Hibernate查询和优化

- Hibernate查询和优化
  - 课前默写
  - 课程回顾
  - 今天任务
  - 教学目标
  - 第十三章 HQL查询
    - 13.1 HQL单表查询
      - 13.1.1 准备项目
      - 13.1.2 HQL单表查询
    - 13.2 HQL多表查询
      - 13.2.1 多表查询分类
      - 13.2.2 HQL多表查询
  - 第十四章 QBC查询
    - 14.1 QBC简介
    - 14.2 QBC使用
  - 第十五章 本地SQL查询
    - 15.1 Hibernate使用SQL语句
  - 第十六章 延迟加载策略
    - 16.1 类级别延迟加载
    - 16.2 关联级别
  - 第十七章 抓取策略
    - 17.1 查询一方单条
    - 17.2 查询一方多条
    - 17.3 查询多方的记录
  - 第十八章 整合连接池
    - 18.1 导入 c3p0的jar包
    - 18.2 Hibernate.cfg.xml整合
    - 18.3 测试是否启动连接池
  - 第十九章 二级缓存和查询缓存
    - 19.1 二级缓存
      - 19.1.1 二级缓存概述
      - 20.1.2 二级缓存使用
    - 19.2 查询缓存
      - 19.2.1 查询缓概述
      - 19.2.2 查询缓存使用
      - 19.2.3 查询缓存总结
  - 总结
  - 作业
  - 面试题

课前默写

```
1. 一对多映射的实现过程
2. 多对多映射的实现过程
3. 一对一映射的实现过程
```

课程回顾

```
1. 一对多映射
2. 多对多映射
3. 一对一映射
```

今天任务

```
1. HQL查询
2. QBC查询
3. 本地SQL查询
4. 延迟加载策略
5. 抓取策略
6. 整合连接池
7. 二级缓存
```

教学目标

```
1. 掌握HQL查询
2. 掌握QBC查询
3. 掌握本地SQL查询
4. 掌握延迟加载策略
5. 掌握抓取策略
6. 掌握整合连接池
7. 掌握二级缓存
```

第十三章 HQL查询

13.1 HQL单表查询

13.1.1 准备项目

- 创建项目: hinernate-03-query
- 引入jar,同前一个项目
- 复制实体(订单,客户),映射,配置,工具类.

13.1.2 HQL单表查询

```
/**
 * 演示HQL的查询（单表查询）
 *     1）全表查询
 *     2）别名查询
```

```
 *    3）条件查询
 *    4）具名查询
 *    5）分页查询
 *    6）查询排序
 *    7）聚合查询
 *    8）投影查询
 */
public class HQLTest1 {
    /**
     * 全表查询
     */
    @Test
    public void test1(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        //注意:
        //1、不能写 select * from Order
        //2、Order是对象名，不是表名
        Query<Order> query = session.createQuery("from Order",
Order.class);
        List<Order> list = query.list();
        for (Order order : list) {
            //为了避免空指针和内存溢出，toString打印时暂时不要打印order中的customer
            System.out.println(order);
        }

        tx.commit();
        session.close();
    }

    /**
     * 别名查询
     */
    @Test
    public void test2(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Query<Order> query = session.createQuery("select o from Order o",
Order.class);
        List<Order> list = query.list();
        for (Order order : list) {
            System.out.println(order);
        }

        tx.commit();
        session.close();
    }
```

```
    /**
     * 条件查询
     */
    @Test
    public void test31(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Query<Order> query = session.createQuery("from Order where orderno
= '201709070001'", Order.class);
        List<Order> list = query.list();
        for (Order order : list) {
            System.out.println(order);
        }

        tx.commit();
        session.close();
    }

    /**
     * 条件查询2
     */
    @Test
    public void test32(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Query<Order> query = session.createQuery("from Order where orderno
= ?", Order.class);
        query.setParameter(0, "201709070001");
        List<Order> list = query.list();
        for (Order order : list) {
            System.out.println(order);
        }

        tx.commit();
        session.close();
    }

    /**
     * 具名查询
     */
    @Test
    public void test4(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
```

```
        Query<Order> query = session.createQuery("from Order where orderno
= :orderno", Order.class);
        query.setParameter("orderno", "201709070001");
        List<Order> list = query.list();
        for (Order order : list) {
            System.out.println(order);
        }

        tx.commit();
        session.close();
    }

    /**
     * 分页查询
     */
    @Test
    public void test5(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Query<Order> query = session.createQuery("from Order",
Order.class);
        //设置起始行，从0开始
        query.setFirstResult(2);
        //设置查询行数
        query.setMaxResults(2);
        List<Order> list = query.list();
        for (Order order : list) {
            System.out.println(order);
        }

        tx.commit();
        session.close();
    }

    /**
     * 查询排序
     */
    @Test
    public void test6(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Query<Order> query = session.createQuery("from Order order by id
desc", Order.class);
        List<Order> list = query.list();
        for (Order order : list) {
            System.out.println(order);
        }
```

```
        tx.commit();
        session.close();
    }

    /**
     * 聚合查询：同样也支持max、min、avg、sum等聚合函数
     */
    @Test
    public void test7(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Query<Long> query = session.createQuery("select count(*) from
Order", Long.class);

        /*List<Long> list = query.list();
        Long count = list.get(0);
        System.out.println(count);*/
        Long count = query.uniqueResult();
        System.out.println(count);

        tx.commit();
        session.close();
    }

    /**
     * 投影查询（查询局部字段）
     */
    @Test
    public void test81(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Query<Object[]> query = session.createQuery("select
orderno,productName from Order", Object[].class);
        List<Object[]> list = query.list();
        for (Object[] objects : list) {
            for (Object object : objects) {
                System.out.print(object+"\t");
            }
            System.out.println();
        }

        tx.commit();
        session.close();
    }

    /**
```

```
     * 投影查询2（查询局部字段）
     */
    @Test
    public void test82(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        //注意创建对应的两个参数的构造方法
        Query<Order> query = session.createQuery("select new
com.qfedu.hibernate.pojo.Order(orderno,productName) from Order",
Order.class);
        List<Order> list = query.list();
        for (Order order : list) {
            System.out.println(order);
        }

        tx.commit();
        session.close();
    }
}
```

13.2 HQL多表查询

13.2.1 多表查询分类

1）内连接查询：`inner join`

2）左连接查询：`left join`

3）右连接查询：`right join`

13.2.2 HQL多表查询

需求: 同时显示客户名称和订单名称

```
/**
 * 演示HQL的查询（多表查询）
 *    1）内连接查询
 *    2）左连接查询
 *    3）右连接查询
 *
 */
public class HQLTest2 {

    /**
     * 内连接查询
     * 效果：只会显示满足条件的数据
     */
    @Test
    public void test1(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        //需求：显示客户名称和订单产品名称
        Query<Object[]> query = session.createQuery("select
c.name,o.productName from Customer c inner join c.orders o",
Object[].class);
        List<Object[]> list = query.list();
        for (Object[] objects : list) {
            for (Object object : objects) {
                System.out.print(object+"\t");
            }
            System.out.println();
        }
        tx.commit();
        session.close();
    }

    /**
     * 左连接查询
     * 效果：左边的数据全部显示
     */
    @Test
    public void test2(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
```

```
        //需求：显示客户名称和订单产品名称
        Query<Object[]> query = session.createQuery("select
c.name,o.productName from Customer c left join c.orders o",
Object[].class);
        List<Object[]> list = query.list();
        for (Object[] objects : list) {
            for (Object object : objects) {
                System.out.print(object+"\t");
            }
            System.out.println();
        }
        tx.commit();
        session.close();
    }

    /**
     * 右连接查询
     * 效果：右边的数据全部显示（和上个测试用例的效果一致）
     */
    @Test
    public void test3(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        //需求：显示客户名称和订单产品名称
        Query<Object[]> query = session.createQuery("select
c.name,o.productName from Order o right join o.customer c",
Object[].class);
        List<Object[]> list = query.list();
        for (Object[] objects : list) {
            for (Object object : objects) {
                System.out.print(object+"\t");
            }
            System.out.println();
        }
        tx.commit();
        session.close();
    }
}
```

第十四章 QBC查询

14.1 QBC简介

QBC(Query By Criteria) API提供了检索对象的另一种方式，它主要由Criteria接口、Criterion接口和Expresson类组成，它支持在运行时动态生成查询语句

使用 `Criteria` 对象进行查询

特点：面向对象方式的查询

注意：5.2版本以后被废弃，推荐使用 JPA Criteria

14.2 QBC使用

```
/**
 * 演示Criteria的查询（单表查询）
 *    1）全表查询
 *    2）条件查询
 *    3）分页查询
 *    4）查询排序
 *    5）聚合查询
 *    6）投影查询
 *
 *    //@deprecated (since 5.2) for Session, use the JPA Criteria
 */
public class QBCTest {
    /**
     * 全表查询
     */
    @Test
    public void test1(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        Criteria ce = session.createCriteria(Customer.class);
        List<Customer> list = ce.list();
        for (Customer customer : list) {
            System.out.println(customer.getName());
        }
        tx.commit();
        session.close();
    }

    /**
     * 条件查询
     */
    @Test
    public void test21(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        Criteria ce = session.createCriteria(Order.class);
        //添加查询条件 orderno = '201709070002'
        ce.add( Restrictions.eq("orderno", "201709070002")  );
```

```
        List<Order> list = ce.list();
        for (Order order : list) {
            System.out.println(order.getProductName());
        }
        tx.commit();
        session.close();
    }

    /**
     * 条件查询2(多条件)
     */
    @Test
    public void test22(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        Criteria ce = session.createCriteria(Order.class);
        //添加查询条件   orderno like '%2017%' and productName like
'%JavaWeb%'
        ce.add( Restrictions.and(  Restrictions.like("orderno", "%2017%") ,
 Restrictions.like("productName", "%JavaWeb%")  )  );
        List<Order> list = ce.list();
        for (Order order : list) {
            System.out.println(order.getProductName());
        }
        tx.commit();
        session.close();
    }

    /**
     * 分页查询
     */
    @Test
    public void test3(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        Criteria ce = session.createCriteria(Order.class);
        //分页查询
        ce.setFirstResult(2);//起始行
        ce.setMaxResults(2);//查询行数
        List<Order> list = ce.list();
        for (Order order : list) {
            System.out.println(order.getProductName());
        }
        tx.commit();
        session.close();
    }

    /**
     * 查询排序
```

```
     */
    @Test
    public void test4(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        Criteria ce = session.createCriteria(Order.class);
        //排序  order by id desc
        //因为我们的项目中也定义了Order类，所以这里的Order使用全名
        ce.addOrder(org.hibernate.criterion.Order.desc("id"));
        List<Order> list = ce.list();
        for (Order order : list) {
            System.out.println(order);
        }
        tx.commit();
        session.close();
    }

    /**
     * 聚合查询
     */
    @Test
    public void test5(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        Criteria ce = session.createCriteria(Order.class);
        //查询总记录数  select count(id)
        //ce.setProjection(Projections.rowCount());
        ce.setProjection(Projections.count("id"));
        Long count = (Long)ce.uniqueResult();

        //查询id的最大值
        //ce.setProjection(Projections.max("id"));
        //Integer count = (Integer)ce.uniqueResult();
        System.out.println(count);
        tx.commit();
        session.close();
    }

    /**
     * 投影查询
     */
    @Test
    public void test6(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        Criteria ce = session.createCriteria(Order.class);
        //投影操作
        ProjectionList pList = Projections.projectionList();
        pList.add(Property.forName("orderno"));
```

```
        pList.add(Property.forName("productName"));
        ce.setProjection(pList);
        List<Object[]> list = ce.list();
        for (Object[] objects : list) {
            for (Object object : objects) {
                System.out.print(object+"\t");
            }
            System.out.println();
        }
        tx.commit();
        session.close();
    }
}
```

第十五章 本地SQL查询

本地sql查询可以直接执行 SQL 语句

- 5.2以后推荐使用 `createNativeQuery`
- 5.2之前使用 `createSQLQuery`

15.1 Hibernate使用SQL语句

```
/**
 * 演示本地 SQL 的查询
 */
public class SQLTest {
    /**
     * 5.2开始支持
```

```
     */
    @Test
    public void test1() {
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        NativeQuery<Order> sqlQuery = session.createNativeQuery("select *
from t_order", Order.class);
        List<Order> list = sqlQuery.list();
        for (Order order : list) {
            System.out.println(order);
        }
        tx.commit();
        session.close();
    }

    /**
     *     5.2之前的用法
     *  以JavaBean对象封装
     */
    @Test
    public void test2() {
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        SQLQuery sqlQuery = session.createSQLQuery("select * from
t_order");
        sqlQuery.addEntity(Order.class);
        List<Order> list = sqlQuery.list();
        for (Order order : list) {
            System.out.println(order);
        }
        tx.commit();
        session.close();
    }

    /**
     *     5.2之前的用法
     *  以对象数组封装
     */
    @Test
    public void test3() {
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();
        SQLQuery sqlQuery = session.createSQLQuery("select * from
t_order");
        List<Object[]> list = sqlQuery.list();
        for(Object[] order : list) {
            for(Object column : order) {
                System.out.print(column);
                System.out.print(" ");
```

```
            }
            System.out.println();
        }
        tx.commit();
        session.close();
    }
}
```

第十六章 延迟加载策略

延迟加载是为了减少程序和数据库的访问次数，提供程序的执行性能。

延迟加载的执行机制：

1）在查询一个对象的时候，不会到数据库查询对象的属性或者其关联的数据

2）在需要使用到对象的属性或关联数据的才会去查询数据库！

按需加载！

16.1 类级别延迟加载

主要是针对属性

```
public class LazyLoadingTest {
    /**
     * 类级别 延迟加载
     */
    @Test
    public void test1(){

        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        //get()：get方法不支持类级别的延迟加载
        /*
        Customer cust = session.get(Customer.class, 1);//debug：此行查询了数据
库
        System.out.println(cust.getName());
        */

        //load()：load方法支持类级别的延迟加载
        Customer cust = session.load(Customer.class, 1);//debug：此行没有查询
数据库
        System.out.println(cust.getName());//debug：此行查询了数据库
        tx.commit();
        session.close();
    }
}
```

结论：

load()：只有 load 方法才支持类级别的延迟加载

get()：get 方法不支持类级别的延迟加载

使用 load() 方法的默认延迟加载策略是延迟加载，可以在配置文件中修改延迟加载策略

```
<class name="Customer" table="t_customer" lazy="false">
```

16.2 关联级别

注意: 测试前先删除前面配置的lazy = "false"

以一对多为例

1）一方：

测试：

```
/**
     * 关联级别 延迟加载（一方：<set/>）
     * 修改一对多的延迟加载配置：<set name="orders" inverse="true" lazy="false">
     */
    @Test
    public void test2(){

        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Customer cust = session.get(Customer.class, 1L);
        //关联订单
        System.out.println(cust.getOrders().size()); //延迟加载的

        tx.commit();
        session.close();

    }
```

结论：

类级别默认使用延迟加载策略，如果不想使用延迟加载策略，那么可以在配置文件中修改延迟加载策略：Customer.hbm.xml

```
<set name="orders" cascade="all" inverse="true" lazy="false">
```

多方

测试：

```
 /**
     * 关联级别 延迟加载（多对一：<many-to-one/>）
     * 修改多对一延迟加载配置：<many-to-one name="customer" class="Customer"
column="customer_id" cascade="all" lazy="false"/>
     */
    @Test
    public void test3(){

        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Order order = session.get(Order.class, 1L);
        System.out.println(order.getCustomer().getName()); // 延迟加载

        tx.commit();
        session.close();
    }
```

结论：

类级别默认使用延迟加载策略，如果不想使用延迟加载策略，那么可以在配置文件中修改延迟加载策略：Order.hbm.xml

```
<many-to-one name="customer" class="Customer" column="customer_id"
cascade="all" lazy="false"/>
```

第十七章 抓取策略

抓取策略，是为了改变 SQL 语句查询的方式，从而提高 SQL 语句查询的效率（优 化 SQL 语句）

可以设置以下三个值：

`fetch="select(默认值)|join|subselect"`

17.1 查询一方单条

在Customer.hbm.xml中配置 `fetch="select"` 会执行两条sql

```
<set name="orders" cascade="all" inverse="true" fetch="select">
```

配置 `fetch="join"` 会执行一条左外连接的sql语句。

注意：如果配置了join，那么延迟加载就会失效！

```
<set name="orders" cascade="all" inverse="true" fetch="join">
```

测试

```
public class FetchingStrategyTest {
    /**
     * 一方：<set/>
     *  fetch="select" ：默认情况，执行两条sql语句
     *  fetch="join"：  把两条sql合并成左外连接查询（效率更高）
     *  注意：如果配置了join，那么延迟加载就会失效！
     */
    @Test
    public void test1(){

        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Customer cust = session.get(Customer.class, 1L);
        System.out.println(cust.getOrders());

        tx.commit();
        session.close();

    }
}
```

17.2 查询一方多条

此时无论设置 `fetch="select"` 还是 `fetch="join"` ，都会执行多条sql语句（n+1）

可以设置 `fetch="subselect"` ，会执行一条带有子查询的sql语句：

```
<set name="orders" cascade="all" inverse="true" fetch="subselect">
```

测试

```
/**
     * 一方：<set/>
     *   需求：在查询多个一方（客户列表）的数据，关联查询多方（订单）的数据
     *   如果fetch的配置是select或join的时候，一共发出n+1条sql语句
     *   fetch="subselect"：  使用子查询进行关联查询
     */
    @Test
    public void test2(){

        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Query<Customer> query = session.createQuery("from Customer",
Customer.class);
        List<Customer> list = query.list();
        for (Customer customer : list) {
            System.out.println(customer.getOrders().size());
        }

        tx.commit();
        session.close();

    }
```

17.3 查询多方的记录

Order.hbm.xml中配置 `fetch="select"` 会执行两条sql

```
<many-to-one name="customer" class="Customer" column="customer_id"
cascade="all" fetch="select"/>
```

配置 `fetch="join"` 会执行一条左外连接的sql语句。

注意：如果配置了join，那么延迟加载就会失效！

```
<many-to-one name="customer" class="Customer" column="customer_id"
cascade="all" fetch="join"/>
```

测试:

```
/**
     * 多方：<many-to-one/>
     *   fetch="select" ：默认情况，执行两条sql语句（支持延迟加载）
     *   fetch="join"：  把两条sql合并成左外连接查询（效率更高）
     *   注意：如果配置了join，那么延迟加载就会失效！
     */
    @Test
    public void test3(){

        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Order order = session.get(Order.class, 1);
        System.out.println(order.getCustomer());

        tx.commit();
        session.close();

    }
```

第十八章 整合连接池

18.1 导入 c3p0的jar包

在hibernate解压目录下 lib/optional/c3p0 中可以找到整合相关的包

如果maven项目,pom配置如下

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-c3p0</artifactId>
   <version>5.2.10.Final</version>
</dependency>
```

18.2 Hibernate.cfg.xml整合

在连接数据库参数的后面添加:

```
<!-- 整合c3p0：必要 -->
<property
name="hibernate.connection.provider_class">org.hibernate.c3p0.internal.C3P0
ConnectionProvider</property>
<!-- c3p0详细配置：可选 -->
<property name="c3p0.min_size">10</property>
<property name="c3p0.max_size">20</property>
```

18.3 测试是否启动连接池

```
public class PoolTest {
    /**
     * 演示连接池的整合
     */
    @Test
    public void test1(){

        Session session = HibernateUtil.openSession();
        session.doWork(new Work(){
            @Override
            public void execute(Connection connection) throws SQLException
{
                System.out.println(connection);
            }

        });

        session.close();
    }
}
```

第十九章 二级缓存和查询缓存

19.1 二级缓存

19.1.1 二级缓存概述

Hibernate 的一级缓存：就是 Session 对象的缓存，而 Session 对象在每次操作之 后都会关闭，那么一级缓存就丢失！

结论：一级缓存只用于一次业务操作内的缓存。

Hibernate 的二级缓存：就是 SessionFactory 的缓存，二级缓存和 SessionFactory 对象的生命周期是一致的，SessionFactory 不消耗，那么二级缓存的数据就不会 丢失！

结论：二级缓存可以用于多次业务操作的缓存。

注意的问题：

1）Hibernate 一级缓存默认是开启的，而且无法关闭。

2）Hibernate 二级缓存默认是关闭的，如果使用需要开启，而且需要引入第三 方的缓存工具，例如 EhCache 等。

20.1.2 二级缓存使用

添加二级缓存jar包

jar包位置: hibernate解压目录下 lib/optional/ehcache下找到相关包!

maven项目,pom文件添加

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-ehcache</artifactId>
   <version>5.2.10.Final</version>
</dependency>
```

配置二级缓存

在hibernate.cfg.xml中配置以下节点:

1. `property` 节点 要放在 `mapping` 节点的上方

```
 <!-- 开启 Hibernate 的二级缓存 -->
 <property
name="hibernate.cache.use_second_level_cache">true</property>
 <!-- 引入 EhCache 的工具 -->
 <property name="hibernate.cache.region.factory_class">
        org.hibernate.cache.ehcache.EhCacheRegionFactory
 </property>
```

2. `class-cache` 节点 要放在 `mapping` 节点的下方

```
<!-- 需要缓存哪个类 -->
   <class-cache usage="read-only"
class="com.qfedu.hibernate.pojo.Customer"/>
```

二级缓存测试

使用二级缓存只打印1条sql，不使用二级缓存会打印2条sql

```
package com.qfedu.hibernate.test;
import org.hibernate.Session;
import org.junit.Test;
import com.qfedu.hibernate.pojo.Customer;
import com.qfedu.hibernate.utils.HibernateUtil;
public class CacheLevelTwoTest {
    /**
     * 演示二级缓存
     */
    @Test
    public void test1(){

        Session session = HibernateUtil.openSession();

        //第1次操作
        Customer cust = session.get(Customer.class, 1L);
        System.out.println(cust.getName());

        //关闭session
        session.close();

        //第2次操作
        session = HibernateUtil.openSession();
        cust = session.get(Customer.class, 1L);
        System.out.println(cust.getName());

        session.close();
    }
}
```

19.2 查询缓存

19.2.1 查询缓概述

hibernate中提供了一级缓存、二级缓存用来提高单个记录查询的效率。查询缓存是为了提高批量查询的效率。何为批量查询呢？就是使用HQL/SQL语句，从数据库中查询多条满足条件的记录。也就是说：load和get缓存中的key是实体对象的主键值；查询缓存中的key则是hql/sql语句。

19.2.2 查询缓存使用

在hibernate4.1.6版本中，为了使用查询缓存，需要进行两步操作：

1.在hibernate.cfg.xml中开启查询缓存，因为hibernate默认情况下会关闭查询缓存。配置方式如下：

```
<property name="hibernate.cache.use_query_cache">true</property>
```

2.在hibernate的Query对象中，通过代码设置开启查询缓存

```
query.setCacheable(true);
```

19.2.3 查询缓存总结

默认hibernate不会开启查询缓存，这是因为查询缓存只有在hql/hql语句语义完全一致的时候，才能命中。而实际查询场景下，查询条件、分页、排序等构成的复杂查询sql语句很难完全一致。可能是hibernate觉得命中率低，所以默认关闭了查询缓存。我们可以根据实际使用情况，决定是否开启查询缓存，唯一的原则就是命中率要尽可能的高。如果针对A表的查询，查询sql语句基本都是完全一致的情况，就可以针对A使用查询缓存；如果B表的查询条件经常变化，很难命中，那么就不要对B表使用查询缓存。这可能就是hibernate使用查询缓存的时候，既要在hibernate.cfg.xml中进行配置，也需要query.setCacheable(true)的原因。

查询缓存只对list有用，对iterate方式无用。iterate不会读也不会写查询缓存，list会读也会写查询缓存。查询缓存中的key是sql语句（这些sql语句会被hibernate解析，保证语义相同的sql，能够命中查询缓存），缓存的value是记录的主键值。

总结

作业

参考数据库文件

使用Hibernate的查询方式完成以下查询：（下面的SQL为原生的SQL作为参考）

1、 1、查询所有学生信息（显示班级名称、按年龄由大到小排序）。

```
SELECT DISTINCT stuId, stuName, stuSex, stuAge, claName FROM allscore ORDER
BY stuAge DESC;
SELECT s.*, c.claName from student s INNER JOIN classes c on s.claId =
c.claId ORDER BY stuAge DESC;
```

2、查询期中考试每个班有多少个人次缺考。

```
SELECT claName, stuName, COUNT(stuId) FROM allscore WHERE score = 0 and
examName LIKE '%2016%期中%' GROUP BY claId, stuId;
SELECT c.claName, s.stuName, COUNT(s.stuId) FROM
student s INNER JOIN classes c on s.claId = c.claId
INNER JOIN score r on s.stuId = r.stuId
INNER JOIN exam e on e.examId = r.examId
 WHERE r.score = 0 and e.examName
LIKE '%2016%期中%' GROUP BY c.claId, s.stuId;
```

3、查询期末考试1班每门课程平均成绩。

```
SELECT subName, AVG(score) from allscore WHERE claName LIKE '%1601%' AND
examName LIKE '%2016%期末%' GROUP BY subId;
```

4、查询期中考试2班拖后腿的学生。（低于总平均成绩）。

```
SELECT stuName, avg(score) FROM allscore WHERE claName LIKE '%1602%' AND
examName LIKE '%2016%期中%' GROUP BY stuId HAVING avg(score) < (SELECT
avg(score) FROM allscore WHERE claName LIKE '%1602%' AND examName LIKE
'%2016%期中%');
```

5、查询期末考试每个学生的总分。（显示班级名称）。

```
SELECT stuName, claName, sum(score) from allscore WHERE examName LIKE
'%2016%期末%' GROUP BY stuId;
```

6、查询期末考试每个班最高总分和最低总分的学生名称和各科成绩。

```
SELECT stuName, claName, score, subName FROM allscore
WHERE examName LIKE '%2016%期末%' AND stuId =
(SELECT stuId from allscore WHERE
examName LIKE '%2016%期末%' and claName LIKE '%1601%' GROUP BY stuId
ORDER BY sum(score) DESC LIMIT 0, 1);
```

7、查询期中考试某个学生低于班级此科目平均分的科目成绩。

```
SELECT stuName, subName, score from allscore WHERE examName LIKE '%期中%'
AND subId = 1 AND stuName = '詹云久' AND score < (SELECT avg(score) from
allscore WHERE examName LIKE '%期中%' AND subId = 1 AND claId = 1);
```

8、查询期中考试每个班每个科目有多少个人不及格。

```
SELECT claName, subName, COUNT(stuId) from allscore WHERE examName LIKE '%期
中%' AND score < 60 GROUP BY claId, subId;
```

面试题

1. Hibernate查询的几种方式
2. 延迟加载策略
3. 连接池在Hibernate中的使用
4. Hibernate中缓存的分类和作用