**Hibernate总结**

**hibernate 简介：**

　　hibernate是一个开源ORM(Object/Relationship Mipping)框架，它是对象关联关系映射的持久层框架，它对JDBC做了轻量级的封装，而我们java程序员可以使用面向对象的思想来操纵数据库。

**为什么要用hibernate(jdbc的缺点)：**

　　1、编程的时候很繁琐，用的try和catch比较多

　　2、jdbc没有做数据的缓存

　　3、没有做到面向对象编程

　　4、sql语句的跨平台性很差

**JDBC的优点：**

　　效率高！

**hibernate的优点：**

　　1、完全的面向对象编程

　　2、hibernate的缓存很牛的，一级缓存，二级缓存，查询缓存   重点

　　3、编程的时候就比较简单了

　　4、跨平台性很强

　　5、使用场合就是企业内部的系统

**hibernate的缺点：**

　　1、效率比较低

　　2、表中的数据如果在千万级别，则hibernate不适合

　　3、如果表与表之间的关系特别复杂，则hibernate也不适合

**Hibernate所需包说明：**

　　　　 说明：

           　　 javassist包是用来创建代理对象的

              　　代理对象的三种创建方式：

                 　　　　   1、jdkproxy

                　　　　    2、cglib

               　　　　     3、javassist

       　　    jta: Java Transaction API，是sun公司给分布式事务处理出来的规范

**hibernate.cfg.xml:**

　　　　主要的用途：

          　　　　告诉hibernate连接数据库的信息，用的什么样的数据库(方言)

          　　　　根据持久化类和映射文件生成表的策略

**五个核心接口：**

* Configuration：负责配置并启动hibernate，创建SessionFactory
* SessionFactory：负责初始化hibernate，创建session对象
  + 1、hibernate中的配置文件、映射文件、持久化类的信息都在sessionFactory中
  + 2、sessionFactory中存放的信息都是共享的信息
  + 3、sessionFactory本身就是线程安全的
  + 4、一个hibernate框架sessionFactory只有一个
  + 5、sessionFactory是一个重量级别的类

Session：负责被持久化对象CRUD操作

* + 1、得到了一个session，相当于打开了一次数据库的连接
  + 2、在hibernate中，对数据的crud操作都是由session来完成的

Transaction：负责事物相关的操作

* + hibernate中的事务默认不是自动提交的

　　设置了connection的setAutoCommit为false

      　　只有产生了连接，才能进行事务的操作。所以只有有了session以后，才能有transaction

* Query：负责执行各种数据库查询

**hibernate.cfg.xml文件的加载：**

1 <?xml version='1.0' encoding='utf-8'?>

2 <!DOCTYPE hibernate-configuration PUBLIC

3 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

4 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

5 <hibernate-configuration>

6 <!--

7 一个sessionFactory代表数据库的一个连接

8 -->

9 <session-factory>

10 <!-- 链接数据库的用户名 -->

11 <property name="connection.username">root</property>

12 <!-- 链接数据库的密码 -->

13 <property name="connection.password">root</property>

14 <!-- 链接数据库的驱动 -->

15 <property name="connection.driver\_class">

16 com.mysql.jdbc.Driver

17 </property>

18 <!-- 链接数据库的url -->

19 <property name="connection.url">

20 jdbc:mysql://localhost:3306/itheima12\_hibernate

21 </property>

22 <!--

23 方言

24 告诉hibernate使用什么样的数据库，hibernate就会在底层拼接什么样的sql语句

25 -->

26 <property name="dialect">

27 org.hibernate.dialect.MySQLDialect

28 </property>

29 <!--

30 根据持久化类生成表的策略

31 validate 通过映射文件检查持久化类与表的匹配

32 update 每次hibernate启动的时候，检查表是否存在，如果不存在，则创建，如果存在，则什么都不做了

33 create 每一次hibernate启动的时候，根据持久化类和映射文件生成表

34 create-drop

35 -->

36 <property name="hbm2ddl.auto">update</property>

37 <property name="show\_sql">true</property>

38 <mapping

39 resource="com/itheima12/hibernate/domain/Person.hbm.xml" />

40 <mapping

41 resource="com/itheima12/hibernate/utils/Person.hbm.xml" />

42 </session-factory>

43 </hibernate-configuration>

**hibernate.cfg.xml文件的加载：**

|  |  |
| --- | --- |
| 属性名字 | 含义 |
| hibernate.show\_sql | 是否把Hibernate运行时的SQL语句输出到控制台，编码阶段便于测试 |
| hibernate.format\_sql | 输出到控制台的语句是否进行排版，便于阅读，建议设置为true |
| hbm2ddl.auto | 可以帮助由java代码产生数据库脚本，进而生成具体的表结构(create|update|create drop|validate) |
| hibernate.default\_schema | 默认的数据库 |
| hibernate.dialect | 配置Hibernate数据库方言，Hibernate可针对特殊的数据库进行优化 |

　　　　　　　　　　　　　　　　　　　　　　　　注：hibernate的前缀可以省略，即：hibernate.dealict==dealect

**表的生成策略：**

1 <id name="pid" type="java.lang.Long">

2 　　<column name="pid" />

3　　　　<generator class="increment" />

4 </id>

　　increment：先找到主键的最大值，在最大值基础上+1

　　identiry：表的自动增长机制，主键必须是数字类型，效率比increment高，但id值不连续

　　sequence：

　　native：

　　assigned：在程序中手动设置主键的值

　　uuid:

**Hibernate的执行流程：**

1. 开始
2. 启动Hibernate
3. 构建Configuration实例

1 Configuration configuration = new Configuration().configure();

1. Configuration实例加载hibernate.cfg.xml文件至内存
2. Configuration实例根据hibernate.cfg.xml文件加载映射文件(\*.hbm.xml)至内存
3. Configuration实例构建一个SessionFactory实例

1 SessionFactory sessionFactory = configuration.buildSessionFactory();

1. SessionFactory实例创建Session实例
2. 1 Session session1 = sessionFactory.openSession(); //不会自动关闭session

2 Session session2 = sessionFactory.getCurrentSession();　　　　//会自动关闭session  
　　 //调用getCurrentSessio()必须在hibernate.cjf.xml文件中进行配置<property name="hibernate.current\_session\_context\_class">thread</property>

1. 由Session实例创建Transaction的一个实例，开启事务

1 Transaction tanTransaction = session1.beginTransaction();

1. 通过Session接口提供的各种方法操作数据库

1 public Serializable save(Object object) throws HibernateException;

2 public Serializable save(String entityName, Object object) throws HibernateException;

3 public Object load(Class theClass, Serializable id, LockMode lockMode) throws HibernateException;

4 public Object load(String entityName, Serializable id, LockMode lockMode) throws HibernateException;

5 public Object load(Class theClass, Serializable id) throws HibernateException;

6 public Object load(String entityName, Serializable id) throws HibernateException;

7 public Object load(String entityName, Serializable id) throws HibernateException;

8 public void update(Object object) throws HibernateException;

9 public void update(String entityName, Object object) throws HibernateException;

10 public void delete(Object object) throws HibernateException;

11 public void delete(String entityName, Object object) throws HibernateException;

12 public Transaction beginTransaction() throws HibernateException;

13 public SQLQuery createSQLQuery(String queryString) throws HibernateException;

14 public Query createQuery(String queryString) throws HibernateException;

15 public void clear();

16 public Object get(Class clazz, Serializable id) throws HibernateException;

17 public Object get(Class clazz, Serializable id, LockMode lockMode) throws HibernateException;

18 public Object get(String entityName, Serializable id) throws HibernateException;

19 public Object get(String entityName, Serializable id, LockMode lockMode) throws HibernateException;

1. 提交事务或回滚事务
2. 1 tanTransaction.commit();

2 tanTransaction.rollback();

1. 关闭Session
2. 1 session1.close();

2 sessionFactory.close();

1. 结束

**查询操作之get和load的区别：**

1 Student student1 = (Student)session.get(Student.class,1);

2 Student student2 = (Student)session.load(Student.class,1);

　　两个方法都是从数据库获取数据

　　　　get():若数据库无此数据，则返回null；

　　　　load():懒加载模式，若数据库无此数据，则会报错ObjectNotfountException异常，

　　所以，load加载数据一定要保证其数据存在。

**flush()**

**openSession与getCurrentSession的区别：**

* getCurrentSession在事务提交或回滚之后会自动关闭，而openSession需要手动关闭。如果使用openSession没有手动关闭，多次使用之后可能会导致连接池溢出。
* openSession每次创建新的Session对象，getCurrentSession使用现有的Session对象。

**hbm常用的设置：**

　　<hibernate-mapping　　//属性

　　　　schema="schemaName"　　//模式的名字

　　　　catelog="catelogName"　　//目录的名称

　　　　default-cascade="cascade\_style"　　//级联风格

　　　　default-access="field|property|ClassName"　　　　//访问策略

　　　　default-lazy="true|false"　　　　//加载策略

　　　　package="packagename"

　　/>

　　<class

　　　　name="ClassName"　　//映射的类名

　　　　table="tableName"　　//对应数据库的映射表名

　　　　batch-size="N"　　　　//抓取策略，一次住多少条数据

　　　　where="condition"　　//条件

　　　　entity-name="EntityName"　　//支持同一个类映射成多个表名(用的少)

　　/>

　　<id

　　　　name="propertyName"

　　　　type="typeName"

　　　　column="colunm\_name"

　　　　length="length"

　　　　<generator class="generatorClass"/>

　　/>

**Query**

1. 获得Hibernate Session对象

1 Session session = sessionFactory.openSession();

1. 编写HQL(Hibernate Query Language)语句

1 String hql = "from Student";　　//这里值得注意的是Student是java bean对象

1. 调用session.createQuery创建查询对象

1 Query query = session.createQuery(hql);

1. 如果HQL语句包含参数则调用Query的setXxx设置参数
2. 调用Query对象的list()或uniqueResult()执行查询

1 List<Student> list = query.list();

**Criteria(条件查询)**

　　Criteria查询语句又叫QBC查询(Query By Criteria)

　　Criterion接口是Hibernate矿浆提供的一个面向对象的查询接口

　　　　Restrictions工厂类  -->Criterion对象

　　使用Criteria对象查询数据的几个步骤：

* 1. 获得Hibernate的Session对象
  2. 通过Session获得Criteria对象
  3. 使用Restrictious的静态方法创建Criterion条件对象
  4. 向Criteria对象添加Criterion查询条件
  5. 执行Criteria的list()或uniqueResult()获得结果

1 Session session = sessionFactory.openSession();

2 Transaction tanTransaction = session.beginTransaction();

3 Criteria criteria = session.createCriteria(Student.class);

4 criteria.add(Restrictions.eq("name", "王五"));

5 List<Student> list = criteria.list();

6 for(Student s : list){

7 System.out.println(s);

8 }

9 tanTransaction.commit();

10 session.close();

11 sessionFactory.close();

**hibernate关联映射**

* 一对多(one to many):在"多"的一方，添加"一"的一方的主键作为外键(学生和班级，在学生的一方添加外键（班级的主键）作为属于哪个班级)
* 多对一(many to one)：
* 一对一(one to one)：在任意一方引入对方主键作为外键，两个主键成为联合主键
* 多对多(many to many)：产生中间关系表，引入两张表的主键作为外键，两个主键成为联合主键

**·反转**

　　inverse属性值为false(默认)，相当于两端都能控制。在实际开发中，如果是一对多的关系，会将"一"的一方的inseverse设置为true，即由"多"的一方来维护关系，如果是多对多的关系，随意一方即可

　　inverse只对<set>、<onetomany>、<manytomany>有效，对<manytoone>、<onetoone>无效。

**·级联**

　　cascade定义的是有关联关系的对象之间的级联关系

* + save-update：执行保存和更新操作时进行级联操作
  + delete：执行删除时进行级联操作
  + delete-orphan
  + all：对所有操作进行级联操作
  + all-delete-orphan
  + none：对所有操作不进行级联操作

　　　　　　★当设置了cascade属性部位none时，hibernate会自动持久化所关联的对象

　　　　　　★cascade属性的设置会带来性能上的变动，需谨慎设置

**Hibernate检索方式**

1. 导航对象图检索方式(前提是必须在对象关系映射文件配置多对一关系)
2. 1 Classes classes = (Classes)session.get(Classe.class, 1);

2 Set<Student> set = classes.getStudents();

1. OID检索方式(get()、load())
2. 1 Classes classes1 = (Classes)session.get(Classe.class, 1);

2 Classes classes2 = (Classes)session.load(Classe.class, 1);

1. QBC检索方式

1 //创建criteria对象

2 Critria criteria = session.createCriteria(Student.class);

3 //设置查询条件

4 Criterion criterion = Restrictious.eq("id", 1);

5 //添加查询条件

6 criteria.add(criterion);

7 //执行查询结果

8 List<Student> cs = criteria.list();

HQL检索方式(Hibernate Query Language)

1. HQL检索方式(Hibernate Query Language)
2. 1 [select/update/delete...][from][where...][group by...]

2 [having...][order by...][asc/desc]

1. 本地SQL检索方式

1 SQLQuery sqlQuery = session.creatSQLQuery("select id,name,age,city from customer");

**投影查询：用来查询对象的部分属性**

1 Configuration configuration = new Configuration().configure();

2 SessionFactory sesionFactory = configuration.buildSessionFactory();

3 Session session = sesionFactory.openSession();

4 Transaction taTransaction = session.beginTransaction();

5

6 String hql = "select p.id,p.name,p.description from Persion p";

7 Query query = session.createQuery(hql);

8 List<Object[]> list = query.list();

9 Iterator iterator = list.iterator();

10 while (iterator.hasNext()) {

11 Object[] object = (Object[]) iterator.next();

12 System.out.println(object[0] + ", " + object[1]);

13 }

14 taTransaction.commit();

15 session.close();

16 sesionFactory.close();

**动态实例查询：**

1 String hql = "select new Person(p.id,p.name,p.description) from Persion p";

　　　　注：必须有构造方法和无参构造

**条件查询：**

1. 按参数位置查询

1 String hql = "from Customer where name like ?";

2 Query query = session.createQuery(hql);

3 query.setString(0, "%wy%");

4 List<Customer> list = query.list();

5 // setDate() 给映射类型为Date的参数赋值

6 // setDouble() 给映射类型为double的参数赋值

7 // setBoolean() 给映射类型为boolean的参数赋值

8 // setInteger() 给映射类型为int的参数赋值

9 // setTime() 给映射类型为date的参数赋值

1. 按参数名称查询
2. 1 String hql = "from Customer where id=:id";
3. 2 Query query = session.createQuery(hql);
4. 3 query.setString("id", "1");

4 List<Customer> list = query.list();

**分页查询：**

　　setFirstResult(int firstResult)

　　setMaxResult(int maxResult)

　　firstResult 默认值 0

　　maxResult 默认值 全部

**lazy(3个懒加载)**

　　需要数据的时候才要加载

　　　　1、类的懒加载(get()数据不存在则返回null，load()如果数据不存在，则报异常)

　　　　　　 session.load方法产生的是代理对象，该代理类是持久化类的子类

　　　　2、集合的懒加载

　　　　　　 set的延迟加载：

　　　　　　       true

　　　　　　       false

　　　　　　       extra

　　　　　　   说明：

　　　　　　       1、因为延迟加载在映射文件设置，而映射文件一旦确定，不能修改了。

　　　　　　       2、延迟加载是通过控制sql语句的发出时间来提高效率的

　　　　3、manytoone的懒加载

　　　　　　 对于性能来说，没有什么影响，所以随便怎么样都行

**抓取策略**

　　　　join:左外连接

　　　　select:默认的值

　　　　subselect:子查询

　　　　（比如:根据需求分析，判断出来该需求分析中含有子查询，所以抓取策略应该用“subselect”）

　　　　说明：

　　　　　　1、因为抓取策略的设置在映射文件中，所以一旦映射文件生成就不能改变了。

　　　　　　2、通过发出怎么样的SQL语句加载集合，从而优化效率的

**一级缓存（生命周期和Session保持一致）**

* 一级缓存位置：

　　　　get():可以把对象放入到一级缓存中，也可以从一级缓存中把对象提取出来

　　　　save():该方法可以把一个对象放入到一级缓存中

　　　　evit():可以把一个对象从session的缓存中清空

　　　　update():可以把一个对象放入到一级缓存(session)中

　　　　clear():清空一级缓存中所有的数据

　　　　close():当调用session.close方法的时候，一级缓存的生命周期就结束了

**创建session的方式**

分布式缓存

一级缓存内存结构

**二级缓存（二级缓存的生命周期和sessionFactory是一致的）**

* **实用场合**

　　　　公开的数据

　　　　   数据基本上不发生变化

　　　　   该数据保密性不是很强

　　　　说明：如果一个数据一直在改变，不适合用缓存。

* **设置二级缓存**

 　　利用的是ehcache实现的二级缓存

    　　1、在hibernate的配置文件中

　　二级缓存的供应商、开启二级缓存：必须有！

　　　　二级缓存的统计机制默认是关闭的，用来验证save方法有没有保存到二级缓存

　　　　2、指定哪个类（或者集合）开启二级缓存（两种方法）

　　　　　　第一种：在映射文件里这么写：

　　　　　　　<class name="com.xjh.hibernate.domain.Classes">

　　　　　　　　<cache usage="read-only"/>

　　　　　　第二种：在hibernate.cfg.xml里这么写：

　　　　　　　　<class-cache usage="reand-only" class=""/>

* **有哪些方法可以把对象放入二级缓存中**

　　　　get方法,list方法可以把一个或者一些对象放入到二级缓存中

* **哪些方法可以把对象从二级缓存中提取出来**

　　　　get方法,iterator方法可以提取

**查询缓存**

 　　一级缓存和二级缓存都是对象缓存:就是把该对象对应的数据库表中的所有的字段全部查询出来了，这种查询在某些场合下会让效率降低。例如：表中的字段特别多，但是程序中所需要的字段却很少。

       查询缓存也叫数据缓存：内存(页面)中需要多少数据就把多少数据放入到查询缓存中。

* **生命周期**

　　　　只要一些数据放入到查询缓存中，该缓存会一直存在，直到缓存中的数据被修改了，该缓存的生命周期就结束了。

* **操作步骤**

1. 在hibernate的配置文件中，开启查询缓存

　　2.使用查询缓存

**总结：**

hibernate总共有三种缓存

　　　　一级缓存解决的问题是在一次请求中，尽量减少和数据库交互的次数，在session.flush之前，改变的是一级缓存的对象的属性。当session.flush的时候才要跟数据库交互，一级缓存解决不了重复查询的问题。一级缓存是对象缓存。

　　　　二级缓存可以把经常不改变、常用的公共的数据放入进来，可以重复查询，利用get方法和iterator方法可以把二级缓存中的数据得到。二级缓存也是对象缓存。

　　　　查询缓存可以缓存数据或者对象，可以利用list方法把查询缓存中的数据放入到缓存中。

　　　　查询缓存中存放的是数据，是数据缓存。