# 第七章：springboot整合SpringDataJPA

## SpringDataJPA介绍

## SpringBoot整合SpringDataJPA

### 编写pom.xml文件

|  |
| --- |
| <!-- druid连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.9</version>  </dependency>  <!-- mysql -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <!-- SpringDataJPA依赖 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency> |

### 编写application.properties文件

|  |
| --- |
| spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver  spring.datasource.url=jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  spring.datasource.username=root  spring.datasource.password=root  spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  #创建数据库的方式类型  spring.jpa.hibernate.ddl-auto=update  #显示sql语句  spring.jpa.show-sql=true |

### 编写实体类

|  |
| --- |
| @Entity//标识是一个实体类  @Table(name="t\_user")//表名  **public** **class** User {  @Id//主键  @GeneratedValue(strategy=GenerationType.***IDENTITY***)//主键自增类型  //当属性名与列名一致时，可以省略@Column  @Column  **private** Integer id;  **private** String userName;  **private** Integer age;  **private** String address;  **public** Integer getId() {  **return** id;  }  **public** **void** setId(Integer id) {  **this**.id = id;  }  **public** String getUserName() {  **return** userName;  }  **public** **void** setUserName(String userName) {  **this**.userName = userName;  }  **public** Integer getAge() {  **return** age;  }  **public** **void** setAge(Integer age) {  **this**.age = age;  }  **public** String getAddress() {  **return** address;  }  **public** **void** setAddress(String address) {  **this**.address = address;  }  @Override  **public** String toString() {  **return** "User [id=" + id + ", userName=" + userName + ", age=" + age + ", address=" + address + "]";  }  } |

### 编写dao接口

|  |
| --- |
| /\*\*  \* 参数一 :当前需要映射的实体  \* 参数二 :当前映射的实体中的主键列的类型  \*/  **public** **interface** UserDao **extends** JpaRepository<User, Integer>{  } |

### 编写测试类

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** Springboot07JpaApplicationTests {  @Resource  **private** UserDao userDao;  @Test  **public** **void** testAddUser() {  User users = **new** User();  users.setAddress("广州市");  users.setAge(20);  users.setUserName("张三");  **this**.userDao.save(users);  }  } |

## SpringDataJPA核心接口

* Repository接口
* CrudRepository接口
* PagingAndSortingRepository接口
* JpaRepository接口
* JpaSpecificationExecutor接口

## Repository接口的使用

### 概述

提供方法名称命名查询方式

提供基于@Query注解查询与更新

### 方法名称命名查询

#### 编写dao接口

|  |
| --- |
| /\*\*  \* Repository接口的方法名称命名查询  \*  \*/  **public** **interface** UserRepository **extends** Repository<User,Integer>{  //注意：方法的名称必须要遵循驼峰式命名规则。findBy(关键字)+属性名称(首字母要大写)+查询条件(首字母大写)  List<User> findByUserName(String name);    List<User> findByUserNameAndAge(String name,**int** age);    List<User> findByUserNameLike(String name);    } |

#### 编写测试类

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** Springboot07JpaApplicationTests {    @Resource  **private** UserDao userDao;    @Resource  **private** UserRepository userRepository;  @Test  **public** **void** testAddUser() {  User users = **new** User();  users.setAddress("广州市");  users.setAge(20);  users.setUserName("张三");  **this**.userDao.save(users);  }    @Test  **public** **void** testFindByUserName() {  //调用查询的方法  List<User> userList = **this**.userRepository.findByUserName("张三");  **for** (User user : userList) {  System.***out***.println(user);  }  }    @Test  **public** **void** testFindByUserNameAndAge() {  //调用查询的方法  List<User> userList = **this**.userRepository.findByUserNameAndAge("张三",20);  **for** (User user : userList) {  System.***out***.println(user);  }  }    @Test  **public** **void** testFindByUserNameLike() {  //调用查询的方法  List<User> userList = **this**.userRepository.findByUserNameLike("张%");  **for** (User user : userList) {  System.***out***.println(user);  }  }  } |

### 基于@Query注解查询与更新

#### 编写dao

|  |
| --- |
| **public** **interface** UserAnnotationQueryRepository **extends** Repository<User, Integer>{  /\*\*  \* 使用HQL语句完成查询  \*/  @Query("from User where userName = ?1")  List<User> findUserByNameUseHQL(String name);    /\*\*  \* 使用SQL语句完成查询  \*/  @Query(value="select \* from t\_user where user\_name = ?1",nativeQuery=**true**)  List<User> findUserByNameUseSQL(String name);    /\*\*  \* 使用SQL语句完成查询  \*/  @Query(value="select \* from t\_user where user\_name like %:name%",nativeQuery=**true**)  List<User> findUserByNameUseSQLWithParam(String name);  //命名参数必须与参数名一致，若想不一致，则需加@Param()注解  @Query(value="select \* from t\_user where user\_name like %:userName% and age>:age",nativeQuery=**true**)  List<User> findUserByNameUseSQLWithParam2(@Param("userName") String name,**int** age);    @Modifying//标记为更新操作  @Query("update User set userName =?1 where id=?2")  **void** updateUser(String username,**int** id);  } |

#### 编写测试类

|  |
| --- |
| @Resource  **private** UserAnnotationQueryRepository userAnnotationQueryRepository;  @Test  **public** **void** testUseHQL() {  List<User> userList = userAnnotationQueryRepository.findUserByNameUseHQL("张三");  **for** (User user : userList) {  System.***out***.println(user);  }  }  @Test  **public** **void** testUseSQL() {  List<User> userList = userAnnotationQueryRepository.findUserByNameUseSQL("张三");  **for** (User user : userList) {  System.***out***.println(user);  }  }  @Test  **public** **void** testUseSQLParam() {  List<User> userList = userAnnotationQueryRepository.findUserByNameUseSQLWithParam("张");  **for** (User user : userList) {  System.***out***.println(user);  }  }  @Test  **public** **void** testUseSQLParam2() {  List<User> userList = userAnnotationQueryRepository.findUserByNameUseSQLWithParam2("张",22);  **for** (User user : userList) {  System.***out***.println(user);  }  }  @Test  @Transactional//@Transactional与@Test 一起使用时 事务是自动回滚的。  @Rollback(**false**)//取消自动回滚  **public** **void** testUpdate() {  **this**.userAnnotationQueryRepository.updateUser("张三丰", 1);  } |

## CrudRepository接口

### 概述

支持简单的增删改查

### 编写接口

|  |
| --- |
| **public** **interface** UserCrudRepository **extends** CrudRepository<User, Integer>{  } |

### 编写测试类

|  |
| --- |
| @Resource  **private** UserCrudRepository userCrudRepository;  //新增  @Test  **public** **void** testAdd() {  User users = **new** User();  users.setAddress("广州市");  users.setAge(22);  users.setUserName("张全蛋");  **this**.userCrudRepository.save(users);  }  //修改  @Test  **public** **void** testSaveUpdate() {  User users = **new** User();  users.setAddress("广州市");  users.setAge(22);  users.setUserName("李四");  users.setId(4);  **this**.userCrudRepository.save(users);  }  //查询对象  @Test  **public** **void** testFindOne() {  Optional<User> user = **this**.userCrudRepository.findById(3);  System.***out***.println(user);  }  //查询所有  @Test  **public** **void** testFindAll() {  List<User> userList = (List<User>) **this**.userCrudRepository.findAll();  **for** (User user : userList) {  System.***out***.println(user);  }  }  //删除  @Test  **public** **void** testDelete() {  **this**.userCrudRepository.deleteById(5);  } |

## PagingAndSortingRepository接口

### 概述

此接口支持分页查询及排序。

### 编写接口

|  |
| --- |
| **public** **interface** UserRepositoryPagingAndSorting **extends** PagingAndSortingRepository<User, Integer>{  } |

### 编写测试类

|  |
| --- |
| @Resource  **private** UserRepositoryPagingAndSorting userRepositoryPagingAndSorting;  //排序测试  @Test  **public** **void** testSort() {  //设置排序字段  Order order = **new** Order(Direction.***DESC***,"id");  Sort sort = **new** ~~Sort~~(order);  //查询  List<User> userList = (List<User>) **this**.userRepositoryPagingAndSorting.findAll(sort );  **for** (User user : userList) {  System.***out***.println(user);  }  }  //排序测试  @Test  **public** **void** testPage() {  //Pageable:封装了分页的参数，当前页，每页显示的条数。注意：当前页是从0开始。  //PageRequest(page,size) page:当前页。size:每页显示的条数  Pageable pageable = **new** ~~PageRequest~~(0, 2);  Page<User> page = **this**.userRepositoryPagingAndSorting.findAll(pageable );  System.***out***.println("总条数："+page.getTotalElements());  System.***out***.println("总页数："+page.getTotalPages());  List<User> userList = page.getContent();  **for** (User user : userList) {  System.***out***.println(user);  }  }    //排序分页测试  @Test  **public** **void** testPageSort() {  Sort sort = **new** ~~Sort~~(**new** Order(Direction.***DESC***,"id"));  Pageable pageable = **new** ~~PageRequest~~(0, 2,sort);  Page<User> page = **this**.userRepositoryPagingAndSorting.findAll(pageable );  System.***out***.println("总条数："+page.getTotalElements());  System.***out***.println("总页数："+page.getTotalPages());  List<User> userList = page.getContent();  **for** (User user : userList) {  System.***out***.println(user);  }  } |

## JpaRepository接口

### 概述

JpaRepository接口与CrudRepository接口用法类似

### 编写接口

|  |
| --- |
| /\*\*  \* 参数一 :当前需要映射的实体  \* 参数二 :当前映射的实体中的主键列的类型  \*  \*/  **public** **interface** UserDao **extends** JpaRepository<User, Integer>{  } |

### 编写测试类

|  |
| --- |
| @Test  **public** **void** testJapRepository() {  List<User> userList = **this**.userDao.findAll();  **for** (User user : userList) {  System.***out***.println(user);  }  } |

## JpaSpecificationExecutor接口

### 概述

### 编写接口

|  |
| --- |
| **public** **interface** UserRepository2 **extends** JpaRepository<User, Integer>,JpaSpecificationExecutor<User> {  } |

### 编写测试类

|  |
| --- |
| @Test  **public** **void** testSearch() {  //封装查询对象  User user = **new** User();  user.setUserName("张");  user.setAge(30);  //封装分页信息  Pageable pageable = **new** ~~PageRequest~~(0, 2,Sort.Direction.***DESC***,"id");  Page<User> page = userRepository2.findAll(**new** Specification<User>() {    @Override  **public** Predicate toPredicate(Root<User> root, CriteriaQuery<?> query, CriteriaBuilder cb) {  // **TODO** Auto-generated method stub  //条件参数对象  Predicate predicate = cb.conjunction();  //用户名不为空  **if**(user.getUserName()!=**null** && !user.getUserName().equals("")) {  //用户名模糊查询  predicate.getExpressions().add(cb.like(root.get("userName"), "%"+user.getUserName()+"%"));  }  //年龄不为空  **if**(user.getAge()!=**null** ) {  //大于当前年龄  predicate.getExpressions().add(cb.ge(root.get("age"), user.getAge()));  }  **return** predicate;  }  },pageable);  //获取用户列表  List<User> userList = page.getContent();  **for** (User user2 : userList) {  System.***out***.println(user2);  }  } |