佟刚Hibernate讲解

对象持久化：

1. 狭义的理解：“持久化”仅仅是指把对象永久保存到数据库中。
2. 广义的理解：“持久化”包括和数据库相关的操作：
3. 保存：把对象永久保存到数据库中。
4. 更新：更新数据库中对象（记录）的状态
5. 删除：从数据库中删除一个对象
6. 查询：根据特定的查询条件，把符合查询条件的一个或多个对象从数据库加载到内 存中。
7. 加载：根据特定的OID，把一个对象从数据库加载到内存中。
8. OID：为了在系统中能够找到所需对象，需要为每一个对象分配一个唯一的标识号。在 关系数据库中称之为主键，而在对象术语中，则称之为对象标识（Object Identifier--OID）

ORM（Object Relation Mapping）：对象关系映射

1. ORM主要解决对象-关系映射

|  |  |
| --- | --- |
| 面向对象概念 | 面向关系概念 |
| 类 | 表 |
| 对象 | 表的行（记录） |
| 属性 | 表的列（记录） |

2、ORM思想：将关系数据库中表的记录映射成为对象，以对象的形式展现，程序员可以 把对数据库的操作转化为对对象的操作。

3、ORM采用元数据来描述对象-关系映射细节，元数据通常采用xml格式，并且存放在专 门的对象-关系映射文件中。

Configuration类：

1. Configuration类负责管理hibernate的配置信息。包含如下内容
2. 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、数据库连接池等（对应hibernate.cfg.xml文件）
3. 持久化与数据表的映射关系（\***.**hbm**.**xml文件）
4. 创建Configuration的两种方式：
5. 属性文件（hibernate.properties）:

Configuration cfg = new Configuration();

2）XML文件（hibernate.cfg.xml）

Configuration cfg = new Configuration()**.**configure();

【注：configure()方法还支持带参数的访问，参数就是hibernate.cfg.xml的文件 路径，默认是 /hibernate.cfg.xml】

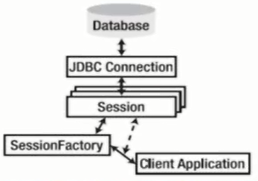
SessionFactory：

1. 针对单个数据库映射关系经过编译后的内存镜像，是线程安全的。
2. SessionFactory对象一旦构造完成，即被赋予特定的配置信息
3. SessionFactory是生成Session的工厂
4. 构造SessionFactory很消耗资源，一般情况下一个应用只初始化一个SessionFactory 对象。
5. Hibernate4.x新增加了一个ServiceRegistry接口，所有基于hibernate的配置或者服 务都必须统一向这个ServiceRegistry注册后才能生效
6. Hibernate4.x创建SessionFactory步骤：

|  |
| --- |
| //1.创建一个SessionFactory对象  Configuration configuration = **new** Configuration().configure();  ServiceRegistry serviceRegistry = **new** ServiceRegistryBuilder().applySettings(configuration.getProperties()).buildServiceRegistry();    SessionFactory sessionFactory = configuration.  buildSessionFactory(serviceRegistry); |

Session：

1. Session是应用程序与数据库之间交互操作的一个单线程对象，是hibernate运作的核 心。所有持久化对象必须在session的管理下才可以进行持久化操作。
2. Session对象的生命周期很短，session关闭就销毁了。
3. Session对象有一个一级缓存，显式执行flush之前，所有的持久层操作的数据都缓存 在session对象处。
4. 相当于JDBC中的Connection。



Hibernate配置文件的两个配置项：

1. hbm2ddl.auto：该属性可以帮助程序员实现正向工程，即由Java代码生成数据库脚本， 进而生成具体的表结构。取值create | update | create-drop | validate

1）create：会根据 \***.**hbm**.**xml文件来生成数据表，但是每次都会先删除上一次的表， 重新生成表。【即使第二次生成的表没有任何改变也会删除第一次生成的表，然后 再生成。】

2）update：最常用的属性值，也会根据 \***.**hbm**.**xml文件生成表，但若 \***.**hbm**.**xml 文件和数据库中对应的表结构不同，hibernate将更新数据表结构，但不会删除已 有的行和列。

3）create-drop：会根据 \***.**hbm**.**xml文件生成表，但是SessionFactory一旦关闭， 表就自动删除。

4）validate：会和数据库中的表进行比较，若 \***.**hbm**.**xml文件中的列在数据表中不存 在，则抛出异常。

Session缓存：（一级缓存）

1、在session接口的实现中包含一系列的Java集合，这些Java集合构成了session缓存。 只要session实例没有结束生命周期，且没有被清理缓存，则存放在它缓存中的对象也 不会结束生命周期。

2、flush()方法：使数据表中的记录和Session缓存中的对象的状态保持一致。为了保持一 致，则可能发送相应的SQL语句。

1）在Transaction的commit()方法中：先调用session的flush()方法，再提交事务。

2）flush()方法可能会发送SQL语句，但不会提交事务。

3）注意：在未提交事务或显式的调用session**.**flush()方法之前，也有可能进行flush() 操作：

①执行HQL或QBC查询，会先得到数据表的最新的记录。

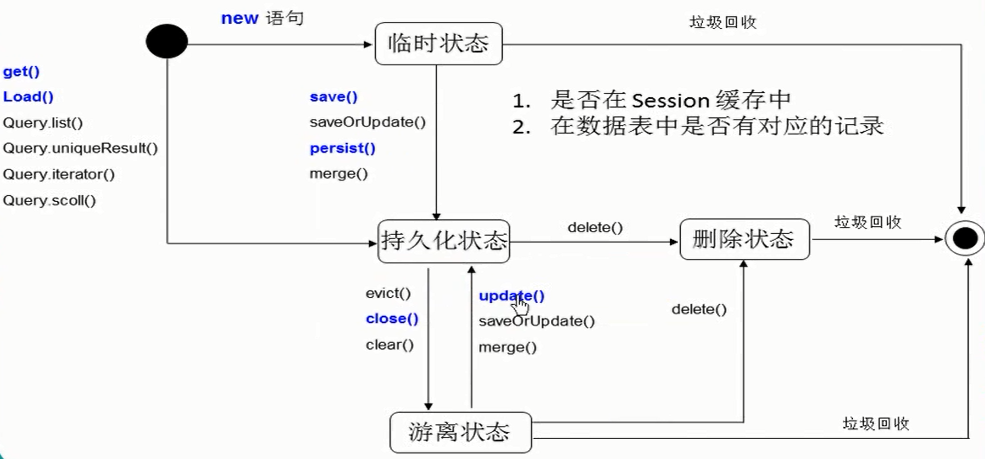
②若记录的ID（主键）是由底层数据库使用自增的方式生成的，则在调用save() 方法时，就会立即发送 insert 语句。因为save()后，必须保证对象的ID是 存在的。

1. refresh()方法：
2. 会强制发送 select 语句，以使session缓存中对象的状态和数据表中对应的记录 保持一致。

4、clear()方法：清理缓存

持久化对象的状态：

1. 站在持久化的角度，hibernate把对象分为4中状态：持久状态、临时（瞬时）状态、 游离（脱管）状态、删除状态。Session的特定方法能使对象从一个状态转换到另一个 状态。
2. 临时（瞬时）状态：
3. 不处在Session的缓存中
4. 在数据库没有对应的记录
5. 持久状态：
6. OID不为null
7. 位于Session缓存中
8. 数据库中有相应的数据
9. 游离（脱管）状态：
10. OID不为null
11. 不再处于session缓存中
12. 一般情况下，游离对象是由持久对象转变过来的，因此在数据库中可能还存在与之 对应的记录。
13. 删除状态：
14. 在数据库中没有和其OID对应的记录
15. 不再处于session缓存中
16. 一般情况下，应用程序不该再使用该删除的对象



Hibernate.cfg.xml的常用属性：

1. C3P0数据库连接池属性：
2. hibernate.c3p0.max\_size：数据库连接池的最大连接数
3. Hibernate.c3p0.min\_size：数据库连接池的最小连接数
4. Hibernate.c3p0.timeout：数据库连接池中连接对象在多长时间没有使用过后，就 应该被销毁。

4）hibernate.c3p0.max\_statements：缓存statements对象的数量。

5）hibernate.c3p0.idle\_test\_period：表示连接池检测线程多长时间检测一次池内的 所有连接对象是否超时。连接池本身不会把自己从连接池中移除，而是专门有一个 线程按照一定的时间间隔来做这件事，这个线程通过比较连接对象最后一次被使用 的时间和当前时间的时间差来和timeout做对比，进而决定是否销毁这个连接对 象。

6）hibernate.c3p0.acquire\_increment：当数据库连接池中的连接耗尽时，同一时刻 获取多少个数据库连接。

2、在hibernate中使用C3P0数据源：【参考hibernate2-2】

1）导入jar包【C3P0的jar包】

2）加入配置

|  |
| --- |
| <!-- 配置C3P0数据源 -->  <!-- 数据库连接池的最大连接数 -->  <property name=*"c3p0.max\_size"*>10</property>  <!-- 数据库连接池的最小连接数 -->  <property name=*"c3p0.min\_size"*>5</property>  <!-- 当数据库连接池中的连接耗尽时，同一时刻获取多少个数据库连接。 -->  <property name=*"c3p0.acquire\_increment"*>2</property>    <!-- 表示连接池检测线程多长时间检测一次池内的所有连接对象是否超时 -->  <property name=*"c3p0.idle\_test\_period"*>2000</property> <!-- 2秒 -->  <!-- 3）数据库连接池中连接对象在多长时间没有使用过后，就应该被销毁。 -->  <property name=*"c3p0.timeout"*>2000</property> <!-- 2秒 -->    <!-- 缓存statements对象的数量 -->  <property name=*"c3p0.max\_statements"*>10</property> |

3、hibernate.jdbc.fetch\_size：实质是调用Statement.setFetchSize()方法设定JDBC的 Statement读取数据的时候每次从数据库中取出的记录条数。【取值100比较合适】

4、hibernate.jdbc.betch\_size：设置对数据库进行批量删除，批量更新和批量插入的时候 的批次大小，类似于设置缓存区的意思。【取值30比较合适】

\*.hbm.xml文件讲解：

1. class元素用于指定类和表的映射：

1）name：指定该持久化类映射的持久化类的类名

2）table：指定该持久化类映射的表名, Hibernate 默认以持久化类的类名作为表名

3）dynamic-insert：若设置为 true, 表示当保存一个对象时, 会动态生成 insert 语 句, insert 语句中仅包含所有取值不为 null 的字段. 默认值为 false

4）dynamic-update：若设置为 true, 表示当更新一个对象时, 会动态生成 update 语句, update 语句中仅包含所有取值需要更新的字段. 默认值为 false

5）select-before-update：设置 Hibernate 在更新某个持久化对象之前是否需要先 执行一次查询. 默认值为 false

6）batch-size：指定根据 OID 来抓取实例时每批抓取的实例数.

7）lazy：指定是否使用延迟加载.

8）mutable：若设置为 true, 等价于所有的 <property> 元素的 update 属性为 false, 表示整个实例不能被更新. 默认为 true.

9）discriminator-value：指定区分不同子类的值. 当使用 <subclass/> 元素来定义 持久化类的继承关系时需要使用该属性

2、property元素：

1）property 元素用于指定类的属性和表的字段的映射

①name：指定该持久化类的属性的名字

②column：指定与类的属性映射的表的字段名. 如果没有设置该属性, Hibernate 将直接使用类的属性名作为字段名.

③type：指定 Hibernate 映射类型. Hibernate 映射类型是 Java 类型与 SQL 类型的桥梁. 如果没有为某个属性显式设定映射类型, Hibernate 会运用反射 机制先识别出持久化类的特定属性的 Java 类型, 然后自动使用与之对应的 默认的 Hibernate 映射类型.

④not-null：若该属性值为 true, 表明不允许为 null, 默认为 false

⑤access：指定 Hibernate 的默认的属性访问策略。默认值为 property, 即： 使用 getter, setter 方法来访问属性。若指定 field, 则 Hibernate 会忽略 getter/setter 方法, 而通过反射访问成员变量

⑥unique：设置是否为该属性所映射的数据列添加唯一约束.

⑦index：指定一个字符串的索引名称。当系统需要 Hibernate 自动建表时, 用 于为该属性所映射的数据列创建索引, 从而加快该数据列的查询.

⑧update：表示这一列的值是否可以被修改。

⑨scale：指定该属性所映射数据列的小数位数, 对 double, float, decimal 等类 型的数据列有效.

⑩formula：设置一个 SQL 表达式，Hibernate 将根据它来计算出派生属性的值。

--》派生属性：并不是持久化类的所有属性都直接和表的字段匹配，持久化类 的有些属性的值必须在运行时通过计算才能得出来, 这种属性称为派生 属性。

--》使用 formula 属性时：formula=“(sql)” 的英文括号不能少

--》Sql 表达式中的列名和表名都应该和数据库对应, 而不是和持久化对象的 属性对应。

--》如果需要在 formula 属性中使用参数, 这直接使用 where cur.id=id 形 式，其中 id 就是参数, 和当前持久化对象的 id 属性对应的列的 id 值 将作为参数传入。

1. Java 时间和日期类型的 Hibernate 映射：

1）在 Java 中，代表时间和日期的类型包括：java.util.Date 和 java.util.Calendar。 此外, 在 JDBC API 中还提供了 3 个扩展了 java.util.Date 类的子类： java.sql.Date, java.sql.Time 和 java.sql.Timestamp, 这三个类分别和标准 SQL 类型中的 DATE, TIME 和 TIMESTAMP 类型对应

2）在标准 SQL 中, DATE 类型表示日期；TIME 类型表示时间；TIMESTAMP 类型 表示时间戳，同时包含日期和时间信息

1. <set>元素的inverse属性：

1）在hibernate中通过对inverse属性操作来决定是由双向关联中的哪一方来维护表 和表之间的关系。Inverse=false的为主动方，inverse为true的为被动方，由主 动方负责维护关联关系。

2）在没有设置inverse=true的情况下，两边都维护父子关系

3）在one-to-many关系中，将many方设为主控方将有助于性能改善。

主键生成策略：（标识符生成器）

1. increment：

1）increment 标识符生成器由 Hibernate 以递增的方式为代理主键赋值

2）Hibernate 会先读取 NEWS 表中的主键的最大值,，而接下来向 NEWS 表中插入 记录时, 就在 max(id) 的基础上递增, 增量为 1。

3）适用范围：

①由于 increment 生存标识符机制不依赖于底层数据库系统, 因此它适合所有 的数据库系统。

②适用于只有单个 Hibernate 应用进程访问同一个数据库的场合, 在集群环境下 不推荐使用它。

③OID 必须为 long, int 或 short 类型, 如果把 OID 定义为 byte 类型, 在运 行时会抛出异常。

1. identity：

1）identity 标识符生成器由底层数据库来负责生成标识符, 它要求底层数据库把主键 定义为自动增长字段类型

2）适用范围:

①由于 identity 生成标识符的机制依赖于底层数据库系统, 因此, 要求底层数据 库系统必须支持自动增长字段类型。支持自动增长字段类型的数据库包括: DB2, Mysql, MSSQLServer, Sybase 等（Oracle不支持）

②OID 必须为 long, int 或 short 类型, 如果把 OID 定义为 byte 类型, 在运 行时会抛出异常

1. sequence：
2. 标识符生成器利用底层数据库提供的序列来生成标识符。

|  |
| --- |
|  |

2）Hibernate 在持久化一个 News 对象时, 先从底层数据库的 news\_seq 序列中获 得一个唯一的标识号, 再把它作为主键值

3）适用范围：

①由于 sequence 生成标识符的机制依赖于底层数据库系统的序列, 因此, 要求 底层数据库系统必须支持序列. 支持序列的数据库包括: DB2, Oracle 等

②OID 必须为 long, int 或 short 类型, 如果把 OID 定义为 byte 类型, 在运 行时会抛出异常

1. Hilo：
2. hilo 标识符生成器由 Hibernate 按照一种 high/low 算法生成标识符, 它从数 据库的特定表的字段中获取 high 值。

|  |
| --- |
|  |

2）Hibernate 在持久化一个 News 对象时, 由 Hibernate 负责生成主键值。hilo 标 识符生成器在生成标识符时, 需要读取并修改 HI\_TABLE 表中的 NEXT\_VALUE 值。

3）适用范围：

①由于 hilo 生存标识符机制不依赖于底层数据库系统, 因此它适合所有的数据库 系统。

②OID 必须为 long, int 或 short 类型, 如果把 OID 定义为 byte 类型, 在运 行时会抛出异常

1. native：

1）native 标识符生成器依据底层数据库对自动生成标识符的支持能力, 来选择使用 identity, sequence 或 hilo 标识符生成器.

2）适用范围:

①由于 native 能根据底层数据库系统的类型, 自动选择合适的标识符生成器, 因 此很适合于跨数据库平台开发

②OID 必须为 long, int 或 short 类型, 如果把 OID 定义为 byte 类型, 在运 行时会抛出异常

缓存：

1. 一级缓存Session缓存：
2. 二级缓存SessionFactory缓存：
3. 内置缓存：hibernate自带的，不可卸载，通常在hibernate的初始化阶段， hibernate会把映射元素据和预定义的SQL语句放到SessionFactory的缓存中， 映射元数据是映射文件（\*.hbm.xml）中数据的复制，该内置缓存是只读的。
4. 外置缓存（二级缓存）：一个可配置的缓存插件，在默认情况下，SessionFactory 不会启用这个缓存插件。外置缓存中的数据是数据库数据的复制，外置缓存的介质 可以是内存或磁盘。
5. 适合放入二级缓存中的数据：

①很少被修改

②不是很重要的数据，允许出现偶尔的并发问题。

1. 不适合放入二级缓存中的数据：

①经常被修改

②财务数据，绝对不允许出现并发问题

③与其它应用共享的数据。

Ehcache.xml文件详解：

1、<diskStore path=*"java.io.tmpdir"*/>

指定一个目录：当Ehcache把数据写到磁盘上时，将把数据写到这个目录下。【我们可 以修改path】

2、

|  |
| --- |
| <defaultCache  maxElementsInMemory=*"10000"*  eternal=*"false"*  timeToIdleSeconds=*"120"*  timeToLiveSeconds=*"120"*  overflowToDisk=*"true"*  /> |

defaultCache：设置缓存的默认数据过期策略

name：设置缓存的名字,它的取值为类的全限定名或类的集合的名字

maxInMemory：设置基于内存的缓存中可存放的对象最大数目

eternal：设置对象是否为永久的，true表示永不过期，此时将忽略timeToIdleSeconds 和 timeToLiveSeconds属性； 默认值是false

timeToIdleSeconds：设置对象空闲最长时间,以秒为单位, 超过这个时间,对象过期。 当对象过期时,EHCache会把它从缓存中清除。如果此值为0,表示对象可以无限期 地处于空闲状态。

timeToLiveSeconds：设置对象生存最长时间,超过这个时间,对象过期。  
 如果此值为0,表示对象可以无限期地存在于缓存中. 该属性值必须大于或等于 timeToIdleSeconds 属性值

overflowToDisk：设置基于内存的缓存中的对象数目达到上限后,是否把溢出的对象写 到基于硬盘的缓存中

3、

|  |
| --- |
| <cache name=*"sampleCache1"*  maxElementsInMemory=*"10000"*  eternal=*"false"*  timeToIdleSeconds=*"300"*  timeToLiveSeconds=*"600"*  overflowToDisk=*"true"*  /> |

Cache：设定具体的命名缓存的数据过期策略。每个命名缓存代表一个缓存区域。

1. 缓存区域(region)：

1）一个具有名称的缓存块，可以给每一个缓存块设置不同的缓存策略。如果没有设置 任何的缓存区域，则所有被缓存的对象，都将使用默认的缓存策略。即： <defaultCache.../>

2）Hibernate在不同的缓存区域保存不同的类/集合。

①对于类而言，区域的名称是类名。如：domain.Student

|  |
| --- |
| <cache name=*"domain.Studnet"*  maxElementsInMemory=*"10000"*  eternal=*"false"*  timeToIdleSeconds=*"300"*  timeToLiveSeconds=*"600"*  overflowToDisk=*"true"*  /> |

②对于集合而言，区域的名称是类名加属性名。如domain.Department.stu 【stu 是Department类中的集合属性名】

|  |
| --- |
| <cache name=*"domain.Department.stu"*  maxElementsInMemory=*"10000"*  eternal=*"false"*  timeToIdleSeconds=*"300"*  timeToLiveSeconds=*"600"*  overflowToDisk=*"true"*  /> |

查询缓存：

1. 默认情况下，设置的缓存对HQL查询时无效，但可以通过以下方式使其有效

①在hibernate.cfg.cml文件中设置开启“查询缓存”。

|  |
| --- |
| <property name=*"cache.use\_query\_cache"*>true</property> |

②调用Query的setCacheable()方法

2、查询缓存依赖于二级缓存