 学了两天的hibernate锁机制，今天写个总结。hibernate锁机制包括悲观锁和乐观锁1.悲观锁：它指的是对数据被外界修改持保守态度。假定任何时刻存取数据时，都可能有另一个客户也正在存取同一笔数据，为了保持数据被操作的一致性，于是对数据采取了数据库层次的锁定状态，依靠数据库提供的锁机制来实现。基于jdbc实现的数据库加锁如下：0select * from account where name="Erica" for 0update.在更新的过程中，数据库处于加锁状态，任何其他的针对本条数据的操作都将被延迟。本次事务提交后解锁。 而hibernate悲观锁的具体实现如下：String sql="查询语句". Query query=session.createQuery(sql). query.setLockMode("对象"，LockModel.UPGRADE). 说到这里，就提到了hiernate的加锁模式：LockMode.NONE ：无锁机制。 LockMode.WRITE ：Hibernate在Insert和Update记录的时候会自动获取。 LockMode.READ ：Hibernate在读取记录的时候会自动获取。 这三种加锁模式是供hibernate内部使用的，与数据库加锁无关 LockMode.UPGRADE：利用数据库的for 0update字句加锁。 在这里我们要注意的是：只有在查询开始之前（也就是hiernate生成sql语句之前）加锁，才会真正通过数据库的锁机制加锁处理。否则，数据已经通过不包含for updata子句的sql语句加载进来，所谓的数据库加锁也就无从谈起。 但是，从系统的性能上来考虑，对于单机或小系统而言，这并不成问题，然而如果是在网络上的系统，同时间会

有许多联机，假设有数以百计或上千甚至更多的并发访问出现，我们该怎么办？如果等到数据库解锁我们再进行下面的操作，我们浪费的资源是多少？--这也就导致了乐观锁的产生。 2.乐观锁： 乐观锁定（optimistic locking）则乐观的认为资料的存取很少发生同时存取的问题，因而不作数据库层次上的锁定，为了维护正确的数据，乐观锁定采用应用程序上的逻辑实现版本控制的方法。 例如若有两个客户端，A客户先读取了账户余额100元，之后B客户也读取了账户余额100元的数据，A客户提取了50元，对数据库作了变更，此时数据库中的余额为50元，B客户也要提取30元，根据其所取得的资料，100-30将为70余额，若此时再对数据库进行变更，最后的余额就会不正确。 在不实行悲观锁定策略的情况下，数据不一致的情况一但发生，有几个解决的方法，一种是先更新为主，一种是后更新的为主，比较复杂的就是检查发生变动的数据来实现，或是检查所有属性来实现乐观锁定。 Hibernate 中透过版本号检查来实现后更新为主，这也是Hibernate所推荐的方式，在数据库中加入一个VERSON栏记录，在读取数据时连同版本号一同读取，并在更新数据时递增版本号，然后比对版本号与数据库中的版本号，如果大于数据库中的版本号则予以更新，否则就回报错误。 以刚才的例子，A客户读取账户余额1000元，并连带读取版本号为5的话，B客户此时也读取账号余额1000元，版本号也为5，A客户在领款后账户余额为500，此时将版本号加1，版本号目前为6，而数据库中版本号为5，所以予以更新，更新数据库后，数据库此时余额为500，版本号为6，B客户领款后要变更数据库，其版本号为5，但是数据库的版本号为6，此时不予更新，B客户数据重

新读取数据库中新的数据并重新进行业务流程才变更数据库。 以Hibernate实现版本号控制锁定的话，我们的对象中增加一个version属性，例如：public class Account { private int version. .... public void setVersion(int version) { this.version = version. } public int getVersion() { return version. } ....}而在映像文件中，我们使用optimistic-lock属性设定version控制，属性栏之后增加一个标签，如下： optimistic-lock="version"> .... 设定好版本控制之后，在上例中如果B 客户试图更新数据，将会引发StableObjectStateException例外，我们可以捕捉这个例外，在处理中重新读取数据库中的数据，同时将 B客户目前的数据与数据库中的数据秀出来，让B客户有机会比对不一致的数据，以决定要变更的部份，或者您可以设计程式自动读取新的资料，并重复扣款业务流程，直到数据可以更新为止，这一切可以在背景执行，而不用让您的客户知道。 但是乐观锁也有不能解决的问题存在：上面已经提到过乐观锁机制的实现往往基于系统中的数据存储逻辑，在我们的系统中实现，来自外部系统的用户余额更新不受我们系统的控制，有可能造成非法数据被更新至数据库。因此我们在做电子商务的时候，一定要小心的注意这项存在的问题，采用比较合理的逻辑验证，避免数据执行错误。 也可以在使用Session的load()或是lock()时指定锁定模式以进行锁定。 如果数据库不支持所指定的锁定模式，Hibernate会选择一个合适的锁定替换，而不是丢出一个例外