Hibernate第三天：

1. 对象状态

2. session缓存

3. lazy懒加载

4. 映射

一对一对映射

组件/继承映射

目标：

一、hibernate查询

二、hibernate对连接池的支持

三、二级缓存

四、Hibernate与Struts小案例（项目中session的管理方式）

# 一、hibernate查询

## 1. 查询概述

1. Get/load主键查询
2. 对象导航查询
3. **HQL查询， Hibernate Query language hibernate 提供的面向对象的查询语言。**
4. Criteria 查询， 完全面向对象的查询（Query By Criteria ,QBC）
5. SQLQuery， 本地SQL查询

缺点：不能跨数据库平台： 如果该了数据库，sql语句有肯能要改

使用场景： 对于复杂sql，hql实现不了的情况，可以使用本地sql查询。

### HQL查询

|  |
| --- |
| **public** **class** App {    **private** **static** SessionFactory *sf*;  **static** {  *sf* = **new** Configuration()  .configure()  .addClass(Dept.**class**)  .addClass(Employee.**class**) // 测试时候使用  .buildSessionFactory();  }  /\*  \* 1) Get/load主键查询  2) 对象导航查询  3) HQL查询， Hibernate Query language hibernate 提供的面向对象的查询语言。  4) Criteria 查询， 完全面向对象的查询（Query By Criteria ,QBC）  5) SQLQuery， 本地SQL查询  \*/  @Test  **public** **void** all() {    Session session = *sf*.openSession();  session.beginTransaction();    //1) 主键查询  // Dept dept = (Dept) session.get(Dept.class, 12);  // Dept dept = (Dept) session.load(Dept.class, 12);    //2) 对象导航查询  // Dept dept = (Dept) session.get(Dept.class, 12);  // System.out.println(dept.getDeptName());  // System.out.println(dept.getEmps());    // 3) HQL查询  // 注意：使用hql查询的时候 auto-import="true" 要设置true，  // 如果是false，写hql的时候，要指定类的全名  // Query q = session.createQuery("from Dept");  // System.out.println(q.list());    // a. 查询全部列  // Query q = session.createQuery("from Dept"); //OK  // Query q = session.createQuery("select \* from Dept"); //NOK, 错误，不支持\*  // Query q = session.createQuery("select d from Dept d"); // OK  // System.out.println(q.list());  // b. 查询指定的列 【返回对象数据Object[] 】  // Query q = session.createQuery("select d.deptId,d.deptName from Dept d");  // System.out.println(q.list());    // c. 查询指定的列, 自动封装为对象 【必须要提供带参数构造器】  // Query q = session.createQuery("select new Dept(d.deptId,d.deptName) from Dept d");  // System.out.println(q.list());    // d. 条件查询: 一个条件/多个条件and or/between and/模糊查询  // 条件查询： 占位符  // Query q = session.createQuery("from Dept d where deptName=?");  // q.setString(0, "财务部");  // q.setParameter(0, "财务部");  // System.out.println(q.list());    // 条件查询： 命名参数  // Query q = session.createQuery("from Dept d where deptId=:myId or deptName=:name");  // q.setParameter("myId", 12);  // q.setParameter("name", "财务部");  // System.out.println(q.list());    // 范围  // Query q = session.createQuery("from Dept d where deptId between ? and ?");  // q.setParameter(0, 1);  // q.setParameter(1, 20);  // System.out.println(q.list());    // 模糊  // Query q = session.createQuery("from Dept d where deptName like ?");  // q.setString(0, "%部%");  // System.out.println(q.list());    // e. 聚合函数统计  // Query q = session.createQuery("select count(\*) from Dept");  // Long num = (Long) q.uniqueResult();  // System.out.println(num);    // f. 分组查询  //-- 统计t\_employee表中，每个部门的人数  //数据库写法：SELECT dept\_id,COUNT(\*) FROM t\_employee GROUP BY dept\_id;  // HQL写法  // Query q = session.createQuery("select e.dept, count(\*) from Employee e group by e.dept");  // System.out.println(q.list());          session.getTransaction().commit();  session.close();  }    // g. 连接查询  @Test  **public** **void** join() {    Session session = *sf*.openSession();  session.beginTransaction();    //1) 内连接 【映射已经配置好了关系，关联的时候，直接写对象的属性即可】  // Query q = session.createQuery("from Dept d inner join d.emps");    //2) 左外连接  // Query q = session.createQuery("from Dept d left join d.emps");  //3) 右外连接  Query q = session.createQuery("from Employee e right join e.dept");  q.list();    session.getTransaction().commit();  session.close();  }    // g. 连接查询 - 迫切连接  @Test  **public** **void** fetch() {    Session session = *sf*.openSession();  session.beginTransaction();    //1) 迫切内连接 【使用fetch, 会把右表的数据，填充到左表对象中！】  // Query q = session.createQuery("from Dept d inner join fetch d.emps");  // q.list();    //2) 迫切左外连接  Query q = session.createQuery("from Dept d left join fetch d.emps");  q.list();    session.getTransaction().commit();  session.close();  }    // HQL查询优化  @Test  **public** **void** hql\_other() {  Session session = *sf*.openSession();  session.beginTransaction();  // HQL写死  // Query q = session.createQuery("from Dept d where deptId < 10 ");    // HQL 放到映射文件中  Query q = session.getNamedQuery("getAllDept");  q.setParameter(0, 10);  System.*out*.println(q.list());    session.getTransaction().commit();  session.close();  }  } |

### Criteria 查询

|  |
| --- |
| //4) Criteria 查询，  @Test  **public** **void** criteria() {    Session session = *sf*.openSession();  session.beginTransaction();    Criteria criteria = session.createCriteria(Employee.**class**);  // 构建条件  criteria.add(Restrictions.*eq*("empId", 12));  // criteria.add(Restrictions.idEq(12)); // 主键查询    System.*out*.println(criteria.list());      session.getTransaction().commit();  session.close();  } |

### SQLQuery， 本地SQL查询

|  |
| --- |
| // 5) SQLQuery， 本地SQL查询  // 不能跨数据库平台： 如果该了数据库，sql语句有肯能要改。  @Test  **public** **void** sql() {    Session session = *sf*.openSession();  session.beginTransaction();    SQLQuery q = session.createSQLQuery("SELECT \* FROM t\_Dept limit 5;")  .addEntity(Dept.**class**); // 也可以自动封装  System.*out*.println(q.list());    session.getTransaction().commit();  session.close();  } |

## 2. 分页查询

分页SQL：

先查询总记录数，再分页查询。

|  |
| --- |
| // 分页查询  @Test  **public** **void** all() {    Session session = *sf*.openSession();  session.beginTransaction();    Query q = session.createQuery("from Employee");    // zong记录数  ScrollableResults scroll = q.scroll(); // 得到滚动的结果集  scroll.last(); // 滚动到最后一行  **int** totalCount = scroll.getRowNumber() + 1;// 得到滚到的记录数，即总记录数    // 设置分页参数  q.setFirstResult(0);  q.setMaxResults(3);    // 查询  System.*out*.println(q.list());  System.*out*.println("总记录数：" + totalCount);    session.getTransaction().commit();  session.close();  } |

# 二、hibernate对连接池的支持

连接池，

作用： 管理连接；提升连接的利用效率！

常用的连接池： **C3P0连接池**

**Hibernate 自带的也有一个连接池，且对C3P0连接池也有支持！**

**Hbm 自带连接池：**

**只维护一个连接，比较简陋。**

**可以查看hibernate.properties文件查看连接池详细配置:**

#################################

### Hibernate Connection Pool ###

#################################

hibernate.connection.pool\_size 1 【**Hbm 自带连接池： 只有一个连接**】

###########################

### C3P0 Connection Pool### 【Hbm对C3P0连接池支持】

###########################

#hibernate.c3p0.max\_size 2 最大连接数

#hibernate.c3p0.min\_size 2 最小连接数

#hibernate.c3p0.timeout 5000 超时时间

#hibernate.c3p0.max\_statements 100 最大执行的命令的个数

#hibernate.c3p0.idle\_test\_period 3000 空闲测试时间

#hibernate.c3p0.acquire\_increment 2 连接不够用的时候， 每次增加的连接数

#hibernate.c3p0.validate false

【Hbm对C3P0连接池支持， 核心类】

告诉hib使用的是哪一个连接池技术。

#hibernate.connection.provider\_class org.hibernate.connection.C3P0ConnectionProvider

Hibernate.cfg.xml 中增加连接池相关配置：

|  |
| --- |
| <!-- 【连接池配置】 -->  <!-- 配置连接驱动管理类 -->  <property name=*"hibernate.connection.provider\_class"*>org.hibernate.connection.C3P0ConnectionProvider</property>  <!-- 配置连接池参数信息 -->  <property name=*"hibernate.c3p0.min\_size"*>2</property>  <property name=*"hibernate.c3p0.max\_size"*>4</property>  <property name=*"hibernate.c3p0.timeout"*>5000</property>  <property name=*"hibernate.c3p0.max\_statements"*>10</property>  <property name=*"hibernate.c3p0.idle\_test\_period"*>30000</property>  <property name=*"hibernate.c3p0.acquire\_increment"*>2</property> |

# 三、二级缓存

Hibernate提供的缓存

有一级缓存、二级缓存。 目的是为了减少对数据库的访问次数，提升程序执行效率！

一级缓存：

基于Session的缓存，缓存内容只在当前session有效，session关闭，缓存内容失效！

特点：

作用范围较小！ 缓存的事件短。

缓存效果不明显。

## 概述

二级缓存：

Hibernate提供了基于应用程序级别的缓存， 可以跨多个session，即不同的session都可以访问缓存数据。 这个换存也叫二级缓存。

Hibernate提供的二级缓存有默认的实现，且是一种可插配的缓存框架！如果用户想用二级缓存，只需要在hibernate.cfg.xml中配置即可； 不想用，直接移除，不影响代码。

如果用户觉得hibernate提供的框架框架不好用，自己可以换其他的缓存框架或自己实现缓存框架都可以。

## 使用二级缓存

查看hibernate.properties配置文件，二级缓存如何配置？

##########################

### Second-level Cache ###

##########################

#hibernate.cache.use\_second\_level\_cache false【二级缓存默认不开启，需要手动开启】

#hibernate.cache.use\_query\_cache true 【开启查询缓存】

## choose a cache implementation 【二级缓存框架的实现】

#hibernate.cache.provider\_class org.hibernate.cache.EhCacheProvider

#hibernate.cache.provider\_class org.hibernate.cache.EmptyCacheProvider

hibernate.cache.provider\_class org.hibernate.cache.HashtableCacheProvider 默认实现

#hibernate.cache.provider\_class org.hibernate.cache.TreeCacheProvider

#hibernate.cache.provider\_class org.hibernate.cache.OSCacheProvider

#hibernate.cache.provider\_class org.hibernate.cache.SwarmCacheProvider

二级缓存，使用步骤

1) 开启二级缓存

2)指定缓存框架

3)指定那些类加入二级缓存

4)测试

测试二级缓存！

## 缓存策略

<class-cache usage="read-only"/> 放入二级缓存的对象，只读;

<class-cache usage="nonstrict-read-write"/> 非严格的读写

<class-cache usage="read-write"/> 读写； 放入二级缓存的对象可以读、写；

<class-cache usage="transactional"/> (基于事务的策略)

## 集合缓存

<!-- 集合缓存[集合缓存的元素对象，也加加入二级缓存] -->

<collection-cache

usage=*"read-write"* collection=*"cn.itcast.b\_second\_cache.Dept.emps"*/>

## 查询缓存

list() 默认情况只会放入缓存，不会从一级缓存中取！

使用查询缓存，可以让list()查询从二级缓存中取！

完整案例：

|  |
| --- |
| Hibernate.cfg.xml |
| <!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 【二级缓存配置】\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->  <!-- a. 开启二级缓存 -->  <property name=*"hibernate.cache.use\_second\_level\_cache"*>true</property>  <!-- b. 指定使用哪一个缓存框架(默认提供的) -->  <property name=*"hibernate.cache.provider\_class"*>org.hibernate.cache.HashtableCacheProvider</property>  <!-- 开启查询缓存 -->  <property name=*"hibernate.cache.use\_query\_cache"*>true</property>  <!-- c. 指定哪一些类，需要加入二级缓存 -->  <class-cache usage=*"read-write"* class=*"cn.itcast.b\_second\_cache.Dept"*/>  <class-cache usage=*"read-only"* class=*"cn.itcast.b\_second\_cache.Employee"*/>  <!-- 集合缓存[集合缓存的元素对象，也加加入二级缓存] -->  <collection-cache usage=*"read-write"* collection=*"cn.itcast.b\_second\_cache.Dept.emps"*/> |
| App 测试类 |
| **public** **class** App {    **private** **static** SessionFactory *sf*;  **static** {  *sf* = **new** Configuration()  .configure()  .addClass(Dept.**class**)  .addClass(Employee.**class**) // 测试时候使用  .buildSessionFactory();  }  // 1. 测试二级缓存的使用  // 没有/有用 二级缓存  @Test  **public** **void** testCache() {  Session session1 = *sf*.openSession();  session1.beginTransaction();  // a. 查询一次  Dept dept = (Dept) session1.get(Dept.**class**, 10);  dept.getEmps().size();// 集合  session1.getTransaction().commit();  session1.close();    System.*out*.println("------");    // 第二个session  Session session2 = *sf*.openSession();  session2.beginTransaction();  // a. 查询一次  dept = (Dept) session2.get(Dept.**class**, 10); // 二级缓存配置好； 这里不查询数据库  dept.getEmps().size();    session2.getTransaction().commit();  session2.close();  }      @Test  **public** **void** listCache() {  Session session1 = *sf*.openSession();  session1.beginTransaction();  // HQL查询 【setCacheable 指定从二级缓存找，或者是放入二级缓存】  Query q = session1.createQuery("from Dept").setCacheable(**true**);  System.*out*.println(q.list());  session1.getTransaction().commit();  session1.close();      Session session2 = *sf*.openSession();  session2.beginTransaction();  q = session2.createQuery("from Dept").setCacheable(**true**);  System.*out*.println(q.list()); // 不查询数据库： 需要开启查询缓存  session2.getTransaction().commit();  session2.close();  }  } |

# 四、项目中session的管理方式

Session的创建方式：

|  |
| --- |
| @Test  **public** **void** testSession() **throws** Exception {  //openSession: 创建Session, 每次都会创建一个新的session  Session session1 = *sf*.openSession();  Session session2 = *sf*.openSession();  System.*out*.println(session1 == session2);  session1.close();  session2.close();    //getCurrentSession 创建或者获取session  // 线程的方式创建session  // 一定要配置：<property name="hibernate.current\_session\_context\_class">thread</property>  Session session3 = *sf*.getCurrentSession();// 创建session，绑定到线程  Session session4 = *sf*.getCurrentSession();// 从当前访问线程获取session  System.*out*.println(session3 == session4);    // 关闭 【以线程方式创建的session，可以不用关闭； 线程结束session自动关闭】  //session3.close();  //session4.close(); 报错，因为同一个session已经关闭了！  } |

# 四、项目应用

重点：

1. Struts与Hibernate一起使用完成案例开发！

2. Open Session In View 模式应用

**需求：**

显示部门信息，部门下的所有员工！

【要求：员工信息在jsp页面显示，使用懒加载完成！】

**数据库设计：**

T\_dept/ t\_employee

**项目框架搭建**

a. 引用的jar文件

struts相关jar文件

hibernate相关jar文件

c3p0连接池/数据库驱动包

b. 配置

struts.xml

hibernate.cfg.xml

web.xml 【struts核心过滤器】

c. 公用类