

基于 JDBC 数据管理与查询优化的研究

韩 兵¹, 李晶晶¹, 方英兰²

(1. 北方工业大学 计算机学院, 北京 100144;

2. 大规模流数据集成与分析技术北京市重点实验室, 北京 100144)

摘 要:在大多数 Web 应用系统中,当用户浏览数据时,每次都向数据库发送查询请求的做法会使系统性能下降,查询速度降低,而利用缓存技术将部分数据缓存在 Web 应用服务中可以减少对数据库的查询操作,提高数据的利用率和检索效率。JDBC 是应用程序与数据库进行交互的桥梁,通过对 JDBC 中类和接口进行修改和扩展,设计了一个基于 JDBC 数据管理的模型,其中重点研究了缓存置换机制。根据系统对缓存项访问的频率、时间间隔以及缓存项占用存储空间的大小等特性,结合最近最少使用算法,提出了一种基于价值函数的缓存置换算法,选择价值最小的缓存项进行置换,并设计模拟实验进行验证。实验结果表明,在 JDBC 管理的缓存空间中利用该算法可以取得较高的缓存命中率,服务器的响应速度也得到了提高。

关键词:JDBC; Web 应用; 热点数据; 缓存置换; 缓存价值; 命中率

中图分类号: TP311.1

文献标识码: A

文章编号: 1673-629X(2018)09-0176-05

doi: 10.3969/j.issn.1673-629X.2018.09.036

Research on Data Management and Query Optimization Based on JDBC

HAN Bing¹, LI Jing-jing¹, FANG Ying-lan²

(1. School of Computer Science, North China University of Technology, Beijing 100144, China;

2. Beijing Key Laboratory on Data Integration and Analysis Technology of Large-scale Stream, Beijing 100144, China)

Abstract: In most Web applications, sending a query request to the database each time when users browse the content will lead to a decline in system performance and a low query speed. Reserving some of the data in Web application services using the caching technology can reduce the query operation of the database and improve the data utilization and retrieval efficiency. JDBC is the bridge between the application and the database, so we design a model based on JDBC data management by modifying and extending the classes and interfaces in JDBC, which focuses on the cache replacement mechanism. Combined with the least recently used algorithm, we propose a cache replacement algorithm of cache value according to characteristics such as frequency of data access, time interval and size of cache entries, selecting the cache item with least value for replacement and designing the simulation experiment for verification. The experiment shows that this algorithm can achieve high data hit rate in the cache space about JDBC management and improve the server response speed.

Key words: JDBC; Web application; hot data; cache replacement; cache value; hit rate

0 引 言

几乎所有的 Web 应用都离不开数据的支撑,因此在数据的访问过程中,性能成为人们关注的焦点。随着互联网及其相关技术的飞速发展,网络用户不断增加,网站的访问数量也急剧增加,使 Web 系统的访问承受的压力越来越大,随之而来的就是系统性能下降,查询速度降低。大量 Web 应用系统分析表明,在访问过程中大量结果集从数据库传输到应用程序,是查询

速度滞后的主要原因。因此,如何减少用户的等待时间,使用户能够有效快速地访问数据,成为当前 Web 应用技术研究的重点和热点。

众所周知,恰当的缓存可以缩短数据传输的距离,缩短系统响应时间,提高数据利用率和检索效率。对于应用服务器端的缓存机制,当前的做法大多是将上次访问过的数据保存起来,并没有区分要缓存的数据是否为热点数据^[1];当需要进行缓存置换时,通常根据

收稿日期: 2017-11-02

修回日期: 2018-03-20

网络出版时间: 2018-05-16

基金项目: 国家自然科学基金(61370051)

作者简介: 韩 兵(1971-),男,硕士,副研究员,CCF 会员(57372M),研究方向为计算机应用、数据库优化、数据分析和数据挖掘;李晶晶(1993-),女,硕士研究生,研究方向为计算机应用。

网络出版地址: <http://cnki.net/kcms/detail/61.1450.TP.20180515.1655.060.html>

缓存对象的访问频率或者最近访问时间间隔来进行置换,考虑的因素较为单一。例如,EhCache 是一种具有代表性的进程内缓存框架^[2-3],通过手工配置来决定缓存哪些数据,并提供了三种缓存置换策略:FIFO、LRU、LFU 用于缓存管理。但是,该缓存框架并不能自主识别哪些是需要缓存的热点数据,缓存置换的方式只能选择以上一种,不能充分利用现有的缓存资源。

1 研究背景

在 Web 应用系统开发中,合理有效地设计和使用缓存是提高系统性能的一个重要方式。在数据库管理方面,MySQL 数据库使用了 query cache^[4-5]进行数据缓存,以 SQL 语句为依据来存储上一次查询后的结果,从而提高数据库的查询效率,减少大量的磁盘 I/O 操作和 CPU 运算。在 Oracle 数据库中也有类似实现,其名称为 buffer cache^[6],与 MySQL 缓存不同的是,它可以通过加入 Hint 标识符让用户自主选择所要缓存的数据。但是这两种方式都是利用了数据库系统自身的机制,虽然可以提高数据库系统的性能,减少 Web 系统等待时间,但是在系统中却无法减少连接数据库的时间开销,而创建数据库连接和关闭连接又是一个非常耗时的活动,所以,不仅会使系统的响应速度下降,而且会消耗大量不必要的系统资源。

JDBC (Java database connectivity, Java 数据库连接)是一个用于连接并操作数据库的规范^[7],主要功能是为多种关系型数据库提供统一访问,自身并没有提供数据缓存功能。目前在 JDBC 的基础上构建了许多更高级的实现,比较常见的有 Hibernate、Mybatis、Spring JDBC 等 Web 框架。使用框架可以降低系统开发的复杂度,减少程序中大量重复的代码。其中,Hibernate 框架和 Mybatis 框架都引入了缓存机制^[2,8],通过提供一级缓存和二级缓存来提高数据的查找效率。一级缓存又称 session 级缓存,只能被当前事务访问,其生命周期对应于事务的生命周期,当事务结束时,缓存的生命周期也就结束了。一级缓存是针对单次操作而设计的,生命周期较短,当单次数据操作完毕后,下一次请求的数据将无法使用上一次缓存的数据,因此对系统性能的改善是有限的。二级缓存 (Session Factory) 是一个可配置的插件,需要通过外部技术实现,存储的介质可以是内存或者硬盘。但是此功能需要用户手工配置,配置信息影响着缓存自身的性能,因此对程序员的要求较高。综上所述,传统的数据库缓存、Web 框架提供的一级缓存和二级缓存并不能有效地实现缓存中数据的共享并减少冗余。且一些早期的 Web 系统的持久层并没有使用任何框架^[9],而是直接使用 JDBC 进行编写,版本迁移比较困难。

对于缓存数据的管理,文献[10]提出了一种基于 Chunk Folding 的自适应多租户缓存管理机制,依据用户当前访问模式,动态生成候选缓存单元集,并通过贪婪算法选取缓存单元集。文献[11]根据用户访问频率建立对应具有生命周期的缓存项,维护缓存的一致性。文献[12]提出了内容分发网络缓存替换策略,根据用户访问特性建立相应的代价公式,并作为计算 Web 内容价值的要素。

对于持久层采用原生 JDBC 编写的 Web 应用系统,JDBC 作为与数据库进行通信的唯一通道,如果能在获取数据返回给用户的同时,通过分析其是否是热点数据,将热点数据缓存起来,下次发送相同请求时,则可以直接从缓存中获取数据,无需重连数据库。这样对数据库的大部分查询就可以转化为对查询结果的直接获取,能够减少用户等待时间,提高系统性能。

2 基于 JDBC 数据管理的模型构建

JDBC 是 Web 应用与数据库交互的桥梁,是标准的 Java API,提供与数据库进行交互的通道。通过使用 JDBC,编程人员可以向不同关系型数据库发送 SQL 命令,实现对数据库中数据的增、删、改、查等操作。

JDBC 数据管理的模型如图 1 所示,模型主要包括三部分:顶层是应用层;中间层是应用服务器,缓存管理器(cache manager, CM)是该层的核心与重点;底层是数据库。

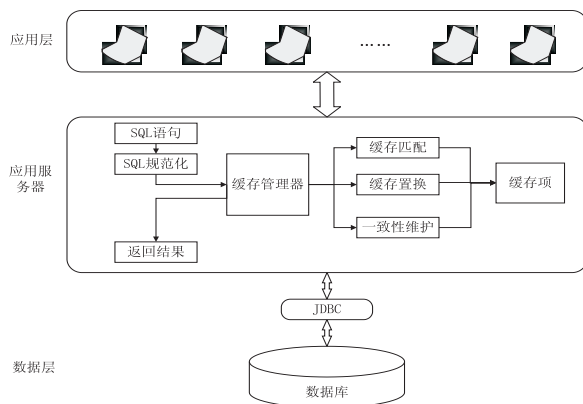


图 1 JDBC 数据管理模型

应用层负责接收用户请求并发送至 Web 应用服务器,并显示数据给用户,响应时间决定着用户体验的好坏。

Web 应用服务器负责应用层与数据库之间的数据处理及交互。在该层中增加缓存数据管理的功能,包括三部分内容:缓存匹配、缓存置换和一致性维护。当接收到用户的查询请求时,首先将查询的 SQL 语句进行规范化处理(去掉多余的空格和换行,并把所有的小写字母变成大写字母)并交由 CM,如果能够在缓存中匹配到相应的目标数据,则将匹配结果返回给应

用层,否则将通过 JDBC 连接数据库,并发送查询命令进行查询,从而获取用户请求的数据。在将查询结果返回给应用层的同时判断其是否是热点数据,如果是则将其保存在缓存空间中。对于缓存空间中的缓存置换^[13],采用基于价值函数的缓存置换算法。当缓存中的数据被修改时,还需要对缓存数据的一致性^[14]进行维护。其中,缓存置换是文中研究的重点内容。数据库则用于存储数据,对数据进行处理。

3 缓存置换机制

由于服务器端内存资源有限,分配给查询结果集的缓存空间大小也是有限的。当缓存中缓存数据达到容量极限后,为了满足新缓存需求,需要进行缓存置换。一个好的缓存置换算法是提升缓存性能的关键。

3.1 缓存对象识别

在 Web 应用中,当用户希望查看自己关心的内容时,Web 应用往往需要向数据库发送查询请求,通过 JDBC 连接数据库来获取相应的结果集,最终将结果呈现给用户。根据时间局部性原理,如果用户访问了一段数据,那么在近期再次访问它的可能性较大。在一个 Web 应用中,用户存在多次查看曾经访问过的页面的可能,这时需要重新向数据库发起查询请求获得相应的查询结果。因此,文中将一个查询请求对应的结果集作为一个缓存项,JDBC 管理的缓存则是由不同的缓存项组成的集合,记为 CS,每个缓存项记为 C ,其缓存价值记为 V 。

为了方便描述,将缓存项的逻辑结构表示为如下的六元组:

$$C = (ID, R, V, L, S, A)$$

元组中每个属性的含义如下:

(1) ID 是缓存项唯一标识,由于缓存项是由查询 SQL 语句获得的,故 ID 表示经过规范化的 SQL 语句。

(2) R 是缓存项的内容,每一个缓存项都包含若干条相似的记录。

(3) V 是缓存项的价值,当需要进行缓存置换时,根据这个属性选取价值小的缓存项进行置换。

(4) L 是缓存对象在缓存中的位置。每当有新的缓存项加入缓存时,总是将其放在缓存中第一个位置。

(5) S 是缓存项占用存储空间的大小,该因素会影响缓存置换。

(6) A 是缓存项从创建开始到需要进行缓存置换时的访问次数,根据此属性可以计算其占总访问次数的比重,从而计算其相对访问频率。

3.2 基于价值函数的缓存置换

当用户通过 Web 应用向 Web 服务器发送查询请求时,首先根据请求的 SQL 语句在 JDBC 管理的缓存

中进行匹配查找,如果能够匹配到,即缓存命中,则将即匹配到的结果返回给用户,并更改该缓存项在缓存中的位置和及其缓存价值。如果不能匹配到,则需要通过 JDBC 从数据库中获取查询的结果集,在将结果集返回给用户的同时将其加入缓存,如果需要进行缓存置换,则使用基于价值函数的缓存置换算法,将缓存空间中价值较小的缓存项移出缓存。缓存置换原理如图 2 所示。

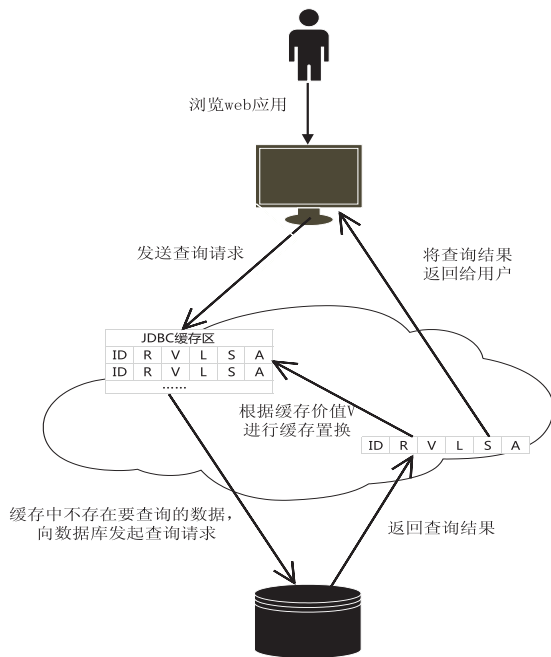


图 2 缓存置换原理

文中采用的缓存置换算法对经典置换算法 (LRU)^[15]作了改进,将新加入的缓存项和最新命中的缓存项都移到缓存空间的第一个位置,缓存项越靠前,表示在与当前时间间隔越短的时间内,该缓存项被访问过。故采用缓存项的位置来计算其 LRU 价值^[16],计算方式如下:

$$V_{LRU}(L_c) = \frac{N - L}{N} \quad (1)$$

其中, N 表示缓存空间中缓存项的总个数; L 表示缓存项在缓存空间中的位置; $V_{LRU}(L_c)$ 表示在位置 L_c 上缓存项的 LRU 价值。

通过式 1 可以看出,每个缓存项都存在一个 LRU 价值,位置值越小,LRU 价值越大,值域在 0 和 1 之间。

同时将缓存项的访问频率所占比重以及缓存项占用的缓存空间大小等因素都考虑在内,故缓存项的缓存价值可以通过式 2 计算:

$$V_c = \frac{\mu_1 \frac{A_c}{A_{sum}} + \mu_2 \frac{N - L}{N}}{S_c} \quad (2)$$

其中, V_c 表示缓存项的缓存价值; A_{sum} 表示总体的访问次数; μ_1 和 μ_2 表示两个可调节的参数, μ_1 表示

缓存项的访问频率对缓存价值的影响权重, μ_2 表示 LRU 价值对缓存价值的影响权重; N 和 L 的定义参见式 1; S_c 表示缓存项占用缓存空间的大小。

基于价值函数的缓存置换算法流程如图 3 所示。

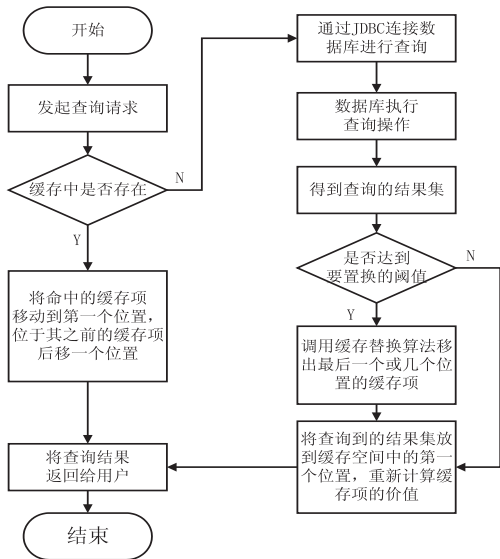


图3 基于价值函数的缓存置换算法流程

当查询请求到来时,如果能够在缓存空间中找到要请求的数据,则将该缓存项移动到缓存中的第一个位置,并将位于其之前的缓存项后移一个位置,表示该缓存项具有较小的访问时间间隔,然后根据式 2 重新计算每个缓存项的价值;否则,Web 应用通过 JDBC 从数据库中获取查询结果,在将其加入缓存空间时判断是否达到缓存置换的阈值,如果是,则根据缓存项的价值从小到大排序,将价值最小的一个或几个缓存项移出缓存,将新的缓存项放入到缓存中的第一个位置;如果未达到缓存置换的阈值,则直接将新的缓存项放到第一个位置,然后将查询结果返回给用户。

记触发缓存置换的阈值为 Y ,待加入的缓存项为 c ,则算法描述如下:

```
开始:
1. if( sizeof( CS ) < Y ) {
2. //缓存空间未满,可以直接插入
3. 将每个缓存项往后移动一个位置
4. 将 c 放入到缓存集合 CS 的第一个位置,且 Ac +1
5. 根据式 2 计算每个缓存项的缓存价值
6. }
7. else {
8. 将 CS 中每个缓存项按照价值 V 进行排序
9. do {
10. 将 V 最大的缓存项移出缓存
11. } while( sizeof( CS ) < Y )
12. 将剩下的每个缓存项往后移动一个位置
13. 将 c 放入到缓存集合 CS 的第一个位置,且 Ac +1
14. 重新计算每个缓存项的缓存价值
```

15. }
结束

4 实验设计与验证

文中研究的是基于 JDBC 数据缓存的管理,通过对缓存项的分析,得出了基于价值函数的缓存置换算法。为了验证算法的有效性,本节设计了一个模拟实验。实验环境中使用的 Web 应用服务器为 Tomcat 服务器,配置为 CPU @ 3.40 GHz,操作系统为 Windows 7,内存为 8 GB,数据库使用 MySQL5.5 版本。

4.1 实验数据

在 Web 应用系统中,数据的访问特征服从 Zipf^[17] 分布,即系统中百分之二十的内容,占据了百分之八十的访问量。因此,实验中使用一段 MATLAB 代码生成符合 Zipf 分布的 10 000 个随机数,范围是 1 至 500,分别对应系统中 500 个查询请求,其中有 80% 的数字是 1 到 500 这 500 个数字中的 20% 的数字,这符合用户的访问规律。式 2 中分别取 μ_1 等于 0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,对应的 μ_2 等于 0.9,0.8,0.7,0.6,0.5,0.4,0.3,0.2,0.1,对于每一对参数,使用基于价值函数的缓存置换算法计算命中率,实验结果如图 4 所示。可以看出,当 $\mu_1=0.8, \mu_2=0.2$ 时,缓存命中率最高。

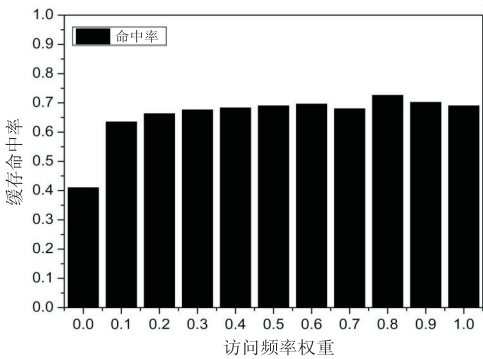


图4 不同参数值对应的不同命中率

4.2 缓存命中率对比分析

根据上述实验得出的结论,取 μ_1 为 0.8、 μ_2 为 0.2 继续进行实验。初始化缓存空间的大小为 50 M,用 10 000 个随机数代表用户的访问序列,依次发送 1 000, 2 000, 3 000, 4 000, 5 000, 6 000, 7 000, 8 000, 9 000, 10 000 个访问序列,通过代码读取访问序列对应的查询请求序列,向 Web 服务器发送相应的 SQL 命令,如果缓存命中则该 SQL 语句对应的缓存项的命中次数加 1,如果未命中则需要通过数据库获取结果,将其保存在缓存空间中。对于传统的 LRU 缓存置换算法,当缓存空间不足时替换出访问时间最早的缓存项。而文中提出的基于价值函数的置换算法则综合考虑了缓存项的访问频率、访问时间间隔和缓存项的大小,选取价

值最小的缓存项进行置换。对于每一个访问序列,分别应用 LRU 置换算法和基于价值函数的置换算法,统计总的命中次数,计算两种算法的命中率,对比结果如图 5 所示。

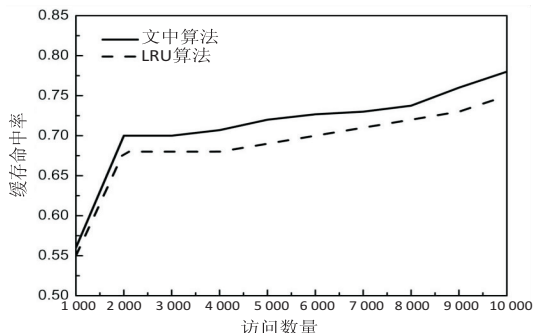


图 5 缓存置换算法命中率对比曲线

从图 5 可以看出,随着访问数量增多,基于价值函数的缓存置换算法的缓存命中率高出传统的 LRU 算法,所以使用基于价值函数的缓存置换算法可以提高 JDBC 缓存的性能。

4.3 响应时间对比分析

实验中对于每一次发送的查询请求,分别记录了使用缓存和不使用缓存时 Web 服务器的响应时间,结果如图 6 所示。从图中可以明显看出,当使用缓存时,服务器的响应时间明显高于不使用缓存时的响应时间。这是因为当使用缓存时,不需要连接数据库就能直接从缓存中获取所需要的数据,减少了连接数据库和在数据库服务器中进行查询的时间,提高了数据访问的效率。因此,文中设计的基于 JDBC 数据管理的模型对于 JDBC 缓存是非常必要的。

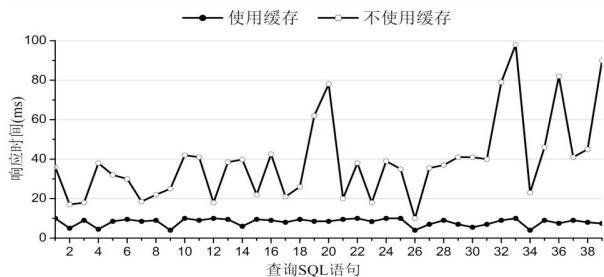


图 6 服务器响应时间对比曲线

5 结束语

通过对 JDBC 中的类和接口进行修改和扩展,设计了一个基于 JDBC 数据管理的模型,综合考虑缓存项的访问频率、访问时间间隔和缓存项占用存储空间的大小等因素,提出了一种基于价值函数的缓存置换算法。通过设计模拟实验,从缓存的命中率、服务器的响应时间两方面进行分析,相比于没有缓存功能的 JDBC,该模型能够获得较高的访问效率,可以有效弥补使用原生的 JDBC 进行开发的 Web 系统的访问效率

问题,提升系统性能。

另外,由于改进后的 JDBC 针对的是全表查询的数据,而实际应用中往往是多表联合查询,因此如何细化缓存粒度,找出更高效的缓存管理方式有待进一步的研究。

参考文献:

- [1] 张明. IaaS 中基于热点数据的存储系统研究与实现[D]. 哈尔滨:哈尔滨工业大学,2015.
- [2] 陈正举. 基于 HIBERNATE 的数据库访问优化[J]. 计算机应用与软件,2012,29(7):144-149.
- [3] 谢杰涛,吴敏,吴娟,等. Web 系统高性能本地数据缓存实现机制[J]. 计算机应用研究,2014,31(7):2074-2077.
- [4] CABRAL S K. MySQL 5.5:improving on the world's most popular open source database[J]. Database Trends and Applications,2011,25(3):30-33.
- [5] 邱林峰,曹学成,柏文阳. 自适应的数据库查询缓存[J]. 计算机工程与应用,2008,44(22):159-161.
- [6] BELKNAP P, DAGEVILLE B, DIAS K, et al. Self-tuning for SQL performance in Oracle database 11g[C]//IEEE 25th international conference on data engineering. Shanghai, China: IEEE,2009:1694-1700.
- [7] ZHANG Yi, ZHANG Luyong. JDBC-based middleware applications in instant message systems[C]//2nd international conference on systems and informatics. Shanghai, China: IEEE,2014:1044-1049.
- [8] 荣艳冬. 关于 Mybatis 持久层框架的应用研究[J]. 信息安全与技术,2015(12):86-88.
- [9] 乔岚. 基于 MyBatis 和 Spring 的 JavaEE 数据持久层的研究与应用[J]. 信息与电脑,2017(8):73-76.
- [10] 姚金成,张世栋,史玉良,等. 基于 Chunk Folding 的多租户数据库缓存管理机制[J]. 计算机学报,2011,34(12):2319-2331.
- [11] 余文涛,李立新,毛秀青,等. 移动环境下一种缓存管理策略[J]. 计算机应用与软件,2014,31(12):178-181.
- [12] 胡伟之,沈富可. 基于 Web 访问特性的缓存替换策略[J]. 计算机应用,2008,28(S2):48-50.
- [13] 吴俊龙,杨清. 基于协同过滤的 Web 缓存替换算法研究[J]. 计算机工程与科学,2015,37(11):2128-2133.
- [14] 蔡建宇,杨树强,贾焰,等. 关系数据库语义缓存的研究进展[J]. 计算机工程与科学,2005,27(10):62-64.
- [15] 王文博,王菁,邢起源,等. 移动即时通讯软件的缓存替换策略[J]. 计算机科学与探索,2015,9(3):292-299.
- [16] 邢起源,王菁,闫阿宾,等. 一种基于社交关系的移动缓存替换算法[J]. 计算机科学,2016,43(6):44-49.
- [17] KUMAR V R, SWATI M. Cache replacement algorithms for coordinated cooperative social wireless networks[J]. International Journal of Computer Science and Mobile Computing, 2014,3(10):718-725.