计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(1):176−181 [doi: 10.15888/j.cnki.csa.006736]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

爬虫系统中标签删除功能的设计及优化①

邓子云

(长沙商贸旅游职业技术学院 经济贸易学院, 长沙 410116)
通讯作者: 邓子云, E-mail: dengziyun@126.com

摘　要: 在用爬虫爬取到大型商品网站的大规模网页数据集后, 要将网页数据集作进一步筛选以得到目标数据集, 筛选之前要做的一项准备工作就是删除网页中多余的标签. 为此, 用递归算法的思想给出了标签删除的算法, 提出了标签删除功能的软件设计思想, 对设计进行了 2 次设计改进及性能优化, 最终采用了 1 个缓冲区维系线程 1 个标签删除线程的双线程设计思想. 实验表明, 优化后的标签删除功能在单机上每 1000 个网页的平均处理时间只需 19.7 s, 处理 20 万个网页只需 1.1 小时.

关键词: 标签删除功能; 递归算法; 双线程设计; 性能实验

引用格式: 邓子云.爬虫系统中标签删除功能的设计及优化.计算机系统应用,2019,28(1):176–181. http://www.c-s-a.org.cn/1003-3254/6736.html

Design and Improvement of Tag Deletion Function in Crawler

DENG Zi-Yun

(College of Economics and Trade, Changsha Commerce &Tourism College, Changsha 410116, China)

Abstract: After crawling to obtain a data set of large web pages on a large commodity site, the data set is screened to further get the target data set. Before screening, preparation must to be done is to delete the redundant tags in the web pages. Therefore, the algorithm of deletion tag is given with the idea of a recursive algorithm. The design idea of tag deletion function is put forward. 2 time design improvements are carried out to optimize the performance. Finally, the design idea of dual thread is adopted. The dual threads are 1 maintain buffer thread and 1 tag deletion thread. In single computer environment, experiments show that the optimized tag deletion function only takes 19.7 seconds for each 1000 pages, and only 1.1 hours for 200 000 web pages.

Key words: tag deletion function; recursive algorithm; dual thread design; performance experiment

在用爬虫爬取到大型商品网站的大规模网页数据集后, 要将网页数据集作进一步筛选以得到目标数据集, 筛选之前要做的一项准备工作就是删除网页中多余的标签. 已有一些文献提出了标签删除功能的算法, 并有工程实现, 但尚未见性能分析及改进的讨论[1–3].

本文作者所在的研究团队已经研发了一种爬取大型商品网站网页数据的爬虫, 可以爬取获得海量的网页数据集. 接下来要编制软件实现标签删除工作. 为让该功能可以实现快速删除, 本文在给出标签删除的算法及软件设计思想后, 还将对设计进行优化.

1　标签删除算法

网页中通常有如下标签可以删除[4,5]:

(1) 头部标签. 包括<head>、<title>、<meta>、<!doctype>、<style>等, 这些与正文内容无关, 其中<!doctype>标签在使用 Jsoup 的 Document 对象解析时结点名称为“#doctype”.

(2) 注释和脚本标签. 包括<script>、<!-- -->, 其中注释标签在使用 Jsoup 的 Document 对象解析时结点名称为“#comment”.

(3) 表单标签. 包括<input>、<select>、<option>等.

① 基金项目: 湖南省自然科学基金 (2017JJ5064)
Foundation item: Natural Science Foundation of Hunan Province (2017JJ5064)
收稿时间: 2018-07-16; 修改时间: 2018-08-10; 采用时间: 2018-08-21; csa 在线出版时间: 2018-12-26

(4) 文本标签. 应去掉内容为空的文本标签, 在使用 Jsoup 的 Document 对象解析时结点名称为“#text”.

(5) 其它标签. 包括<object>、<param>、<img>、
等. 这些标签分别表示控件对象、控件对象参数、图片、换行等, 不影响网页正文文本内容.

可采用递归算法, 在遍历网页树结构时删除这些多余的标签, 伪代码如算法 1 所示.

算法 1. removeTag(*nodeTree*)

```
//功能: 删除网页结构树中多余的标签
//参数说明: nodeTree, 要删除多余结点的网页结构树, 递归时为子树
返回值: 无
//遍历网页结构树
For i=1 to nodeTree.topNode.childNodeSize
  //得到第 i 棵子树
  childNodeTree=nodeTree.getChildNodeTree(i)
  //声明要删除的标签集合
  collectionDeleted={"head","title",…,"br"}
  /*如果当前结点是要删除的标签或为内容为空的文本标签*/
  If (childNodeTree.topNode.name in
     collectionDeleted) or
     (childNodeTree.topNode.name equal
     "#text" and childNodeTree.topNode.content
     is empty) Then
     childNodeTree.delete() //删除当前子树
  Else
     removeTag(childNodeTree) //递归调用
     i=i+1
  End If
End For
```

算法 1 传入的参数是网页结构树, 在递归调用时传入的是子树. 算法 1 实际上是按深度遍历的方法遍历网页结构树. 现有提出的算法还有按层次遍历的方法[1−3], 这种算法需要事先得到树的深度, 以便逐层访问, 故算法 1 的设计相对更为简便, 2 种方法的应用效果和实际性能相当.

算法 1 采用一个 **For** 循环依次访问当前结点的子树. 在循环体中, 先声明一个要删除的标签的集合, 再判断当前子树的顶点名称是否在这个集合中, 是则删除当前子树. 如果当前子树是文本结点且文本内容为空, 也应予以删除. 如果当前结点是不应删除的结点, 则递归调用算法 1, 传入当前子树继续执行删除标签工作. 递归调用后再将 *i* 值增 1. *i* 值为什么不在 **For** 循环中直接设置步长为 1, 而在判定为不用删除并递归调用后再增 1 呢? 来看如图 1 所示的树结构.

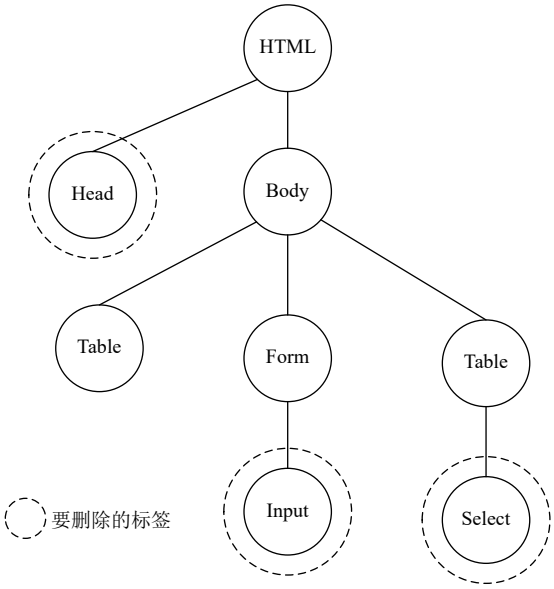


图 1 算法 1 计算的树结构示例

向算法 1 传入如图 1 所示的树结构时, **For** 循环体第 1 次执行时, 先得到 head 标签, 予以删除该<head>标签为顶点的子树 (实际上该树只有<head>一个结点). **For** 循环体第 2 次执行时, <html>标签的子结点数量由 2 变为 1, 继续得到<body>标签为顶点的子树, 则递归调用算法 1, 传入<body>标签为顶点的树. 在对<body>标签为顶点的子树执行完 removeTag() 方法后, 再将 *i* 值增 1. 可见, 在判定为不用删除时再增 1 的原因是因为删除后树顶点的子结点数量减少了一个.

2 标签删除功能的设计、优化及性能分析

下面先给出软件开发及性能实验的环境, 再提出软件设计的思想、性能实验的情况和优化的策略.

2.1 实验环境

为简便起见, 研究团队采用了自己使用的笔记本计算机作为实验环境, 在单机上安装 SQL Server 2017 Desktop Edition 作为数据库, 用 Eclipse Java EE IDE for Web Developers Neon.3 Release (4.6.3) 作为开发工具软件. 计算机硬件及操作系统配置如表 1 所示.

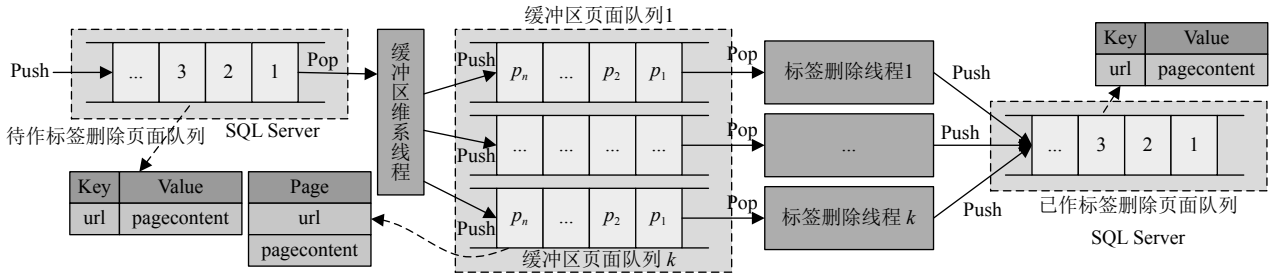
2.2 设计优化过程

对 SQL Server 的操作需要有一个缓冲区, 以满足快速处理网页的需要. 这个缓冲区数据结构及标签删除功能实现的设计如图 2 所示. 为取得较好的网页处理效率, 研究团队对设计做了 2 次改进. 为便于考察设计改进后的标签删除功能处理网页的效率, 按图 2(a)

取 1 个缓冲区维系线程 1 个标签删除线程 (A1S1D)、1 个缓冲区维系线程 2 个标签删除线程 (A1S2D), 图 2(b) 取 1 个缓冲区维系线程 1 个标签删除线程 (B1S1D)、1 个缓冲区维系线程 2 个标签删除线程 (B1S2D), 图 2(C) 取 1 个缓冲区维系线程 1 个标签删除线程 (C1S1D)5 种情况进行对比分析, 如图 3(a) 和图 3(b) 所示.

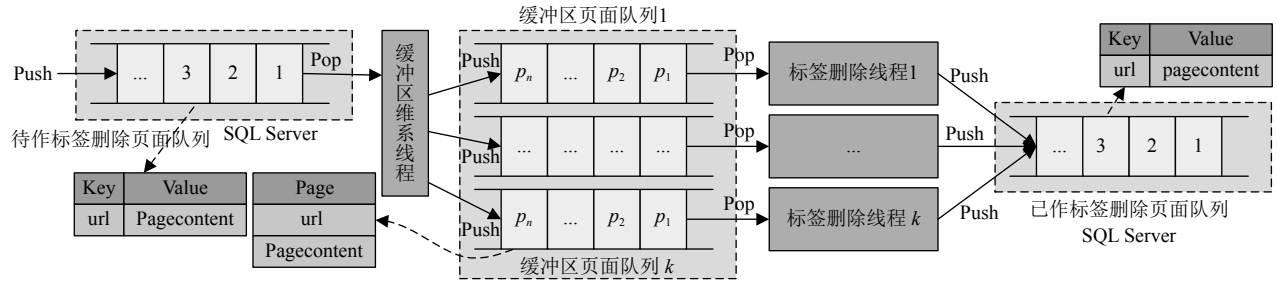
表 1　实验用计算机配置

项目	配置
计算机机型	ThinkPad T460s
CPU	Intel(R) Core(TM) i5-6200U CPU @ 2.30 GHZ(双核心四线程)
内存	8 GB RAM
硬盘	512 GB 固态硬盘
操作系统	Windows 10



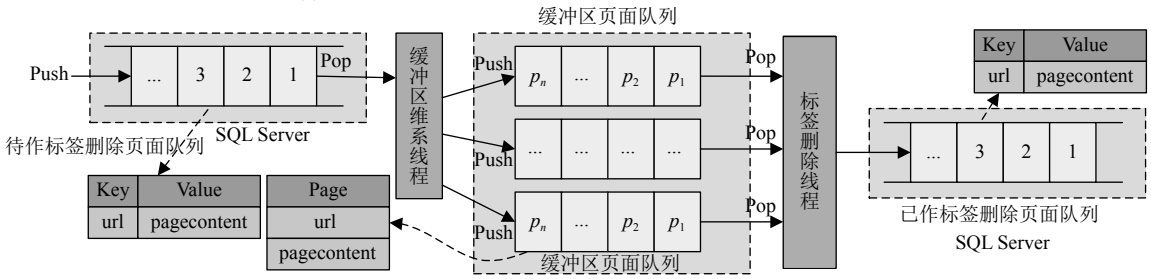
数据库连接：*Org·springframework·jdbc·datasource·DriveranagerDateSourece*

(a) 采用 *k* 个标签删除线程和 DriverManagerDataSource 连接数据库



数据库连接: *Org·apache·commons·dbcp·Basicsatasource* 连接数据库

(b) 采用 *k* 个标签删除线程和 Basicdatasource 连接数据库



数据库连接: *Org·apache·commons·dbcp·Basicdatasource*

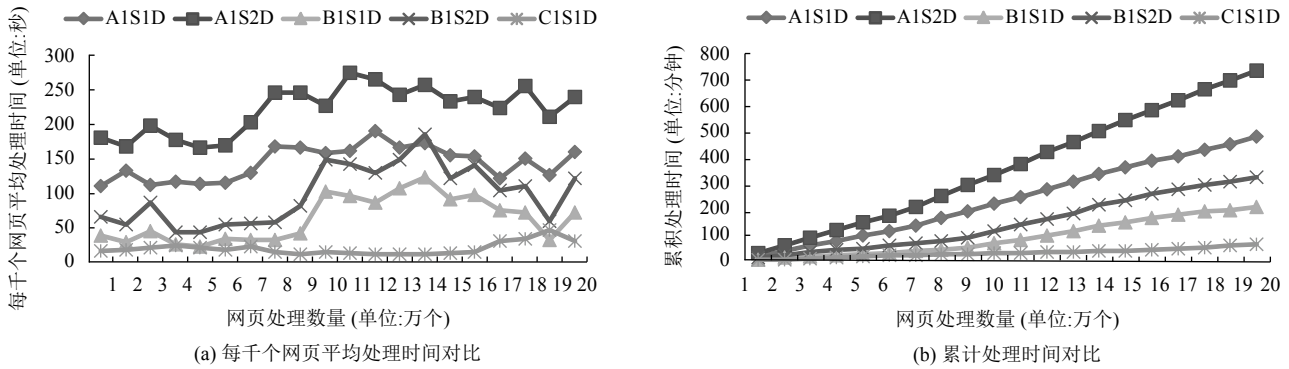
(c) 采用双线程设计和 Basicdatasource 连接数据库

图 2　缓冲区数据结构及标签删除功能实现的设计

最初的图 2(a) 的设计考虑有 2 个方面的因素:

(1) 队列和缓冲区的存储. 队列存放在 SQL Server 的表中, 待标签删除页面队列为一个表, 已作标签删除页面队列为另一个表, 并使用 Spring 的组件包中的 Driver_ManagerDataSource 类来封装连接数据库的数据源. 这些队列中的元素采取 Key-Value 结构, 这里将 Key 设置为网页的网址 url, 将 Value 设置为网页的内容. 缓冲区页面队列中的元素均为 Page 对象, 这个对象有两个属性, 即网页的网址 url 和网页的内容 pageContent. 假定缓冲区中有 k 个队列, 对应着为 k 个标签删除线程提供要删除的页面数据, 又假定缓冲区的一个队列最多有 n 个元素, 一个 url 平均长度为 l Byte, 页面内容平均长度为 m Byte, 则缓冲区大小为:

$$bufferSize = k \times n \times (l + m) \quad (1)$$



(a) 每千个网页平均处理时间对比 (b) 累计处理时间对比

图 3 5 种情况的每 1000 个网页平均处理时间和累计网页处理时间

设定 k=10, n=10, l=300, m=40 000, 则缓冲区大小为:

$$\begin{aligned} bufferSize &= k\times n\times(l+m)\\ &= 10\times 10\times(300+40000)\\ &= 4030000\ \text{Byte} \end{aligned} \quad (2)$$

故一共需要 4 MB 左右的内存空间, 这是当前的主流服务器配置可以接受的. 那么, 缓冲区维系列队怎么知道要将待过虑的页面数据放入到哪个缓冲区页面的队列中呢? 这需要查找出缓冲区页面队列中的最短队列, 再 push 操作. 而标签删除线程则是查找缓冲区页面列队中的对应队列的首元素, 再作如算法 1 所示的标签删除算法. 算法运行完成后将已删除标签的页面队列放入到 SQL Server 中的已删除页面队列中.

(2) 多线程的设计思想. 采用一个缓冲区维系线程, 每次从待作标签删除页面队列中 popup 操作取出一个页面, 再作 push 操作放入到缓冲区页面的队列中, 以为缓冲区中各个标签删除线程的队列源源不断的提供数据. 理论上认为标签删除线程数越多, 处理网页的效率就会越高.

从图 3 可以看出, 采用 A1S1D 比 A1S2D 的网页处理效率更高, 每 1000 个网页平均处理效率约要快约 35%, 20 万个网页处理效率约高 35%. 而且 A1S1D 的处理效率也不高, 处理 20 万个网页约需要 7.99 小时. 那是什么原因呢? 经过分析, 研究团队发现, 其一, 数据库操作占据了缓冲区维系线程和标签处理线程的大量时间, 因为使用了 org.springframework.jdbc.datasource._DriverManagerDataSource 连接数据库, 每次都会新建和关闭一个数据库连接; 其二, 由于标签处理线程只对一个 SQL Server 数据库表作 insert 操作, 而对单机的 SQL Server 的单个表加大 Java 线程操作的并发数并不能提升效率, 反而由于数据库的锁机制会降低效率.

这就需要作出改进, 策略如下:

(1) 改为使用 org.apache.commons.dbcp._BasicDataSource 数据库连接池, 从而减少数据库操作和关闭的操作, 节省整体数据库操作的时间.

(2) 标签处理线程只设一个.

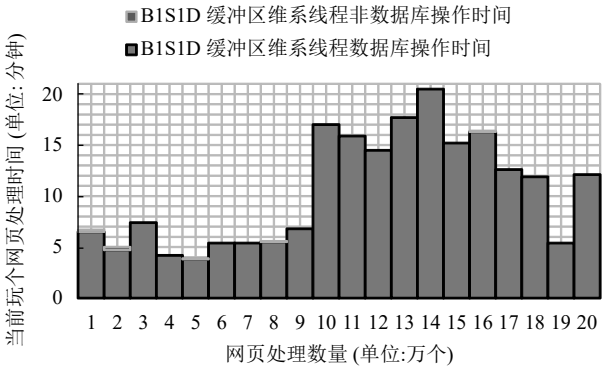
如此改进后, 设计如图 2(b) 所示, 再行作测试实验. 为便于做对比分析, 改用连接池后, 仍按图 2(b) 所示的设计采取标签处理线程 1 个和标签处理线程 2 个 2 种情况作对比, 如图 3 中的 B1S1D 和 B1S2D 所示, 从图中可看出, 网页处理效率加快了, B1S1D 比 A1S1D 每 1000 个网页处理效率约提升 56%, 20 万个网页处理效率也提升约 56%, 仅需 3.48 小时; B1S2D 处理 20 万个网页需 5.42 小时, 效率明显不如 B1S1D.

B1S1D 还有改进的空间. 通过运用 Spring 的 AOP 特性, 研究团队在数据库操作的方法前后用 Advice 记录下了数据操作的时间, 如图 4 所示. 图 4 的左边的图表示了数据库操作与非数据库操作的时间情况, 右边的图用百分比占比表示了数据库操作与非数据库操作的时间占比. 从图 4 的 B1S1D 的情况来看, 缓冲区维系线程中数据库操作的时间占比很大, 可见一个一个从数据库表中查询出数据是比较耗时的, 使得缓冲区维系线程供应网页数据不及时, 成为标签删除功能的瓶颈. 故对 B1S1D 改进的策略是使用批处理, 具体方法如下文.

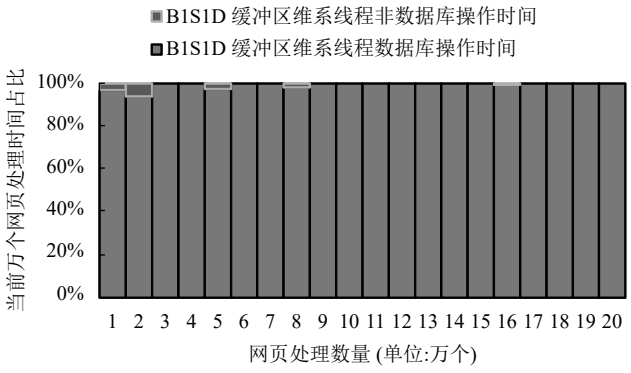
(1) 缓冲区维系线程每次从数据库表中查询出 500 条网页数据, 以 100 个网页数据为一个块向缓冲区中填充队列.

(2) 缓冲区以 100 个网页数据为一个数据块, 设置 10 个数据块, 保持一个 1000 个网页数据的缓冲区.

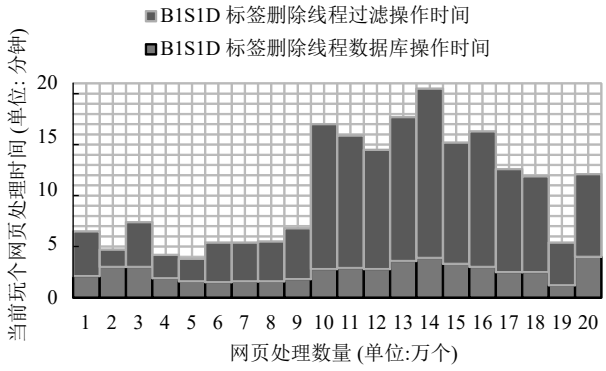
(3) 标签删除线程一次取出 100 个网页数据作批处理, 相应的 insert 操作也采用批处理, 每次向数据库表中插入 100 个删除标签后的网页数据.



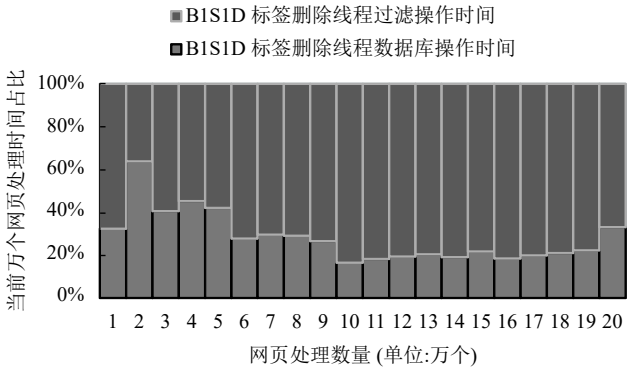
(a) B1S1D缓冲区维系线程处理时间



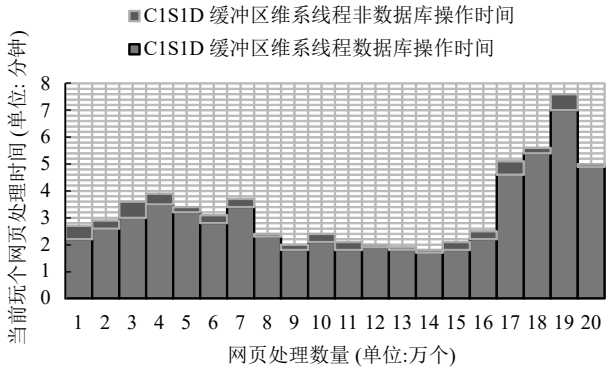
(b) B1S1D缓冲区维系线程处理时间占比



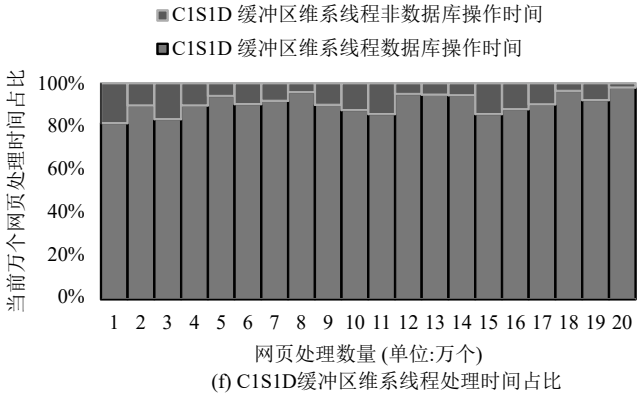
(c) B1S1D 标签删除线程处理时间



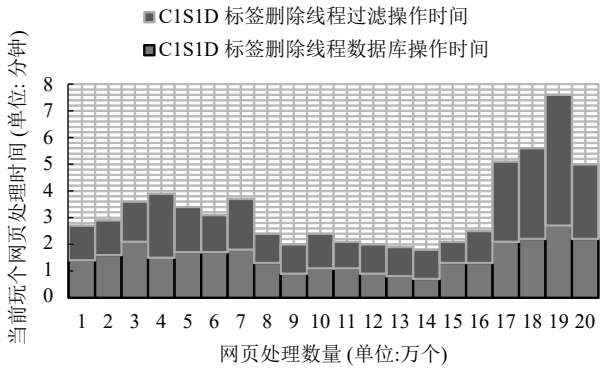
(d) B1S1D 标签删除线程处理时间占比



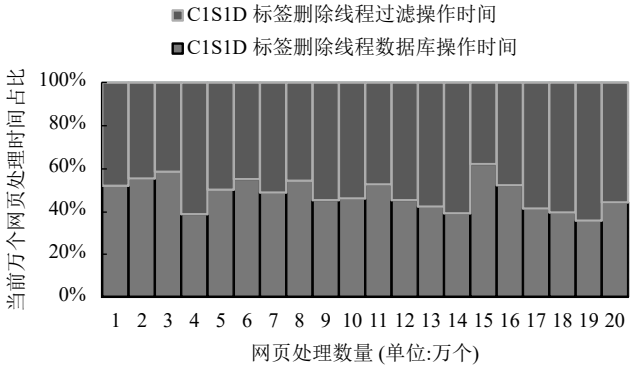
(e) C1S1D 缓冲区维系线程处理时间



(f) C1S1D缓冲区维系线程处理时间占比



(g) C1S1D 标签删除线程处理时间



(h) C1S1D 标签删除线程处理时间占比

图 4 B1S1D 和 C1S1D 的处理时间对比

改进后的设计如图 2(c) 所示, 再行作测试实验. 实验结果如图 3 和图 4 所示, C1S1D 每 1000 个网页的平均处理时间只需 19.7 秒, 处理 20 万个网页只需 1.1 小时. 后续我们不再讨论引擎的问题, 后续的各种过滤方法均采用 C1S1D 类似的结构设计. C1S1D 在 Spring 容器中的设计思想如图 5 所示.

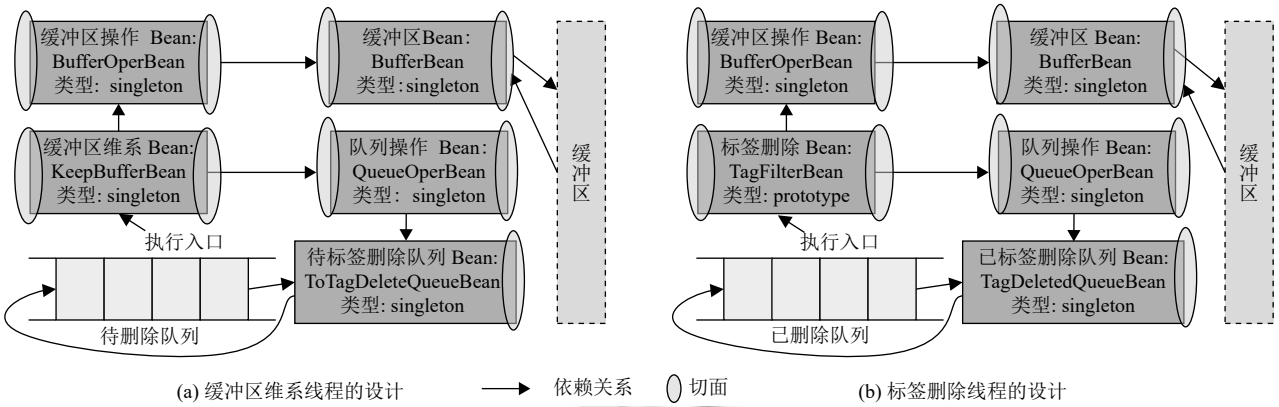


图 5 标签删除功能的 Sprint AOP 和 IoC 设计思想

综合上述分析, 课题研究团队对设计优化的方法是从“并行、连接池、批处理”3 个重要方面进行, 即:

(1) 采用双线程设计思想, 使系统可以并行工作;

(2) 采用连接池和数据缓冲区, 加快数据处理速度;

(3) 采用批处理方式, 一次成批处理方式操作数据库, 使效率数倍于逐条记录处理方式.

2.3 改进后设计的 Spring 实现

为实现如图 2(c) 所示的设计, 采用了 Spring 的 AOP 和 IoC 设计思想, 如图 5 所示, 其中, 图 5(a) 为缓冲区维系线程的设计, 图 5(b) 为标签删除线程的设计[6]. Spring 容器支持异步的任务执行器, 缓冲区维系线程和标签删除线程之间可互不影响, 并行执行.

从依赖关系来看, 线程 Bean 依赖于缓冲区操作 Bean 和队列操作 Bean. 缓冲区操作 Bean 和队列操作 Bean 再分别依赖于缓冲区 Bean 和队列 Bean. 缓冲区 Bean 和队列 Bean 分别封装缓冲区和队列. 这种设计思想就是延用的 DAO (Data Access Object, 数据访问对象) 的设计思想. 切面位于每个 Bean 的方法执行前后, 可根据方法的不同进行不同的 Advice 增强, 比较常用的是日志记录、效率分析等.

3 结束语

给出了用递归思想设计的标签删除算法, 采用 Eclipse 开发工具用 Java 语言开发实现了标签删除功能. 提出了标签删除功能的设计思想, 并作了 2 次性能优化和设计改进. 由最初的多线程设计思想, 改为最终的双线程设计思想, 并利用数据库连接池作网页数据的批量处理. 最终将性能提升至在单机上标签删除功能每 1000 个网页的平均处理时间只需 19.7 秒, 处理 20 万个网页只需 1.1 小时.

参考文献

1 黄仁, 王良伟. 基于主题相关概念和网页分块的主题爬虫研究. 计算机应用研究, 2013, 30(8): 2377–2380, 2409. [doi: 10.3969/j.issn.1001-3695.2013.08.034]

2 孟繁疆, 姬祥, 袁琦, 等. 农产品价格主题搜索引擎的研究与实现. 东北农业大学学报, 2016, 47(9): 64–71. [doi: 10.3969/j.issn.1005-9369.2016.09.009]

3 吴洁明, 冀单单, 韩云辉. 基于 Web 的 DCI 垂直搜索引擎的研究与设计. 计算机工程与设计, 2013, 34(4): 1481–1487. [doi: 10.3969/j.issn.1000-7024.2013.04.066]

4 Sahami M, Heilman T D. A web-based kernel function for measuring the similarity of short text snippets. Proceedings of the 15th International Conference on World Wide Web. Edinburgh, Scotland. 2006. 377–386.

5 Ilbahar E, Cebi S. Classification of design parameters for e-commerce websites: A novel fuzzy Kano approach. Telematics and Informatics, 2017, 34(8): 1814–1825. [doi: 10.1016/j.tele.2017.09.004]

6 Deng ZY, Zhang J, He TQ. Automatic combination technology of fuzzy CPN for OWL-S web services in supercomputing cloud platform. International Journal of Pattern Recognition and Artificial Intelligence, 2017, 31(7): 1759010. [doi: 10.1142/S0218001417590108]