河北师范大学软件学院
Software College of Hebei Normal University

HIBERNATE

第八讲
Hibernate检索方式

BIG DATA Java与大数据分析

Java与移动智能设备开发

知识回顾

- Session缓存
- Hibernate对象的生命周期
- Hibernate操作持久化对象

目录

检索方式简介

- 导航对象图检索方式：
  - 根据已经加载的对象，导航到其他对象。
- OID检索方式：
  - 按照对象的OID来检索对象。Session 的 get() 和 load() 方法提供了这种功能。

检索方式简介

- HQL检索方式：
  - 使用面向对象的HQL查询语言。
- QBC检索方式：
  - 使用QBC（Query By Criteria）API来检索对象。这种API封装了基于字符串形式的查询语句，提供了更加面向对象的接口。
- 本地SQL检索方式：
  - 使用本地数据库的SQL查询语句。

HQL(Hibernate Query Language)简介

- HQL是一种面向对象的查询语言，和SQL查询语言有些类似。
- 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。
- Query接口是HQL查询接口，提供各种查询功能。

HQL实体检索

- 检索 USER 表的所有记录。
  - HQL语句中关键字大小写无关，但习惯将关键字小写。
  - from 关键字后面是类名不是数据库表名，类名需区分大小写。

```
Query query = session.createQuery("from User");
List userList = query.list();
```

  - 等价于：

```
String hql = "select u from User u";
Query query = session.createQuery(hql);
List userList = query.list();
```

HQL实体检索

- where 子句。
  - where 子句中给出的是类的属性名而不是数据库表字段名，其中属性名必须区分大小写。

```
String hql = "from User where userName='张三'";
Query query = session.createQuery(hql);
List userList = query.list();
```

HQL where子句

- HQL支持的各种运算符。

程序中指定的连接类型	HQL运算符
比较运算符	=、<>、>=、<=、>、<、is null、is not null
范围运算符	in、not in、between...and、not between...and
字符串模式匹配运算符	like
逻辑运算符	and、or、not

HQL使用别名查询

- HQL检索一个类的实例时，如果查询语句的其它地方需要引用它，可以给类指定一个别名。
  - as 关键字用来指定别名，as 关键字也可以省略。

```
from User where userName='张三'
from User as u where u.userName='张三'
from User u where u.userName='张三'
```

HQL支持多态查询

- 多态查询指查询出当前类以及所有子类的实例。
  - Employee 有两个子类：HourlyEmployee 和 SalariedEmployee。

```
Query query
     = session.createQuery("from Employee");
List employeeList = query.list();
```

HQL检索单个对象

- HQL查询返回结果方法：
  - list()：返回List类型的查询结果，返回所有满足条件的对象。
  - uniqueResult()：返回单个对象。

```
Query query = session.createQuery("from User u
                                   where u.userName='张三'");
User user = (User)query.uniqueResult();
```

HQL分组与排序

- order by 子句。

```
from User u order by u.userName
from User u order by u.userName desc
from User u order by u.userName, u.id desc
```

- group by 子句。

```
select count(u) from User u group by u.age
```

- having 子句。

```
select count(u) from User u group
by u.age having count(u)<4
```

HQL参数绑定

- 前述HQL中查询参数均直接在HQL中表达。

```
String hql
     = "from User as u where u.userName='张三'";
String hql
     = "from User as u where u.userName =" + name;
```

- 缺陷：
  - 代码更加零乱，可读性降低；
  - 难以进行性能优化；
  - 引入额外的安全风险。

HQL参数绑定

- 在HQL查询语句中按照参数位置绑定参数。
  - setParameter() 参数位置从0开始。

```
String hql = "from User u where u.userName=?";
Query query = session.createQuery(hql);
query.setParameter(0, "张三");
List<User> userList = query.list();
```

HQL参数绑定

- 在HQL查询语句中按参数名称绑定参数。

```
String hql = "from User u
        where u.userName=:name";
Query query = session.createQuery(hql);
query.setParameter("name", "张三");
List<User> userList = query.list();
```

HQL参数绑定

- HQL的绑定参数方法：
  - setParamter() 绑定任意类型的参数。
  - setProperties() 用于把命名参数与一个对象的属性值绑定，并且参数名称要与对象属性名称一致。

HQL实体更新

- 不使用HQL的实体更新。

```
Transaction tx = session.beginTransaction();
User user = (User) session.get(User.class, 1);
user.setUserName("Tom");
tx.commit();
```

- HQL实现实体更新的方式。

```
Transaction tx = session.beginTransaction();
String hql = "update User set userName='Tom'
        where id=2";
Query query = session.createQuery(hql);
int ret = query.executeUpdate();
tx.commit();
```

HQL实体删除

- 实体删除。

```
Transaction tx = session.beginTransaction();
String hql = "delete from User where id = 1";
Query query  = session.createQuery(hql);
int ret = query.executeUpdate();
tx .commit();
```

HQL子查询

- HQL支持在 where 子句中嵌入子查询语句，并且子查询语句必须放在括号内。
  - 查询订单数量大于0的所有用户：

```
from User u
where 0<(select count(o) from u.orderSet o)
```

  - 对应的SQL语句：

```
select * from user u
where 0<(select count(o) from orders o
          where u.id=o.userId )
```

HQL子查询

- HQL子查询说明以下几点：
  - 子查询分为相关子查询和无关子查询；
    - 相关子查询：子查询语句引用了外层查询语句定义的别名。
    - 无关子查询：子查询语句没有引用外层查询语句定义的别名。
  - HQL子查询功能依赖于底层数据库对子查询的支持；
  - HQL子查询返回的是多条记录，使用以下关键字量化。
    - all、any、some、in、exists。

HQL子查询

- 如果HQL子查询的是集合，HQL提供了一组操作集合的函数。
  - size()，获得集合中元素的个数；
  - maxIndex()，对于建立索引的集合，获得最大索引值；
  - minIndex()，对于建立索引的集合，获得最小索引值；
  - elements()，获得集合中所有元素。

HQL分页查询

- 做批量查询时，如果数据量很大就需要分页功能，HQL提供了用于分页查询的方法：
  - setFirstResult(int firstResult)
    - 设定从哪个对象开始检索。
  - setMaxResult(int maxResult)
    - 设定一次检索对象的数目。

HQL引用查询

- 引用查询指在映射文件中定义查询语句。
  - 在O/R映射xml文件中，用与<calss>元素同级的<query name="XXX">元素定义一个HQL查询语句。

```
<query name="findUser">from User</query>
```

  - 程序中通过session.getNamedQuery("XXX")调用对应的HQL。

```
Query query
     = session.createNamedQuery("findUser",
                                User.class);
List userList = query.list();
```

QBC(Query By Criteria)简介

- Query By Criteria(QBC) 可以看作是传统SQL的对象化表示。
- 它主要由Criteria接口，Criterion接口，Expression类组成。

QBC表达式

- 检索姓名为 Erica 的所有用户。

```
Criteria criteria=session.createCriteria(User.class);
Criterion c1= Restrictions.eq("userName", "张三");
criteria.add(c1);
List result = criteria.list();
```

- 步骤：
  - 调用 Session 的 createCriteria() 创建 Criteria 实例；
  - 通过 Restrictions 设定查询条件；
  - 调用 Criteria 实例的 list() 方法执行查询。

Restrictions类

运算类型	方法	描述
比较运算符	Restrictions.eq	等于
	Restrictions.ne	不等于
	Restrictions.gt	大于
	Restrictions.ge	大于等于
	Restrictions.lt	小于
	Restrictions.le	小于等于
	Restrictions.isNull	等于空值
	Restrictions.isNotNull	非空值

Restrictions类

运算类型	方法	描述
范围运算符	Restrictions.in	等于列表中的某个值
	Restrictions.not(Restrictions.in)	不等于列表中的任意值
	Restrictions.between	大于等于值1小于等于值2
字符串模糊匹配	Restrictions.like	字符串模糊匹配 like
逻辑运算符	Restrictions.and	逻辑与
	Restrictions.or	逻辑或
	Restrictions.not	逻辑非

本地SQL查询

- HQL 和 QBC 查询，Hibernate 会生成标准的 SQL 语句适合不同的数据库平台。
- 有时需要根据底层数据库生成特殊的 SQL 查询语句，Hibernate 对本地 SQL 查询提供了内置支持。

本地SQL查询

- 查询所有的用户信息：

```
String sql = "select * from  user";
NativeQuery query = session.createNativeQuery(sql,
    User.class);
List list = query.list();
```

- 步骤：
  - 调用session.createNativeQuery()创建NativeQuery实例，并指定查询的实体类型；
  - 调用NativeQuery实例的list() 方法执行查询（如果查询单个对象，调用uniqueResult() ）。

目录

检索策略

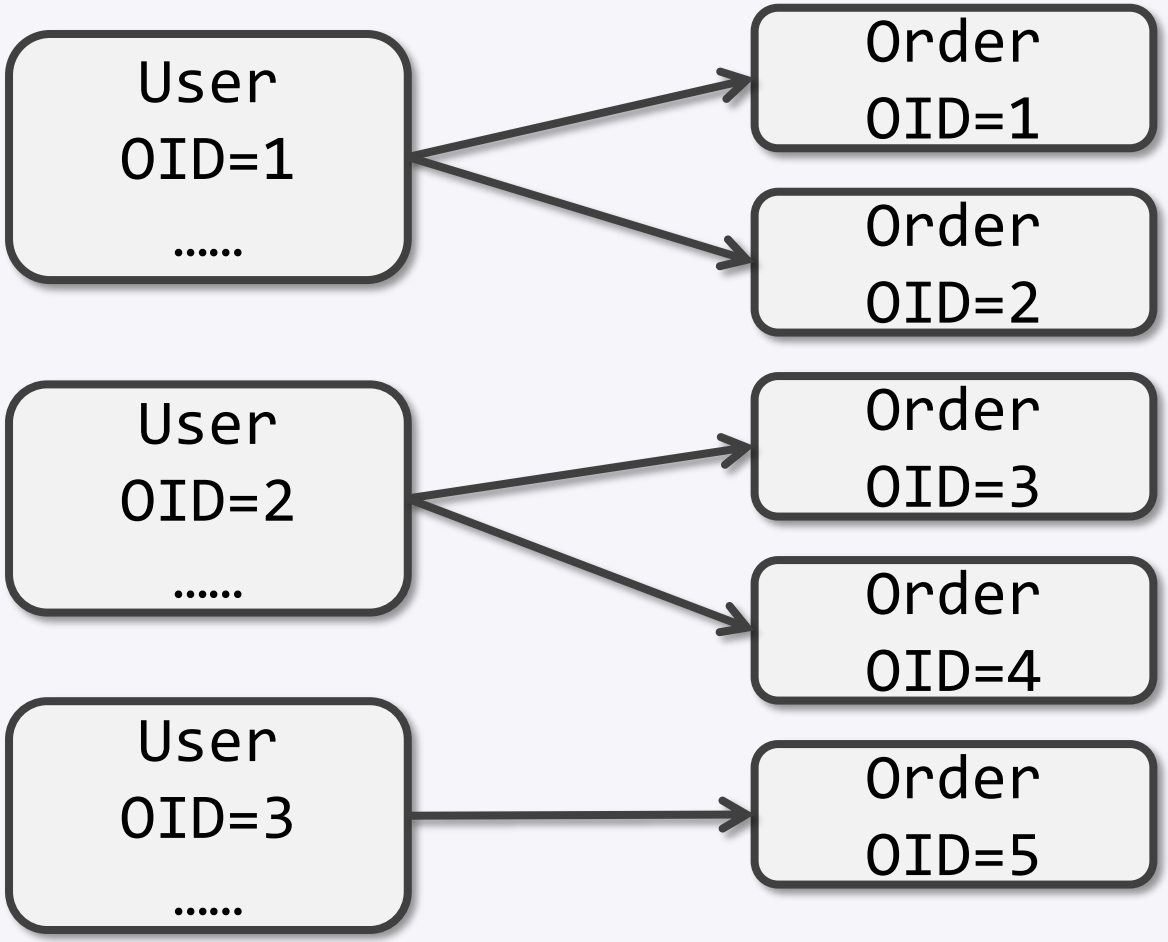
- 立即检索：立即加载检索方法指定的对象。
  - 加载多于需要的对象白白浪费内存空间；
  - select 语句数量多，频繁访问数据库，影响系统性能。
- 延迟检索：延迟加载检索方法指定的对象。
  - 避免多加载应用程序不需要访问的数据对象。

检索策略

- 迫切左外连接检索：利用SQL外连接查询功能加载检索方法指定对象。
  - 减少执行select语句的数量，减少数据库访问，提高系统性能。

引例

- User与Order的对象在内存中关联关系图。



检索执行分析

```
Query query = session.createQuery("from User");
List userList = query.list();
```

- Hibernate在执行检索方法时，要获取以下两种信息：
  - 类级别的检索策略：Hibernate检索方法指定的检索对象（User）的检索策略；
  - 关联级别的检索策略：与检索方法指定的检索对象相关联的对象（Order）的检索策略。

类级别和关联级别可选的检索策略

检索策略的作用域	可选的检索策略	默认的检索策略	影响到的检索方法
类级别	立即检索	立即检索（除Session的load()默认延迟检索）	影响Session的load()方法
	延迟检索		
关联级别	立即检索	默认延迟检索	影响所有检索方法
	延迟检索		
	迫切左外连接检索		影响session的load()和get()方法

三种检索策略的运行机制

检索策略类型	类级别	关联级别
立即检索	立即加载 检索方法指定的对象	立即加载与检索方法指定的对象相关联的对象
延迟检索	延迟加载 检索方法指定的对象	延迟加载与检索方法指定的对象相关联的对象
迫切左外连接检索	不适用	通过左外连接加载与检索方法指定的对象相关联的对象

类级别检索策略

- Hibernate类级别可选的检索策略：
  - 立即检索（加载）：映射配置文件中<class>元素的lazy 属性设置为 false。

```
<class name="User" table="USER" lazy="false">
```

  - 延迟检索（加载）：映射配置文件中<class>元素的lazy 属性设置为 true。

```
<class name="User" table="USER" lazy="true">
```

类级别检索策略 – 立即检索

- 立即检索
  - Hibernate立即执行 "select* from user where id=1" 。

```
User user = session.load(User.class,
                         new Integer(1));
```

类级别检索策略 – 延迟检索

- 延迟检索
  - 创建 User 的代理类实例（代理类是 Hibernate 动态生成的User的扩展类）；
  - Hibernate 创建的 User 代理类的实例仅仅初始化了 OID 属性，其他属性均为 null；
  - 当程序第一次访问代理类实例时（比如user.getX()），Hibernate会初始化代理类实例，执行 select 语句；
  - 如果程序访问 User 的 getId() 方法时，Hibernate并不会初始化代理类实例，因为 id 值已经存在。

类级别检索策略 – 延迟检索

- 类级别的检索策略只会影响到 Session 的 load() 方法，对 get() 和其它查询不起作用。
- 延迟加载对 load() 方法的影响：
  - 如果数据库中不存在对应的对象不会抛出异常，只有在调用 user.getXX() 时才会抛异常；
  - 代理类实例只能在当前 Session 范围内初始化；
  - Hibernate.initialize() 方法可以显示初始化代理类实例。

关联级别检索策略 – 一对多和多对多

- Hibernate需要确定以下检索策略：
  - User 类级别的检索策略；
  - User 一对多关联的Order对象的检索策略，User.hbm.xml 中<set>元素 lazy 和 outer-join 属性。

```
User user = session.get(User.class,
                        new Integer(1));
```

关联级别检索策略 – 一对多和多对多

- 在映射文件中用 <set>元素 来配置 一对多 和 多对多 关联关系。

lazy	outer-join	检索策略
flase	false	立即检索策略。
false	true	迫切左外连接检索，在配置文件中如果有多个<set>元素，只允许一个设置 outer-join=true。
true	false	延迟加载，优先考虑的检索策略。
true	true	迫切左外连接检索。

关联级别检索策略 - 多对一

- Hibernate需要确定以下检索策略：
  - Order 类级别的检索策略；
  - Order 多对一关联的 User 对象检索策略，Order.hbm.xml 中 <many-to-one>元素 outer-join 属性和 User.hbm.xml 中<class>元素的 lazy 属性。

```
Order order = session.get(Order.class,
                          new Integer(1));
```

关联级别检索策略 – 多对一和一对一

- 在映射文件中用 <many-to-one>元素 和 <one-to-one>元素 来分别设置多对一和一对一关联关系。
- <many-to-one>元素的 outer-join 属性：
  - auto、true、false三种取值。

关联级别检索策略 - 多对一

<many-to-one>元素的outer-join属性值	对应one方<class>元素的lazy属性值	检索many方(Orders)对象时，对关联的one方(Users)对象的检索策略
auto	false	迫切左外连接检索
auto	true	延迟检索
true	true or false	迫切左外连接检索
false	false	立即检索
false	true	延迟检索

使用注解配置检索策略

- fetch 参数可以设置为 FetchType.LAZY 或者 FetchType.EAGER。
  - EAGER：通过 outer join select 直接获取关联的对象。
  - LAZY(默认值)：在第一次访问关联对象的时候才会触发相应的 select 操作。

本章小结

- Hibernate检索方式
  - HQL检索方式
  - QBC检索方式
  - 本地SQL检索方式
- Hibernate检索策略
  - 立即检索
  - 延迟检索
  - 迫切左外连接检索

练习

THANK YOU