## 加载策略(优化查询)

### 策略的种类

* 立即检索：立即查询，在执行查询语句时，立即查询所有的数据。
* 延迟检索：延迟查询，在执行查询语句之后，在需要时在查询。（懒加载）

### 策略的应用:

类级别的加载策略.

关联级别的加载策略.

#### 类级别加载策略:

**1. get/load**

get: 立即查询数据库,将数据初始化

load: hbm文件中,class元素的lazy属性绝定类级别load方法的加载策略

true:先返回一个代理对象.使用代理对象的属性时,才去查询数据库.

false: 与get一样,会立即加载数据

|  |
| --- |
| @Test  **public** **void** run1() {  //类级别  Session session = HibernateUtil.getCurrentSession();    Transaction ts = session.beginTransaction();  //1立即  // Customer customer = (Customer) session.get(Customer.class, 1);  //2延迟  Customer customer = (Customer) session.load(Customer.**class**, 1);        //打印  System.*out*.println(customer.getCid());  System.*out*.println(customer.getCname());  ts.commit(); |

#### 关联级别检索

##### 一对多或多对多

#### 介绍

* 容器<set> 提供两个属性：fetch、lazy

lazy：关联对象是否延迟。

fetch：确定使用sql格式

* fetch：join、select、subselect

join：底层使用迫切左外连接

select：使用多个select语句（默认值）

subselect：使用子查询

* lazy：false、true、extra

false：立即

true：延迟（默认值）

extra：极其懒惰

**集合(一对多):**

set

lazy: 是否对set数据使用懒加载

true:(默认值) 对集合使用才加载

false: 集合将会被立即加载

extra: 极其懒惰,如果使用集合时,之调用size方法查询数量, Hibernate会发送count语句,只查询数量.不加载集合内数据.

fetch : 决定加载集合使用的sql语句种类

select: (默认值) 普通select查询

join: 表链接语句查询集合数据

subselect: 使用子查询 一次加载多个Customer的订单数据

select true 默认值, 会在使用集合时加载,普通select语句

select false 立刻使用select语句加载集合数据

select extra 会在使用集合时加载,普通select语句,如果只是获得集合的长度,会发送Count语句查询长度.

join true 查询集合时使用表链接查询,会立刻加载集合数据

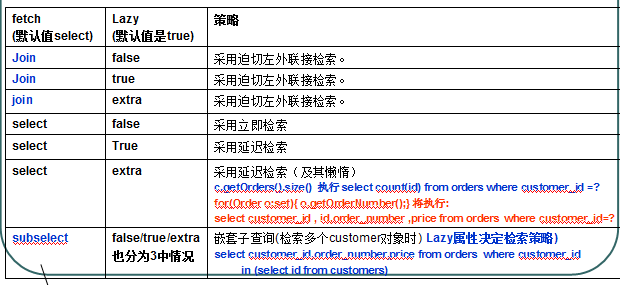
join false 查询集合时使用表链接查询,会立刻加载集合数据

join extra 查询集合时使用表链接查询,会立刻加载集合数据

subselect true 会在使用集合时加载,子查询语句

subselect false 会在查询用户时,立即使用子查询加载客户的订单数据

subselect extra 会在使用集合时加载,子查询语句,如果只是获得集合的长度,会发送Count语句查询长度.



### 多对一

#### 介绍

* <many-to-one fetch="" lazy=""> (<one-to-one>)
* fetch取值：join、select

join：底层使用迫切左外连接

select：多条select语句

* lazy取值：false、proxy、no-proxy

false：立即

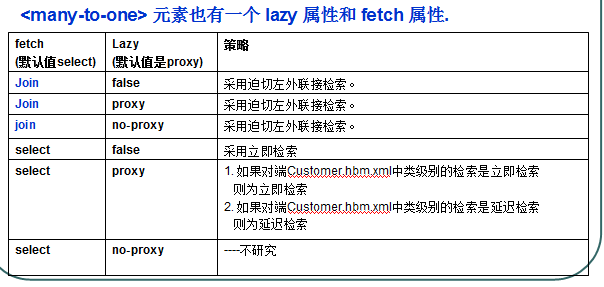
proxy：采用关联对象 类级别检索的策略。lazy="proxy"是否延迟，取决关联对象 类级别检索策略。

订单 关联 客户 （多对一）

订单 立即获得 客户，需要在客户Customer.hbm.xml <class lazy="false">

订单 延迟获得 客户，需要在客户Customer.hbm.xml <class lazy="true">

no-proxy 不研究



## 检索总结

|  |  |  |  |
| --- | --- | --- | --- |
| 检索策略 | 优点 | 缺点 | 优先考虑使用的场合 |
| 立即检索 | 对应用程序完全透明，不管对象处于持久化状态还是游离状态，应用程序都可以从一个对象导航到关联的对象 | (1)select语句多  (2)可能会加载应用程序不需要访问的对象，浪费许多内存空间。 | (1)类级别  (2)应用程序需要立即访问的对象  (3)使用了二级缓存 |
| 延迟检索 | 由应用程序决定需要加载哪些对象，可以避免执行多余的select语句，以及避免加载应用程序不需要访问的对象。因此能提高检索性能，并节省内存空间。 | 应用程序如果希望访问游离状态的代理类实例，必须保证她在持久化状态时已经被初始化。 | (1)一对多或者多对多关联  (2)应用程序不需要立即访问或者根本不会访问的对象 |
| 表连接检索 | (1)对应用程序完全透明，不管对象处于持久化状态还是游离状态，都可从一个对象导航到另一个对象。  (2)使用了外连接，select语句少 | (1)可能会加载应用程序不需要访问的对象，浪费内存。  (2)复杂的数据库表连接也会影响检索性能。 | (1)多对一或一对一关联  (2)需要立即访问的对象  (3)数据库有良好的表连接性能。 |

Web 三层开发 页面==》action🡺service.=>dao层

Lazy=”true”

Dao: User u=session.load(User.class,1L);//user orders <set lazy=”extra”>

Service:dao.findById(1L);

Action: request.setAttribute(“user”,user); 转发到页面 session已经关闭了

页面jsp:${user.uname} ${user.orders}

1. 勤快点查找
2. 业务层，业务层知道页面要展示哪些数据，获得以下你要战士的数据。

User

## 技术分析之Hibernate框架的查询方式

### 1. 唯一标识OID的检索方式

session.get(对象.class,OID)

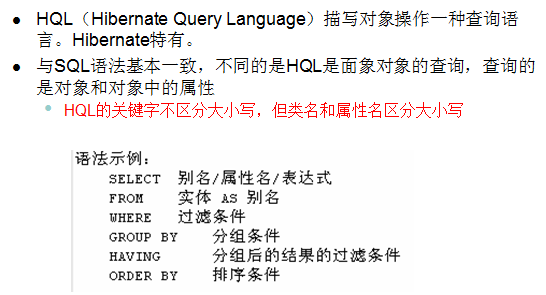
session.load(User.class,oid);

### 2. 对象的导航的方式

User u=session.get(User.class,1L);

u.getOrders();

### 3. HQL的检索方式



Query query = session.createQuery("hql语句") --> 对象，对象属性

Hibernate Query Language -- Hibernate的查询语言

### 4. QBC的检索方式

Query By Criteria -- 条件查询

### 5. SQL检索方式（了解）

SQLQuery sqlQuery = session.createSQLQuery("sql 语句") --->表，表字段（列）

sqlQuery.list() 查询所有

sqlQuery.uniqueResult() 查询一个

本地的SQL检索

技术分析之HQL的查询方式概述

1. HQL的介绍

HQL(Hibernate Query Language) 是面向对象的查询语言, 它和 SQL 查询语言有些相似

在 Hibernate 提供的各种检索方式中, HQL 是使用最广的一种检索方式

2. HQL与SQL的关系

HQL 查询语句是面向对象的,Hibernate负责解析HQL查询语句, 然后根据对象-关系映射文件中的映射信息, 把 HQL 查询语句翻译成相应的 SQL 语句.

HQL 查询语句中的主体是域模型中的类及类的属性

SQL 查询语句是与关系数据库绑定在一起的. SQL查询语句中的主体是数据库表及表的字段

## 技术分析之HQL的查询演示

### 1. HQL基本的查询格式

支持方法链的编程，即直接调用list()方法

简单的代码如下

session.createQuery("from Customer").list();

### 2. 使用别名的方式

可以使用别名的方式

session.createQuery("from Customer c").list();

session.createQuery("select c from Customer c").list();

### 3. 排序查询

排序查询和SQL语句中的排序的语法是一样的

升序

session.createQuery("from Customer order by cust\_id").list();

降序

session.createQuery("from Customer order by cust\_id desc").list();

### 4. 分页查询

Hibernate框架提供了分页的方法，咱们可以调用方法来完成分页

两个方法如下

setFirstResult(a) -- 从哪条记录开始，如果查询是从第一条开启，值是0

setMaxResults(b) -- 每页查询的记录条数

演示代码如下

List<LinkMan> list = session.createQuery("from LinkMan").setFirstResult(0).setMaxResults().list();

### 5. 带条件的查询

setParameter("?号的位置，默认从0开始","参数的值"); 不用考虑参数的具体类型

按位置绑定参数的条件查询（指定下标值，默认从0开始）

按名称绑定参数的条件查询（HQL语句中的 ? 号换成 :名称 的方式）

例如代码如下

|  |
| --- |
| Query query = session.createQuery("from Linkman where lkm\_name like ? order by lkm\_id desc");  query.setFirstResult(0).setMaxResults(3);  query.setParameter(0, "%熊%");  List<Linkman> list = query.list();  for (Linkman linkman : list) {  System.out.println(linkman);  } |

### 6. HQL的投影查询

1. 投影查询就是想查询某一字段的值或者某几个字段的值

2. 投影查询的案例

如果查询多个字段，例如下面这种方式

|  |
| --- |
| List<Object[]> list = session.createQuery("select c.cust\_name,c.cust\_level from Customer c").list();  for (Object[] objects : list) {  System.out.println(Arrays.toString(objects));  } |

如果查询两个字段，也可以把这两个字段封装到对象中

先在持久化类中提供对应字段的构造方法

使用下面这种HQL语句的方式

|  |
| --- |
| List<Customer> list = session.createQuery("select new Customer(c.cust\_name,c.cust\_level) from Customer c").list();  for (Customer customer : list) {  System.out.println(customer);  } |

### 7.技术分析之聚合函数查询

**1. 获取总的记录数**

|  |
| --- |
| Session session = HibernateUtils.getCurrentSession();  Transaction tr = session.beginTransaction();  List<Number> list = session.createQuery("select count(c) from Customer c").list();  Long count = list.get(0).longValue();  System.out.println(count);  tr.commit(); |

**2. 获取某一列数据的和**

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

List<Number> list = session.createQuery("select sum(c.cust\_id) from Customer c").list();

Long count = list.get(0).longValue();

System.out.println(count);

tr.commit();

### 8 多表的查询进来使用HQL语句进行查询，HQL语句和SQL语句的查询语法比较类似。

笛卡尔积：在开发中，尽量避免出现笛卡尔积 ，这种结果集没有任何意义。

\* 内连接查询 按照条件查找数句

\* 显示内连接

\* select \* from customers c inner join orders o on c.cid = o.cno;

\* 隐式内连接

\* select \* from customers c,orders o where c.cid = o.cno;

\* 外连接查询

\*

\* select \* from customers c left join orders o on c.cid = o.cno;

\* 右外连接

\* select \* from customers c right join orders o on c.cid = o.cno;

#### HQL的多表查询



1.交叉连接 ，等效 sql 笛卡尔积

2.隐式内连接，等效 sql 隐式内连接

3.内连接，等效sql内连接

4.迫切内连接，hibernate底层使用 内连接。

5.左外连接，等效sql左外连接

6.迫切左外连接，hibernate底层使用 左外连接

7.右外连接，等效sql右外连接

**迫切和非迫切：**

\* 非迫切返回结果是Object[]

\* 迫切连接返回的结果是对象，把客户的信息封装到客户的对象中，把订单的信息封装到客户的Set集合中。

#### 内连接查询

\* 内连接使用 inner join ，默认返回的是Object数组

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

List<Object[]> list = session.createQuery("from Customer c inner join c.linkmans").list();

for (Object[] objects : list) {

System.out.println(Arrays.toString(objects));

}

tr.commit();

\* 迫切内连接:inner join fetch ，返回的是实体对象

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

List<Customer> list = session.createQuery("from Customer c inner join fetch c.linkmans").list();

Set<Customer> set = new HashSet<Customer>(list);

for (Customer customer : set) {

System.out.println(customer);

}

tr.commit();

#### 左外连接查询

\* 左外连接: 封装成List<Object[]>

\* 迫切左外连接

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

List<Customer> list = session.createQuery("from Customer c left join fetch c.linkmans").list();

Set<Customer> set = new HashSet<Customer>(list);

for (Customer customer : set) {

System.out.println(customer);

}

tr.commit();

## 技术分析之QBC检索方式

QBC：Query By Criteria 按条件进行查询

### 1. 简单查询，使用的是Criteria接口

List<Customer> list = session.createCriteria(Customer.class).list();

for (Customer customer : list) {

System.out.println(customer);

}

### 2. 排序查询

\* 需要使用addOrder()的方法来设置参数，参数使用org.hibernate.criterion.Order对象

\* 具体代码如下：

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

// 设置排序

criteria.addOrder(Order.desc("lkm\_id"));

List<Linkman> list = criteria.list();

for (Linkman linkman : list) {

System.out.println(linkman);

}

tr.commit();

### 3. 分页查询

\* QBC的分页查询也是使用两个方法

\* setFirstResult();

\* setMaxResults();

\* 代码如下;

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

// 设置排序

criteria.addOrder(Order.desc("lkm\_id"));

criteria.setFirstResult(0);

criteria.setMaxResults(3);

List<Linkman> list = criteria.list();

for (Linkman linkman : list) {

System.out.println(linkman);

}

tr.commit();

### 4. 条件查询（Criterion是查询条件的接口，Restrictions类是Hibernate框架提供的工具类，使用该工具类来设置查询条件）

\* 条件查询使用Criteria接口的add方法，用来传入条件。

\* 使用Restrictions的添加条件的方法，来添加条件，例如：

\* Restrictions.eq -- 相等

\* Restrictions.gt -- 大于号

\* Restrictions.ge -- 大于等于

\* Restrictions.lt -- 小于

\* Restrictions.le -- 小于等于

\* Restrictions.between -- 在之间

\* Restrictions.like -- 模糊查询

\* Restrictions.in -- 范围

\* Restrictions.and -- 并且

\* Restrictions.or -- 或者

\* 测试代码如下

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

// 设置排序

criteria.addOrder(Order.desc("lkm\_id"));

// 设置查询条件

criteria.add(Restrictions.or(Restrictions.eq("lkm\_gender", "男"), Restrictions.gt("lkm\_id", 3L)));

List<Linkman> list = criteria.list();

for (Linkman linkman : list) {

System.out.println(linkman);

}

tr.commit();

### 5. 聚合函数查询（Projection的聚合函数的接口，而Projections是Hibernate提供的工具类，使用该工具类设置聚合函数查询）

\* 使用QBC的聚合函数查询，需要使用criteria.setProjection()方法

\* 具体的代码如下

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

Criteria criteria = session.createCriteria(Linkman.class);

criteria.setProjection(Projections.rowCount());

List<Number> list = criteria.list();

Long count = list.get(0).longValue();

System.out.println(count);

tr.commit();

## 技术分析之SQL查询方式（了解）

1. 基本语法

Session session = HibernateUtils.getCurrentSession();

Transaction tr = session.beginTransaction();

SQLQuery sqlQuery = session.createSQLQuery("select \* from cst\_linkman where lkm\_gender = ?");

sqlQuery.setParameter(0,"男");

sqlQuery.addEntity(Linkman.class);

List<Linkman> list = sqlQuery.list();

System.out.println(list);

tr.commit();

## 命名查询