortingRepository接口的使用

定义UserPagingAndSortingRepository接口

```
package com.kazu.springdatajpa01.dao;

import com.kazu.springdatajpa01.entity.User;
import org.springframework.data.repository.PagingAndSortingRepository;

public interface UserPagingAndSortingRepository extends PagingAndSortingRepository<User,Integer> {
}
```

测试类

新增测试类，命名为UserPagingAndSortingRepositoryTest

```java
package com.kazu.springdatajpa01;

import com.kazu.springdatajpa01.dao.UserPagingAndSortingRepository;
import com.kazu.springdatajpa01.entity.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;
import java.util.Iterator;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserPagingAndSortingRepositoryTest {

    @Resource
    private UserPagingAndSortingRepository userPagingAndSortingRepository;

    /**
     * 排序查询
     */
    @Test
    public void testFindAllBySort(){
        //创建排序对象（参数1：升序或降序，参数2：排序属性名）
        Sort sort = new Sort(Sort.Direction.DESC,"id");
        //调用查询方法
        Iterable<User> userIterable = userPagingAndSortingRepository.findAll(sort);
        //获取查询迭代器
        Iterator<User> userIterator =  userIterable.iterator();
        //循环遍历
        while(userIterator.hasNext()){
            //获取每一个用户对象
            User user = userIterator.next();
            System.out.println(user);
        }
    }

    /**
     * 排序分页查询
     */
    @Test
    public void testFindAllBySortAndPage(){
        int pageIndex = 1;//当前页码
        int pageSize =2;//每页显示数量
        //调用查询方法并指定分页参数
        Iterable<User> userIterable = userPagingAndSortingRepository.findAll(PageRequest.of((pageIndex-
1)*pageSize,pageSize,Sort.Direction.ASC,"id"));
        //获取查询迭代器
        Iterator<User> userIterator =  userIterable.iterator();
        //循环遍历
        while(userIterator.hasNext()){
            //获取每一个用户对象
            User user = userIterator.next();
            System.out.println(user);
        }
    }
}
```

Sort和PageRequest对象具体方法参考源码

4.JpaRepository接口

JpaRepository接口的方法

```
package org.springframework.data.jpa.repository;

import java.util.List;
import org.springframework.data.domain.Example;
import org.springframework.data.domain.Sort;
import org.springframework.data.repository.NoRepositoryBean;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.repository.query.QueryByExampleExecutor;

@NoRepositoryBean
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>,
QueryByExampleExecutor<T> {
    List<T> findAll();
    List<T> findAll(Sort var1);
    List<T> findAllById(Iterable<ID> var1);
    <S extends T> List<S> saveAll(Iterable<S> var1);
    void flush();
    <S extends T> S saveAndFlush(S var1);
    void deleteInBatch(Iterable<T> var1);
    void deleteAllInBatch();
    T getOne(ID var1);
    <S extends T> List<S> findAll(Example<S> var1);
    <S extends T> List<S> findAll(Example<S> var1, Sort var2);
}
```

> 重点关注查询方法，通过源码和CrudRepository接口相比较，JpaRepository接口将默认实现的查询结果换成了List。

JpaRepository接口的使用

定义UserJpaRepository接口

```
package com.kazu.springdatajpa01.dao;

import com.kazu.springdatajpa01.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserJpaRepository extends JpaRepository<User,Integer> {
}
```

测试类

> 新增UserJpaRepositoryTest测试类

```
package com.kazu.springdatajpa01;

import com.kazu.springdatajpa01.dao.UserJpaRepository;
import com.kazu.springdatajpa01.entity.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;
import java.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class UserJpaRepositoryTest {

    @Resource
    private UserJpaRepository userJpaRepository;

    @Test
    public void testFindAll(){
```

```
        //调用查询用户的方法
        List<User> userList = userJpaRepository.findAll();
        for (User user : userList) {
            System.out.println(user);
        }
    }
}
```

5.JpaSpecificationExecutor接口

JpaSpecificationExecutor接口的方法

```
package org.springframework.data.jpa.repository;

import java.util.List;
import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.lang.Nullable;

public interface JpaSpecificationExecutor<T> {
    Optional<T> findOne(@Nullable Specification<T> var1);

    List<T> findAll(@Nullable Specification<T> var1);
    //分页查询（支持动态查询+分页查询+排序）
    Page<T> findAll(@Nullable Specification<T> var1, Pageable var2);

    List<T> findAll(@Nullable Specification<T> var1, Sort var2);

    long count(@Nullable Specification<T> var1);
}
```

> 重点关注Page findAll(@Nullable Specification var1, Pageable var2);此方法

JpaSpecificationExecutor接口的使用

定义接口

```
package com.kazu.springdatajpa01.dao;

import com.kazu.springdatajpa01.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;

/**
 * 分页查询（支持动态查询+分页查询+排序）
 */
public interface UserJpaSpecificationExecutor extends
JpaRepository<User,Integer>,JpaSpecificationExecutor<User> {
}
```

测试类

```
package com.kazu.springdatajpa01;
```

```
import com.kazu.springdatajpa01.dao.UserJpaSpecificationExecutor;
import com.kazu.springdatajpa01.entity.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;
import java.util.List;


@RunWith(SpringRunner.class)
@SpringBootTest
public class UserJpaSpecificationExecutorTest {


    @Resource
    private UserJpaSpecificationExecutor userJpaSpecificationExecutor;


    @Test
    public void test(){
        User user = new User();
        user.setUserName("李");
        user.setAge(20);
        user.setAddress("广州");
        //排序等定义
        Sort sort = new Sort(Sort.Direction.ASC,"id");
        //分页的定义
        PageRequest pageable = PageRequest.of(0,2,sort);
        Page<User> page = userJpaSpecificationExecutor.findAll(new Specification<User>() {
            @Override
            public Predicate toPredicate(Root<User> root, CriteriaQuery<?> criteriaQuery,
CriteriaBuilder cb) {
                //获取条件参数对象
                Predicate predicate = cb.conjunction();
                //user对象不为空
                if(user!=null){
                    //用户名
                    if(user.getUserName()!=null && !user.getUserName().equals("")){

 predicate.getExpressions().add(cb.like(root.get("userName"),"%"+user.getUserName()+"%"));
                    }
                    //年龄
                    if(user.getAge()!=null){
                        predicate.getExpressions().add(cb.ge(root.get("age"),user.getAge()));
                    }
                    //地址
                    if(user.getAddress()!=null && !user.getAddress().equals("")){

 predicate.getExpressions().add(cb.like(root.get("address"),"%"+user.getAddress()+"%"));
                    }
                }
                return predicate;
            }
        },pageable);
        //获取用户列表
        List<User> users = page.getContent();
        for (User user1 : users) {
            System.out.println(user1);
        }
        System.out.println("总页数："+page.getTotalPages());
        System.out.println("总记录数："+page.getTotalElements());
        System.out.println("当前页码："+page.getNumber());
        System.out.println("每页显示数量："+page.getSize());
    }
```

```
}
```