JPQL 就是一种查询语言，具有与 SQL 相类似的特征，JPQL 是完全面向对象的，具备继承、多态和关联等特性，和 hibernate HQL 很相似。

查询语句的参数

JPQL 语句支持两种方式的参数定义方式：命名参数和位置参数。。在同一个查询语句中只允许使用一种参数定义方式。

命令参数的格式为：“：+参数名”
例：

```
Query query = em.createQuery("select p from Person p where
p.personid=:Id");
query.setParameter("Id",new Integer(1));
```

位置参数的格式为“?+位置编号”
例：

```
Query query = em.createQuery("select p from Person p where
p.personid=?1");
query.setParameter(1,new Integer(1));
```

如果你需要传递 java.util.Date 或 java.util.Calendar 参数进一个参数查询，你需要使用一个特殊的 setParameter()方法，相关的 setParameter 方法定义如下：

```
public interface Query
{
//命名参数查询时使用，参数类型为 java.util.Date
Query setParameter(String name, java.util.Date value, TemporalType temporalType);
//命名参数查询时使用，参数类型为 java.util.Calendar
Query setParameter(String name, Calendar value, TemporalType temporalType);
//位置参数查询时使用，参数类型为 java.util.Date
Query setParameter(int position, Date value, TemporalType temporalType);
//位置参数查询时使用，参数类型为 java.util.Calendar
Query setParameter(int position, Calendar value, TemporalType temporalType);
}
```

因为一个 Date 或 Calendar 对象能够描述一个真实的日期、时间或时间戳.所以我们需要告诉 Query 对象怎么使用这些参数，我们把 javax.persistence.TemporalType 作为参数传递进 setParameter 方法，告诉查询接口在转换 java.util.Date 或 java.util.Calendar 参数到本地 SQL 时使用什么数据库类型。

下面通过实例来学习 JPQL 语句，例子的 entity Bean 有 Person, Order, OrderItem，他们之间的关系是：一个 Person 有多个 Order,一个 Order 有多个 OrderItem。

JPQL 语句的大小写敏感性：除了 Java 类和属性名称外，查询都是大小写不敏感的。所以，SeLeCT 和 sELEct 以及 SELECT 相同的，但是 com.foshanshop.ejb3.bean.Person 和 com.foshanshop.ejb3.bean.PERSon 是不同的，person.name 和 person.NAME 也是不同的。

命名查询

可以在实体 bean 上通过**@NamedQuery** or **@NamedQueries** 预先定义一个或多个查询语句，减少每次因书写错误而引起的 BUG。通常把经常使用的查询语句定义成命名查询。

定义单个命名查询：

```
@NamedQuery(name="getPerson", query= "FROM Person WHERE personid=?1")
@Entity
public class Person implements Serializable{
```

如果要定义多个命名查询，应在@javax.persistence.NamedQueries 里定义@NamedQuery：

```
@NamedQueries({
    @NamedQuery(name="getPerson", query= "FROM Person WHERE personid=?1"),
    @NamedQuery(name="getPersonList", query= "FROM Person WHERE age>?1")
})
@Entity
public class Person implements Serializable{
```

当命名查询定义好了之后，我们就可以通过名称执行其查询。代码如下：

```
Query query = em.createNamedQuery("getPerson");
query.setParameter(1, 1);
```

排序(order by)

"ASC"和"DESC"分别为升序和降序，JPQL 中默认为 asc 升序

例：

```
//先按年龄降序排序，然后按出生日期升序排序
Query query = em.createQuery("select p from Person p order by p.age desc, p.birthday asc");
```

查询部分属性

通常来说，都是针对 Entity 类的查询，返回的也是被查询的 Entity 类的实体。JPQL 也允许我们直接查询返回我们需要的属性，而不是返回整个 Entity。在一些 Entity 中属性特别多的情况，这样的查询可以提高性能

例：

```
//只查询我们感兴趣的属性(列)
```

```
Query query=em.createQuery("select p.personid, p.name from Person p order by
p.personid desc ");
//集合中的元素不再是 Person,而是一个 Object[]对象数组
List result = query.getResultList();
if (result!=null){
Iterator iterator = result.iterator();
while( iterator.hasNext() ){
Object[] row = ( Object[]) iterator.next();
int personid = Integer.parseInt(row[0].toString());
String PersonName = row[1].toString();
。。。。
}
}
```

查询中使用构造器(Constructor)

JPQL 支持将查询的属性结果直接作为一个 java class 的构造器参数，并产生实体作为结果返回。**例如上面的例子只获取 person entity bean 的 name and personid 属性，我们不希望返回的集合的元素是 object[]，而希望用一个类来包装它。就要用到使用构造器。**

例：

```
public class SimplePerson {
  private Integer personid;
private String name;
     。。。。
public SimplePerson() {
}
public SimplePerson(Integer personid, String name) {
this.name = name;
this. personid = personid;
}
}
```

查询代码为：

```
//我们把需要的两个属性作为 SimplePerson 的构造器参数，并使用 new 函数。
Query query = em.createQuery("select new com.foshanshop.ejb3.bean.SimplePerson(p.
personid, p.name) from Person p order by p.personid desc");
//集合中的元素是 SimplePerson 对象
List result = query.getResultList();
if (result!=null){
Iterator iterator = result.iterator();
while( iterator.hasNext() ){
     SimplePerson simpleperson = (SimplePerson) iterator.next();
。。。。
}
```

```
}
```

聚合查询(Aggregation)

JPQL 支持的聚合函数包括：

1. AVG()
2. SUM()
3. COUNT()，返回类型为 Long，注意 count(*)语法在 hibernate 中可用，但在 toplink 其它产品中并不可用
4. MAX()
5. MIN()

例：

```
//获取最大年龄
Query query = em.createQuery("select max(p.age) from Person p");
Object result = query.getSingleResult();
String maxAge = result.toString();
//获取平均年龄
query = em.createQuery("select avg(p.age) from Person p");
//获取最小年龄
query = em.createQuery("select min(p.age) from Person p");
//获取总人数
query = em.createQuery("select count(p) from Person p");
//获取年龄总和
query = em.createQuery("select sum(p.age) from Person p");
```

如果聚合函数不是 select...from 的唯一一个返回列，需要使用"GROUP BY"语句。**"GROUP BY"应该包含 select 语句中除了聚合函数外的所有属性。**

例：

```
//返回男女生各自的总人数
Query query = em.createQuery("select p.sex, count(p) from Person p group by p.sex");
//集合中的元素不再是 Person,而是一个 Object[]对象数组
List result = query.getResultList();
```

如果还需要加上查询条件，需要使用"HAVING"条件语句而不是"WHERE"语句

例：

```
//返回人数超过 1 人的性别
Query query = em.createQuery("select p.sex, count(p) from Person p group by p.sex
having count(*)>?1");
//设置查询中的参数
query.setParameter(1, new Long(1));
//集合中的元素不再是 Person,而是一个 Object[]对象数组
List result = query.getResultList();
```

关联(join)

JPQL 仍然支持和 SQL 中类似的关联语法：

left out join/left join

inner join

left join fetch/inner join fetch

left out join/left join 等，都是允许符合条件的右边表达式中的 Entiies 为空（需要显式使用 left join/left outer join 的情况会比较少。）

例：

```
//获取 26 岁人的订单,不管 Order 中是否有 OrderItem
select o from Order o left join o.orderItems where o.ower.age=26 order by o.orderid
```

inner join 要求右边的表达式必须返回 Entities。

例：

```
//获取 26 岁人的订单,Order 中必须要有 OrderItem
select o from Order o inner join o.orderItems where o.ower.age=26 order by o.orderid
```

！！重要知识点：在默认的查询中，Entity 中的集合属性默认不会被关联，集合属性默认是延迟加载(lazy-load)。那么，left fetch/left out fetch/inner join fetch 提供了一种灵活的查询加载方式来提高查询的性能。

例：

```
private String QueryInnerJoinLazyLoad(){
// 默认不关联集合属性变量(orderItems)对应的表
Query query = em.createQuery("select o from Order o inner join o.orderItems where
o.ower.age=26 order by o.orderid");
List result = query.getResultList();
if (result!=null && result.size()>0){
//这时获得 Order 实体中 orderItems( 集合属性变量)为空
Order order = (Order) result.get(0);
//当需要时，EJB3 Runtime 才会执行一条 SQL 语句来加载属于当前 Order 的
//OrderItems
Set<OrderItem> list = order.getOrderItems();
Iterator<OrderItem> iterator = list.iterator();
if (iterator.hasNext()){
  OrderItem orderItem =iterator.next();
  System.out.println ("订购产品名："+ orderItem.getProductname());
  }
}
```

上面代码在执行"select o from Order o inner join o.orderItems where o.ower.age=26 order by o.orderid"时编译成的 SQL 如下（他不包含集合属性变量(orderItems)对应表的字段）：

```
select order0_.orderid as orderid6_, order0_.amount as amount6_, order0_.person_id
as
person4_6_, order0_.createdate as createdate6_ from Orders order0_ inner join
OrderItems
orderitems1_ on order0_.orderid=orderitems1_.order_id, Person person2_ where
order0_.person_id=person2_.personid and person2_.age=26 order by order0_.orderid
```

上面代码当执行到 Set<OrderItem> list = order.getOrderItems();时**才会执行一条 SQL 语句来加载**属于当前 Order 的 OrderItems，编译成的 SQL 如下：

```
select orderitems0_.order_id as order4_1_, orderitems0_.id as id1_,
orderitems0_.id as id7_0_,
orderitems0_.order_id as order4_7_0_, orderitems0_.productname as productn2_7_0_,
orderitems0_.price as price7_0_ from OrderItems orderitems0_ where
orderitems0_.order_id=?
order by orderitems0_.id ASC
```

这样的查询性能上有不足的地方。为了查询 N 个 Order，我们需要一条 SQL 语句获得所有的 Order 的原始对象属性，但需要另外 N 条语句获得每个 Order 的 orderItems 集合属性。**为了避免 N+1 的性能问题，我们可以利用 join fetch 一次过用一条 SQL 语句把 Order 的所有信息查询出来**

例子

```
//获取 26 岁人的订单,Order 中必须要有 OrderItem
Query query = em.createQuery("select o from Order o inner join fetch o.orderItems
where
o.ower.age=26 order by o.orderid");
```

上面这句 HPQL 编译成以下的 SQL：

```
select order0_.orderid as orderid18_0_, orderitems1_.id as id19_1_, order0_.amount
as
amount18_0_,order0_.person_id as person4_18_0_, order0_.createdate as
createdate18_0_,
orderitems1_.order_id as order4_19_1_, orderitems1_.productname as
productn2_19_1_,
orderitems1_.price as price19_1_, orderitems1_.order_id as order4_0__,
orderitems1_.id as id0__
from Orders order0_ inner join OrderItems orderitems1_ on
order0_.orderid=orderitems1_.order_id, Person person2_ where
order0_.person_id=person2_.personid and person2_.age=26 order by order0_.orderid,
```

```
orderitems1_.id ASC
```

上面由于使用了 fetch,这个查询只会产生一条 SQL 语句，比原来需要 N+1 条 SQL 语句在性能上有了极大的提升

排除相同的记录 DISTINCT

使用关联查询，我们很经常得到重复的对象，如下面语句：

```
"select o from Order o inner join fetch o.orderItems order by o.orderid "
```

当有 N 个 orderItem 时就会产生 N 个 Order,而有些 Order 对象往往是相同的，这时我们需要使用 DISTINCT 关键字来排除掉相同的对象。

例：

```
select DISTINCT o from Order o inner join fetch o.orderItems order by o.orderid
```

比较 Entity

在查询中使用参数查询时，参数类型除了 String, 原始数据类型(int, double 等)和它们的对象类型(Integer, Double 等),也可以是 Entity 的实例。

例：

```
//查询某人的所有订单
Query query = em.createQuery("select o from Order o where o.ower =?1 order by o.orderid");
Person person = new Person();
person.setPersonid(new Integer(1));
//设置查询中的参数
query.setParameter(1,person);
```

批量更新(Batch Update)

HPQL 支持批量更新

例：

```
//把所有订单的金额加 10
Query query = em.createQuery("update Order as o set o.amount=o.amount+10");
//update 的记录数
int result = query.executeUpdate();
```

批量删除(Batch Remove)

例：

```
//把金额小于 100 的订单删除,先删除订单子项,再删除订单
Query query = em.createQuery("delete from OrderItem item where item.order in(from Order as o where o.amount<100)");
query.executeUpdate();
query = em.createQuery("delete from Order as o where o.amount<100");
query.executeUpdate();//delete 的记录数
```

使用操作符 NOT

```
//查询除了指定人之外的所有订单
Query query = em.createQuery("select o from Order o where not(o.ower =?1) order by o.orderid");
Person person = new Person();
person.setPersonid(new Integer(2));
//设置查询中的参数
query.setParameter(1,person);
```

使用操作符 BETWEEN

```
select o from Order as o where o.amount between 300 and 1000
```

使用操作符 IN

```
//查找年龄为 26,21 的 Person
select p from Person as p where p.age in(26,21)
```

使用操作符 LIKE

```
//查找以字符串"li"开头的 Person
select p from Person as p where p.name like 'li%'
```

使用操作符 IS NULL

```
//查询含有购买者的所有 Order
select o from Order as o where o.ower is [not] null
```

使用操作符 IS EMPTY

IS EMPTY 是针对集合属性(Collection)的操作符。可以和 NOT 一起使用。注:低版权的 Mysql 不支持 IS EMPTY

```
//查询含有订单项的所有 Order
select o from Order as o where o.orderItems is [not] empty
```

使用操作符 EXISTS

[NOT]EXISTS **需要和子查询配合使用**。注：低版权的 Mysql 不支持 EXISTS

```
//如果存在订单号为 1 的订单，就获取所有 OrderItem
select oi from OrderItem as oi where exists (select o from Order o where o.orderid=1)
//如果不存在订单号为 10 的订单，就获取 id 为 1 的 OrderItem
select oi from OrderItem as oi where oi.id=1 and not exists (select o from Order o where o.orderid=10)
```

字符串函数

JPQL 定义了内置函数方便使用。这些函数的使用方法和 SQL 中相应的函数方法类似。包括：

1. CONCAT 字符串拼接
2. SUBSTRING 字符串截取
3. TRIM 去掉空格
4. LOWER 转换成小写
5. UPPER 装换成大写
6. LENGTH 字符串长度
7. LOCATE 字符串定位

例：

```
//查询所有人员，并在姓名后面加上字符串"_foshan"
select p.personid, concat(p.name, '_foshan') from Person as p
//查询所有人员,只取姓名的前三个字符
select p.personid, substring(p.name,1,3) from Person as p
```

计算函数

HPQL 定义的计算函数包括：

ABS 绝对值
SQRT 平方根
MOD 取余数
SIZE 取集合的数量

例：

```
//查询所有 Order 的订单号及其订单项的数量
select o.orderid, size(o.orderItems) from Order as o group by o.orderid
//查询所有 Order 的订单号及其总金额/10 的余数
select o.orderid, mod(o.amount, 10) from Order as o
```

子查询

子查询可以用于 WHERE 和 HAVING 条件语句中

例：

```
//查询年龄为 26 岁的购买者的所有 Order
select o from Order as o where o.ower in(select p from Person as p where p.age =26)
```

结果集分页

有些时候当执行一个查询会返回成千上万条记录，事实上我们只需要显示一部分数据。这时我们需要对结果集进行分页，QueryAPI 有两个接口方法可以解决这个问题：**setMaxResults()和 setFirstResult()**。

setMaxResults 方法设置获取多少条记录

setFirstResult 方法设置从结果集中的那个索引开始获取（假如返回的记录有 3 条，容器会自动为记录编上索引，索引从 0 开始，依次为 0，1，2）

例：

```
public List getPersonList(int max,int whichpage) {
try {
int index = (whichpage-1) * max;
Query query = em.createQuery("from Person p order by personid asc");
List list = query.setMaxResults(max).
     setFirstResult(index).
     getResultList();
em.clear();//分离内存中受 EntityManager 管理的实体 bean，让 VM 进行垃圾回收
return list;
} catch (Exception e) {
e.printStackTrace();
return null;
}
}
```