1. Hibernate 框架的工作流程

a. 读取并解析配置文件

b.读取并解析映射信息，创建 SessionFactory

c.打开 Sesssion
d.创建事务 Transation
e.持久化操作
f.提交事务
g.关闭 Session
h.关闭 SesstionFactory

1. Hibernate 框架中的核心接口有哪些，这些接口的具体功能是什么

核心接口有：session,sessionFactory,transaction,query,configuration.

a) Session 接口:Session 接口负责执行被持久化对象的 CRUD 操作(CRUD 的任务是完成与数据库的交流，包含了很多常见的 SQL 语句。)。

b) SessionFactory 接口:SessionFactroy 接口负责初始化 Hibernate。它充当数据存 储源的代理，并负责创建 Session 对象。

c) Configuration 接口:Configuration 接口负责配置并启动 Hibernate，创建 SessionFactory 对象。

d) Transaction 接口:Transaction 接口负责事务相关的操作。

e) Query 和 Criteria 接口:Query 和 Criteria 接口负责执行各种数据库查询。它可以使用 HQL 语言或 SQL 语句两种表达方式。

1. Hibernate 中的 Session 对象表示什么?它与 Web 程序中的 Session 是一样的机制吗

Hibernate 中的 Session 对象代表与数据库之间的一次操作，它的概念介于 Connection 和 Transaction 之间，也称为持久化管理器，因为它是与持久化有关的操作接口。它通过 SessionFactory 打开，在所有的工作完成后，需要关闭。

它与 Web 层的 HttpSession 没有任何关系，Web 层的 HttpSession 是指一个作用域。

1. Hibernate.cfg.xml 配置文件中，应该包含哪些具体的配置内容
    1. Hibernate 运行的底层信息：数据库的 URL、用户名、密码、JDBC 驱动类，数据库 Dialect, 连接池等。
    2. Hibernate 映射文件（*.hbm.xml）。

2. 简述 Hibernate 的主键机制，针对 Oracle 数据库，有几种主键机制可以适用

A, 数据库提供的主键生成机制。identity、sequence（序列）。

B, 外部程序提供的主键生成机制。increment （递增） ，hilo（高低位） ，seqhilo（使用序列的高低 位 ），uuid.hex(使用了 IP 地址+JVM 的启动时间（精确到 1/4 秒)+系统时间+一个计数器值（在 JVM 中唯 一）),uuid.string。

C, 其它。native（本地），assigned（手工指定），foreign（外部引用）

针对 Oracle 数据库，有 sequence，uuid.hex，native，assigned，foreign 主键机制可以适用

1. 请简述 Hibernate 中 cascade,inverse，constrained 几个属性的区别

cascade（级联） :

是操作主表或者从表时，要不要自动操作从表或者主表，比如，保存主表的时候，要不要也默认保存从表,cascade 的值主要有四种：
none,all,delete,save-update。

Inverse:

是指要不要交出控制权,值有 true(交出控制权，不再维护双方的关系)和 false(不交出控制权，继续维护双方的关系)。

constrained:

表示当前引用对象的主键是否作为当前对象的主键参考,true 为是,false 为否.

1. Hibernate 有几种数据查询方式，这几种数据查询方式的优缺点
    1. 使用主键 id 加载对象(load(),get());
    2. Criteria: 通过面向对象化的设计，将数据查询条件封装为一个对象。Criteria 本身只是一个查询容器，查询条件通过 criteria.add 方法添加到 criteria 查询实例中。
    3. HQL（Hibernate Query Language）针对 hibernate 的查询语言，完全面向对象，理解继承，多态和关联之类的概念。HQL 配备了很强大的查询语言，在语法结构上类似 SQL，但 HQL 是面向对象的查询语言。
    4. Native sql: 使用数据库的原生 sql 语句来查询。

优缺点：

a) criteria 最适合动态查询,但不太适合统计查询,qbe 还不够强大.只适合简单的查询.

b) hql 功能很强大,适合各种情况,但是动态条件查询构造起来很不方便.

c) Native sql 可以实现特定的数据库的 sql.但是可移植性并不好.

1. Hibernate 中的延迟机制的原理，以及 Hibernate 中数据有几种延迟加载方式?

延迟加载机制是为了避免一些无谓的性能开销而提出来的,所谓延迟加载就是当在真正需要数据的时候，才真正执行数据加载操作。

Hibernate 中提供了三种延迟加载方式分别是

A. 实体对象的延迟加载

B. 集合的延迟加载

C. 属性的延迟加载

1. Hibernate 中 Load 和 Get 两种方法查询数据的区别

load 是采用延迟机制(load 语句不读库，等使用非主键时才去读库)，而 get 不采用延迟机制(get 语句时马上读库)。

a.当数据库不存在对应 ID 数据时，调用 load()方法将会抛出 ObjectNotFoundException 异常，get()方法将返回 null.

b.当对象.hbm.xml 配置文件<class>元素的 lazy 属性设置为 true 时,调用 load()方法时则返回持久对象的代理 类实例，此时的代理类实例是由运行时动态生成的类，该代理类实例包括原目标对象的所有属性和方法，该代理类实例的属性除了 ID 不为 null 外，所在属性为 null 值，查看日志并没有 Hibernate SQL 输出，说明没有执行查询操作，当代理类实例通过 getXXX()方法获取属性值 时，Hiberante 才真正执行数据库查询操作。当对象.hbm.xml 配置文件<class>元素的 lazy 属性设置为 false 时，调 用 load()方法则是立即执行数据库并直接返回实体类，并不返回代理类。而调用 get()方法时不管 lazy 为何值，都直接返回实体类。

c. load()和 get()都会先从 Session 缓存中查找，如果没有找到对应的对象，则查询 Hibernate 二级缓存，再找不到该对象，则发送一条 SQL 语句查询。

总之对于 get 和 load 的根本区别，一句话，hibernate 对于 load 方法认为该数据在数据库中一定存在，可以放心的使用代理来延迟加载，如果在使用过程中发现了问题，只能抛异常；而对于 get 方法，hibernate 一定要获取到真实的数据，否则返回 null。

1. Hibernate 如何实现对象之间一对一的映射。一对一的映射有几种方式

A.以主键关联:关联的两个实体共享一个主键

具体映射：

（主表 User）

<class name=”com.softfz.pojo.TUser” table=”T_USER”>

<id name=”userid” type=”java.lang.Long”>

<column name=”USERID” precision=”22″ scale=”0″ />

<generator class=”sequence”>

<param name=”sequence”>seq_t_user</param>

</generator>

</id>

<one-to-one name=”userInfo” cascade=”all”></one-to-one>

</class>

（从表 UserInfo）

<class name=” com.softfz.pojo.UserInfo” table=”T_USERINFO ”>

<id name=”userid”>

<!- userInfo 的主键来源 user，也就是共享 user 的主键 ->

<generator class=”foreign”>

<param name=”property”>user</param>

</generator>

</id>

<!- one-to-one 标签的含义，指示 hibernate 怎么加载它的关联对象，默认根据主键加载，constrained=”true”， 表明当前主键上存在一个约束，UserInfo 的主键作为外键参照了 user ->

<one-to-one name=”user” constrained=”true” />

</class>

B.一对一以外键关联： 两个实体各自有不同的主键，但是一个实体有一个外键引用另一个实体的主键。

（从表 UserInfo）

<class name=" com.softfz.pojo.UserInfo" table="T_USERINFO ">

<id name="userInfoId" type="java.lang.Long">

<column name="USERID" precision="22" scale="0" />

<!- userInfo 的主键来源由序列生成 ->

<generator class="sequence">

<param name="sequence">seq_t_userinfo</param>

</generator>

</id>

<!—may-to-one 表示 user 和 userinfo 是一对多的关系，userinfo 的外键 - userid 参考 user 的主键，unique 表示这是一种特殊的一对多->

<many-to-one name="user" column="userid" unique="true">

</many-to-one>

</class>

1. Hibernate 如何实现对象之间的一对多映射。并且如何对 Set 集合中的列表数据进行排序

（主表 User）

<class name="com.test.hibernate.User" table="TBL_USER">

<id name="id" column="userId"><generator class="native" /></id>

<set name="addresses" cascade="all" inverse="true" >

<!- 从表的外键字段 ->

<key column=”userid”/>

<one-to-many class=”com.test.hibernate.Address”/>

</set>

</class>

（主表 Address）

<class name=”com.test.hibernate.Address” table=”TBL_ADDRESS”>

<id name=”id” column=”addressId”> <generator class=”native” /></id>

<!—column 表示从表的外键字段 userid 参考了 user 的主键 ->

<many-to-one name=”user” class=”com.softfz.pojo.TUser” column=”userid”>

</many-to-one>

</class>

1. Hibernate 如何实现对象之间的多对多的映射

（主表 User）

<class name=” com.softfz.pojo.User” table=”T_USER”>

<id name=”id” column=”userId”><generator class=”native”/></id>

<set name=”roles” table=”t_user_role” cascade=”save-update” >

<!- column 指中间表的外键字段 ->

<key column=”useridd”></key>

<!- column 指与从表相关联的中间表的外键字段 ->

<many-to-many class=”com.softfz.pojo.TRole” column=”roleidd”>

</many-to-many>

</set>

</class>

（从表 role）

<class name=”com.softfz.pojo.TRole” table=”T_ROLE” >

<id name=”id” column=”userId”><generator class=”native”/></id>

<set name=”users” table=”t_user_role” cascade=”save-update”>

<!- column 指中间表的外键字段 ->

<key column=”roleid”></key>

<!- column 指与从表相关联的中间表的外键字段 ->

<many-to-many class=”com.softfz.pojo.TUser” column=”userid”>

</many-to-many>

</set>

</class>

1. Hibernate 框架中，如何实现对象数据之间的内连接操作

hql = ”Select a,b From Orderinfo a,Orderdetail b where a.autoid = b.orderid”;

特点：无需配置 Orderinfo 和 Orderdetail 的关联关系。

1. Hibernate 框架中，如何实现对象数据这间的左外连接操作

hql = ”Select a From Orderinfo a left join a.orderDetails”;

特点：必须配置 Orderinfo 与 orderDetails 之间的关联关系。

1. 如何在 Hibernate 中实现对数据的批量删除和批量更新

通过 Hibernate 的 session.delete(“from TUser”)进行批量操作有如下缺点：

(1)　占用大量内存，必须把 1 万个 TUser 对象先加载到内存，然后一一通过主键删除他们。

(2)　执行的 delete 语句的数目太多，每个 delete 语句只能更新一个 Customer 对象，必须通过 1 万条 delete 语句才能删除一万个 TUser 对象，频繁的访问数据库，会大大降低应用的性能。

直接通过 Hibernate　API 进行批量更新和批量删除都不值得推荐。而直接通过 JDBC　API 执行相关的 SQL 语句或调用相关的存储过程，是批量更新和批量删除的最佳方式，这两种方式都有以下好处：

(1)　无需把数据库中的大批量数据先加载到内存中，然后逐个更新或修改他们，因此不会消耗大量内存。

(2)　能在一条 SQL 语句中更新或删除大批量的数据。