SpringDataJPA

Day01（JPA基础）

# 课程提要

1. 什么是jpa
2. 什么是SpringDataJpa
3. 入门程序
4. Jap对象介绍
5. 使用jpa实现CRUD操作

# 什么是JPA

## 什么是ORM

ORM（Object-Relational Mapping） 表示对象关系映射。在面向对象的软件开发中，通过ORM，就可以把对象映射到关系型数据库中。只要有一套程序能够做到建立对象与数据库的关联，操作对象就可以直接操作数据库数据，就可以说这套程序实现了ORM对象关系映射。简单的说：ORM就是建立实体类和数据库表之间的关系，从而达到操作实体类就相当于操作数据库表的目的。ORM是一种解决问题的思想。

## JPA概念

JPA是Java Persistence API的简称，中文名Java持久层API，是JDK 5.0注解或XML描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中。

JPA由EJB 3.0软件专家组开发，作为JSR-220实现的一部分。但它又不限于EJB 3.0，你可以在Web应用、甚至桌面应用中使用。JPA的宗旨是为POJO提供持久化标准规范。总的来说，JPA包括以下3方面的技术：

* ORM映射元数据，JPA支持XML和JDK 5.0注解两种元数据的形式，元数据描述对象和表之间的映射关系，框架据此将实体对象持久化到数据库表中；
* JPA 的API，用来操作实体对象，执行CRUD操作，框架在后台替我们完成所有的事情，开发者从繁琐的JDBC和SQL代码中解脱出来。
* 查询语言，这是持久化操作中很重要的一个方面，通过面向对象而非面向数据库的查询语言查询数据，避免程序的SQL语句紧密耦合。

## jpa的优势

* **标准化**

JPA 是 JCP 组织发布的 Java EE 标准之一，因此任何声称符合 JPA 标准的框架都遵循同样的架构，提供相同的访问API，这保证了基于JPA开发的企业应用能够经过少量的修改就能够在不同的JPA框架下运行。

* **容器级特性的支持**

JPA框架中支持大数据集、事务、并发等容器级事务，这使得 JPA 超越了简单持久化框架的局限，在企业应用发挥更大的作用。

* **简单方便**

JPA的主要目标之一就是提供更加简单的编程模型：在JPA框架下创建实体和创建Java 类一样简单，没有任何的约束和限制，只需要使用 javax.persistence.Entity进行注释，JPA的框架和接口也都非常简单，没有太多特别的规则和设计模式的要求，开发者可以很容易的掌握。JPA基于非侵入式原则设计，因此可以很容易的和其它框架或者容器集成

* **查询能力**

JPA的查询语言是面向对象而非面向数据库的，它以面向对象的自然语法构造查询语句，可以看成是Hibernate HQL的等价物。JPA定义了独特的JPQL（Java Persistence Query Language），JPQL是EJB QL的一种扩展，它是针对实体的一种查询语言，操作对象是实体，而不是关系数据库的表，而且能够支持批量更新和修改、JOIN、GROUP BY、HAVING 等通常只有 SQL 才能够提供的高级查询特性，甚至还能够支持子查询。

* **高级特性**

JPA 中能够支持面向对象的高级特性，如类之间的继承、多态和类之间的复杂关系，这样的支持能够让开发者最大限度的使用面向对象的模型设计企业应用，而不需要自行处理这些特性在关系数据库的持久化。

## Jpa的供应商

JPA 的目标之一是制定一个可以由很多供应商实现的API，并且开发人员可以编码来实现该API，而不是使用私有供应商特有的API。因此开发人员只需使用供应商特有的API来获得JPA规范没有解决但应用程序中需要的功能。尽可能地使用JPA API，但是当需要供应商公开但是规范中没有提供的功能时，则使用供应商特有的API。

* **Hibernate**

JPA是需要Provider来实现其功能的，Hibernate就是JPA Provider中很强的一个，应该说无人能出其右。从功能上来说，JPA就是Hibernate功能的一个子集。Hibernate 从3.2开始，就开始兼容JPA。Hibernate3.2获得了Sun TCK的JPA(Java Persistence API) 兼容认证。

只要熟悉Hibernate或者其他ORM框架，在使用JPA时会发现其实非常容易上手。

* **OpenJPA**

OpenJPA 是 Apache 组织提供的开源项目，它实现了 EJB 3.0 中的 JPA 标准，为开发者提供功能强大、使用简单的持久化数据管理框架。OpenJPA 封装了和关系型数据库交互的操作，让开发者把注意力集中在编写业务逻辑上。OpenJPA 可以作为独立的持久层框架发挥作用，也可以轻松的与其它 Java EE 应用框架或者符合 EJB 3.0 标准的容器集成。

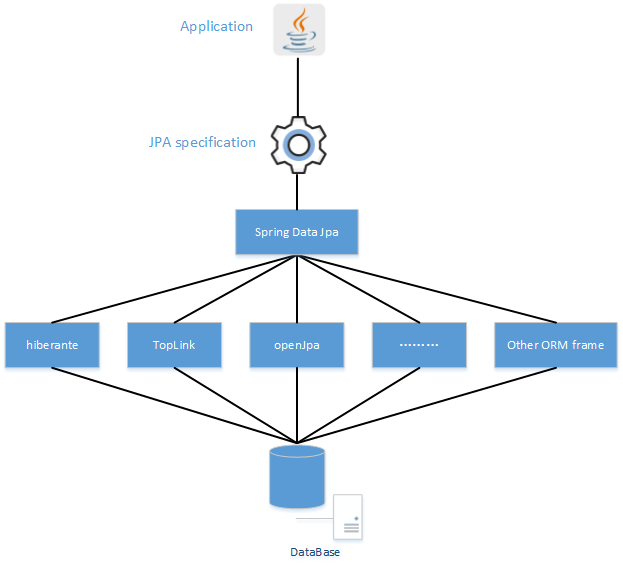
* **TopLink**

TopLink，是位居第一的Java对象关系可持续性体系结构，原署WebGain公司的产品，后被Oracle收购，并重新包装为Oracle AS TopLink。TOPLink为在关系数据库表中存储 Java 对象和企业 Java 组件 (EJB) 提供高度灵活和高效的机制。TopLink 为开发人员提供极佳的性能和选择，可以与任何数据库、任何应用服务器、任何开发工具集和过程以及任何 J2EE 体系结构协同工作。

# 什么是SpringDataJpa

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！

Spring data jpa是在JPA规范下提供了Repository层的实现，但是使用哪一种ORM需要你来决定（默认使用Hibernate JPA的实现）。虽然ORM框架都实现了JPA规范，但是在不同的ORM框架之间切换仍然需要编写不同的代码，而通过使用Spring data jpa能够方便大家在不同的ORM框架之间进行切换而不要更改代码。并且spring data jpa 对Repository层封装的很好，也省去了不少的麻烦。



spring data jpa、jpa以及ORM框架之间的关系

# JPA入门程序

## 开发环境

开发环境使用jdk1.9（1.8也可以）

开发工具使用IntelliJ IDEA

数据库：mysql（Sql脚本见参考资料）

Maven工程pom文件：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.itheima</groupId>  <artifactId>spring-data-first</artifactId>  <version>1.0-SNAPSHOT</version>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.hibernate.version>5.0.7.Final</project.hibernate.version>  <maven.compiler.source>1.9</maven.compiler.source>  <maven.compiler.target>1.9</maven.compiler.target>  </properties>   <dependencies>  <!-- junit -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <!--<scope>test</scope>-->  </dependency>   <!-- hibernate对jpa的支持包 -->  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-entitymanager</artifactId>  <version>${project.hibernate.version}</version>  </dependency>   <!-- c3p0 -->  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-c3p0</artifactId>  <version>${project.hibernate.version}</version>  </dependency>   <!-- log日志 -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>   <!-- Mysql and MariaDB -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.6</version>  </dependency>  <!--如果是jdk1.8不需要引入此jar包-->  <dependency>  <groupId>javax.xml.bind</groupId>  <artifactId>jaxb-api</artifactId>  <version>2.3.0</version>  </dependency>  </dependencies>   </project> |

## 入门程序

### 需求

向cst\_customer表插入一条数据。

### 配置JPA的核心配置文件

在工程的classpath下创建一个名为META-INF的文件夹，由于我们是maven工程所以可以把此文件放到src/main/resources目录下。在此文件夹下创建一个名为persistence.xml的配置文件。注意文件名必须是persistence.xml不可改动。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <persistence xmlns="http://java.sun.com/xml/ns/persistence" version="2.0">  <!--持久化单元配置  name:持久化单元名称  transaction-type:事务模式。RESOURCE\_LOCAL：本地事务 JTA：分布式事务  -->  <persistence-unit name="myJpa" transaction-type="RESOURCE\_LOCAL">  <!--配置JPA提供商-->  <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>  <properties>  <!--连接数据库配置-->  <property name="javax.persistence.jdbc.user" value="root"/>  <property name="javax.persistence.jdbc.password" value="root"/>  <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>  <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/spring-data"/>   <!--jpa实现方的配置-->  <!--是否打印sql语句-->  <property name="hibernate.show\_sql" value="true"/>  <!--是否格式化sql语句-->  <property name="hibernate.format\_sql" value="true"/>  <!--是否自动创建数据库表  可选值：create、update、none  create：程序自动创建数据库表，如果表存在则先删除后创建  update：程序自动创建数据库表，如果表存在则不创建。  none:不会创建表  -->  <property name="hibernate.hbm2ddl.auto" value="create"/>   </properties>  </persistence-unit> </persistence> |

### 创建表对象的实体类

|  |
| --- |
| package com.itheima.pojo;  import javax.persistence.\*;  @Entity @Table(name = "cst\_customer") public class Customer {  @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  @Column(name="cust\_id")  private long custId;   @Column(name="cust\_name")  private String custName;   @Column(name="cust\_source")  private String custSource;   @Column(name="cust\_industry")  private String custIndustry;   @Column(name="cust\_level")  private String custLevel;   @Column(name="cust\_address")  private String custAddress;   @Column(name="cust\_phone")  private String custPhone;   public long getCustId() {  return custId;  }   public void setCustId(long custId) {  this.custId = custId;  }   public String getCustName() {  return custName;  }   public void setCustName(String custName) {  this.custName = custName;  }   public String getCustSource() {  return custSource;  }   public void setCustSource(String custSource) {  this.custSource = custSource;  }   public String getCustIndustry() {  return custIndustry;  }   public void setCustIndustry(String custIndustry) {  this.custIndustry = custIndustry;  }   public String getCustLevel() {  return custLevel;  }   public void setCustLevel(String custLevel) {  this.custLevel = custLevel;  }   public String getCustAddress() {  return custAddress;  }   public void setCustAddress(String custAddress) {  this.custAddress = custAddress;  }   public String getCustPhone() {  return custPhone;  }   public void setCustPhone(String custPhone) {  this.custPhone = custPhone;  } } |

### 测试代码

|  |
| --- |
| public class CustomerTest {   @Test  public void testAddCustomer() throws Exception {  EntityManagerFactory factory = Persistence.*createEntityManagerFactory*("myJpa");  EntityManager entityManager = factory.createEntityManager();  EntityTransaction transaction = entityManager.getTransaction();  transaction.begin();  Customer customer = new Customer();  customer.setCustName("张三");  customer.setCustAddress("北京");  customer.setCustIndustry("旅游");  customer.setCustLevel("vip");  customer.setCustPhone("13600132258");  customer.setCustSource("网络");  entityManager.persist(customer);  transaction.commit();  entityManager.close();  factory.close();  } } |

## 细节说明

### 常用注解的说明

|  |
| --- |
| @Entity  作用：指定当前类是实体类。  @Table  作用：指定实体类和表之间的对应关系。  属性：  name：指定数据库表的名称  @Id  作用：指定当前字段是主键。  @GeneratedValue  作用：指定主键的生成方式。。  属性：  strategy ：指定主键生成策略。  @Column  作用：指定实体类属性和数据库表之间的对应关系  属性：  name：指定数据库表的列名称。  unique：是否唯一  nullable：是否可以为空  inserttable：是否可以插入  updateable：是否可以更新  columnDefinition: 定义建表时创建此列的DDL  secondaryTable: 从表名。如果此列不建在主表上（默认建在主表），该属性定义该列所在从表的名字搭建开发环境[重点] |

### JPA中的主键生成策略

通过annotation（注解）来映射hibernate实体的,基于annotation的hibernate主键标识为@Id, 其生成规则由@GeneratedValue设定的.这里的@id和@GeneratedValue都是JPA的标准用法。

JPA提供的四种标准用法为TABLE,SEQUENCE,IDENTITY,AUTO。

具体说明如下：

**IDENTITY:主键由数据库自动生成（主要是自动增长型）**

用法：

|  |
| --- |
| @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  **private** Long custId; |

**SEQUENCE：根据底层数据库的序列来生成主键，条件是数据库支持序列。**

用法：

|  |
| --- |
| @Id  @GeneratedValue(strategy = GenerationType.***SEQUENCE***,generator="payablemoney\_seq")  @SequenceGenerator(name="payablemoney\_seq", sequenceName="seq\_payment")  **private** Long custId;  -----------------------------------------------------------  //@SequenceGenerator源码中的定义  @Target({TYPE, METHOD, FIELD})  @Retention(RUNTIME)  **public** **@interface** SequenceGenerator {  //表示该表主键生成策略的名称，它被引用在@GeneratedValue中设置的“generator”值中  String name();  //属性表示生成策略用到的数据库序列名称。  String sequenceName() **default** "";  //表示主键初识值，默认为0  **int** initialValue() **default** 0;  //表示每次主键值增加的大小，例如设置1，则表示每次插入新记录后自动加1，默认为50  **int** allocationSize() **default** 50;  } |

**AUTO：主键由程序控制**

用法：

|  |
| --- |
| @Id  @GeneratedValue(strategy = GenerationType.***AUTO***)  **private** Long custId; |

**TABLE：使用一个特定的数据库表格来保存主键**

用法：

|  |
| --- |
| @Id  @GeneratedValue(strategy = GenerationType.***TABLE***, generator="pk\_gen")  @TableGenerator(name = "pk\_gen",  table="tb\_generator",  pkColumnName="gen\_name",  valueColumnName="gen\_value",  pkColumnValue="PAYABLEMOENY\_PK",  allocationSize=1  )  **private** Long custId;  --------------------------------------------------------  //@TableGenerator的定义：  @Target({TYPE, METHOD, FIELD})  @Retention(RUNTIME)  **public** **@interface** TableGenerator {  //表示该表主键生成策略的名称，它被引用在@GeneratedValue中设置的“generator”值中  String name();  //表示表生成策略所持久化的表名，例如，这里表使用的是数据库中的“tb\_generator”。  String table() **default** "";  //catalog和schema具体指定表所在的目录名或是数据库名  String catalog() **default** "";  String schema() **default** "";  //属性的值表示在持久化表中，该主键生成策略所对应键值的名称。例如在“tb\_generator”中将“gen\_name”作为主键的键值  String pkColumnName() **default** "";  //属性的值表示在持久化表中，该主键当前所生成的值，它的值将会随着每次创建累加。例如，在“tb\_generator”中将“gen\_value”作为主键的值  String valueColumnName() **default** "";  //属性的值表示在持久化表中，该生成策略所对应的主键。例如在“tb\_generator”表中，将“gen\_name”的值为“CUSTOMER\_PK”。  String pkColumnValue() **default** "";  //表示主键初识值，默认为0。  **int** initialValue() **default** 0;  //表示每次主键值增加的大小，例如设置成1，则表示每次创建新记录后自动加1，默认为50。  **int** allocationSize() **default** 50;  UniqueConstraint[] uniqueConstraints() **default** {};  }  //这里应用表tb\_generator，定义为 ：  CREATE TABLE tb\_generator (  id NUMBER NOT NULL,  gen\_name VARCHAR2(255) NOT NULL,  gen\_value NUMBER NOT NULL,  PRIMARY KEY(id)  ) |

### 对象使用方法介绍

#### Persistence工具类

Persistence工具类主要作用是用于获取EntityManagerFactory对象的 。通过调用该类的createEntityManagerFactory静态方法，根据配置文件中持久化单元名称创建EntityManagerFactory。

|  |
| --- |
| //1. 创建 EntitymanagerFactory  @Test  String unitName = "myJpa";  EntityManagerFactory factory= Persistence.*createEntityManagerFactory*(unitName); |

#### EntityManagerFactory

EntityManagerFactory 接口主要用来创建 EntityManager 实例

|  |
| --- |
| //创建实体管理类  EntityManager em = factory.createEntityManager(); |

由于EntityManagerFactory 是一个线程安全的对象（即多个线程访问同一个EntityManagerFactory 对象不会有线程安全问题），并且EntityManagerFactory 的创建极其浪费资源，所以在使用JPA编程时，我们可以对EntityManagerFactory 的创建进行优化，只需要做到一个工程只存在一个EntityManagerFactory 即可

#### EntityManager

在 JPA 规范中, EntityManager是完成持久化操作的核心对象。实体类作为普通 java对象，只有在调用 EntityManager将其持久化后才会变成持久化对象。EntityManager对象在一组实体类与底层数据源之间进行 O/R 映射的管理。它可以用来管理和更新 Entity Bean, 根椐主键查找 Entity Bean, 还可以通过JPQL语句查询实体。

我们可以通过调用EntityManager的方法完成获取事务，以及持久化数据库的操作

方法说明：

|  |
| --- |
| getTransaction : 获取事务对象  persist ： 保存操作  merge ： 更新操作  remove ： 删除操作  find/getReference ： 根据id查询 |

#### EntityTransaction

在 JPA 规范中, EntityTransaction是完成事务操作的核心对象，对于EntityTransaction在我们的java代码中承接的功能比较简单

|  |
| --- |
| begin：开启事务  commit：提交事务  rollback：回滚事务 |

# 使用JPA实现CRUD操作

## 提取工具类

|  |
| --- |
| public class JpaUtil {   private static class EntityManagerFactoryHolder {   //EntityManagerFactory在系统中单例存在  private static EntityManagerFactory *factory* = Persistence.*createEntityManagerFactory*("myJpa");    public static EntityManagerFactory getFactory() {  return *factory*;  }   }   public static EntityManager getEntityManager() {  //获得EntityManager对象  return EntityManagerFactoryHolder.*getFactory*().createEntityManager();  }   } |

## 添加数据

|  |
| --- |
| @Test public void addCustomer() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //获得事务管理器  EntityTransaction transaction = entityManager.getTransaction();  //开启事务  try {  transaction.begin();  //创建一个Customer对象  Customer customer = new Customer();  customer.setCustName("宋江");  customer.setCustAddress("山东");  customer.setCustPhone("13322445566");  customer.setCustLevel("普通客户");  //把customer对象添加到数据库  entityManager.persist(customer);  //提交事务  transaction.commit();   } catch (Exception e) {  //系统发生异常，回滚。  transaction.rollback();  } finally {  //释放资源  entityManager.close();  }  } |

## 删除数据

|  |
| --- |
| @Test public void deleteCustomer() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //获得事务管理器  EntityTransaction transaction = entityManager.getTransaction();  //开启事务  try {  transaction.begin();  //根据id查询客户信息  Customer customer = entityManager.find(Customer.class, 1l);  //根据id删除客户信息  entityManager.remove(customer);  //提交事务  transaction.commit();   } catch (Exception e) {  //系统发生异常，回滚。  transaction.rollback();  } finally {  //释放资源  entityManager.close();  }  } |

## 修改数据

|  |
| --- |
| @Test public void updateCustomer() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //获得事务管理器  EntityTransaction transaction = entityManager.getTransaction();  //开启事务  try {  transaction.begin();  //根据id查询客户信息  Customer customer = entityManager.find(Customer.class, 2l);  //修改客户信息  customer.setCustName("武松");  //更新客户信息  entityManager.merge(customer);  //提交事务  transaction.commit();   } catch (Exception e) {  //系统发生异常，回滚。  transaction.rollback();  } finally {  //释放资源  entityManager.close();  } } |

## 查询数据

### 根据id查询

在删除和修改的数据的代码中，我们已经使用了find方法来查询数据，此方法为即时加载方法。也就是说查询之后马上得到数据。

|  |
| --- |
| @Test public void findCustomerById() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //根据id查询客户信息  Customer customer = entityManager.find(Customer.class, 2l);  //打印结果  System.*out*.println(customer);  //释放资源  entityManager.close(); } |

在jpa中还提供一个根据id查询数据的方法getReference，此方法和find方法可以起到相同的效果，不同的是此方法为延迟加载（懒加载）方法。

|  |
| --- |
| @Test public void findCustomerById2() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //根据id查询客户信息  Customer customer = entityManager.getReference(Customer.class, 2l);  //打印结果  System.*out*.println(customer);  //释放资源  entityManager.close(); } |

### 使用JPQL查询

JPQL全称Java Persistence Query Language，基于首次在EJB2.0中引入的EJB查询语言(EJB QL),Java持久化查询语言(JPQL)是一种可移植的查询语言，旨在以面向对象表达式语言的表达式，将SQL语法和简单查询语义绑定在一起·使用这种语言编写的查询是可移植的，可以被编译成所有主流数据库服务器上的SQL。

其特征与原生SQL语句类似，并且完全面向对象，通过类名和属性访问，而不是表名和表的属性。

#### 查询全部

|  |
| --- |
| @Test public void findAll() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //JPQL查询语句  String sql = "from Customer";  //创建查询对象  Query query = entityManager.createQuery(sql);  //执行查询  List<Customer> list = query.getResultList();  for (Customer c:list) {  System.*out*.println(c);  } } |

#### 分页查询

|  |
| --- |
| @Test public void findAllPage() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //JPQL查询语句  String sql = "from Customer";  //创建查询对象  Query query = entityManager.createQuery(sql);  //设置起始行号  query.setFirstResult(0);  //设置每页的行数  query.setMaxResults(3);  //执行查询  List<Customer> list = query.getResultList();  for (Customer c:list) {  System.*out*.println(c);  } } |

#### 带条件查询

|  |
| --- |
| @Test public void findByCriteria() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //JPQL查询语句  String sql = "from Customer where custName like ?";  //创建查询对象  Query query = entityManager.createQuery(sql);  //设置查询条件。参数1：参数的索引从1开始，参数2：查询参数  query.setParameter(1, "%武%");  //执行查询  List<Customer> list = query.getResultList();  for (Customer c:list) {  System.*out*.println(c);  } } |

#### 排序查询

|  |
| --- |
| @Test public void findBySort() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //JPQL查询语句  String sql = "from Customer where custName like ? order by custId desc";  //创建查询对象  Query query = entityManager.createQuery(sql);  //设置查询条件。参数1：参数的索引从1开始，参数2：查询参数  query.setParameter(1, "%宋%");  //执行查询  List<Customer> list = query.getResultList();  for (Customer c:list) {  System.*out*.println(c);  } } |

#### 统计查询

|  |
| --- |
| @Test public void findByCount() {  //获得EntityManager对象  EntityManager entityManager = JpaUtil.*getEntityManager*();  //JPQL查询语句  String sql = "select count(\*) from Customer where custName like ?";  //创建查询对象  Query query = entityManager.createQuery(sql);  //设置查询条件。参数1：参数的索引从1开始，参数2：查询参数  query.setParameter(1, "%宋%");  //执行查询  Object result = query.getSingleResult();  System.*out*.println(result); } |