一种基于 key-value 数据库的快速地名地址输入提示方法*

曹广顺[1], 禹 维[2], 朱欣焰[1], 佘 冰[1]

(1. 武汉大学 测绘遥感信息工程国家重点实验室, 武汉 430079; 2. 警用地理信息技术公安部重点实验室, 江苏常州 213022)

摘 要: 现有地址输入提示方法涉及标准地址和 POI 的研究较少,地址字符串的索引,大多采用 Trie(字典)树索引,Trie 树建立时内存消耗巨大,面临海量数据,问题突出。针对以上问题,提出一种基于 Key-Value 数据库的快速地名地址输入提示方法。该方法基于 Trie 树结构进行改进,降低了地址索引的复杂度;基于 Key-Value 数据库构建 Trie 树,避免了内存消耗巨大的问题。实验结果表明,基于 Key-Value 数据库构建的 Trie 树索引较基于内存构建的 Trie 树索引在事务响应性能和内存消耗方面具有明显的优势和效率。

关键词: Trie 树; key-value 数据库; 地名地址; 输入提示

中图分类号: TP311.13 **文献标志码**: A **文章编号**: 1001-3695(2017)11-3334-05

doi:10.3969/j.issn.1001-3695.2017.11.030

Rapid address searching pre-prompt method based on key-value database

Cao Guangshun[1], Guo Wei[2], Zhu Xinyan[1], She Bing[1]

(1. *State Key Laboratory for Information Engineering in Surveying, Mapping & Remote Sensing, Wuhan University, Wuhan* 430079, *China*; 2. *Key Laboratory of Ministry of Public Security Police Geographic Information Technology, Changzhou Jiangsu* 213022, *China*)

Abstract: The existed address searching pre-prompt for standard address and poi information is relatively few. For the index of the address string, most will use trie tree to index address. But Trie tree consumes huge memory in its establishment. Especially when facing with massive data, the problem is particularly prominent. In view of these problems, this paper proposed a rapid address searching pre-prompt method based on Key-Value database. The method improved the structure of Trie tree, and reduced the complexity of address index. At the same time, the method avoided consuming huge memory in Trie tree establishment because of basing on Key-Value database to build the Trie tree. The experiment results showed that the Trie tree index based on the Key-Value database construction has obvious advantages and efficiency in terms of transaction response performance and memory consumption compared with the Trie tree index based on memory construction.

Key words: Trie tree; Key-Value database; address; pre-prompt

0 引言

输入提示技术[1~3]是根据用户输入的关键词,在搜索框下方实时提示与关键词有着某种联系的提示信息。该技术主流的实现方式是将关键词交给数据库进行精确匹配、相似性匹配[4~6]等检索。输入提示技术的提出极大地提高了输入效率,同时也成为了输入技术的转折点。因此,该技术的应用十分广泛[1],如文本搜索、多媒体与信息检索、互联网搜索引擎、地址搜索等。地址信息[7]可谓与人们的生活密切相关,日常生活中,经常会使用电子地图提供的地址信息进行导航;在某些专业领域,很多的文档及历史记录都需要附带地址信息,以便后续的时空分析[7]。可以说地址信息的核心作用是将各类信息上图,改变原有分散独立的信息状态。地址信息与生活息息相关,不可避免地会经常输入地址查询。而输入提示技术极大地提高了输入效率,因此,输入提示技术在地址信息查询中的应用如火如荼。国内外有很多地图服务商都将输入提示技术应用到地址信息查询中,如国内有百度地图、高德地图、腾讯地图等,国外有 Google 地图、Bing 地图等。

以上地图服务商提供的地址输入提示技术虽然应用十分便捷,但还是存在一些不足:a)以上地图服务商提供的地址输入提示技术针对标准地址数据和 POI 信息的研究相对较少,无法满足某些专业领域对标准地址信息和 POI 信息输入提示的需求;b)针对一些网络相对封闭的专业领域,现有地图服务商提供的地址输入提示服务无法应用,例如,公安领域,由于公安业务的特殊性,公安业务平台无法与互联网互通,即使百度地图现在提供输入的接口,但是当前依然无法应用至网络相对封闭的公安业务平台;c)现有的地址输入提示方法主流采用精确匹配或者模糊匹配[4]等方法实现,匹配效率较低。有的为了快速响应,提高匹配效率,采用 Trie 构建索引进行匹配,例

收稿日期: 2016-11-29; **修回日期**: 2017-01-17 **基金项目**: 警用地理信息技术公安部重点实验室开放基金(2016LPGIT05);公安部重点研究计划课题(2013022DYJ018);国家科技支撑计划资助项目(2012BAH35B03);测绘遥感信息工程国家重点实验室专项科研经费资助项目

作者简介: 曹广顺(1992-),男,黑龙江哈尔滨人,硕士研究生,主要研究方向为地理信息系统开发与应用、数据挖掘(2290168104@qq.com);禹维(1981-),男,副教授,主要研究方向为三维建模;朱欣焰(1963-),男,教授,博士,主要研究方向为空间信息服务、空间数据库等;佘冰(1987-),男,博士(后),主要研究方向为空间统计分析、地理信息抽取.

如文献[4]提出基于哈希和双数组 trie 树的多层次地址匹配算法,在实现多层次地址匹配算法时将地址数据构建成 Trie 树进行匹配;文献[8]重点讨论了用 Trie 树的方式实现从内存中检索完成搜索提示。以上两篇文献的实现方法确实提高了关键词匹配效率,但是 Trie 树建立对内存消耗相对较大,尤其是面对海量数据时,几乎全部依靠内存构建 Trie 树难度相对较大。

基于以上问题,本文提出一种基于 Key-Value 数据库的快速地名地址输入提示方法。该方法基于 Trie 树结构进行改进,在每个子节点处添加以从根节点到该关键词的字符串为前缀的提示词信息,降低了地址索引的复杂度,提高了地址匹配的效率;基于 Key-Value 数据库建立拼音 Trie 树,解决了基于内存建立 Trie 树时内存消耗巨大的问题;针对标准地址数据及 POI 信息进行研究,满足了某些专业领域对标准地址信息及 POI 信息输入提示的需求;服务端采用 Vert. x 进行服务开发,Vert. x-Web 可以实现自己生成 Web 服务器,满足了网络相对封闭领域对地址信息输入提示服务的需求。

1　关键数据结构

1.1　地址数据结构

标准地址数据和 POI 数据作为本文的研究对象,本文根据其特点设计了一种地址数据结构[9],该数据结构主要有以下属性:地址名称、地址唯一标志 ID、地址经度、地址纬度、地址所属行政区。该数据结构主要用于存储海量地址数据,如表 1 所示。

表 1　地址数据结构

名称	类型	说明	名称	类型	说明
addressName	String	地址名称	address_lat	String	地址纬度
addressId	int	地址唯一标志 ID	district	String	地址所属行政区
address_lng	String	地址经度			

1.2　Key-value 数据库结构

Key-value 数据库[10~12]又称键值数据库,其核心思想是以键值对的数据存储模型进行数据存储,并提供持久化机制和数据同步等功能,是一种以键值数据存储为基础的数据库管理系统(DBMS)。其具有以下特性[10~12]:海量存储和高吞吐能力、较强扩展性、数据元组格式较为灵活。这三个特性使得 Key-Value 数据库与常见的关系型数据库在数据模型定义、存储对象以及数据共享冗余方面有着本质上的区别[10~12]。近几年 Key-Value 数据库以其高效的海量数据处理能力和较强的可扩展性走进人们的视野,并在诸多的领域应用中彰显其优势。

本文使用了 H2 数据库中的 key-value 数据库——MVStore[13]。MVStore 是一个多版本、持久化的,以 LZF 为写入策略的 Key-Value 存储系统,是作为 H2 的新一代存储子系统设计。它不需要 JDBC,可以在应用中直接使用。MVStore 具有以下特点[13]:基于多版本页数据结构;以 java. util. Map 接口为基础进行 Key-Value 存取;支持内存、普通文件、加密文件、压缩文件多种存储形式;同时支持事物并发读写。在 MVStore 中,包含一系列特定名称的 MVMap。MVMap 是持久化的 Map,区别于 JDK 中内存的 HashMap 或者 TreeMap。因此,依托 MVStore 中持久化的 MVMap 建立 Trie 树和存储海量地址数据也是本文的亮点所在。

1.3　Trie 树数据结构

Trie[14~16]是数据结构[9]中一种树型结构,是一种哈希树的变种。其核心思想[17]是利用字符串的公共前缀来节约存储空间,最大限度减少了无谓字符的比较,以此达到提高查询效率的目的。Trie 树有 3 三个基本性质[16]:a)根节点不包含字符,除根节点外每一个节点都只包含一个字符;b)从根节点到某一节点,路径上经过的字符连接起来,为该节点对应的字符串;c)每个节点的所有子节点包含的字符都不相同。Trie 树节点 node 包含了节点关键字、是否是某个词的结尾以及子节点信息等,这种 Trie 树结构查询以某个前缀(前缀的长度为 N)开始的所有词,它的事件复杂度[15]则为 O(以该词为前缀的最长词汇的长度,M),最好情况下,$M=N$,但是往往 M 都大于 N,时间复杂度 $O(M)\geqslant O(N)$。

本文构建的拼音 Trie 树使得查找时间复杂度稳定在 $O(N)$。要使得时间复杂度稳定在 O(字符长度),即要在搜索到关键词的最后一个字时就停止搜索,并且能够得出以该关键词为前缀的完整词条。因此,为了提高搜索效率,降低时间复杂度,建立拼音 Trie 树时在子节点处添加了截至该字为起始词的词条 id 集合(最多 10 个),使得搜索不必要穷尽这条线即可得到相应个数的词条。另外,将得到的词条按照词条长度进行排序提示。拼音 Trie 树节点结构如代码 1 所示。其结构示意图如图 1 所示,其中灰色部分为该字位于某一个词的末尾。

代码 1　拼音 Trie 树节点结构

```
node{
  char key;                    //节点关键字
  int nodeId;                  //节点 id
List<Integer> addressIds;
  /*截至该字为起始词的词条 id 集合(最多 10 个)。其中,Integer 是地址数据唯一标志 id 的类型*/
  HashMap<Character,Integer> children;
  /*指向下一个节点的指针结构,用来存放当前字在词中的下一个字的位置,其中 Character 是节点关键字的类型,Integer 是节点 Id 类型*/
}
```

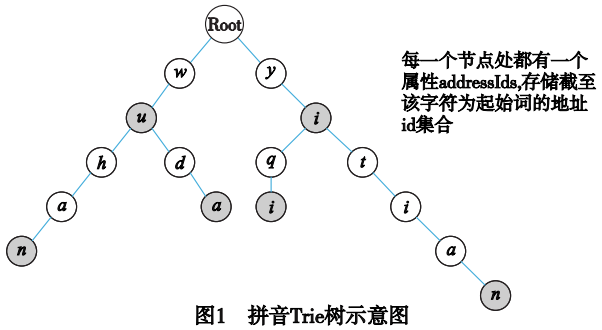


图1　拼音Trie树示意图

2　基于 key-value 的 Trie 树算法

本文中基于 Key-Value 数据库的快速地名地址输入提示方法的核心算法是基于 Key-Value 的 Trie 树算法,该算法实现过程中主要包括地址数据存储、Trie 树建立以及地址索引三个方面。

2.1　地址数据存储

海量地址源数据是 Excel 形式,本文在此处主要是将海量的 excel 地址数据存储在 MVStore 持久化的 MVMap 中。其中 MVMap 的 key 是一条地址数据的唯一标志 id,其 value 是上述地址数据结构所构成可序列化的类,再将海量地址源数据存储

在 Key-Value 数据库 MVStore 中。由于 Key-Value 数据库的高吞吐和高扩展性,在数据存储和索引方面优于关系型数据库,避免了关系型数据库索引时的查询表操作,大幅度提高了效率。

2.2　Trie 树建立

构建拼音 Trie 树的基本算法是遍历全部地址数据。在遍历全部地址数据时,需要将标准地址进行拆分,拆分的原则是根据省、市和区信息进行拆分。例如,“湖北省武汉市经济开发区向家尾后村 73 号 1 栋”,其拆分后得到“湖北省武汉市经济开发区向家尾后村 73 号 1 栋”“武汉市经济开发区向家尾后村 73 号 1 栋”“经济开发区向家尾后村 73 号 1 栋”“向家尾后村 73 号 1 栋”四种形式,这样可以避免输入词与数据源不匹配的问题。将拆分后的地址数据字符串转为拼音,然后把字符串中每一个字母在 Trie 树中进行添加。添加中,如果某个字母字符已经在 Trie 树中存在,则共享该字符;如果不存在,则创建新节点,添加到 Trie 树中。由于本文 Trie 树结构的设计,在每个节点处会添加地址信息的 id 集合,通过该地址 id 可以获得地址的详细信息,如说坐标信息、所属区信息等。另外,由于本文 Trie 树节点存储方式的设计,每一个节点都要存储在持久化的节点 MVMap 中。其中,key 是节点 id,value 是 Trie 树节点数据结构所构成的可序列化的类。作为建立 Trie 树最重要的一步添加节点的算法描述如代码 2 所示。

代码 2　Trie 树建立添加节点算法描述

说明:存储地址数据的是 MVMap addressMap,存储 Trie 树节点的是 MVMap nodeMap;节点唯一标志 nodeId 是从 1 自增长的 Integer。

```
void addNode(String word,String addressId)
  if nodeMap 包含根节点 then
      node < - nodeMap 中根节点
  else
      node < - 新建 nodeId 为 0 的节点
  endif
  for i = 0 to word. length
      c = word 的第 i 个字符
      if node 孩子节点信息中不包含字符 c then
         if nodeMap 中包含当前 nodeId 的节点 then
             nodeChildren < - nodeMap 中当前 nodeId 的节点
         else
             nodeChildren < - 以字符 c 创建一个节点
             nodeChildren 的节点 id 设置为当前 nodeId
             nodeId 加 1
        endif
        node 孩子节点信息. put(nodeChildren 的关键词,node-
Children 的 id)
      else
          id < - 通过字符 c 在 node 孩子节点信息中获得节点 id
          nodeChildren < - 通过 id 获得 nodeMap 中节点
      endif
      //孩子节点中添加 addressId
      if nodeChildren 地址 id 集合数量 < 10 then
          nodeChildren 的地址 id 集合添加 addressId
          putNodeMap(nodeChildren)
      else
          for j = 0 to nodeChildren 地址 id 集合数量
              addressName < - 通过地址 id 在 addressMap 中获
得地址名称
              比较 addressName 和 word 的长度,保留地址名称
长度较短的地址 id 在 nodeChildren 地址 id 集合中
              putNodeMap(nodeChildren)
          end for
      end if
      node = nodeChildren
  end for
end addNode
```

以上算法具体流程如图 2 所示。

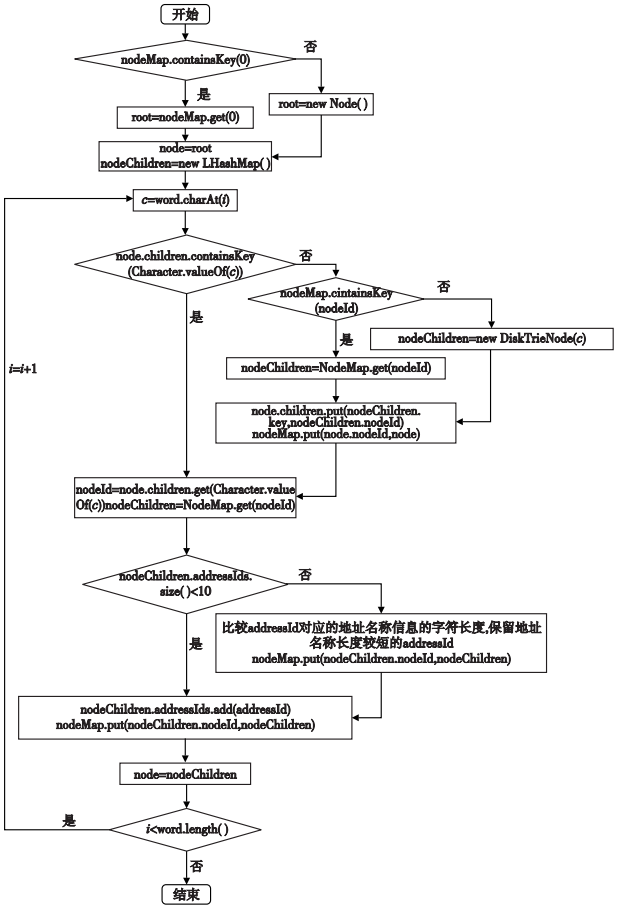


图2　Trie树添加节点算法流程

2.3　地址索引

假设现在已经创建一棵 Trie 树,那么地址索引算法是首先在持久化的节点 MVMap 中获得根节点;然后遍历查询词的每个字符去 Trie 树中进行匹配,直到遍历结束,获得尾部节点处的 addressIds 属性,通过 addressId 在持久化的地址 MVMap 中获得地址详细信息。具体算法描述如代码 3 所示。

代码 3　Trie 树地址索引算法

输入:查询关键词 word。
输出:与关键词相关的地址信息 addressInfos。

```
void searchAddress(String word)
  isExist < - true
  node < - nodeMap 中根节点
  for i = 0 to word. length
      c = word 第 i 个字符
      node_id = 通过字符 c 在 node 孩子节点信息中获得节点 id
      nodeChildren < - 通过 node_id 获得 nodeMap 中节点
      if nodeChildren 是 null then
          isExist = false
          退出此循环
      end if
      node = nodeChildren
  end for
  if isExist 是 True then
      for (遍历 node 地址 id 集合)
          addressInfos. add(通过地址 id 在 addressMap 中获得地
址详细信息)
      end for
  end if
  return addressInfos
end searchAddress
```

以上算法具体流程如图 3 所示。

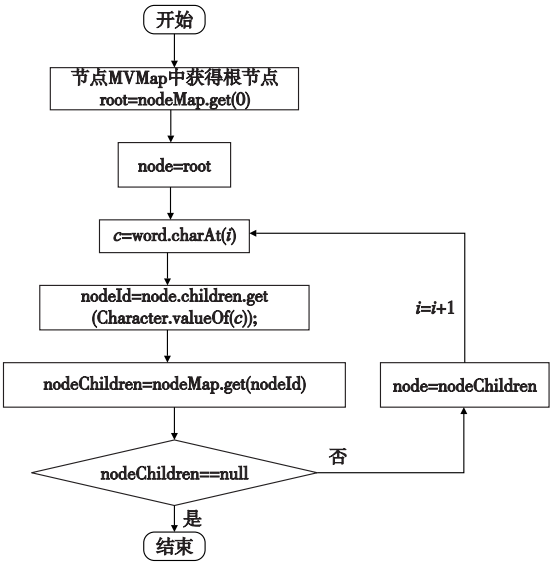


图3　地址索引算法流程

3　实验分析

3.1　实验概述

为验证本文提出的基于 Key-Value 数据库的快速地名地址输入提示方法的有效性和优势,与基于内存的快速地名地址输入提示方法进行对比实验,包括性能压力测试、内存消耗统计及功能测试等。本文采用武汉市标准地址数据及其相应的变化数据、武汉市 POI 数据分别基于 Key-Value 数据库和基于内存建立拼音 Trie 树索引。其中武汉市标准地名地址数据和武汉市 POI 数据总计约 200 万。表 2 描述了系统的测试环境。

表 2　系统测试环境

处理器	IntellCoreI i5 CPU M380 @ 2.53 GHz 2.53 GHz
安装内存(RAM)	48 GB
硬盘	500 GB
系统类型	64 位操作系统
测试数据	武汉市标准地址数据及其变化数据和 POI 数据

3.2　实验结果

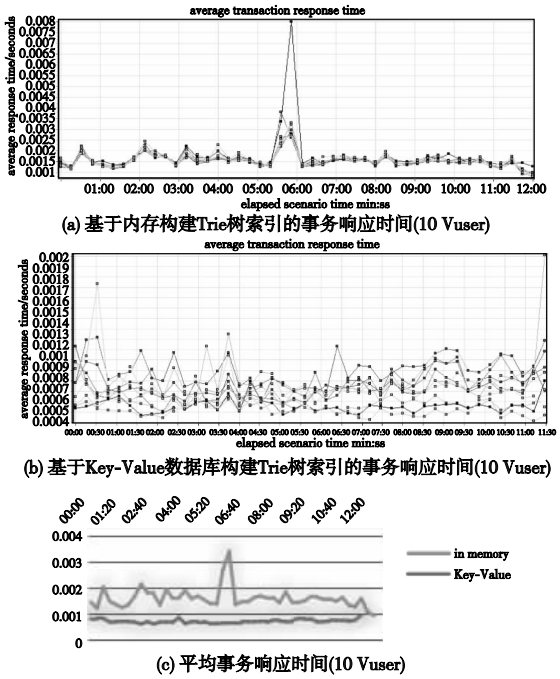
3.2.1　并发对事务响应时间影响

本节采用 LoadRunner 软件分别对基于以上两种方式构建的拼音 Trie 树索引进行性能、压力测试。LoadRunner 是一种预测系统行为和性能的负载测试工具,通过以模拟一定数量用户实施并发负载及实时性能监测的方式来进行系统服务性能压力测试。针对基于内存和 Key-Value 构建的拼音 Trie 树,本文通过逐渐增加 Vuser 的数量来观察多用户并发对两种方式构建的 Trie 树索引事务响应时间的影响。同时,分别设计了 10 个事务,以此来总体反映事务响应性能。

1)10 个 Vuser 进行测试

LoadRunner 模拟 10 个用户同时访问两种方式构建的拼音 Trie 树索引。其中,每个用户访问事务为 10 个。测试结果如图 4 所示。

由图 4(a)(b)可以看出,10 个用户并发访问两种 Trie 树索引时,事务响应时间都相对比较稳定,基于内存的 Trie 树响应时间稳定在 0.002 5 s 以下,虽然有一个事务的响应时间在某一时刻达到 0.008 s,但是可以将其视为偶然情况;基于 Key-Value 的 Trie 树响应时间稳定在 0.002 s 以下。图 4(c)是根据



(a) 基于内存构建Trie树索引的事务响应时间(10 Vuser)

(b) 基于Key-Value数据库构建Trie树索引的事务响应时间(10 Vuser)

(c) 平均事务响应时间(10 Vuser)

图4　10个Vuser时两种Trie树事务响应时间对比

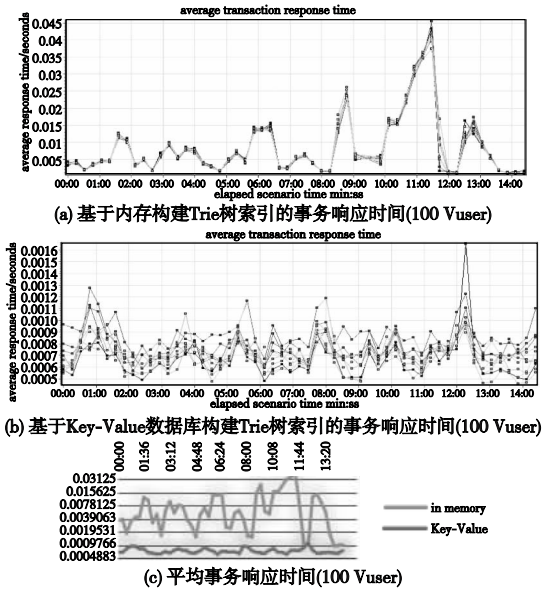
两种 Trie 树在各个时间点上 10 个事务响应时间的均值绘制而成,其均值概要如表 3 所示。由图 4(c)中的对比结果以及表 3 发现,10 个用户并发访问时,基于 Key-Value 的 Trie 树索引事务平均响应时间较基于内存的 Trie 树索引事务平均响应时间快约 2 倍。

表 3　平均事务响应时间平均值(10 Vuser)

in memory	Key-Value
0.001 588 655	0.000 703 94

2)100 个 Vuser 进行测试

LoadRunner 模拟 100 个用户分别对两种 Trie 树索引进行事务访问,测试结果如图 5 所示。



(a) 基于内存构建Trie树索引的事务响应时间(100 Vuser)

(b) 基于Key-Value数据库构建Trie树索引的事务响应时间(100 Vuser)

(c) 平均事务响应时间(100 Vuser)

图5　100个Vuser时两种Trie树事务响应时间对比

由图 5(a)(b)可以看出,100 个用户并发访问两种 Trie 树时,基于内存的 Trie 树事务响应时间随着用户数量的增加波动相对较大,其最大峰值达到 0.045 s;基于 Key-Value 的 Trie 树事务响应时间随着用户数量的增加相对比较稳定,其峰值达到 0.001 6 s。图 5(c)是根据两种 Trie 树在各个时间点上 10

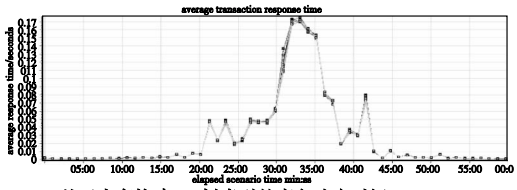
个事务响应时间的均值绘制的,其均值概要如表 4 所示。通过两种 Trie 树平均事务响应时间的比较和表 4 发现,100 个用户并发访问时,基于 Key-Value 的 Trie 树较基于内存的 Trie 树事务平均响应时间快约 10 倍。

表 4　平均事务响应时间平均值(100 Vuser)

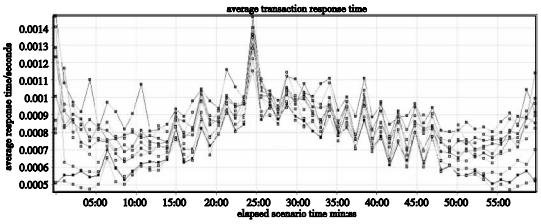
in memory	Key-Value
0.008 408 771	0.000 727 219

3)1 000 个 Vuser 进行测试

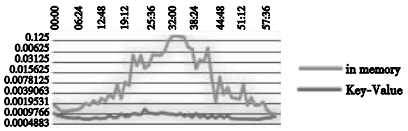
LoadRunner 同时模拟 1 000 个用户分别对两种 Trie 树索引进行事务访问,测试结果如图 6 所示。



(a) 基于内存构建Trie树索引的事务响应时间(1 000 Vuser)



(b) 基于Key-Value数据库构建Trie树索引的事务响应时间(1 000 Vuser)



(c) 平均事务响应时间(1 000 Vuser)

图6　1 000个Vuser时两种Trie树事务响应时间对比

由图 6(a)(b)可以看出,1 000 个用户同时并发访问时,基于内存的 Trie 树事务响应时间随着用户数量的增加明显变慢,其峰值为 0.17 s,随后随着用户数量的减少,事务响应时间才逐渐变快;基于 Key-Value 的 Trie 树事务响应时间随着用户数量的增加也有变慢的趋势,但是总体相对趋于稳定,一直稳定在 0.001 4 s 以下。图 6(c)结果是根据两种 Trie 树在各个时间节点上 10 个事务响应时间的均值绘制的,其均值概要如表 5 所示。通过该对比结果和表 5 发现,当 1 000 个用户并发访问时,基于 Key-Value 的 Trie 树较基于内存的 Trie 树事务平均响应时间快约 30 倍,前者在事务响应性能方面较后者具有明显的优势和效率。

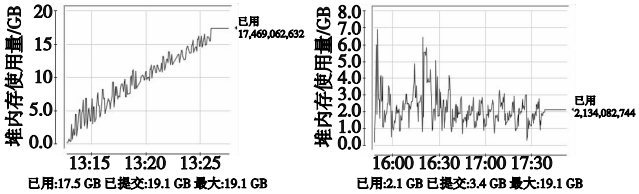
表 5　平均事务响应时间平均值(1 000 Vuser)

in memory	Key-Value
0.027 393 146	0.000 814 614

比较 10、100、1 000 个用户访问同一种 Trie 树索引时发现,基于内存的 Trie 树索引事务响应时间随着用户数量的增加逐渐变慢,事务平均响应时间由 10 个用户时的 0.001 6 s 增加到 100 个用户时 0.008 s,再增加到 1 000 个用户时的 0.027 s;而基于 Key-Value 的 Trie 树索引事务响应时间却随着用户数量的增加变化较小,相对趋于稳定,一直稳定在 0.000 8 s 左右。根据以上分析发现,本文提出的基于 Key-Value 数据库的快速地名地址输入提示方法相对基于内存的快速地名地址输入提示方法在事务响应时间方面具有一定的有效性和优势,尤其是在多用户并发访问时,前者优势愈发明显。

3.2.2　内存消耗统计

本节采用 JConsole 来检测两种方式构建 Trie 树时内存的消耗情况。JConsole 是一个基于 JMX 的 GUI 工具,用于连接正在运行的 JVM,可以监控正在运行的 JVM 堆内存使用量的情况,监控结果如图 7 所示。



(a) 基于内存构建Trie树内存消耗　(b) 基于Key-Value构建Trie树内存消耗

图7　基于两种方式构建Trie树内存消耗对比

观察图 7(a)(b)发现,基于内存构建 Trie 树时,随着地址数据量的增加,内存消耗量在逐渐增加,直到所有数据全部添加进 Trie 树后,内存消耗情况趋于稳定,稳定在 17.5 GB 左右;而基于 Key-Value 构建 Trie 树时,内存消耗一直稳定在 8 GB 以下,可见,基于内存构建 Trie 树内存消耗约是基于 Key-Value 构建 Trie 树内存消耗的 2 倍。而且基于 Key-Value 构建 Trie 树只需一次构建,而后不需要消耗内存即可继续使用已构建的 Trie 树;基于内存构建 Trie 树却是每次都需要消耗内存构建 Trie 树。根据以上分析发现,基于 Key-Value 构建 Trie 树确实避免了基于内存构建 Trie 树时内存消耗巨大的问题,尤其是面临海量数据时,前者优势愈发明显。

4　原型系统功能

本文针对武汉市标准地址数据及 POI 信息进行研究,服务后端使用 Java、Vert. x 进行服务部署开发。根据用户输入的关键词,返回相应的预提示地址信息和 POI 信息,原型系统功能实现如图 8 所示。

(a)　(b)

图8　基本查询提示案例

图 8 所示的功能完全满足了某些专业领域对标准地址信息和 POI 信息输入提示的需求,同时本文使用 Vert. x 进行服务部署开发,解决了现有地址输入提示服务无法应用于网络相对封闭的领域(如公安领域)的问题。

5　结束语

本文提出一种基于 key-value 数据库的快速地名地址输入提示方法,该方法基于 Trie 树进行改进,在 Trie 树的节点处添加从根节点处到该节点处关键词为起始的 TOP-10 的地址数据 id,降低后续查询的时间复杂度。实验验证,基于 Key-Value 数据库构建拼音 Trie 树索引,解决了以往基于内存构建 Trie 树时内存消耗巨大的问题。同时,基于 Key-Value 数据库构建的拼音 Trie 树索引在多用户并发访问时事务响应时间明显优于基于内存构建的拼音 Trie 树索引。此外,本文针对武汉市标准地址及 POI 信息进行研究,满足了如公安这样网络相对封闭的专业领域对标准地址输入提示服务的需求。(下转第　页)

操作。

图10 数据库迁移之后的MongoDB中的part集合的一个文档

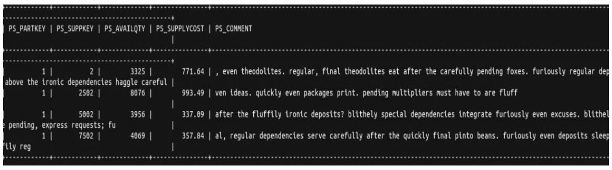


图11 part_supp表记录以数组的形式嵌入到part集合中

3 结束语

本文对传统 RDBMS 向文档型非关系数据库 MongoDB 数据模型自动转换、数据迁移以及混合应用架构下数据同步技术进行了研究，提出了关系有向图动态识别和生成算法、基于关系有向图的关系及关系间参照完整性向 MongoDB 集合的转换算法、基于 RDBMS 日志的 RDBMS 数据向 MongoDB 数据迁移算法。本文的研究主要在数据库层次的转换和应用方面，然而对于架构在 RDBMS 上的应用系统而言，一旦 RDBMS 需要被 MongoDB 所替代，并不能实现在不修改上层应用的条件下，上层应用可直接访问 MongoDB，也不能实现传统 RDBMS 上的 DML 操作能直接转换为 MongoDB 的数据操作。这是本文后续将进步一研究的内容。

参考文献：

[1] Han Jing, Haihong E, Le Guan, *et al*. Survey on NoSQL database [C]//Proc of the 6th International Conference on Pervasive Computing and Applications. 2011:363-366.

[2] Stonebraker M. SQL databases vs. NoSQL databases[J]. Communications of the ACM, 53(4):10-11.

[3] Zhu Weiping, Li Mingxin, Chen Huan. Using MongoDB to implement textbook management system instead of MySQL[C]//Proc of the 3rd International Conference on Communication Software and Networks. 2011:303-305.

[4] 雷德龙，郭殿升，陈崇成，等. 基于 MongoDB 的矢量空间数据云存储与处理系统[J]. 地球信息科学学报，2014，16(4):507-516.

[5] Liana S, Marius B, Dumitu D B. Auto mapping MySQL to NoSQL MongoDB[C]//Proc of Federated Conference on Computer Science and Information Systems. 2016.

[6] Jia Tianyu, Zhao Xiaomeng, Wang Zheng, *et al*. Model transformation and data migration from relational database to MongoDB[C]//Proc of IEEE International Congress on Big Data. 2016.

[7] 黄伟钗. MongoDB 的形式化模型和研究[D]. 广州：中山大学，2013.

[8] Kanade A, Gopal A, Kanade S. A study of normalization and embedding in MongoDB[C]//Proc of IEEE International Advance Computing Conference. 2014.

[9] Zhao Gansen, Huang Weichai, Liang Shunlin, *et al*. Modeling MongoDB with relational model [C]//Proc of the 4th International Conference on Emerging Intelligent Data and Web Technologies. 2013: 115-121.

[10] Roijackers J, Fletcher G. Bridging SQL and NoSQL[D]. [S. l.]: Eindhoven University of Technology, 2012.

[11] 陶勇，丁维明. 数据库中规范化与反规范化设计的比较与分析[J]. 计算机技术与发展，2006，16(4):107-109.

(上接第 页)对于用户输入词的容错问题、根据用户输入词频次进行提示排序等问题，有待进一步深入研究。

参考文献：

[1] 官连军，丁光耀，张燕妮. 支持多种模糊处理的英文智能预提示输入引擎[J]. 成都信息工程学院学报，2012，27(2):131-135.

[2] Li Guoliang, Ji Shengyue, Li Chen, *et al*. Efficient fuzzy full-text type-ahead search[J]. The Vldb Journal, 2011, 20(4):617-640.

[3] Feng Jianhua, Li Guoliang. Efficient fuzzy type-ahead search in XML data[J]. IEEE Trans on Knowledge & Data Engineering, 2010, 24(5):882-895.

[4] 徐聪，张丰，杜震洪，等. 基于哈希和双数组 trie 树的多层次地址匹配算法[J]. 浙江大学学报：理学版，2014，41(2):217-222.

[5] 郑志宏，郑志高，王玉婷. 模糊查找算法的设计实现[J]. 情报科学，2010，28(6):915-918.

[6] 苗兰芳，杨传斌. 模糊串匹配算法及其应用[J]. 小型微型计算机系统，1996，17(10):72-76.

[7] 严潇，芮维，樊亚新，等. 融合空间计算的 POI 地理编码方法[J]. 计算机应用研究，2016，33(9):2652-2655.

[8] 杜星，王洪波. 基于 Trie 树的搜索提示设计与实现[EB/OL]. [2014-11-15]. http://www.paper.edu.cn/releasepaper/content/201411-288.

[9] 严蔚敏，吴伟民. 数据结构[M]. 北京：清华大学出版社，2002.

[10] 肖红玉，陈海，黄静. Key-value 数据库的研究与应用[J]. 自动化与仪器仪表，2010(4):87-89.

[11] 苏翔宇. Key-value 数据库及其应用研究[J]. 电脑知识与技术，2012，8(5):1009-1011.

[12] 周沫. Key-value 数据库的发展与展望[J]. 信息与电脑：理论版，2012(7):152-153.

[13] MVStore. Overview of MVStore[EB/OL]. [2016-11-25]. http://www.h2database.com/html/mvstore.html

[14] 李琪，郭敏. 基于字典树的基数排序算法[J]. 硅谷，2011(14):192-193，151.

[15] 乐娟. 基于 Trie 树的京剧术语语义词典[J]. 计算机工程，2011，37(S1):30-32.

[16] 蔡琪，刘东霞. 基于 Trie 树的快速电子词典开发[J]. 现代电子技术，2014，37(12):90-92.

[17] 刘丽霞，张志强. 基于 Trie 树的相似字符串查找算法[J]. 计算机应用，2013，33(8):2375-2378.