# 基于 HIBERNATE 的数据库访问优化

陈 正 举
（工业和信息化部电信研究院 北京 100083）

**摘 要** 数据库的访问机制和效率是影响信息管理系统性能的重要因素。在分层应用体系系统结构中，数据库访问通过独立的逻辑层次——数据持久层来实现；持久层的访问机制直接影响信息系统访问数据库的效率。SSH 是一种典型的基于 J2EE 的轻量级软件开发分层体系结构，采用基于 ORM 原理的插件 HIBERNATE 实现数据持久层。可尝试改善其持久层访问机制来优化数据库运行效率。结合应用实例，对 HIBERNATE 的 ORM 机制、HQL 查询语言和缓存机制的优化机制进行了探索；对基于 HIBERNATE 的持久层数据库访问效率优化的策略进行了尝试和改进，并将其放到招生系统数据库访问优化的实践进行检验。结合实践结果提出了具体建议，希望能够对类似基于 J2EE 轻量级软件开发架构的信息系统分层数据库访问机制的优化提供参考。

**关键词** 招生系统 数据持久层 ORM HIBERNATE DAO 模式

**中图分类号** TP311.13 **文献标识码** A

## HIBERNATE-BASED DATABASE ACCESS OPTIMIZATION

Chen Zhengju
(*China Academy of Telecommunication Research of MIIT, Beijing 100083, China*)

**Abstract** Database access mechanism and efficiency are important factors that influence the performance of the information management system. In tiered application architectures, database access is achieved through an independent logic level——data persistence layer; the persistence layer access mechanism can directly influence the efficiency of database accessed by the information system. SSH is a typical J2EE-based light-weight software development tired architecture. It uses HIBERNATE, a plug-in based on the principle of ORM, to implement the data persistence layer. The paper tries to optimize database operation efficiency by improving its data persistence layer access mechanism. In the paper, a real case is integrated to explore HIBERNATE ORM mechanism, HQL query language and cache mechanism's optimization mechanism. HIBERNATE-based persistent layer database access efficiency optimization policy is tried and improved, then checked by being applied to enrollment system database access optimization practice. Practice results are cited for detailed suggestions. The author hopes to provide references for information system layered database access mechanism optimization based on J2EE light-weight software development architectures and the like.

**Keywords** Enrollment system Data persistent layer ORM HIBERNATE DAO mode

## 0 引 言

J2EE 是一套面向企业应用的体系结构。J2EE 通过提供中间层集成框架来满足多种需求。在 J2EE 的多层开发体系中，数据持久层随着应用系统开发规范的进步从小到大，作用越来越被开发人员所重视。

HIBERNATE 是一种的持久层框架，采用 ORM 机制实现数据持久化。HIBERNATE 是一种 ORM 映射工具，它不仅提供了从 JAVA 类到数据表的映射，也提供了数据查询和恢复等机制[1]。

招生系统就是一个实现了轻量级 J2EE 架构（SSH）的教学管理信息系统。考生可以通过系统进行网上报名、资格确认等活动，教师则可以通过系统对考生、考试等进行管理，从而提高招生报名工作各环节的效率，降低管理成本。随着使用系统的用户不断增加，用户处理吞吐量增大，对于 HIBERNATE 访问数据库在具体系统中的实现更应该予以关注和改进，以期提高应用程序对数据库的访问效率，改进系统的性能。HIBERNATE 针对 ORM 提供了多种的映射方式，针对数据库访问提供了多种缓存机制，如何实现映射和利用各种缓存机制都会对系统和数据库的性能带来影响。那么，探索具体系统中数据持久层的较优实现方式来提高应用程序对数据库的访问效率就显得比较迫切，也是具有现实意义的。

## 1 数据持久化与 ORM

### 1.1 数据持久化与持久层技术

#### 1.1.1 数据的持久化

数据的持久化就是将数据转化为持久的数据。持久的数据就是指保存在掉电后也不会丢失数据的存储设备中的数据。在计算机中这样的设备就是硬盘（磁盘）。那么数据的持久化就

收稿日期：2011－06－02。陈正举，硕士，主研领域：计算机应用。

是将数据保存到可掉电的存储设备中的过程。在计算机中这样的过程通常就是将内存中的数据保存到磁盘的操作。

在计算机领域,将数据固化到磁盘的数据持久化的手段多种多样,可以将数据存储在关系数据库或者磁盘文件和 XML 文件中[2]。

1.1.2　持久层技术

持久层是针对数据持久化提出来的。持久层是软件开发模型领域的一个基本概念。软件开发技术的升级带来了软件体系结构的升级,人们已经不满足仅仅应用层与数据层分离的软件体系结构,已经向更多、更细致的层次划分探索更好的实践。软件体系结构的层次模型也就经历了图 1 描述的几个阶段。

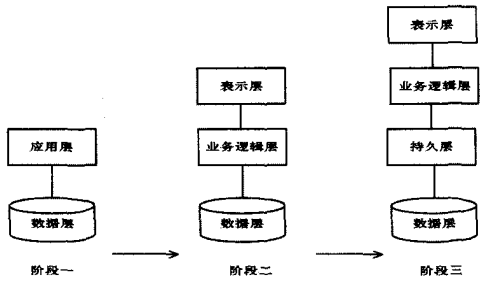


图 1　软件体系结构模型演变图

阶段一实现了应用层与数据层的分离。数据层用来保存需要进行持久化的数据,应用层专门用来接收用户输入,进行业务逻辑处理然后反馈处理结果。阶段二实现了业务逻辑与显示逻辑的分离,就是大家熟知的 MVC 模式。表示层用来与用户交互,业务逻辑层用来处理业务逻辑和进行数据的持久化操作。较阶段一来讲,降低了程序开发的复杂度。阶段三是在阶段二的基础上发展起来的。体现了人们对软件开发难度进一步降低和软件开发效率的追求。将阶段二中的业务逻辑层的两项功能分别部署到拆分后的两个层次中,即业务逻辑层专门负责核心业务逻辑的处理,持久层专门负责对象的持久化操作。这样一方面降低了业务逻辑层的复杂度,一方面又可以将数据的持久化工作交给其他组件来完成。

SSH 就是一种实现了分层体系结构的轻量级 J2EE 框架。前端采用 STRUTS MVC 框架,中间层采用 SPRING,后台采用 HIBERNATE。这样就采用了三种技术分别实现了软件体系阶段三所描述的三个层次的功能。

通过以上的介绍可以看到,持久层是更细致划分的软件结构体系中的一个逻辑层次,而不仅仅是简单的持久化操作[3],应该从整个系统开发的角度来进行统筹。它应该与系统的其他部分有较为清晰和严格的边界,并能够提供完整的数据持久化的解决方案。这就是持久层技术。

### 1.2　ORM 概述

ORM 全称是 Object-Relational Mapping,中文就是"对象-关系映射"。它的实质就是将关系数据(库)中的业务数据用对象的形式表示出来,并通过面向对象的方式将这些对象组织起来,实现系统业务逻辑的过程[4]。因此在对象-关系映射中涉及到的两个关键点是:OBJECT(对象)和 RELATION(关系),分别代表了 JAVA 编程中的对对象的操作和对关系型数据库的访问。

### 1.3　HIBERNATE 概述

HIBERNATE 是采用 ORM 机制实现数据持久层的 JAVA 组件[5]。HIBERNATE 是一种新的 ORM 映射工具,它不仅提供了从 JAVA 类到数据表的映射,也提供了数据查询和恢复等机制。

图 2 显示了 HIBERNATE 的体系结构。

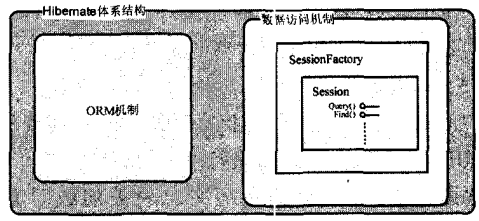


图 2　HIBERNATE 体系结构

## 2　基于 HIBERNATE 的优化方法和优化指标

### 2.1　HIBERNATE 映射优化

HIBERNATE 的 ORM 机制帮助解决了数据库中的关系数据向 JAVA 对象的映射。使得关系数据能够被以对象的形式持有,同时又能够通过面向对象操作将对象中的数据持久化到数据库。比如将一张数据库表的关系型数据装载到持久类以供查询,而对该持久类数据的任何修改又可以通过调用 SESSION.save()方法持久到数据库的这张表中。可以说这是实现了对象与表之间一对一的对象←→关系映射。然而实际应用中我们更多的时候需要很多的表与映射类,并且要处理表与表之间的关系,而仅仅实现了表到对象映射的映射类是不能体现表之间关系的。HIBERNATE 给这个问题的解决提供了很好的方法——"对象←→关联"映射。顾名思义,这个方法就是在"对象←→关系"映射了表结构的基础上来实现表间关系到对象间关系的映射。

那么,如何找到表与表之间的关系、如何把握和配置对象之间的关联就能够决定不同的编程成本和数据库访问策略,直接影响程序质量和数据库访问效率。因此,根据应用的实际情况做好 HIBERNATE 对象关联的映射能够极大地改善程序质量和对数据库的访问效率。

### 2.2　SQL 语句的优化

2.2.1　使用正确的查询方法

get()和 load()方法用来得到单个持久化对象;还有就是 QUERY 对象的 list()和 iterator()方法。get()和 load()方法的主要区别在于对二级缓存的使用上。load()方法会使用二级缓存,而 get()方法在一级缓存没有找到的情况下会直接查询数据库,不会去二级缓存中查找。在使用中,对使用了二级缓存的对象进行查询时最好使用 load()方法。

2.2.2　使用正确的抓取策略

抓取策略就是指当应用程序需要利用关联关系进行对象获取的时候,HIBERNATE 获取对象的策略。

HIBERNATE3 提供了这样几种抓取策略:

① 连接抓取:就是在 SELECT 语句中使用外连接的方式来获得关联对象。通过循环扫描当前对象对应的表来找到与关联表对应的记录,如果找不到则给关联表的字段附空值。

② 查询抓取:通过外键的方式执行数据库查询。HIBERNATE 通过另外一条 SELECT 语句来抓取当前对象的关联对象,通常这个 SELECT 语句要在访问到关联对象的时候才会执行。

③ 子查询抓取:HIBERNATE 通过另外一条 SELECT 语句来抓取当前对象的关联对象的方式。与查询抓取的区别在于它

所采用的 SELECT 语句的方式为子查询而不是通过外连接。

④ 批量抓取:对查询抓取的优化,根据主键或者外键的列表通过单条 SELECT 语句实现对象的批量抓取。

2.2.3 正确运用抓取时机

有了抓取策略还不够,要想提高系统的性能还要把握好抓取时机,HIBERNATE 的抓取时机有以下的选择:

① 立即抓取:关联对象在宿主对象被加载时会被立即加载。

② 延迟集合抓取:关联对象只有在被应用程序访问到时才会被抓取,这是集合关联对象的默认行为。

③ 延迟代理抓取:返回的是单值关联对象,不在对其进行 get()操作时抓取,而是直到调用其某个方法的时候才会抓取这个对象。

④ 延迟属性加载:在关联对象某个属性被访问的时候才进行关联对象该属性的抓取。

从以上的介绍中可以看到,延迟加载的关联对象的访问应该在其宿主对象存在的时候进行,如果其宿主对象(或 SESSION)已经结束,那么在访问关联对象的时候就会抛出异常。因此应该在事务提交之前完成对关联对象的访问。

## 2.3 HIBERNATE 缓存的应用

HIBERNATE 提供的一级、二级缓存策略能够在不同层级实现持久化对象的缓存,对象缓存集中体现了 ORM 的优越性,因为这种缓存属于细颗粒度,针对表的记录级别,透明化访问,在不改变程序代码的情况下可以极大提升 WEB 应用对数据库的访问性能。因此,对于缓存的正确使用是 HIBERNATE 性能优化的重中之重。

HIBERNATE 的一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。二级缓存是一种可选的插件缓存,在使用之前需要进行一些配置。借助二级缓存对持久化对象散装数据的缓存,可以避免数据库的重复查询,减少数据库的请求次数,因此能够大幅度地节省被缓存数据的查询时间。进而改善了数据库的运行性能。

同时,二级缓存的细节配置关系着缓存自身的效率,比如,持久化对象的缓存时间等。应该根据信息系统的实际使用情况,正确运用缓存。

## 2.4 性能调整的指标

本文讨论的是基于 HIBERNATE 的持久层数据库访问优化,因此可以将性能比较的指标分为两类:一类是应用服务器端的,一类是数据库服务器端的。

在数据库服务器端,系统使用的是 ORACLE 数据库。通常认为评价 ORACLE 数据库系统的性能指标主要有系统吞吐量和数据库用户响应时间、数据库命中率、内存使用情况以及所需的磁盘 I/O 量[6]。

服务器端衡量的性能指标主要是与操作系统相关的。如:磁盘读取页数(Page/sec)、每秒软性页面失效数(Page Faults/sec)、页的硬故障(Page Reads/sec)、进程分配内存数(Private Bytes)、内存页数(Work Set)、可用内存(Available Bytes)、处理器队列线程数(Processor Queue Length)、处理器使用率(% processor time)、物理磁盘利用率(Physical Disk. % Disk Time)等。

# 3 招生信息系统优化策略分析

## 3.1 招生信息系统

3.1.1 系统综述

招生系统是学院教学信息管理平台的重要组成部分,承担着接收考生网上报名和考试、考生管理等重要工作。对提高学院的招生工作的服务质量和加强招生工作的管理、监控力度都起了重要作用。系统采用了 SSH 这种轻量级的 J2EE 应用架构,实现了表现层、业务逻辑层和数据持久层的分离,提高了系统的可扩展性和可维护性。

3.1.2 报名信息浏览功能简介

学生在报名系统成功注册并录入报名信息后,即可通过报名信息浏览功能查看自己的报名信息是否正确,如果有误可以使用修改报名信息功能来进行修改。网上报名流程如图 3 所示。

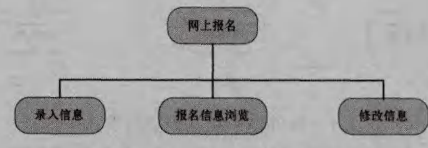


图3 网上报名功能模块图

招生系统的学生报名信息浏览功能的并发量是很大的,而且学生报名后会反复检查确认信息正确与否,因此这个功能是系统中一个既重要又频繁使用的功能。

## 3.2 系统性能分析

3.2.1 压力测试用例和性能指标

3.2.1.1 用例环境配置

(1) 应用服务器

硬件:DELL GX270 Intel Pentium 4 Processor 处理器 1G 内存 Intel(R) PRO/1000 MT Network Connection。

软件:Windows 2003 Server Tomcat 5.5。

(2) 数据库服务器

硬件:i386Architecture Intel(R) Xeon(TM) CPU 2.40GHz 2G 内存 82546EB Gigabit Ethernet Controller。

软件:SUSE Linux Enterprise Server 10 (i586) Oracle Database 10g Enterprise Edition Release 10.2.0.1.0。

3.2.1.2 用例简介

动作:登录系统→浏览报名信息→再次浏览报名信息→退出系统。

测试场景:模拟 50 个用户同时在线。用户登录系统两次浏览自己的报名信息。

3.2.1.3 性能指标

性能指标可参见表 1。

表1 优化前性能指标表

| 内容 \ 数值(scale=1) | 优化前 |
|---|---|
| 应用服务器端 | |
| 每秒点击次数 | 106.15 |
| 吞吐量(字节/秒) | 667454 |
| %物理磁盘利用率 | 0.19 |
| 物理磁盘每秒读操作次数 | 3.85 |

| 数值(scale = 1) / 内容 | 优化前 |
|---|---|
| 磁盘读取页数 | 11.44 |
| 每秒软性页面失效数 | 10.29 |
| 进程分配内存数 | 11.20 |
| 内存页数 | 34.24 |
| 可用内存 | 16.22 |
| 处理器使用率 | 0.75 |
| 处理器队列线程数 | 17.13 |
| 第一次浏览响应时间(秒) | 0.76 |
| 第二次浏览响应时间(秒) | 0.3 |
| 每秒处理事务数 | 1.39 |
| 数据库服务器端 | |
| 总解析数量 | 6274.25 |
| 解析时间 | 1.26 |
| 硬解析时间 | 0.57 |
| 缓存命中率 | 88.2% |
| SQL 执行总数 | 6205.25 |
| 内存排序数 | 816.5 |
| 处理器调用次数 | 31526.63 |
| 并发等待时间 | 7.75 |
| SQL 运行时内存(兆字节) | 24.5 |
| 数据库时间(秒) | 5.97 |
| % 数据库服务时间 | 96.4 |
| 数据库服务时间(秒) | 5.74 |
| 处理器时间(秒) | 5.75 |
| % 处理器服务时间 | 98.19 |
| 处理器服务时间(秒) | 5.62 |
| SQL 运行时间(秒) | 3.49 |

3.2.2　性能问题的分析

3.2.2.1　源代码数据库访问接口

浏览信息功能一共用到 15 个 HIBERNATE 的持久类,每个持久类都唯一对应数据库中的一个表。

上述所有记录的提取都是只对单个独立的持久类(单表)的,没有作任何的持久类(表)关联。这样就使得程序中充斥了大量的 DAO 代码来分别对每个持久类进行实例化,一方面使得程序变得繁琐,不利于开发和维护,另一方面使得服务器要不断地处理用户递交的大量 SQL,频繁地与数据库进行交互。因此使得第一次浏览的响应时间 0.76 和第二次浏览响应时间 0.3 比较长。

跟踪 HIBERNATE 传到数据库的 SQL 语句发现,该功能涉及到的 15 张数据库表逐一被访问过,共向数据库递交了 42 条 SQL 语句。其中,有的表还需要被重复访问。因为一个学生可以录入多条工作经验信息,相应的就有多个职位,而为了得到每个职位的类代码都必须重新读一遍职位表。这种连续提交多个单表查询的 SQL 抓取信息,再将得到的信息汇总返回给用户的查询方式就使得数据库要连续执行一个 SESSION 提交过来的不同的 SQL,执行一个返回一个的数据,然后再执行下一个。这样就使得数据库连续忙于解析不同的 SQL 语句和为一个连接提供服务。因此,数据库解析查询语句的数量 5674.25、查询语句的解析时间 1.26 和查询语句的执行时间 3.49 都较高,使得数据库时间 5.97 和数据库占用 CPU 时间 5.75 都比较高。

3.2.2.2　数据抓取策略

查看 SQL 语句的写法。几乎每条 SQL 都使用了'*',遍历了表中的所有字段。这种写法在表中所有字段的信息都要用到的情况下是合理的。而事实上,以上大部分表的属性信息并没有完全被使用到。比如学生信息表中的 STU_CODE 和 PICTURE 等字段就没有被使用,而原程序在实例化映射类时没有指定具体字段,于是 HIBERNATE 将该映射类对应的数据库表中的所有字段,包括像 PICTURE 这样的重量级字段都取出来了,这些对用户来说无用的信息不仅浪费了数据库资源还耗费了应用服务器的内存。

3.2.2.3　数据库方面

(1) 各表中只有主键没有任何外键,这样各表数据不能形成参照完整性约束[12],不利于信息的完整性和维护。

(2) 由于数据库端的缓冲区设置比较小为 128M,因此没能给各表的查询缓存足够信息,因此数据库的缓存命中率 88.2% 偏低。

## 4　优化实例分析

结合招生系统学生信息浏览功能,基于具体 ORM 工具 HIBERNATE 的系统性能优化分析实例。

### 4.1　HIBERNATE 映射优化

(1) 整理各表之间的关联关系

根据系统业务层和显示层的逻辑,得到如下表间关系:

一对一的关联有:学生与注册信息。

一对多的关联有:学生与简历、学生与大学所修课程、学生与项目经验、学生与工作经验、学生与技能。

多对一的关联有:学生与大学(因为 GDS_STUDENTINFO 表只包含了学生的毕业学校信息,所以一个学生只对应了一个学校)、注册信息与专业、注册信息与研究方向、学生技能与技能类别信息、学生技能与技能信息、工作经验与职位、学生与薪水水平、职位与职位类别信息。

对以上表间关系不甚合理之处加以调整,调整各表主键及外键的相互关系。

(2) 配置持久类间的关联关系

修改相应类的 ORM 映射类(*.JAVA)和映射文件(*.HBM.XML)。将类间关系都配置为双向,且由"多"的一方负责数据维护效率更好一些。

(3) 对程序进行调整

上面谈到该功能对持久类的操作贯穿于系统业务层和表示层,没有达到轻量级 J2EE 框架 MVC 模式的要求。这就使得服务器在业务层和编译表示层 JSP 文件时要进行两次数据库连接操作,不仅由于增加了数据库的访问次数而降低了系统反应时间,而且因为业务代码散布在 JSP 页面中也不利于代码的理解与维护。因此,将表示层需要的数据在业务层进行提取,统一存入相应类别的信息表单,再提交到表示层,这样就无需在表示层对持久类进行操作,避免了又一次的数据库连接。

(4) 小结

应用服务器方面,由于用户需要的信息通过持久类的关联第一时间都被装载到服务器,因此有效地提高了服务器的事务处理能力,系统吞吐量由 667454 增加到 772866.8;物理磁盘利用率由 0.19 降到 0.08,物理磁盘每秒读取次数由 3.85 降到

3.05,说明服务器第一时间将该功能涉及到的各相关类加载到内存,节省了磁盘开销。同时,也增加了内存的使用效率,磁盘读取页数由11.43降低到4.5,每秒软性页面失效数由10.29降低到5.8;在各项反应时间基本不变情况下,增加了每秒事务处理能力,每秒平均处理事务数由1.39增加到1.6。但由于服务器一次加载到内存的数据较之前明显增加,因此,服务器工作集数量由34.24增加到69.40,表明内存负荷有较大增加;同时,也更多地占用了处理器时间,处理器使用率由0.75上升到0.81。数据库方面,由于递交到数据库的SQL数量的减少以及一级缓存的帮助使得数据库解析和处理SQL语句的负担明显减轻,因此,解析时间(由1.26降到0.56)和硬解析时间(由0.57降到0.01)均有明显下降,SQL语句执行时间由3.49降到3.2;同时由于加大了数据库的缓存,因此缓冲区高速缓存命中率由88.2%提高到94%。由于以上因素的共同作用使得数据库时间(由5.97降到5.2)和数据库处理器时间(由5.75降到5.12)均有明显下降,提高了数据库的访问效率。

## 4.2 HQL的优化

### 4.2.1 抓取策略优化

(1) 优化策略

批量抓取是对查询抓取的优化方案,通过指定一个主/外键列表,HIBERNATE使用单条SELECT语句获取一批对象实例或集合,分为单端代理的批量抓取和集合代理的批量抓取。多对一的关联关系中可以设置"一"的一端为批量抓取,这样当"一"的一端的记录达到批量抓取的数量(batch-size指定的值)时,HIBERNATE会执行一次查询全部返回这些记录;而在一对多的关联关系中可以在集合级别定义批量抓取。将"一"的一端持有的"多"端集合设置为批量抓取也可以大大减少HIBERNATE的查询次数,提高查询效率。

HIBERNATE3默认使用的是查询抓取。上述实例采用的是默认抓取策略,下面将抓取策略调整为查询抓取的优化方案——批量抓取。根据HIBERNATE的查询计划,有些表需要进行多次查询,因此将这些表对应的持久类设置为批量抓取。需要修改持久类的映射文件(*.HBM.XML)。

(2) 优化结果

HIBERNATE提交的查询语句数量由上次优化后的36条减少到14条,大大地减少了数据库的访问次数。

批量查询改善了数据库的访问方式,减少了应用服务器查询数据库的次数,有效地减少了与数据库进行通信所需的系统资源,这表现在以下几个方面:内存工作集由69.40降到8.54,大大减小了内存负荷,处理器利用率(由0.81降到0.69)降低,处理器队列有小幅下降(由18.9降到14.97);同时,第一次浏览的响应时间有明显的改善(由0.88降到0.49)。数据库方面,由于递交到数据库查询语句数量的大量减少使得数据库解析查询语句的数量(由5794.12降到5280.87)、解析时间(由0.56降到0.52)、和硬解析时间(由0.01降到0.005)都有不同程度的减少;同时数据库查询语句的执行总数由6098.62降到5573.37,用户CPU的调用数也由33904.88降到30614.75。进而节省了数据库时间(由5.2降低到4.95)和数据库占用的CPU时间(由5.12降到4.86)。

### 4.2.2 抓取时机优化

#### 4.2.2.1 延迟抓取

(1) 优化策略

HIBERNATE对集合持久化对象采用延迟集合抓取,对单独持久化对象采用延迟代理抓取。比如在一对多关系中,"一"的一方拥有另一方的一个对象集合,HIBERNATE对多的一方采用的就是集合延迟抓取,而对"一"的一方采用的就是延迟代理抓取。那么,宿主对象什么时候抓取另一个对象的集合会对应用系统及数据库的性能产生影响。默认情况下HIBERNATE采用的是立即抓取,如果持久化对象之间的关联较多,那么应用程序就不得不在第一时间将所有相互关联的对象全部持久化,这就造成在对象的持久化过程中应用程序事务反应时间和数据库时间的明显波动。因此,将关联对象的抓取时机配置为延迟抓取。

(2) 优化过程

① OPENSESSIONINVIEW

SSH轻量级J2EE框架利用SPRING对HIBERNATE数据持久层的访问进行了封装,招生系统就利用了SPRING提供的DAO接口对数据持久层进行访问。那么,就将HIBERNATE的事务局限在业务方法中了。因此,一个持久类的DAO接口只对应一个HIBERNATE事务周期,当延迟加载一个持久类的关联对象时,由于取得宿主类的事务已经结束,因此无法持久化关联对象。为了避免这种情况,就需要将一次用户请求作为一个事务来进行处理。利用SPRING提供的OPENSESSIONINVIEW过滤器来实现这一点。需要修改工程的配置文件WEB.XML。

② 配置延迟抓取

修改相应类的映射文件(*.HBM.XML)。在一对多关系的"一"的一方配置集合类为延迟抓取,添加lazy = true;在一对一关系中配置单值对象的延迟代理抓取,添加lazy = proxy。

#### 4.2.2.2 属性延迟加载

(1) 优化策略

通过属性延迟加载过滤掉不使用的字段,节省数据库查询和应用响应时间。

(2) 优化过程

① 确定需要属性延迟加载的类,将持久类中一般情况下不需要使用的属性配置成延迟加载。修改持久类配置文件,在属性定义中加入lazy = true。

② 利用类增强器对上述的二进制CLASS文件进行强化处理。本文选用了当前较为流行的快速开发工具ANT来实现二进制文件的强化。

### 4.2.3 优化结果

从HIBERNATE提交到数据库的SQL可以看到,SQL语句的执行顺序发生了调整。直到程序中调用了持久类的具体方法要求返回数据时,HIBERNATE才提交相应数据库表的查询请求。比如,查询用户工作经验里职位信息的语句就从之前的第5个执行调整到了现在的第13个执行。

根据实验结果,由于在数据提取过程中过滤掉了像学生照片这样重量级的BLOB类型字段,因此服务器不需要开辟更多内存空间来存放这些用不到的属性信息,这就使得服务器不需要开辟更多的磁盘空间作为虚拟内存来处理数据,因此,磁盘读取页数明显减少(由7.71降到3.69);同时,第一次浏览响应时间也有小幅降低(由0.45降到0.42)。数据库方面,数据库并发等待时间(由3.87降到3.25)和查询运行时内存(由24.2降到21.2)都相对减少;数据库时间也由4.46降到4.28。数据库的使用效率再次得到提高。

### 4.3 利用缓存优化查询

#### 4.3.1 选择查询方法

原程序中对某些映射类的实例化和信息的提取采用的是如下的方法:

```
Instance a = (Instance) getHibernateTemplate()
        .get("Instance", ID);
```

可以看到这里用的是 HIBERNATE 的 get()方法提取的信息。前面已经谈过了 HIBERNATE 的查询方法的区别,get()方法在一级缓存没有找到的情况下会直接查询数据库,不会去二级缓存中查找,而 load()方法则会使用二级缓存。在使用中,对使用了二级缓存的对象进行查询时最好使用 load()方法。

#### 4.3.2 打开二级缓存

在 HIBERNATE 配置文件(HIBERNATE. CFG. XML)中指定二级缓存的提供者类,在 HIBERNATE. CFG. XML 中添加如下内容:

```
<PROPERTY
    name = " hibernate. cache. provider _ class" > org. hibernate.
cache. EhCacheProvider
</PROPERTY >
```

在 HIBERNATE 配置文件的同级目录增加 EHCACHE 配置文件(EHCACHE. XML),EHCACHEPROVIDER 通常会到 HIBERNATE 配置文件的同级目录下去查找这个配置文件。文件内容如下:

```
<? XML version = "1. 0" encoding = "UTF - 8"? >
<ehcache XMLns:xsi = "http://www. w3. org/2001/XMLSchema - in-
stance" xsi:noNamespaceSchemaLocation = "ehcache. xsd" >
<diskSTOre path = "java. io. tmpdir"/ >
<defaultCache
    maxElementsInMemory = "10000"
    eternal = "false"
    timeToIdleSeconds = "120"
    timeToLiveSeconds = "120"
    overflowToDisk = "true"
    diskPersistent = "false"
    diskExpiryThreadIntervalSeconds = "120"
    memoryStoreEvictionPolicy = "LRU"/ >
</ehcache >
```

在需要进行缓存的实体对象的映射文件中配置缓存的策略,与不使用二级缓存的 JAVA 对象的区别就在于需要前面所介绍 < CACHE > 元素来配置此对象的缓存策略。可以将 HIBERNATE 的缓存策略定义为"read-write"。因此应该在映射文件中增加如下的配置:

```
<CACHE usage = "read-write"/ >
```

#### 4.3.3 优化结果

学生第二次浏览信息时已经不再重复递交第一次浏览信息时递交的查询语句了。因此,递交的查询语句数量只有原来的一半。

通过应用二级缓存,服务器和数据库效率都得到了一定的优化。服务器方面,每秒点击命中次数(由 105.27 增加到 128.74)、吞吐量(由 661776.1 增加到 809322.6)和平均每秒事务处理量(由 1.39 增加到 1.69)都有较大幅度提高;在第一次浏览响应时间基本不变的情况下,第二次浏览响应时间(由 0.23 降到 0.11)有明显降低。数据库方面,由于大量减少了对数据库的查询,数据库的解析总数(由 4669.25 降到 2781)、解析时间(由 0.44 降到 0.27)和查询语句的执行时间(由 2.98 降到 2.4)都有明显减少;同时,数据库查询执行总数(由 4935.5 降到 3120.37)、数据库处理器的调用次数(由 26906.13 降到 15468.88)和查询语句运行时内存(由 21.2 降到 16.9)都有明显减少;由于上述各项指标优化的共同作用,数据库时间由 4.28 降到 3.25,数据库处理器占用时间由 4.29 降到 3.15。说明二级缓存明显减轻了数据库的负荷,有效地提高了数据库的服务效率。

## 5 结　语

对系统性能的调优不应该是等到系统出现问题时才进行,而应该贯穿系统设计、开发和应用的整个过程。因此,系统调优工作是一个多层次、多角度的循环往复的过程。

不同的优化策略会对性能产生不同的影响。如果数据库配置不合理,那么无论怎么优化应用都无法达到预期性能要求;如果数据库设计本身不合理也会限制应用性能的提升;只有先配置好持久类之间的关联才能够简化数据库访问的步骤和代码,才能够利用 HIBERNATE 的批量抓取功能;如果只配置持久类属性的延迟加载而忽略对象的延迟加载,那么,不需要的关联对象的信息会被提取出来,比起对象延迟加载的性能不会有提高,因为,对数据库行数据提取的优化比对列数据提取的优化更为有效,特别是在数据量较大的情况下。因此,对招生系统基于 HIBERNATE 的数据库访问性能优化的有效步骤应该从数据库配置开始:

① 调整数据库参数,比如内存参数的调整;
② 优化数据库表设计;
③ 优化应用程序;
④ 优化 HIBERNATE 的数据抓取策略;
⑤ 优化 HIBERNATE 的数据抓取时机;
⑥ 优化 HIBERNATE 的数据访问机制,比如利用缓存。

当然不同优化方法的代价也不同。对数据库内存参数的调整更多地占用了系统的资源,因此,在硬件条件允许的情况下可以对独立性较强的参数进行优化,以免给其他优化带来影响;持久化对象的关联可以更好地实现持久层对数据库的访问以及体现程序的逻辑性,但却占用操作系统的更多内存和硬盘资源;属性的延迟加载尽管优化了数据库的访问效率,但由于二进制码增强导致类文件增大,因此更多地占用了服务器的 CPU 时间;二级缓存在极大地优化了查询效率的同时,给持久化数据的同步带来了困难。如果对数据库中的数据进行修改,同时还要更新二级缓存中的相应数据或者清空二级缓存。这就增加了数据更新的时间。因此,针对不同的方法,在实际系统中必须根据具体情况进行适当的折衷。

## 参 考 文 献


[1] 宫生文,王宁. Hibernate 作为 J2EE 数据持久层的分析和研究[J]. 计算机与信息技术,2006(4):40 - 42.

[2] 计磊,李里,周伟. 精通 J2EE—Eclipse、Struts、Hibernate、Spring 整合应用案例[M]. 北京:人民邮电出版社,2006(8):9 - 10.

[3] 陈天河. Struts、Hibernate、Spring 集成开发宝典[M]. 北京:电子工业出版社,2007(5):248 - 249.

[4] 林学鹏. Grove——. NET 中的 ORM 实现[C]//MSDN——. NET 开发,2005-06-30.

[5] 朱庆伟,吴宇红. 一种对象/关系映射框架的分析和应用[J]. 电子科技,2004(1):54 - 60.

[6] Whalen E, Schroeter M. Oracle 性能调整与优化[M]. 高艳春,周兆确,唐艳军,译. 北京:北京人民邮电出版社,2002.