Hibernate3\_Day4

学习内容：

1. Hibernate事务支持(事务的隔离级别)
2. session管理-(手动opensession,本地线程session)
3. 二级缓存（基本二级缓存）：结构、配置、使用(ehcache)
4. 查询缓存的配置使用
5. 性能监测—了解—看看二级缓存到底命中率是什么样的。（了解）
6. JPA注解
7. 其他：hbm反转生成等

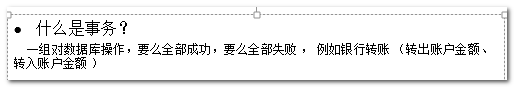
学习目标：

1. Hibernate的二级缓存的配置和使用,知道缓存到底是什么
2. JPA注解的编写

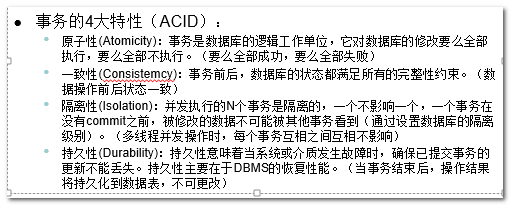
# Hibernate事务支持

## 事务隔离性引发的问题和解决方案

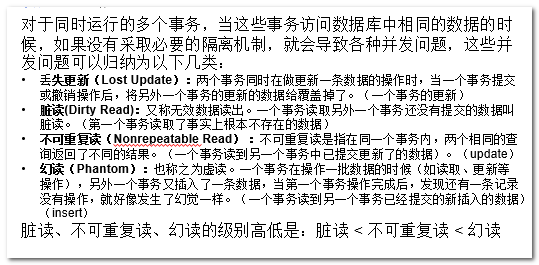
### 什么是数据库事务？

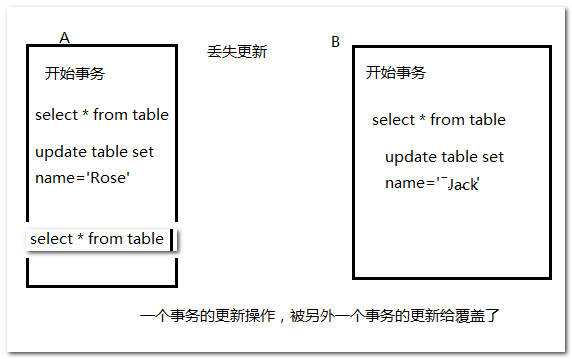


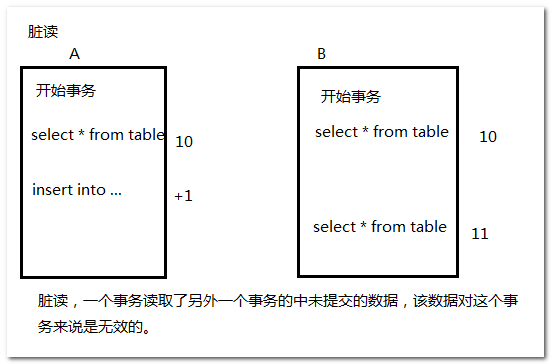
### 事务的4个特性



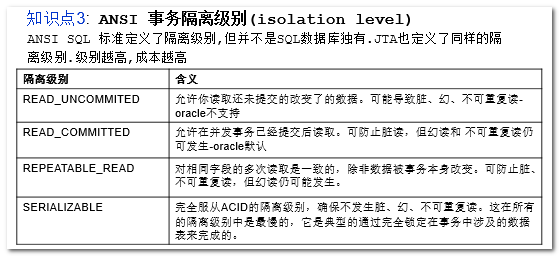
### 事务隔离性引发的问题？







### 事务的隔离级别



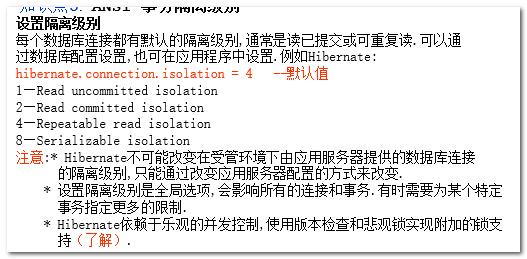
如何选择隔离级别呢？

在使用数据库时候，隔离级别越高，安全性越高 ，性能越低

实际开发中，不会选择最高或者最低隔离级别，选择 READ\_COMMITTED（oracle 默认）、REPEATABLE\_READ (mysql默认)

## Hibernate设置隔离级别

### Hibernate如何设置隔离级别？

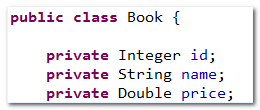


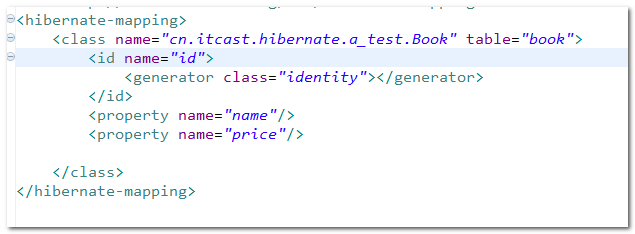
### 测试隔离级别—脏读

图书信息的保存和查询

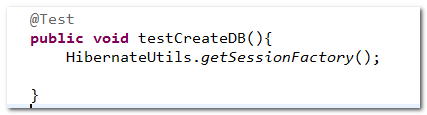
基本步骤：搭建环境-建表-设置不同的隔离级别

第一步：编写实体类、HBM、映射添加、建表测试

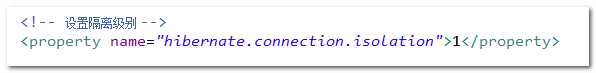








第二步：设置隔离级别



第三步：编码测试：---模拟脏读

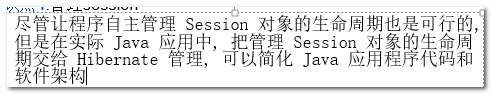
构建两个方法(可通过junit来调试)来模拟两个进程：一个保存，一个查询。分别执行：

注意：Oracle不支持uncommited isolation的隔离级别，仅可以在mysql下可以测试。

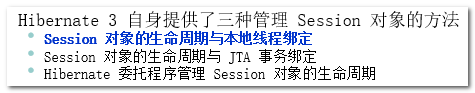
|  |
| --- |
| @Test  **public** **void** testIsolation1(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //保存数据  Book book = **new** Book();  book.setName("Java");  book.setPrice(998d);  session.save(book);    session.getTransaction().commit();  session.close();      }    @Test  **public** **void** testIsolation2(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //查询数据  System.*out*.println(session.createQuery("from Book").list());    session.getTransaction().commit();  session.close();      } |

# Session管理

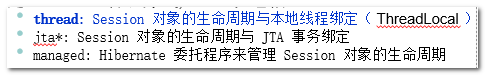
### 如何管理session



### session管理的方法



对应配置说明：

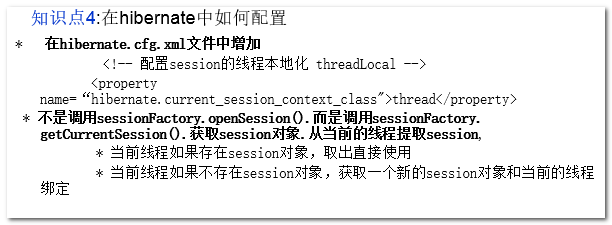


简单的说：在单系统的情况下，就是两种：程序（手动）管理和hibernate框架（线程）管理。

### 线程管理和程序管理的区别

目标：了解opensession和getCurrentSession的区别

要使用线程管理session，需要在配置文件中显式的打开



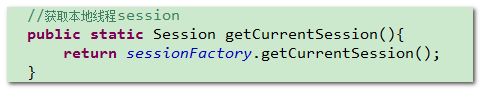
修改session管理的方式：

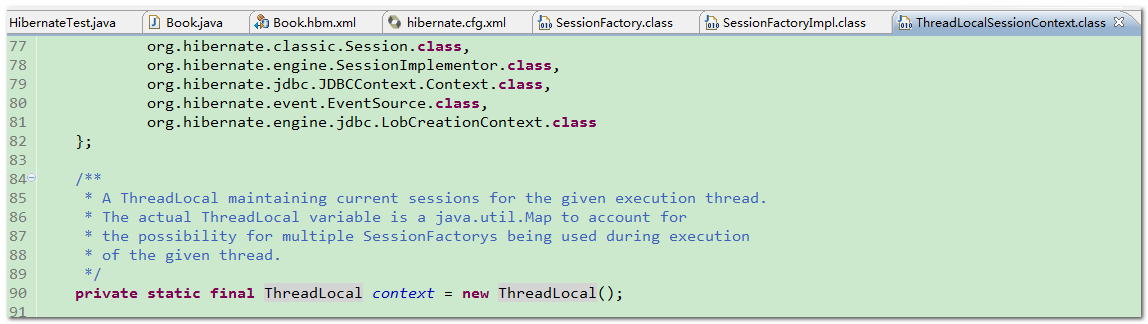


目标:测试两种线程管理的session对象的区别:

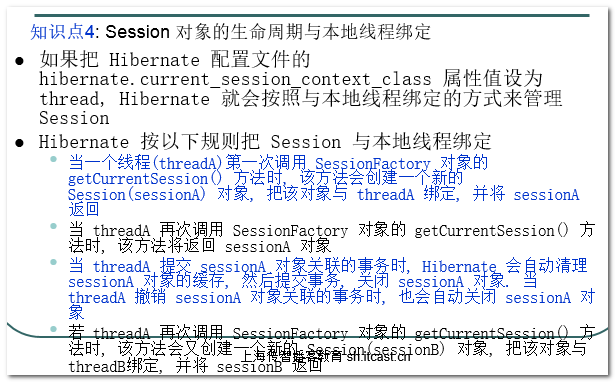
|  |
| --- |
| @Test  //目标：测试手动session管理和本地线程sessin管理的区别（是否同一个对象，是否需要自己管理）  **public** **void** testSessionmanager(){  //手动管理  Session session1 = HibernateUtils.*openSession*();//每一次open，都是一个新的session  System.*out*.println(session1.hashCode());  Session session2 = HibernateUtils.*openSession*();//每一次open，都是一个新的session  System.*out*.println(session2.hashCode());    //释放资源  session1.close();  session2.close();      //本地线程管理  SessionFactory sessionFactory = HibernateUtils.*getSessionFactory*();  Session session3=sessionFactory.getCurrentSession();//此时初始化了一个session  System.*out*.println(session3.hashCode());  Session session4=sessionFactory.getCurrentSession();//会拿到线程中已经初始化过了的session  System.*out*.println(session4.hashCode());    // session3.close();  // session4.close();//session已经被关闭了，会报错。    //如果本地线程管理，session可以不用手动关闭。底层：当线程结束的时候，自动销毁绑定的session  //本地线程管理的session是在一个事务中有效的。一个事务中，session只会new 一个。多次调用session还是用的一个session，节约了资源    } |

工具类的改造：





### 小结

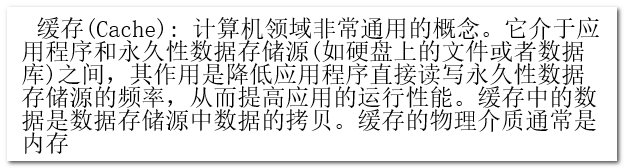


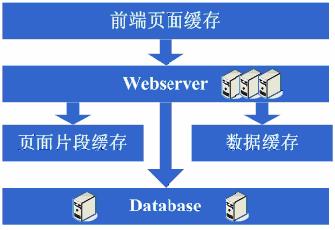
当线程关闭的时候，session会自动关闭！

# 二级缓存

## 二级缓存的相关概念

### 什么是缓存





javaweb应用的缓存一般分两种：页面缓存和数据缓存。我们这里的缓存是指数据缓存。

数据缓存的作用：缓存位于程序和数据库之间，可减少程序访问数据库频率。

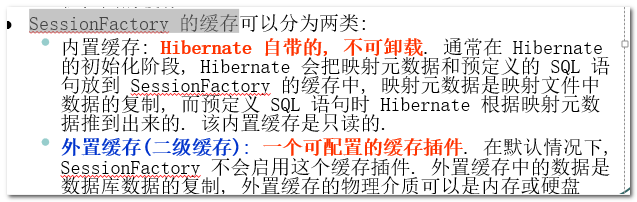
### 什么是Hibernate的二级缓存（与一级缓存对比）

Hibernate中提供了两个级别的缓存:

一级缓存 是session级别的缓存，它是属于事务范围的缓存,生命周期是session的生命周期， （一个线程 绑定一个Session， 对应一份一级缓存， 一级缓存无法实现多用户之间数据共享）,它是hibernate的内置缓存,由hibernate管理的,一般情况下无需进行干预.

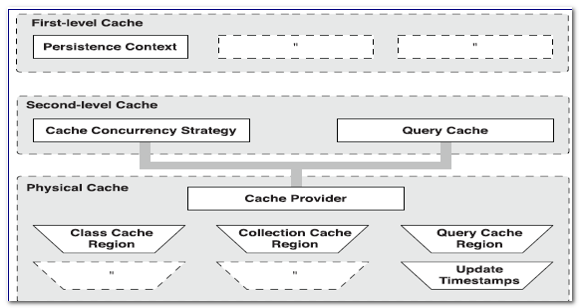
二级缓存 是sessionFactory 级别的缓存，它属于进程级别的缓存 （一个项目 只会对应一个SessionFactory对象， sessionFactory缓存数据 实现多用户之间共享 ），二级缓存是可插拔的。(解耦合思想)

### SessionFactory 级别的缓存分类



### 二级缓存的结构

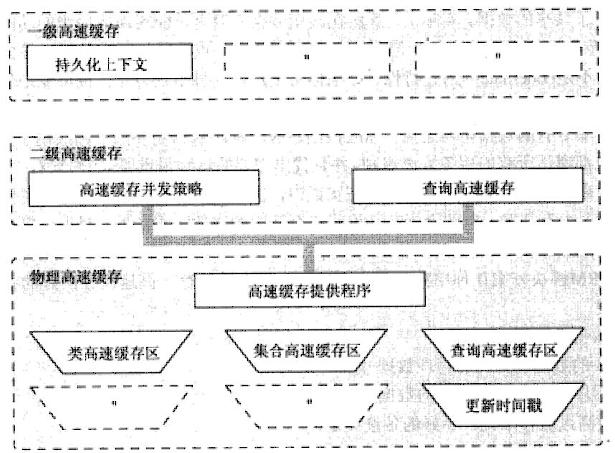
Hibernate缓存的结构：英文版:



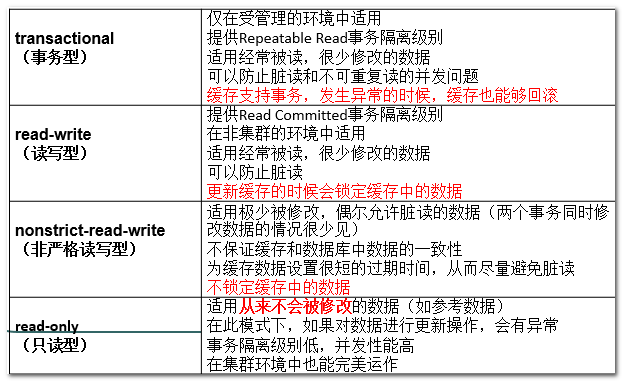
二级缓存一般分为”普通”二级缓存策略和查询缓存策略两种.但通常大家说的二级缓存主要是指普通的二级缓存策略.

内存存储区域:类级别的缓存区域,集合级别的缓存区域,更新时间戳,查询缓存的区域.

中文版:



### 二级缓存的并发策略

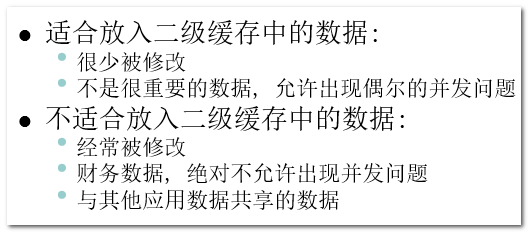


从概念上说：

read-write策略：缓存数据既能读也能写(比如”经常”更新的数据)

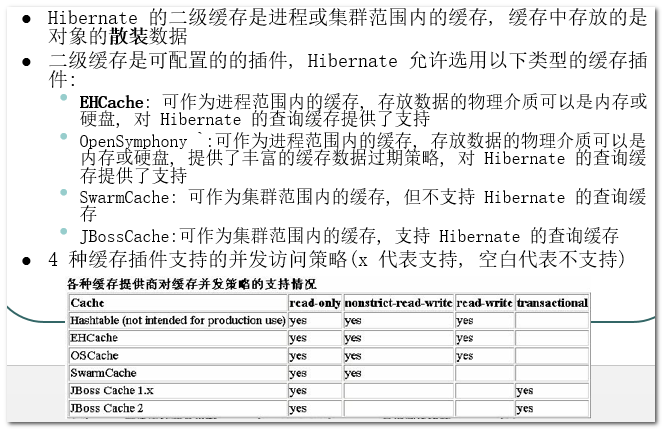
read-only策略：缓存数据一般只用来读。(比如系统参数，地区的分类)，并发效率高！

### 二级缓存的适用和不适用场景



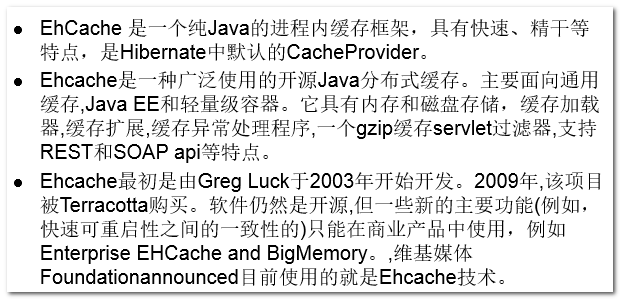
缓存的适用场景就是多查少改。

### 二级缓存提供商



我们课程采用ehcache。

### ehcache基本介绍



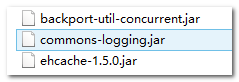
**memcache**

## 二级缓存的基本配置

几件事情：1。开启二级缓存，2，配置二级缓存提供商。3将ehcache的jar导入+核心文件配置

步骤：

第一步：导入ehcache的jar包(3个)

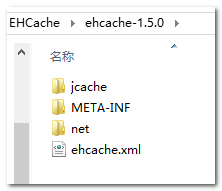


ehcache依赖 backport-util-concurrent 和 commons-logging

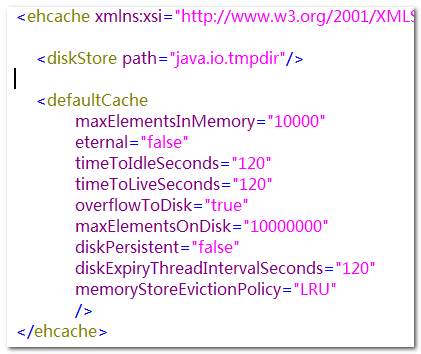
提示：本课程使用的是ehcache的1。x的版本，该版本需要依赖其他jar包，所以，你在导入jar的时候，别忘了其他两个。但2。x之后的版本，依赖jar都已经集成到jar中，所以不需要额外的第三方jar。

第二步：配置ehcache默认的核心配置文件ehcache.xml(名字固定)（放在类路径下）

解压 ehcache的jar ，将根目录 ehcache-failsafe.xml 改名为 ehcache.xml 复制 src



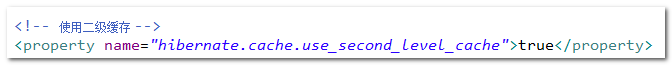
去掉注释：



第三步：配置Hibernate核心配置文件：启用二级缓存

在hibernate.cfg.xml中配置

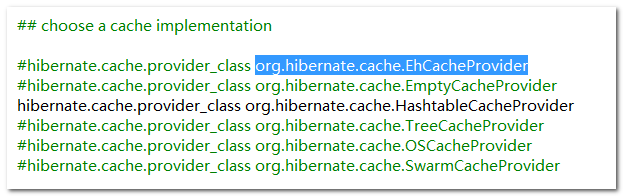
<!-- 开启二级缓存,其实hibernate默认就是开启的,这里显示的指定一下 -->



第四步：配置二级缓存提供商

 <!-- 指定二级缓存产品的提供商 -->

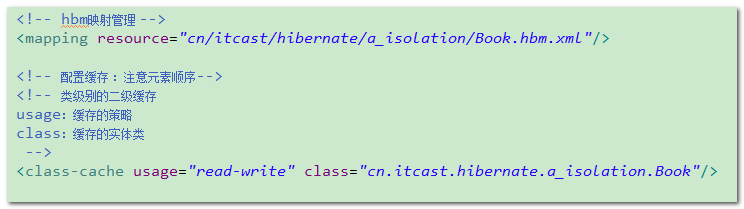




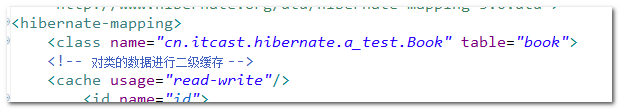
第五步：配置要缓存的目标数据（类）和并发策略（你要缓存哪些数据-对象）

这里可以有两种方法配置：

方法一： 在hibernate.cfg.xml 进行配置



方法二： 在XXX.hbm.xml 进行配置

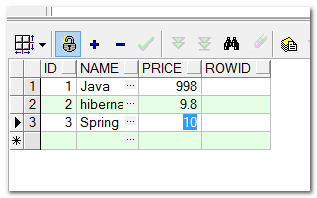


局部的配置会覆盖全局的配置。

## 二级缓存的使用

### 测试二级缓存的存在性

准备数据(手动插入数据)：



测试：Book先toString

//如果二级缓存生效，那么再次跨session查询的时候，就不需要发出sql查询数据库。

|  |
| --- |
| //测试二级缓存的存在性：  //如果二级缓存生效，那么再次跨session查询的时候，就不需要发出sql查询数据库。  @Test  **public** **void** testSecondCacheExist(){  Session session1 = HibernateUtils.*openSession*();  session1.beginTransaction();  //查询操作  //操作：立即发出sql语句，向数据库查询数据，并将查询出来的数据放入一级缓存和快照，  //因为开启了二级缓存，因此，还向二级缓存放入数据  Book book1 = (Book)session1.get(Book.**class**, 1);  System.*out*.println(book1);    session1.getTransaction().commit();  //关闭session，关闭了一级缓存  session1.close();    //再开启session  Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();  //查询操作  //Hibernate会先判断是否有一级缓存，如果有，则从一级缓存获取数据，如果没有，看看有没有二级缓存，如果有，则从二级缓存获取数据  //如果二级缓存还没有呢？发sql从数据库查询。  Book book2 = (Book)session2.get(Book.**class**, 1);  System.*out*.println(book2);    session2.getTransaction().commit();  session2.close();    } |

查看输出结果。

### 二级缓存可以自动同步到一级缓存

|  |
| --- |
|  |

结论:二级缓存可以自动同步到一级缓存,而且会自动引用散装数据的地址作为一级缓存的数据的地址.

任何的查询，都会往一级缓存放，一级缓存是hibernate的操作的核心缓存。

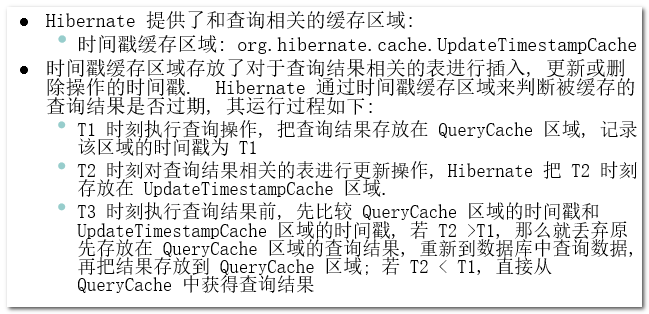
### 一级缓存自动同步到二级缓存(前提二级缓存打开)

|  |
| --- |
| @Test  //一级缓存自动同步到二级缓存  **public** **void** testFirstCacheAutoFromSecondSession(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //数据放入一级缓存，同时也放入二级缓存  Book book1 =(Book)session.get(Book.**class**, 1);  book1.setName("Java");//修改一级缓存，同时二级缓存也被自动同步  System.*out*.println(book1.getName());  session.getTransaction().commit();  //关闭session，清空一级缓存  session.close();    Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();  //直接从二级缓存查询数据，不再发送sql语句  Book book3 =(Book)session2.get(Book.**class**, 1);  System.*out*.println(book3.getName());    session2.getTransaction().commit();  session2.close();  } |

缓存同步的顺序是：默认先放一级缓存，然后再从一级缓存同步到二级缓存。二级缓存的数据是可以来自于一级缓存的。

### 更新时间戳区域—了解

作用和运行过程：



换个说法：

作用：记录hibernate对表中数据最后更新时间 。当有风吹草动的时候，用来判断要不要更新二级缓存。

运行过程：

1、 第一次查询，将结果放入二级缓存 ，查询时间 t1

2、 hibernate通过update 语句对表中数据 更新，同时更新时间戳区域，记录更新时间t2

3、 第二次查询， hibernate会自动去比较（只有发现是更改数据的那个时间戳的时候才会去自动比较）t1和t2的时间，如果t2>t1 ，在缓存后，数据被更新过，缓存的数据不是最新的，重新发送SQL进行查询，更新二级缓存区域

代码：

主要看有没有更新的操作，保证数据的及时性。

|  |
| --- |
| @Test  //更新时间戳的测试（了解，无法直接对该区域进行数据操作，内部操作）  **public** **void** testUpdateTimestamps(){  Session session1 = HibernateUtils.*openSession*();  session1.beginTransaction();  //放入一级二级  Book book1 = (Book)session1.get(Book.**class**, 1);//时间记录到更新时间戳的区域，T1  System.*out*.println(book1);    //手动发出更新语句:无法对一级缓存造成影响，也就不会让一级缓存自动同步到二级缓存  // Query query = session1.createSQLQuery("update t\_book set name='java2222' where id=1");  Query query = session1.createQuery("update Book set name='java2222' where id=1");//t2  //执行  query.executeUpdate();    //旧的数据  System.*out*.println(book1);    //当flush的时候，除了自动同步数据库外，还会自动同步二级缓存，而无需再次查询  session1.getTransaction().commit();  session1.close();    //再开启session  Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();  //查询操作  //Hibernate如果要从二级缓存获取数据的话，它会自动判断更新时间戳，如果更新后的时间t2>t1  //那么Hibernate会认为数据过时了，会自动重新发出sql语句。再放入二级缓存  Book book2 = (Book)session2.get(Book.**class**, 1);  System.*out*.println(book2);    session2.getTransaction().commit();  session2.close();  } |

目的：保证二级缓存最新。

注意：更新时间戳是有hibernate自动维护的，你不需要维护。

### 类级别缓存区域的存储特性-散装数据

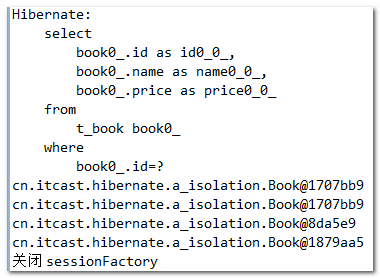
散装数据：不是一个完整对象，是个Object[],当查询出来的时候，临时组装为对象（new PO().setter）。意味着，每次从二级缓存查询的数据的物理地址都不一样。

去掉book的toString

修改代码(要求:多个session)

|  |
| --- |
| //目标：二级缓存的散装数据  @Test  **public** **void** testSecondCachesanzhuangdata(){  //开session  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //查询数据，放入一级缓存，因为我们打开了二级缓存，那么此时，数据也会放入二级缓存。  Book book1 = (Book) session.get(Book.**class**, 1);  System.*out*.println(book1);    session.getTransaction().commit();  session.close();    //关上一个session，再开一个session  Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();  //从二级缓存获取:  Book book3 = (Book) session2.get(Book.**class**, 1);  System.*out*.println(book3);    session2.getTransaction().commit();  session2.close();    //关上一个session，再开一个session  Session session3 = HibernateUtils.*openSession*();  session3.beginTransaction();  //从二级缓存获取:每次取的数据地址不同，说明不是一个对象，为什么？因为二级缓存存的是散装数据  Book book4 = (Book) session3.get(Book.**class**, 1);  System.*out*.println(book4);    session3.getTransaction().commit();  session3.close();      } |

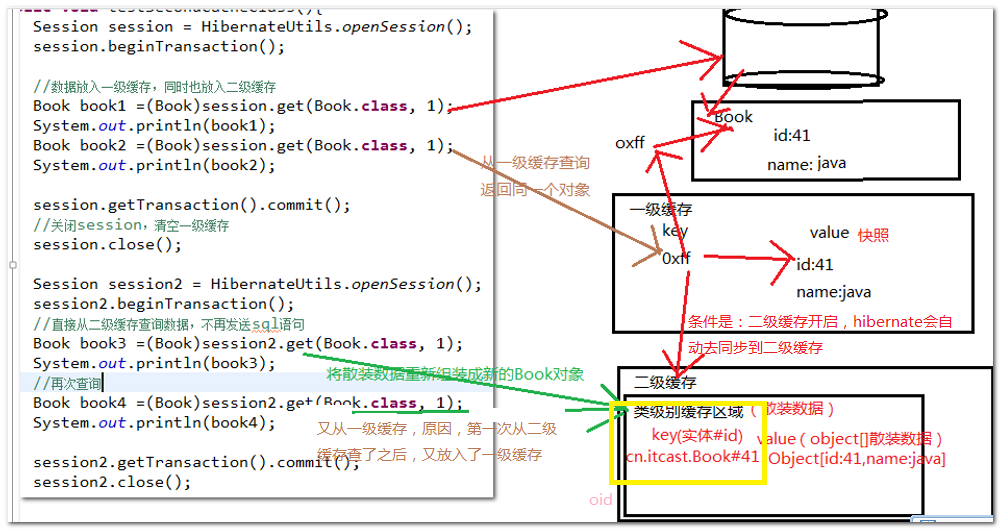
输出结果地址不同：



结论:每次从二级缓存获取数据，对象地址是不同的 ！

原因：类级别的缓存区域存放的是散装数据，每次从二级缓存读取的时候，会重新创建新的对象。

理论图解



散装数据是Object[]的目的是为了存放不同的数据，为了通用！

当再次从二级缓存查询数据的时候。临时组装对象。

### 类级别的二级缓存的读写方式

get，load可读写，但Query.list只有写的功能没有读取的功能（可以用get来读取），

原因：二级缓存类级别的缓存区域的数据的读取必须是”通过#id”进行查询

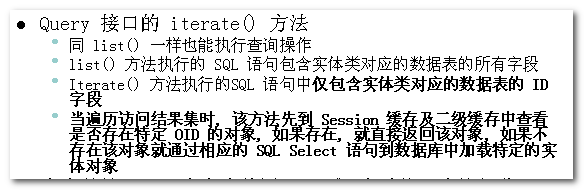
|  |
| --- |
| @Test  //二级缓存的读写  //查询（get、load、query.list等）可以写，读：只能一个一个根据id来读取  **public** **void** testSecondCacheWriteRead(){  Session session1 = HibernateUtils.*openSession*();  session1.beginTransaction();  //批量将数据放入二级缓存  session1.createQuery("from Book").list();//一级二级都 有二楼    session1.getTransaction().commit();  session1.close();    //从二级缓存读取  Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();  //无法直接读取二级缓存，仍然发出sql语句查询数据库  //query.list没有读取二级缓存的能力！  // session2.createQuery("from Book").list();    //可以使用get或load来读取  Book book = (Book)session2.get(Book.**class**, 2);  System.*out*.println(book);  session2.getTransaction().commit();  session2.close();    } |

### Query接口Iterator方法

iterrate与list的区别：

Query的list 方法，只能写入二级缓存，不能读取

Query的iterate方法，返回迭代器，是可以读取二级缓存的



iterate方法---》查询出来集合（Iterator<对象>），但对象只有id，没其他属性，（像load），变量获取数据的时候，像load一样，优先从一级缓存、二级缓存获取数据。

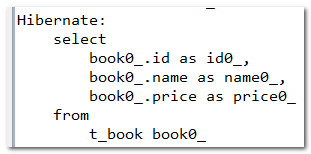
将列表 变成一个个小的load操作了。

list方法和iterator方法有什么不一样的呢?请看下面的示例:

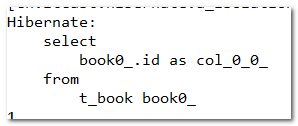
|  |
| --- |
| @Test  //二级缓存的读写  //批量获取二级缓存的数据query.iterator方法  **public** **void** testSecondCachequeryiterator(){  Session session1 = HibernateUtils.*openSession*();  session1.beginTransaction();  //批量将数据放入二级缓存  session1.createQuery("from Book").list();//一级二级都 有    session1.getTransaction().commit();  session1.close();    //从二级缓存读取  Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();  //使用iterator方法  Iterator<Book> iterator = session2.createQuery("from Book").iterate();  //iterate确实可以查询出book对象的列表，但列表中的对象只有id  **while** (iterator.hasNext()) {  //会在遍历的时候，会自动根据id从二级缓存查询数据，类似于一个个小的get或load  Book book = iterator.next();//就会被初始化，会自动从二级缓存拉去数据  System.*out*.println(book);//  }    session2.getTransaction().commit();  session2.close();    } |

结论:

Query.list()发出的语句是直接查询出所有字段的数据.

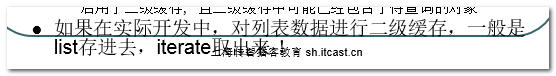


Query.iterate()发出的语句是仅查询出主键id的字段的数据



iterate.next()发出的语句是根据id来查询的,如果二级缓存中有对应记录,则不发出sql语句.

【补充扩展】iterater.next()是被重写了



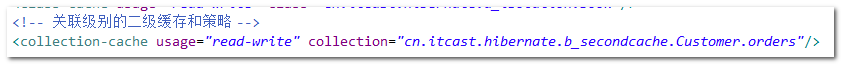
### 关联集合级别的缓存区域存储特性

特性：关联集合级别的缓存区域只会缓存OID ，具体数据会保存类级别缓冲区中

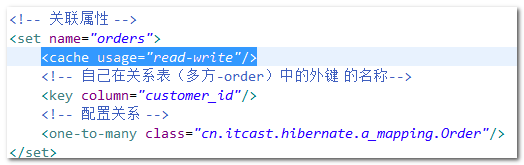
数据准备：将上次课的Customer和order拿过来。

配置：两种方法：

第一种：在hibernate.cfg.xml配置集合级别缓存



第二种：在hbm中配置



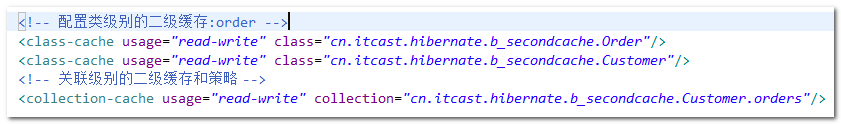
测试代码：

|  |
| --- |
| @Test  //关联集合的缓存区域  **public** **void** testCollectionCache(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //将数据放入二级缓存  //将customer对象放入一级、二级缓存  Customer customer =(Customer) session.get(Customer.**class**, 1);  //通过导航查询订单,order对象也会放入一级缓存、二级缓存。  System.*out*.println(customer.getOrders().size());    session.getTransaction().commit();  session.close();//关闭一级缓存  //-------------------  Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();  //从二级缓存获取customer  Customer customer2 =(Customer) session2.get(Customer.**class**, 1);  //通过导航查询订单？为什么不从二级缓存获取呢？  //原因：没有配置关联集合的缓存区域，它无法知道你的这些order是为了上面的customer而缓存的，不知道他们之间有关系  //导航查询获取的是属性，缓存的是属性  System.*out*.println(customer2.getOrders().size());    // Order order=(Order)session2.get(Order.class, 1);  // System.out.println(order);  session2.getTransaction().commit();  session2.close();  } |

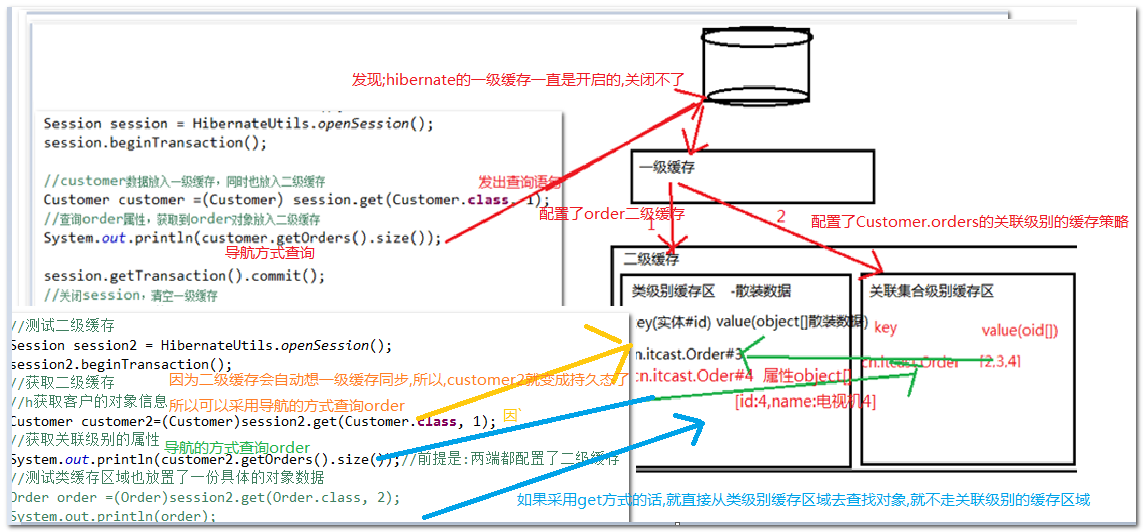
上述例子运行后并没有缓存关联集合对象。原因，集合缓存区必须依赖类缓存区，否则，不能使用到集合缓存。

为什么？因为，集合缓存区，只能缓存集合中的数据id。还要依赖于类级别的缓存区

因此，类缓存必须配置：



图解：



通过分析，发现：

关联集合缓存区域只保存oid(关系)，而且它必须依赖于类级别缓存区域来存储具体的po对象的数据。

缓存区的小结:

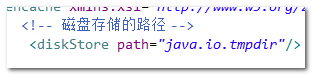
类缓存区域，缓存对象散装数据（object[属性1，属性2，。。。。]）的结果数据 ， 集合缓存区必须依赖类缓存区，才能使用，他是类级别缓存策略的一个补充。缓存的属性OID。

## ehcache缓存的详细配置使用

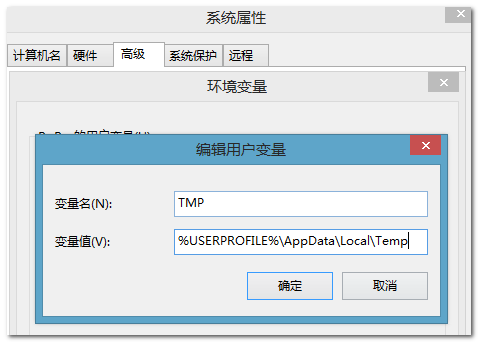
### echcache配置文件解读

使用ehcache 缓存框架，默认加载 src下 ehcache.xml 配置文件

* <diskStore> 缓存数据在硬盘临时存储位置

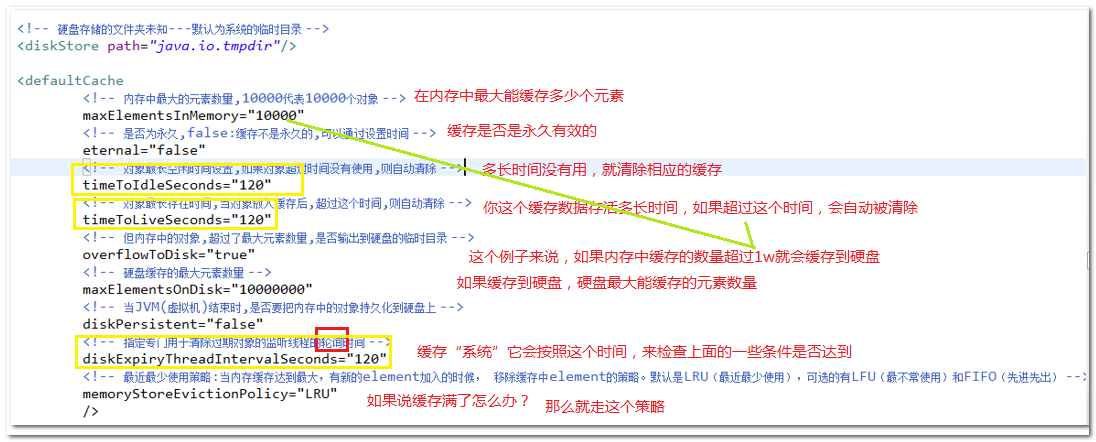


默认查找系统环境变量 TEMP



* <defaultCache> 默认缓存配置





|  |
| --- |
| LRU:最近最少使用:根据使用的频率来判断  A 元素 截至到一个时刻,它被访问了10次  B元素 截至到一个时刻,它被访问了9次  C元素 截至到一个时刻,它被访问了20次  问题:如果缓存满了的话,谁会被最新清除掉,答案是:B  LFU:最不常使用的:根据时间来判断  A 元素 最后一次访问的时间是00:10  B元素 最后一次访问的时间是00:09  C元素 最后一次访问的时间是00:20  问题:如果缓存满了的话,谁会被最新清除掉,,答案是:B  FIFO:先进先出--管道--根据顺序有关系  放入缓存的顺序:A---B---C  问题:如果缓存满了的话,谁会被最新清除掉,,答案是:A |

### 缓存数据持久化到硬盘

目标：两个：自定义缓存区域设置，另外一个就是持久化到硬盘的参数配置。

为什么要自定义缓存呢？因为业务需要，可能不同的实体需要不同的缓存策略。

自定义的方式：使用<cache> 自定义缓存区域 ， 通过name属性 标识缓存区域名称。

自定义echcache配置文件，并测试内存缓存区存满后，自动将数据持久化到硬盘：

修改ehcache.xml：

|  |
| --- |
| <diskStore path="z:"/>  <!-- 自定义缓存区域  name:和Hibernate集成的时候，配置你要缓存的po对象类型，一旦配置了自定义的缓存策略，就不走默认的策略  -->  <cache name=*"cn.itcast.hb.a\_isolation.Book"*  maxElementsInMemory=*"1"*  eternal=*"false"*  timeToIdleSeconds=*"1200"*  timeToLiveSeconds=*"1200"*  overflowToDisk=*"true"*  maxElementsOnDisk=*"10000000"*  diskPersistent=*"false"*  diskExpiryThreadIntervalSeconds=*"120"*  memoryStoreEvictionPolicy=*"LRU"*  /> |

|  |
| --- |
| @Test  //测试内存缓存区存满后,持久化到硬盘  **public** **void** testOverflowToDisk(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //测试批量插入30个数据  // for (int i = 0; i < 30; i++) {  // Book book = new Book();  // book.setName("book"+i);  // book.setPrice(100d+i);  //  // session.save(book);  // }    List<Book> bookList = session.createQuery("from Book").list();  System.*out*.println(bookList);    session.getTransaction().commit();  session.close();  } |
|  |

提示：由于sessionFactory在运行立刻关闭，缓存还没来得及写入到硬盘，这里可以使用断点暂停的方式来测试缓存写入硬盘。

提示：如果你在hibernate.cfg.xml中和hbm.xml中都配置了缓存的配置，那么会以hbm.xml为准。

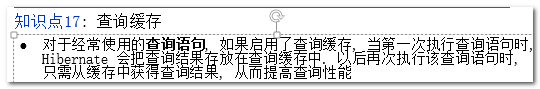
一般会缓存：系统层面：参数，常量、业务层面：订单分类、经常查询的东西，人员信息。注意：缓存有容量。

会将数据存入数据库，然后当系统启动的时候，加载到缓存（内存）。spring会整合ehcache，很容易清除更改缓存的内容，无需重启系统。更便于维护。

## 查询缓存

上面的二级缓存只能通过load.get,query.iterate来获取.query.list是只能存,不能取.

### 什么是查询缓存



查询缓存是基本二级缓存的补充，也是二级缓存的一部分，是一种特殊的二级缓存。

主要用来保存经常查询的sql语句和“结果”。

查询缓存是一种特殊的二级缓存，有人称之为三级缓存。（根本没有三级缓存）.

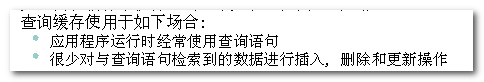
基本二级缓存和查询缓存的区别:（查询条件不同，返回结果不同）

基本二级缓存，查询条件是记录的id ，返回整条记录散装数据 --- 可封装为实体对象;

查询缓存，查询条件 使用任何sql语句，返回任何sql执行结果,缓存的结果更加灵活. （可解决query.list的无法读取二级缓存的缺陷。）

但查询缓存有个缺点:每次必须用代码手动激活开启.

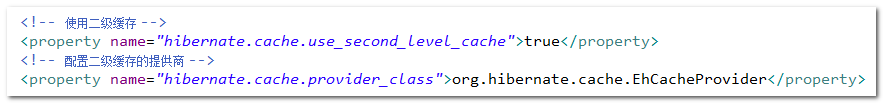
### 查询缓存的应用场合



多查少改的场景

### 使用步骤

1. 设置二级缓存提供商



扩展：一般情况下，使用查询缓存的时候，我们会同时开启基本的二级缓存策略。

1. 开启查询缓存



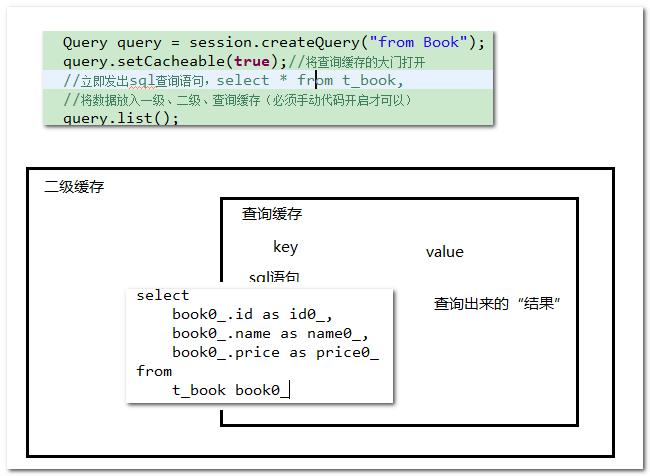
1. 程序中激活使用：



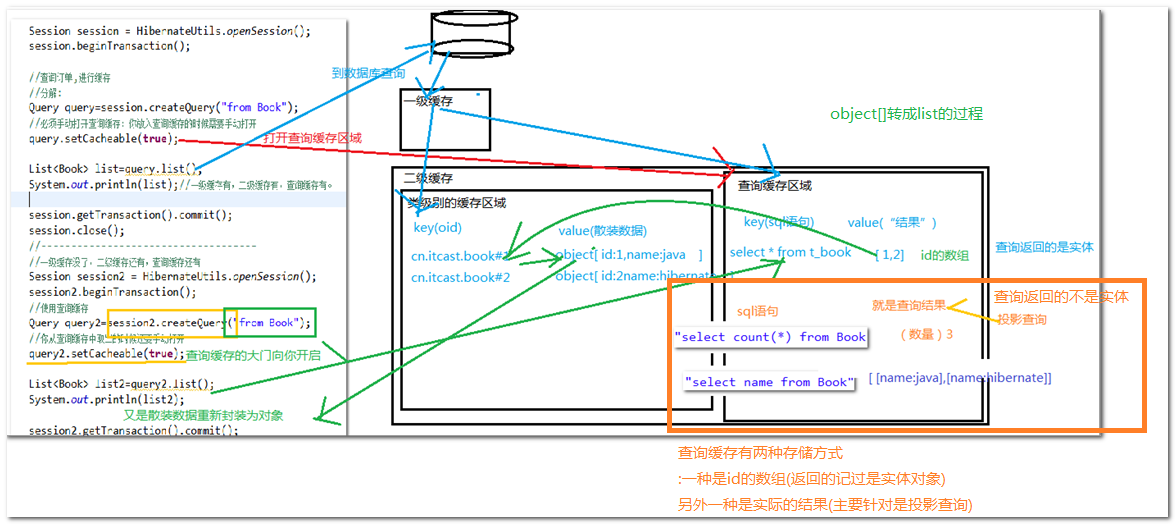
|  |
| --- |
| //测试查询缓存  @Test  **public** **void** testQueryCache(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //将数据放入二级缓存（缓存了语句和结果）  Query query = session.createQuery("from Book");  //手动打开二级缓存  query.setCacheable(**true**);  //当打开二级缓存之后，那么你再list查询，会将数据放入一级、二级、查询缓存  List list = query.list();  System.*out*.println(list);      session.getTransaction().commit();  session.close();  //----------------  Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();    //从查询缓存中获取查询结果  Query query2 = session2.createQuery("from Book");  //获取查询缓存的时候也需要先开启查询缓存的大门  query2.setCacheable(**true**);  //查询：从查询缓存中查询  List list2 = query2.list();  System.*out*.println(list2);      session2.getTransaction().commit();  session2.close();    } |

### 查询缓存原理分析

查询缓存的一般理解：



通过查询缓存来缓存的数据和读取原理（通过跟踪源码得到的结论）：



作用：二级缓存不能缓存的东东，就可以在查询缓存中缓存。查询缓存又是二级缓存的补充。

但是,如果有一级缓存尽量用一级缓存,有普通二级缓存尽量用普通二级缓存,实在没办法了就用查询缓存.

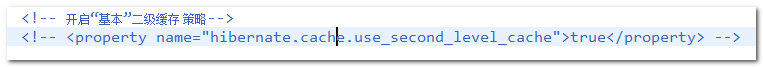
* 实体查询：(目标:验证如果是查询结果可封装为实体的,从查询缓存中取出的是散装数据)

|  |
| --- |
| @Test  //测试查询缓存对应的散装数据  **public** **void** testQueryCacheSanZhuang(){  //使用查询缓存  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //查询订单,进行缓存  Query query=session.createQuery("from Book");  //必须手动打开查询缓存：你放入查询缓存的时候需要手动打开  query.setCacheable(**true**);    List<Book> list=query.list();  System.*out*.println(list.get(0).hashCode());//一级缓存有，二级缓存有，查询缓存有。    session.getTransaction().commit();  session.close();  //------------------------------------  //一级缓存没了，二级缓存还有，查询缓存还有  Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();  //使用查询缓存  Query query2=session2.createQuery("from Book");  //你从查询缓存中取出的时候还要手动打开  query2.setCacheable(**true**);    List<Book> list2=query2.list();  System.*out*.println(list2.get(0).hashCode());    session2.getTransaction().commit();  session2.close();        //------------------------------------  //一级缓存没了，二级缓存还有，查询缓存还有  Session session3 = HibernateUtils.*openSession*();  session3.beginTransaction();  //使用查询缓存  Query query3=session3.createQuery("from Book");  //你从查询缓存中取出的时候还要手动打开  query3.setCacheable(**true**);    List<Book> list3=query3.list();  System.*out*.println(list3.get(0).hashCode());    session3.getTransaction().commit();  session3.close();  } |

* 投影查询(目标:验证如果是查询结果不可以封装为实体的,从查询缓存中取出的是同一个对象)

|  |
| --- |
| @Test  //测试查询缓存  **public** **void** testQueryCache(){  ///放入  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //将查询结果放入查询缓存(一级缓存，基本二级缓存)  // Query query = session.createQuery("select name from Book");  Query query = session.createQuery("select count(b) from Book b");  //手动开启查询缓存  query.setCacheable(**true**);//开启查询缓存的大门了，如果查询，则会将语句缓存到查询缓存。  // List list =query.list();  // System.out.println(list.get(0).hashCode());  **long** count1 =(Long) query.uniqueResult();  System.*out*.println(count1);    session.getTransaction().commit();  session.close();      //取出  Session session2 = HibernateUtils.*openSession*();  session2.beginTransaction();  //query.list来取二级缓存，普通的二级缓存是不能取到数据（by id），  // Query query2 = session2.createQuery("select name from Book");  Query query2 = session2.createQuery("select count(b) from Book b");  query2.setCacheable(**true**);////开启查询缓存的大门了,就可以从查询缓存中查找需要的数据  // List list2 = query2.list();  // System.out.println(list2.get(0).hashCode());    **long** count2 =(Long) query2.uniqueResult();  System.*out*.println(count2);    session2.getTransaction().commit();  session2.close();    } |

查询缓存小结：

1. 查询缓存区存放的是键值对，key是sql语句，value是“结果”(如果不是实体，就直接存放最终结果，如果是实体，则存放oid数组,具体实体数据存放在类缓存区(散装数据).)
2. ~~从某种意义上说，查询缓存策略不完全依赖于基本二级缓存的查询策略：~~

【缓存使用小结】

* 一级缓存：在session中缓存数据,内置自带的缓存，无法关闭，还能更新数据（快照功能）。get load
* 二级缓存：跨session来缓存数据了，任何查询来放入缓存数据，但是取：get ，load，query.iterator（根据id来获取）
* 查询缓存：上面都不能满足，直接缓存sql语句，和查询结果。取：相同的sql就能取了。

## 监测性能—了解

### 为什么要监测性能？

实际应用中，不是说配置了二级缓存就一定会有效果，需要通过某种途径查看缓存的使用率。

Hibernate SessionFactory 提供了一个统计功能（默认是关闭的）：

|  |  |
| --- | --- |
| [Statistics](mk:@MSITStore:D:\itcast\bak\文档\Hibernate3.2API.chm::/org/hibernate/stat/Statistics.html) | [**getStatistics**](mk:@MSITStore:D:\itcast\bak\文档\Hibernate3.2API.chm::/org/hibernate/SessionFactory.html#getStatistics())()            Get the statistics for this session factory |

通过Statistics这个对象，对二级缓存和查询缓存 进行使用监控

采用会话工厂的统计方法SessionFactory.getStatistics()，包含二级缓存的统计

### 监测步骤

1. 在hibernate.cfg.xml开启统计功能

|  |
| --- |
| <!-- 开启统计-监控-->  <property name=*"hibernate.generate\_statistics"*>true</property> |

1. 程序中通过SessionFactory 获取监控数据

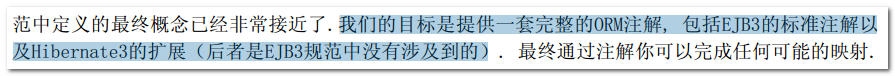
|  |
| --- |
| @Test  //缓存使用率性能监控(前提是开启了性能监视)  **public** **void** testStatistics(){  SessionFactory sessionFactory = HibernateUtils.*getSessionFactory*();  //通过会话工厂，获取统计对象  Statistics statistics = sessionFactory.getStatistics();  //初始状态下，命中次数和丢失次数都是0  System.*out*.println("二级缓存的命中次数:"+statistics.getSecondLevelCacheHitCount());  System.*out*.println("二级缓存的丢失次数:"+statistics.getSecondLevelCacheMissCount());    Session session1 = sessionFactory.openSession();  //原理讲解  //查询前2条订单信息，放入二级缓存  session1.createQuery("from Book where id<=2").list();    session1.close();    System.*out*.println("-换一个session-------------------------");    Session session2 = sessionFactory.openSession();  //查询1号订单  Book book1 = (Book) session2.get(Book.**class**, 1);  System.*out*.println(book1);  Book book2 = (Book) session2.get(Book.**class**, 2);  System.*out*.println(book2);    System.*out*.println("二级缓存的命中次数:"+statistics.getSecondLevelCacheHitCount());  System.*out*.println("二级缓存的丢失次数:"+statistics.getSecondLevelCacheMissCount());    System.*out*.println("-------------------------");    //查询1号订单  Book book4 = (Book) session2.get(Book.**class**, 3);  System.*out*.println(book4);    System.*out*.println("二级缓存的命中次数:"+statistics.getSecondLevelCacheHitCount());  System.*out*.println("二级缓存的丢失次数:"+statistics.getSecondLevelCacheMissCount());  session2.close();  } |

提示：平时系统正常运行的时候，不要开这个功能，消耗性能。

# JPA注解开发

jpa是sun公司的一个ORM规范，只有接口和注解，没有具体实现。

jpa是EJB3中的子规范。



hibernate中有两套注解规范：一套jpa，一套自己的；

使用注解开发，效率高！

## 单表常用注解

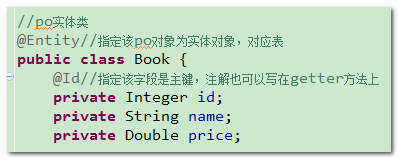
新建工程：hibernate3\_day4\_jpa

导入jar、配置文件、工具类等

第一步：建立实体

第二步：最简注解示例: (使用了注解的默认值)

【最最小化配置】：

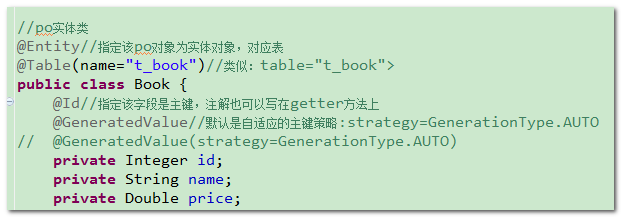


第三步：Hibernate.cfg.xml配置映射:



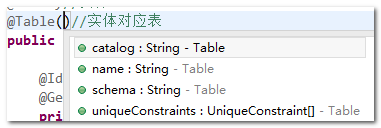
建表测试（略）

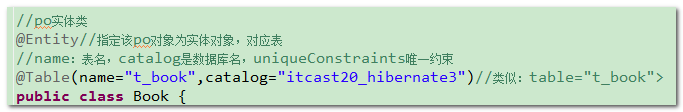
【推荐标准最小化配置】：



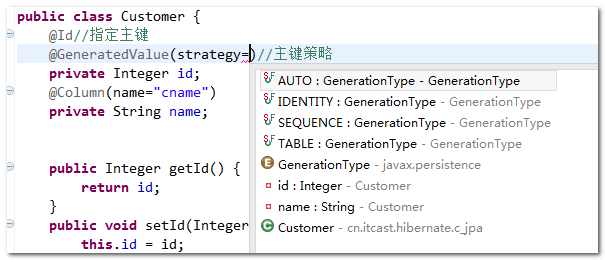
更多常用注解注解

实体和表本身相关：



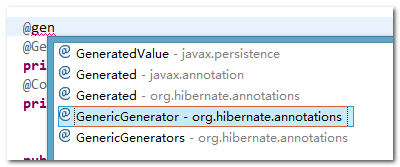


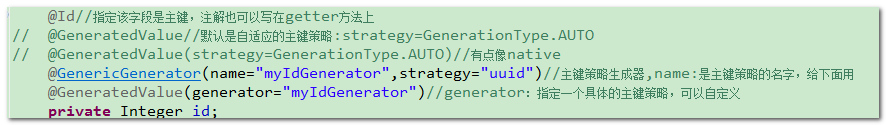
主键相关的：



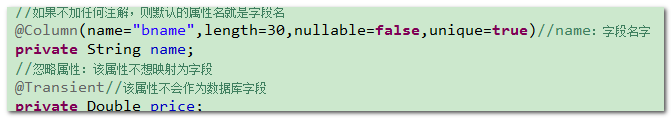
Auto相当与native,默认值

自定义主键策略(下面使用hibernate的实现):

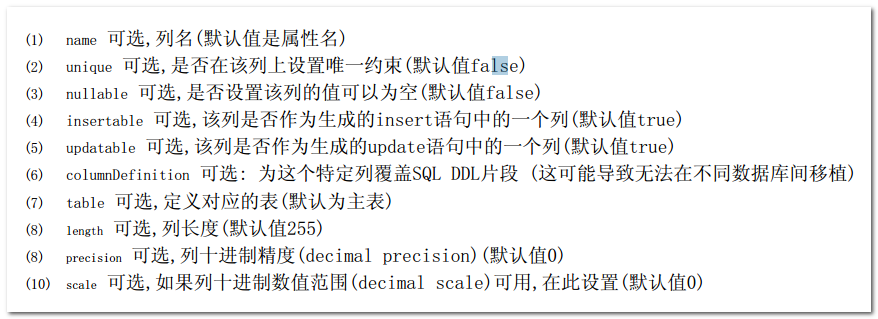




其他字段相关的：



属性字段官方参考配置：



【较完整配置】：



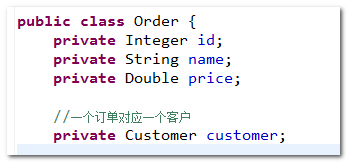
【补充】：

注解：可以放到属性上面设置，也可以在getter方法上设置，效果一样。但是：要么都放属性，要么都放getter，不能混着用。

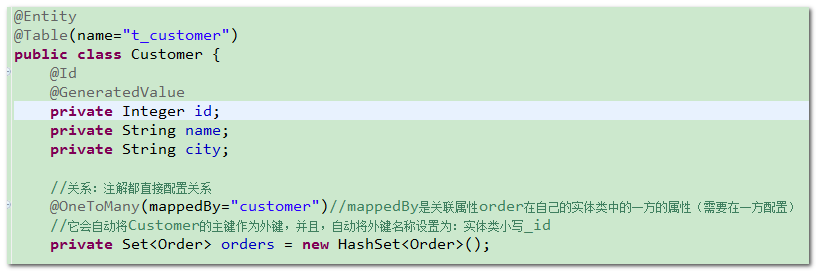
## 多表常用注解

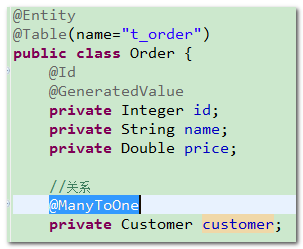
### 一对多

新建订单表实体类，并与客户表建立实体关系。



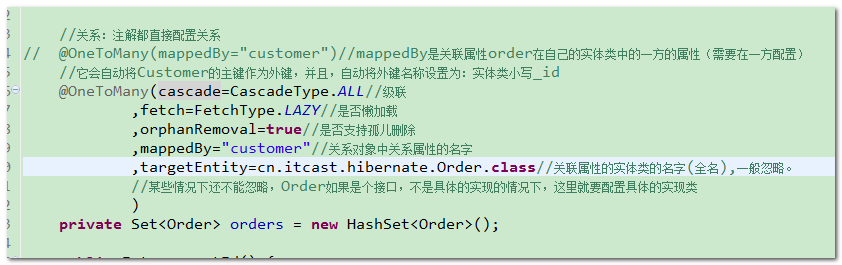
最小化配置：



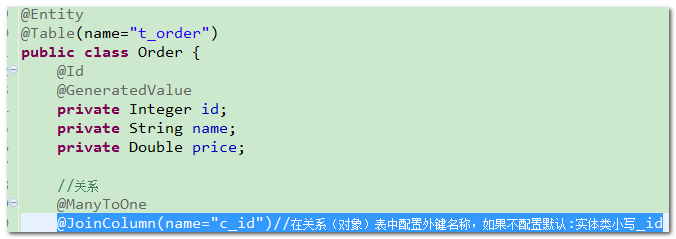


更多详细配置：

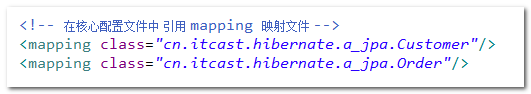
Customer.java



Order.java



映射文件加入到核心文件：

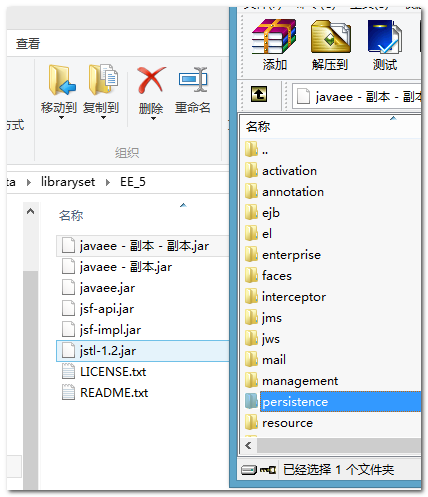


建表测试。。。--单表可能不报错，多表会报错

结果可能报错：



原因（包冲突了） javaee.jar 包含不完整JPA规范；解决：手动删除 javaee.jar中 javax.persistence 包

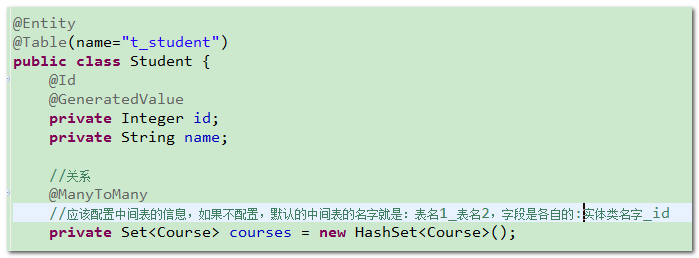


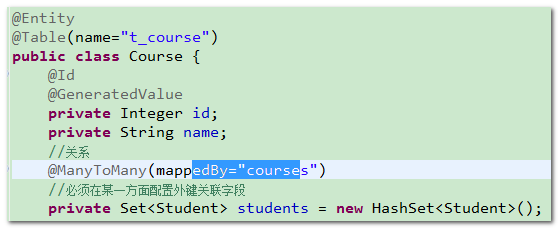
### 多对多

示例：学生和课程

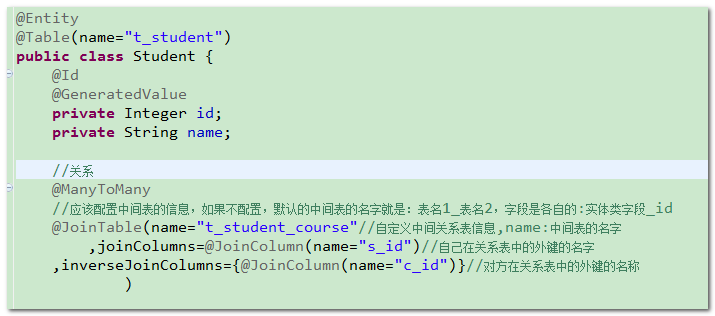
建立实体类，并加上注解：

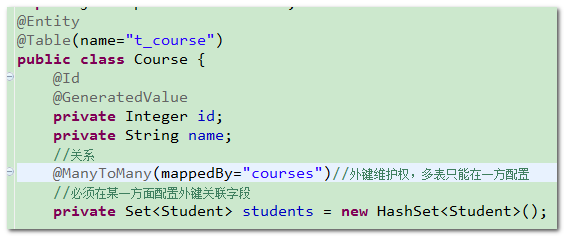
【最小化配置】：





【更多配置】：



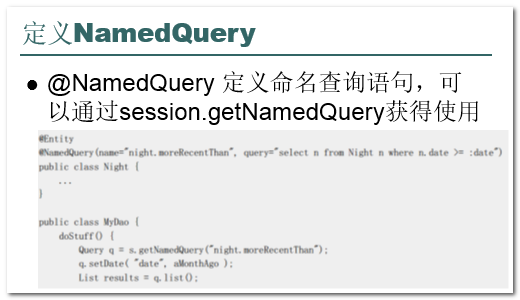


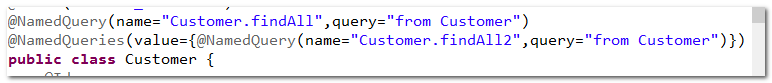
在核心配置文件中配置映射：



建表测试：。。。。。

## 命名查询





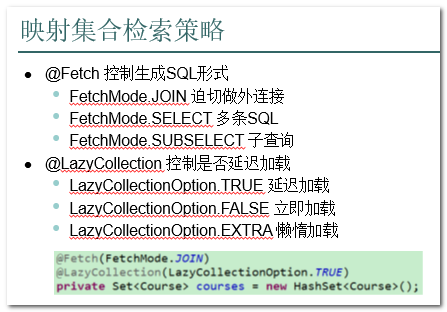
测试：

|  |
| --- |
| **public** **void** test02(){  Session session= HibernateUtils.*getCurrentSession*();  session.beginTransaction();  //save  // Customer c =new Customer();  // c.setName("xiaoming");  //  // Order o = new Order();  // o.setName("电视机");  // o.setPrice(998d);  // Order o2 = new Order();  // o2.setName("电视机2");  // o2.setPrice(9981d);  //  // c.getOrders().add(o);  //  // session.save(c);    //query  List<Customer> list = session.getNamedQuery("Customer.findAll").list();  System.*out*.println(list.get(0).getOrders());    Customer c =(Customer)session.get(Customer.**class**, 1);  System.*out*.println(c.getOrders().size());    session.getTransaction().commit();  //session.close();    } |

## 抓取策略

类抓取策略：

关联集合抓取策略：

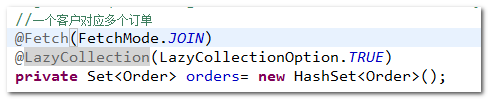


jpa的：

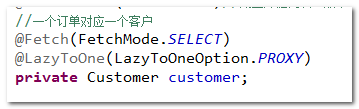


Hibernate的：

一方：



多方：



## 缓存策略

在实体上打开二级缓存策略：



也可以在核心文件配置，但建议是在实体上配置。

类级别的二级缓存



集合级别的二级缓存



最终：

|  |
| --- |
| //客户：一方  @Entity  @Table(name="t\_customer")  //hql  @NamedQuery(name="Customer.findAll" ,query="from Customer")//配置一个命名查询  @NamedQueries({@NamedQuery(name="Customer.findAll" ,query="from Customer"),@NamedQuery(name="Customer.findAll2" ,query="from Customer")})  //sql  @NamedNativeQuery(name="Customer.findcount",query="select count(\*) from t\_customer")//配置一个  @NamedNativeQueries({@NamedNativeQuery(name="Customer.findcount",query="select count(\*) from t\_customer"),@NamedNativeQuery(name="Customer.findcount2",query="select count(\*) from t\_customer")})  @Cacheable(**true**)//jpa打开了缓存  @Cache(usage=CacheConcurrencyStrategy.*READ\_ONLY*)  **public** **class** Customer {  @Id  @GeneratedValue  **private** Integer id;//oid属性  **private** String name;  **private** String city;  //关联集合属性  @OneToMany(mappedBy="customer"//自己在关联的对象中的属性（告诉jpa，两个对象怎么关联的，一方的id关联的）  ,cascade=CascadeType.*ALL*//级联配置  ,fetch=FetchType.*LAZY*//是否懒加载，默认是懒加载  ,targetEntity=cn.itcast.hibernate.a\_singlepo.Order.**class**//orders里面的元素对应的实体类的类型  //实体类可以有接口或抽象类，这里配置的实现类  )  @Fetch(FetchMode.*SELECT*)//hibernate的抓取策略  @LazyCollection(LazyCollectionOption.*TRUE*)//hiernate  @Cache(usage=CacheConcurrencyStrategy.*READ\_WRITE*)  **private** Set<Order> orders = **new** HashSet<Order>(); |

# 扩展

## myeclipse工具反转生成hbm映射

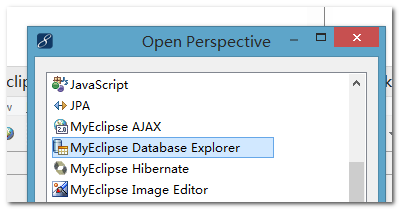
hibernate初衷：通过设计po（domain领域模型设计），来映射实际表。

但企业级开发一般都是先设计表，在根据表来编写hbm映射。

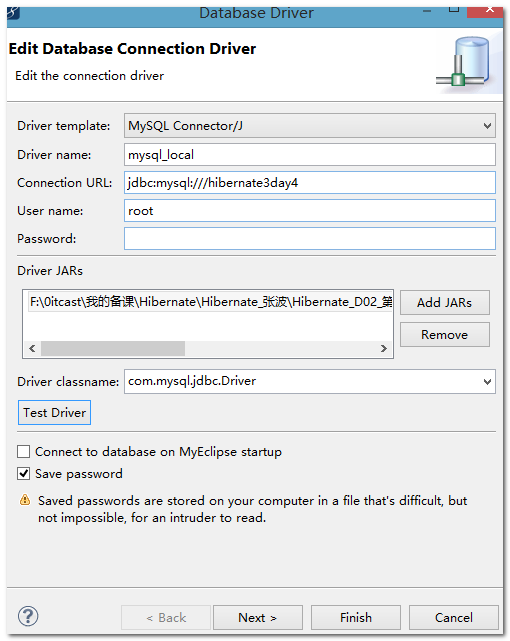
对于已经存在的表，我们可以借助myeclipse的工具内置的Hibernate反转引擎，来根据表自动生成hbm、实体类等

第一步：在数据库视图中建立数据库连接

切换myeclipse视图到Database Explorer

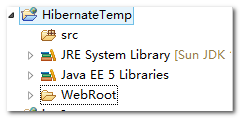


建立数据库连接

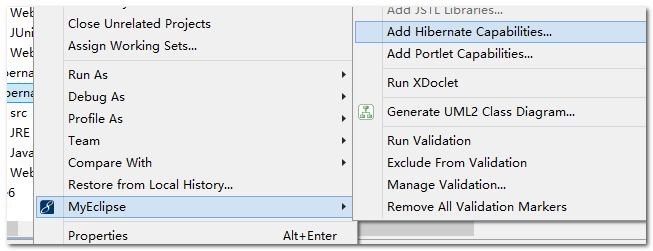


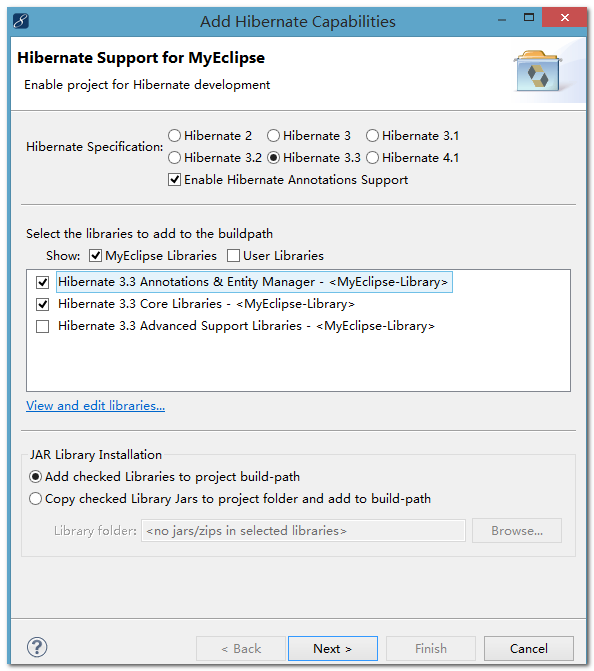
第二步：建立存放生成Hibernate相关内容的web项目临时载体

新建web工程

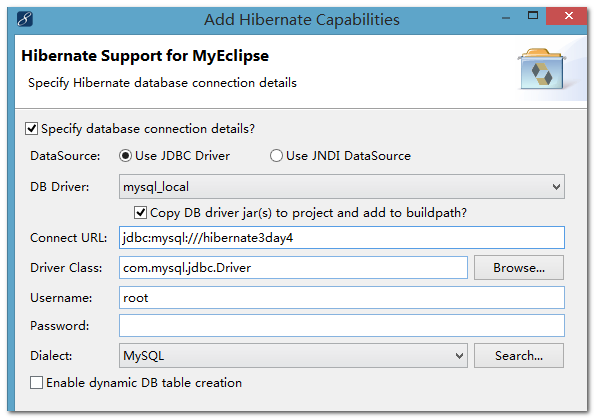


对web项目添加Hibernate能力支持（目的是导入必要的jar，--是myeclipse给你导入的）

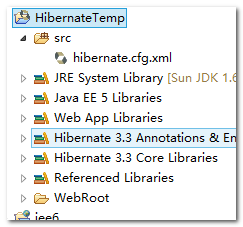


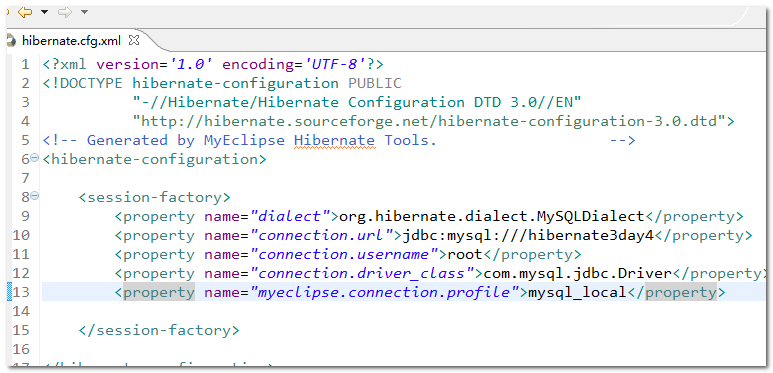


上面用来添加jar包的



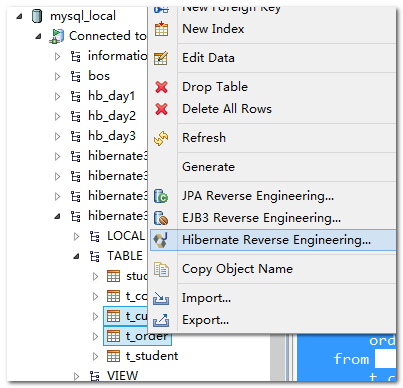
自动生成了 依赖包 以及核心配置文件：



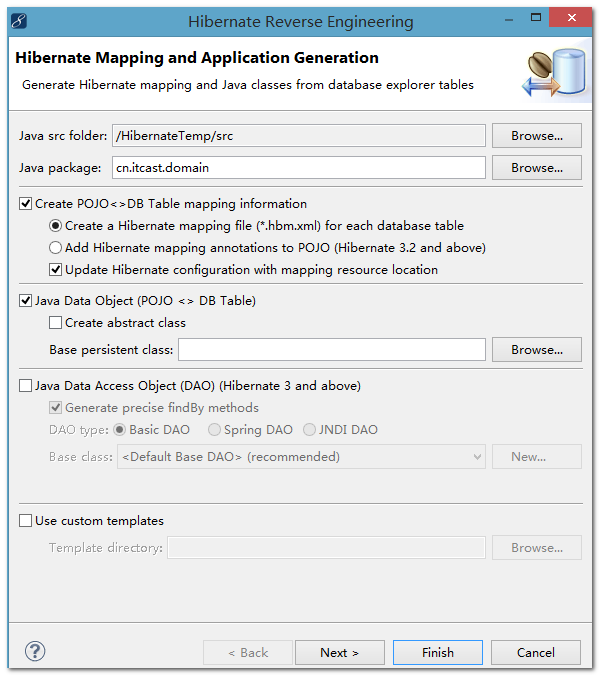


第三步：反转生成

切换回数据视图。

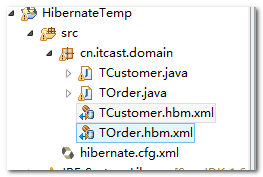


生成hbm：

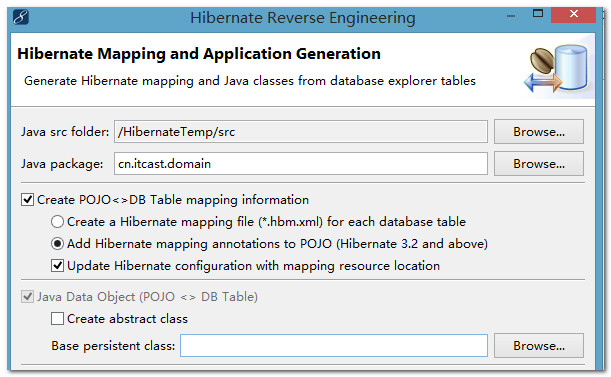


建议输入包：你将来工程用的什么包，你这里就指定什么包。如果你不指定，则里面的内容需要修改。

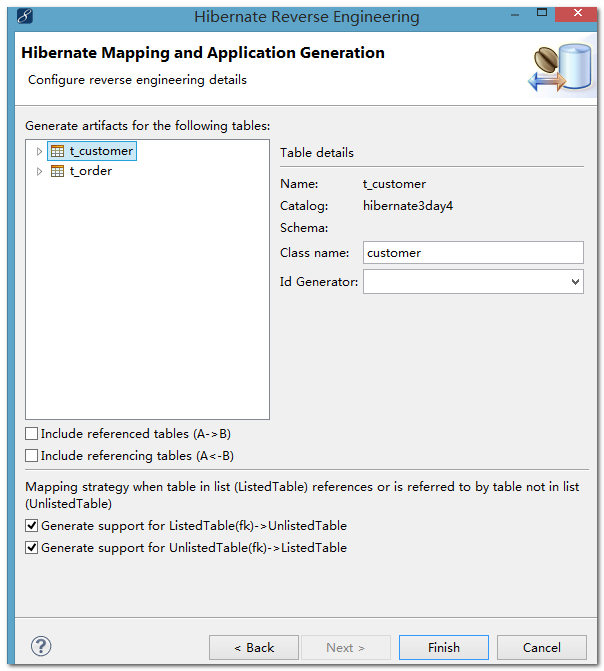
结果：



生成JPA注解方式：

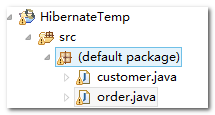


修改默认生成的类名：



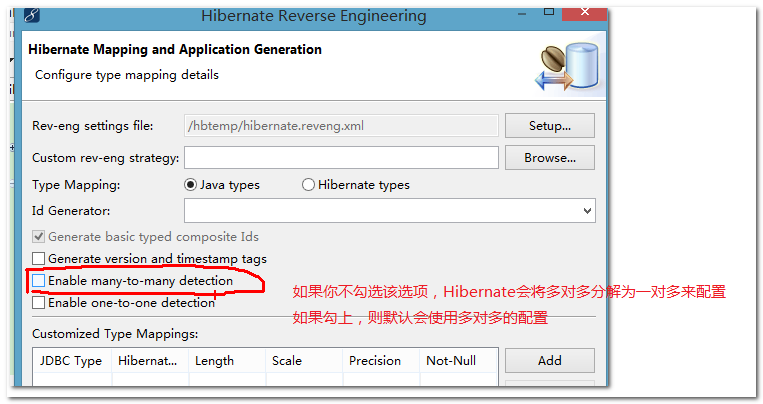
**自定义的类名要加上包名**

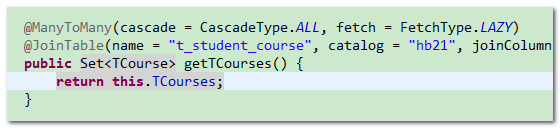
生成结果：



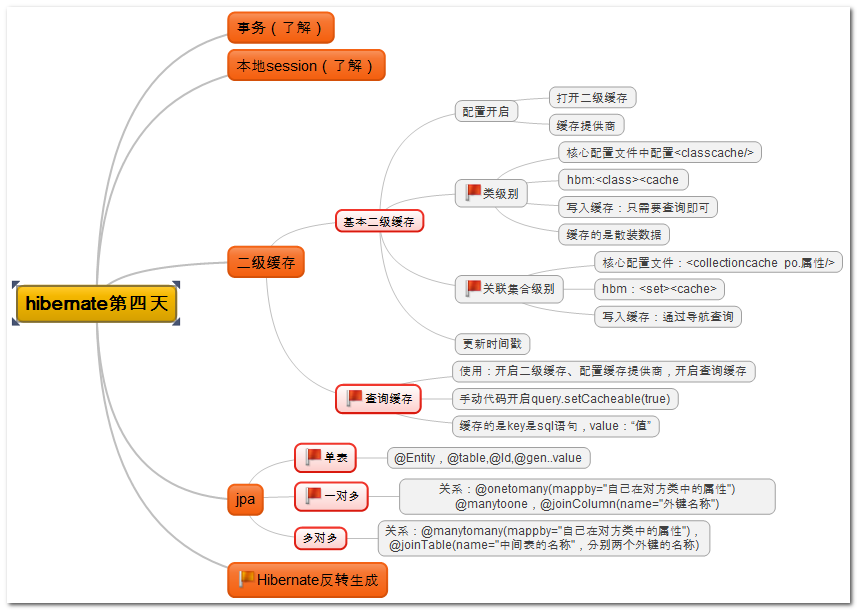
最后,你就可以将,生成东东,拷贝到你的正式工程中了..

多对多注意：





# 小结和重点



1. 缓存的概念自己体会
2. 二级缓存的配置和使用
3. 查询缓存的配置和使用
4. ehcache
5. jpa-主要熟悉单表，一对多
6. 反转生成

整个Hibernate的核心内容：

1. CRUD（save,update,快照更新,delete,get,load,query.list）
2. 理论：一级缓存，快照原理，二级缓存、查询缓存。
3. 性能优化：抓取策略、缓存的优化
4. 多表：导航查询（抓取策略：fetch、lazy、batch）（当单表用）
5. Jpa会配置