**Hibernate的性能优化**

Hibernate是对JDBC的轻量级封装，因此在很多情况下Hibernate的性能比直接使用JDBC存取数据库要低。然而，通过正确的方法和策略，在使用Hibernate的时候还是可以非常接近直接使用JDBC时的效率的，并且，在有些情况下还有可能高于使用JDBC时的执行效率。

AD： 在进行Hibernate性能优化时，需要从以下几个方面进行考虑：

数据库设计调整。

HQL优化。

API的正确使用(如根据不同的业务类型选用不同的集合及查询API)。

主配置参数(日志、查询缓存、fetch\_size、batch\_size等)。

映射文件优化(ID生成策略、二级缓存、延迟加载、关联优化)。

一级缓存的管理。

针对二级缓存，还有许多特有的策略。

事务控制策略。

数据的查询性能往往是影响一个应用系统性能的主要因素。对查询性能的影响会涉及到系统软件开发的各个阶段，例如，良好的设计、正确的查询方法、适当的缓存都有利于系统性能的提升。

系统性能的提升设计到系统中的各个方面，是一个相互平衡的过程，需要在应用的各个阶段都要考虑。并且在开发、运行的过程中要不断地调整和优化才能逐步提升系统的性能。

**设计阶段的考虑问题**

一个良好的数据库结构有利于系统性能的提升。这里所说的良好结构的数据库并不单纯是指满足数据库设计范式的数据库结构。这是因为，按照数据库范式所设计的数据库只能说在结构上是最优的，没有冗余数据等问题，但在生产过程中并不一定能获得最佳的性能。有时候适当地增加一些数据的冗余虽然增加了数据维护的难度，但可以极大地简化业务的查询，提高数据检索的效率。

在使用Java访问数据库的时候，还存在另外一个问题，就是面向对象的Java语言与关系型数据库之间的矛盾。在这两者之间必然要涉及到一个相互转化的问题，对于这个问题是否能够正确的处理也是影响系统性能的一个重要因素。

综合以上提出的各种问题，在数据库设计阶段要综合考虑以下三个方面的因素。

Java建模

在建立Java对象模型的时候，要考虑数据库持久化的方便性，所建立的Java对象模型应该可以很容易地被数据所存储，并且数据库中表的结构也是越简单越好。

数据库结构

在设计数据库结构的时候也要考虑到是否可以很容易地用Java对象去表示。这里并不是简单的一个表对应一个对象的直接转换，更重要的是转换后的Java对象应该能够描述出数据间的关系。

所以在设计阶段，对于Java对象和数据库结构要进行综合考虑，也就是可以从两个方向进行考虑，毕竟两者之间不是一个时代的产物，设计的结果应该在两者之间达到一个平衡，虽然不能每一方都达到最优，但也不能造成有一方结构很差的情况。就像装水的木桶，最矮的板子决定水桶的容量。

业务需求

前面两个因素都是纯技术方面的考虑，在设计的过程中，更重要的是要紧扣业务需求。这是因为任何的软件系统都是以业务为中心的，那么对于系统的设计也不例外，在设计的阶段就应该考虑业务实现的方便性以及执行的效率。一个良好的结构设计不但使业务功能的实现变得非常容易并且可以避免很多复杂的操作，还可以达到提升系统性能的目的。

设计阶段是整个应用系统开发中的根基，其对软件的影响仅次于对系统需求的把握。所以在设计阶段应该对整个软件系统有一个整体的考虑，这里所说的具体设计也只是设计阶段中的很少的一部分，综合考虑多方面的因素才能达到更佳的性能。

**[Hibernate优化问题](http://www.cnblogs.com/wangbinjr/archive/2010/04/26/2354222.html)**

初用HIBERNATE的人也许都遇到过性能问题，实现同一功能，用HIBERNATE与用JDBC性能相差十几倍很正常，如果不及早调整，很可能影响整个项目的进度。

　　大体上，对于HIBERNATE性能调优的主要考虑点如下:

　　\* 数据库设计调整

　　\*　HQL优化

　　\* API的正确使用(如根据不同的业务类型选用不同的集合及查询API)

　　\* 主配置参数(日志，查询缓存，fetch\_size, batch\_size等)

\* 映射文件优化(ID生成策略，二级缓存，延迟加载，关联优化)

　　\* 一级缓存的管理

　　\* 针对二级缓存，还有许多特有的策略

　　\* 事务控制策略。

　　1、 数据库设计

　　a) 降低关联的复杂性

　　b) 尽量不使用联合主键

　　c) ID的生成机制，不同的数据库所提供的机制并不完全一样

　　d) 适当的冗余数据，不过分追求高范式

　　2、 HQL优化

　　HQL如果抛开它同HIBERNATE本身一些缓存机制的关联，HQL的优化技巧同普通的SQL优化技巧一样，可以很容易在网上找到一些经验之谈。

　　3、 主配置

　　a) 查询缓存，同下面讲的缓存不太一样，它是针对HQL语句的缓存，即完全一样的语句再次执行时可以利用缓存数据。但是，查询缓存在一个交易系统(数据变更频繁，查询条件相同的机率并不大)中可能会起反作用:它会白白耗费大量的系统资源但却难以派上用场。

　　b) fetch\_size，同JDBC的相关参数作用类似，参数并不是越大越好，而应根据业务特征去设置

　　c) batch\_size同上。

　　d) 生产系统中，切记要关掉SQL语句打印。

　　4、 缓存

　　a) 数据库级缓存:这级缓存是最高效和安全的，但不同的数据库可管理的层次并不一样，比如，在ORACLE中，可以在建表时指定将整个表置于缓存当中。

　　b) SESSION缓存:在一个HIBERNATE SESSION有效，这级缓存的可干预性不强，大多于HIBERNATE自动管理，但它提供清除缓存的方法，这在大批量增加/更新操作是有效的。比如，同时增加十万条记录，按常规方式进行，很可能会发现OutofMemeroy的异常，这时可能需要手动清除这一级缓存:Session.evict以及Session.clear

　　c) 应用缓存:在一个SESSIONFACTORY中有效，因此也是优化的重中之重，因此，各类策略也考虑的较多，在将数据放入这一级缓存之前，需要考虑一些前提条件:

　　i. 数据不会被第三方修改(比如，是否有另一个应用也在修改这些数据?)

　　ii. 数据不会太大

　　iii. 数据不会频繁更新(否则使用CACHE可能适得其反)

　　iv. 数据会被频繁查询

　　v. 数据不是关键数据(如涉及钱，安全等方面的问题)。

　　缓存有几种形式，可以在映射文件中配置:read-only(只读，适用于很少变更的静态数据/历史数据)，nonstrict-read-write，read-write(比较普遍的形式，效率一般)，transactional(JTA中，且支持的缓存产品较少)

　　d) 分布式缓存:同c)的配置一样，只是缓存产品的选用不同，在目前的HIBERNATE中可供选择的不多，oscache, jboss cache，目前的大多数项目，对它们的用于集群的使用(特别是关键交易系统)都持保守态度。在集群环境中，只利用数据库级的缓存是最安全的。

　　5、 延迟加载

　　a) 实体延迟加载:通过使用动态代理实现

　　b) 集合延迟加载:通过实现自有的SET/LIST，HIBERNATE提供了这方面的支持

　　c) 属性延迟加载:

　　6、 方法选用

　　a) 完成同样一件事，HIBERNATE提供了可供选择的一些方式，但具体使用什么方式，可能用性能/代码都会有影响。显示，一次返回十万条记录(List/Set/Bag/Map等)进行处理，很可能导致内存不够的问题，而如果用基于游标(ScrollableResults)或Iterator的结果集，则不存在这样的问题。

　　b) Session的load/get方法，前者会使用二级缓存，而后者则不使用。

　　c) Query和list/iterator，如果去仔细研究一下它们，你可能会发现很多有意思的情况，二者主要区别(如果使用了Spring，在HibernateTemplate中对应find,iterator方法):

　　i. list只能利用查询缓存(但在交易系统中查询缓存作用不大)，无法利用二级缓存中的单个实体，但list查出的对象会写入二级缓存，但它一般只生成较少的执行SQL语句，很多情况就是一条(无关联)。

　　ii. iterator则可以利用二级缓存，对于一条查询语句，它会先从数据库中找出所有符合条件的记录的ID，再通过ID去缓存找，对于缓存中没有的记录，再构造语句从数据库中查出，因此很容易知道，如果缓存中没有任何符合条件的记录，使用iterator会产生N+1条SQL语句(N为符合条件的记录数)

　　iii. 通过iterator，配合缓存管理API，在海量数据查询中可以很好的解决内存问题，如:

　　while(it.hasNext()){

　　YouObject object = (YouObject)it.next();

　　session.evict(youObject);

　sessionFactory.evice(YouObject.class, youObject.getId());

　　}

　　如果用list方法，很可能就出OutofMemory错误了。

　　iv. 通过上面的说明，我想你应该知道如何去使用这两个方法了。

　　7、 集合的选用

　　在HIBERNATE 3.1文档的“19.5. Understanding Collection performance”中有详细的说明。

　　8、 事务控制

　　事务方面对性能有影响的主要包括:事务方式的选用，事务隔离级别以及锁的选用

　　a) 事务方式选用:如果不涉及多个事务管理器事务的话，不需要使用JTA，只有JDBC的事务控制就可以。

　　b) 事务隔离级别:参见标准的SQL事务隔离级别

　　c) 锁的选用:悲观锁(一般由具体的事务管理器实现)，对于长事务效率低，但安全。乐观锁(一般在应用级别实现)，如在HIBERNATE中可以定义VERSION字段，显然，如果有多个应用操作数据，且这些应用不是用同一种乐观锁机制，则乐观锁会失效。因此，针对不同的数据应有不同的策略，同前面许多情况一样，很多时候我们是在效率与安全/准确性上找一个平衡点，无论如何，优化都不是一个纯技术的问题，你应该对你的应用和业务特征有足够的了解。

　　9、 批量操作

　　即使是使用JDBC，在进行大批数据更新时，BATCH与不使用BATCH有效率上也有很大的差别。我们可以通过设置batch\_size来让其支持批量操作。

　　举个例子，要批量删除某表中的对象，如“delete Account”，打出来的语句，会发现HIBERNATE找出了所有ACCOUNT的ID，再进行删除，这主要是为了维护二级缓存，这样效率肯定高不了，在后续的版本中增加了bulk delete/update，但这也无法解决缓存的维护问题。也就是说，由于有了二级缓存的维护问题，HIBERNATE的批量操作效率并不尽如人意!