面向 Redis 的数据序列化算法研究

孙杜靖,李玲娟
(南京邮电大学 计算机学院,江苏 南京 210003)

摘 要:为了解决实时计算中半结构化和非结构化数据的存储问题,借助内存数据库 Redis 可以存储键值型数据和支持全内存运算的优势,结合文件序列化、图像序列化、JSON 序列化和 Java 对象序列化技术,设计了面向 Redis 的半结构化和非结构化数据的序列化算法。该序列化算法不仅解决了半结构化和非结构化数据无法直接存入 Redis 的问题,而且由于在序列化过程中实现了对这些数据的深拷贝,使得反序列化可以完美地还原初始数据。此外,序列化过程还支持通过加解密来保障数据安全。基于 Storm 平台的实验结果表明,所设计的序列化算法快速、有效且性能稳定。在海量数据实时计算中,无论使用哪种开发语言,将该算法与 Redis 数据库结合,既能利用 Redis 带来的高读写效率,又能存储任何半结构化和非结构化数据对象而无需重复开发代码。

关键词:Redis;序列化;半结构化;非结构化;Storm
中图分类号:TP391 **文献标识码**:A **文章编号**:1673-629X(2017)05-0077-05
doi:10.3969/j.issn.1673-629X.2017.05.017

Investigation on Data Serialization Algorithm for Redis

SUN Du-jing, LI Ling-juan
(School of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: In order to deal with the problem of storing semi-structured and unstructured data in real-time calculation, a data serialization algorithm for Redis is designed. It takes advantage of Redis which can store key-value data and support full memory operation and uses the technologies of file serialization, image serialization, JSON serialization and Java object serialization. The algorithm can not only solve the problem that the semi-structured and unstructured data cannot be directly stored into Redis, but also enable the deserialization to restore the original data perfectly by keeping a deep copy of the data. In addition, encrypting and decrypting can be added to the serialization process to ensure the security of data. The experimental results on Storm platform show that the proposed algorithm is fast, effective and stable. In the real-time processing of massive data integration, this algorithm with Redis can not only make reading and writing highly efficient, but also store any semi-structured and unstructured data without rewriting code no matter which programming language is employed.

Key words: Redis; serialization; semi-structured; unstructured; Storm

0 引 言

大数据时代的到来使得半结构化数据、非结构化数据迅猛增长,而处理和存储这些数据的需求也日益增长。传统的关系型数据库只能存储结构化数据,并且对于高并发的数据写操作、海量数据的存储和快速查询以及服务器的横向扩展[1]显得无能为力。为了解决这些问题,NoSQL[2](Not Only SQL)技术应运而生。NoSQL 主要分为四类:键值(Key-Value)存储数据库、列存储数据库、文档型数据库和图形数据库。键值存储数据库使用哈希来构建数据库,列存储数据库善于分布式存储,文档型、图形数据库顾名思义在文档型、图数据处理方面优势明显。

Redis(Remote Dictionary Server)[3] 就是 NoSQL 中属于键值存储数据库的一个产品,并且是一个内存型数据库,全内存运算和存储使其在高并发的操作下仍能保持高性能读写,是已知性能最快的 Key-Value 数据库[4]。Key-Value 模型的内存数据库,支持多种语言接口,如 C++、C#、Java、JavaScript、Python 等。

收稿日期:2016-06-20 **修回日期**:2016-09-22 **网络出版时间**:2017-03-07
基金项目:国家自然科学基金资助项目(61302158,61571238)
作者简介:孙杜靖(1992-),女,硕士研究生,CCF 会员,研究方向为流数据挖掘;李玲娟,教授,CCF 会员,研究方向为数据挖掘、信息安全、分布式计算。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20170307.0922.086.html

Key-Value 数据库利用哈希表维护 Key 值到具体数据(Value)的映射,通过 Key 值可以方便高效地查询数据[5]。Redis 通过缓存数据库查询结果,减少对硬盘的访问次数,其缓存数据库全部加载在内存中进行操作,定期通过异步快照或者日志操作将数据库数据 flush 到硬盘上进行保存。因采用纯内存操作,每秒可以处理超过 10 万次读写操作,Redis 性能非常高。Redis 还有其他一些优势,比如提供了丰富的数据结构、支持主从复制、完善的持久化机制等等。

实时计算中需要频繁读取数据,Redis 通过访问缓存数据库读取数据,可以保证计算过程中读取需求的实时响应、数据库数据的实时更新,因此 Storm[6] 等实时计算平台借助 Redis 键值存储和内存操作的优势将能够更好地完成流数据的实时处理任务。但是,采用全内存的 Key-Value 存储形式,Redis 虽然能满足实时处理的需求,却不能直接存储和处理大量半结构化及非结构化数据,而这些数据是 Storm 等流式实时计算平台必须面对的。

为此,设计了面向 Redis 的序列化算法,并通过在 Storm 平台上的实现结果,证明了该算法在保证 Redis 高性能读写的前提下,解决实时计算中半结构和非结构化数据的存储问题。

1 面向 Redis 的数据序列化算法设计思想

如上所述,要充分利用 Redis 的优势,需要解决 Redis 作为键值型数据库不能直接存储和处理 Storm 等流式实时计算平台必须面对的半结构化及非结构化数据的问题。为解决这一问题,可以利用 Redis 能够存储二进制流的特性,先将半结构化及非结构化数据进行序列化,然后存入 Redis 中,读取后再进行反序列化,使 Redis 存储各种数据成为可能。基于此思路,设计了面向 Redis 的序列化算法,该算法考虑了半结构化数据、非结构化数据和需要保密的数据。非结构化数据的序列化包括:采用文件流的序列化和图数据序列化;半结构化数据的序列化包括:采用 JSON[7] 的序列化和采用 Java 对象的序列化。

图 1 给出了算法的总体思想。

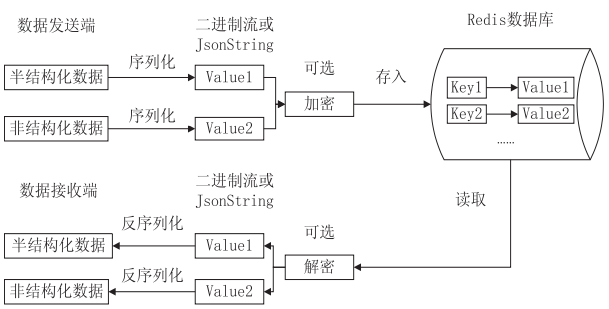

图 1 面向数据的半结构化及非结构化数据序列化

数据发送端将半结构化或非结构化数据先序列化成二进制流或 JSON 字符串,如有敏感数据,可以将其加密,然后存入 Redis 数据库;数据更新时,以半结构化数据为例,Value1 值会随之改变,但是存入的键 Key1 值不变,即 Redis 数据库中存入的 Value1 值也是实时更新的,如果数据消费者想要查看此半结构化数据,可通过事先约定好的键 Key1 值来获取数据,如曾经加密过,则需解密;然后将获得的二进制流或 JSON 字符串进行反序列化,得到原始数据。由于 Redis 数据库中存入的数据是实时更新的,数据消费者获取的数据也是最近存入的数据,对于实时更新的半结构化数据只采用一个 KEY 值进行存储的原因有三:一是通信双方可以事先规定好 KEY 值,无需频繁修改;二是 Redis 为内存型数据库,读写速率快,完全可以支持数据的实时更新和实时读取;三是若采用不同时刻不同 KEY 值,在实时计算中,会迅速占满内存,使 Redis 可能无法再存入新的数据。

2 面向 Redis 的数据序列化算法描述

2.1 非结构化数据的序列化

非结构化数据即为不方便用传统数据库二维逻辑表来存储的数据,包括文档、文本、图像、音频、视频等等。传统数据库通过创建一个三字段(编号 number、内容描述 varchar(1024)、内容 blog)的表来对其进行索引,需要的人工参与量比较大,面对海量非结构化数据的存储与检索时,该方法显然不可能在较短时间内将其整理入库,数据的价值也无法很好地发挥。所设计的基于 Redis 的数据序列化算法采用将非结构化数据进行序列化后存入内存数据库 Redis 中的方案,能很好地解决此类问题,具体包括采用文件流的序列化和图数据序列化。

1)采用文件流序列化非结构数据。

对非结构化数据序列化分为两个过程:序列化和反序列化,对应的方法分别是 writeObject() 和 readObject()。

(1)writeObject():将非结构化数据 A 序列化(序列化过程)。

输入:非结构化数据 A,可以是文档、图像、音频、视频等等,路径 P

输出:字节数组

步骤为:

①while(A 更新一次)

②将 A 写入到底层输入流 A′;

③通过文件输出流将 A′以字节流的方式保存到指定路径 P;

④end while

(2) readObject( ):将指定路径中的对象转化为原始非结构化数据(反序列化过程)。

输入:路径 P

输出:原始非结构化数据 A

步骤为:

①将路径 P 中的对象读入原始字节流 A′;

②从输入流中读取数据对象类数据 A′;

③将 A′转换成抽象 Object 类 A;

④return A

数据发送方通过 writeObject( )函数将非结构化数据序列化,然后将序列化的结果作为 value 存入 Redis 的某一路径下,数据接收方先根据数据发送方存入的 key 取出相应的字节流,然后通过 readObject( )方法,将字节流反序列化为抽象类,而原始数据类型可能是文档、图像、音频或视频,反序列化后格式不变,可将其下载到某个路径下查看,如若知道其为文档,可以将其向下转型成文件类,直接进行其他数据处理。

2)对图数据的序列化。

图像是非结构化数据的一种,视频、图像的实时计算处理过程中需要对大量照片进行存储、计算。在线系统中,在线数据的备份和恢复和对原始数据进行测试需要使用到快照[7-8],而快照由于其实时更新,并且需要被不断读取和计算,对其进行存储并保证其能被快速读取显得尤为重要。由于图数据使用频繁,频繁的文件读写会增加时间开销;另一方面,图数据占用内存过大时,所涉及的内存消耗和网络传输会很大,为了避免文件读写以减少时间开销、内存消耗和网络传输,采用如下的图像序列化方案。该方案分为两个过程:序列化和反序列化,对应的方法分别是 writeImage( )和 byte2image( )。

(1) writeObjectwriteImage( ):将图像转化成字节数组(序列化过程)。

输入:实时更新的某张图像 A

输出:字节数组

步骤为:

①while(A 更新一次)

②将图像 A 以某种格式(如 png)写入内存;

③字节输出流捕获内存缓冲区的数据 A′,转换成字节数组 B;

④end while

⑤return B

(2) byte2image( ):将从 Redis 读取的字节数组 B 转化为图像(反序列化过程)。

输入:字节数组 B

输出:原始图像 A

步骤为:

①将字节数组 B 写入图像文件输出流中;

②输出流探测图像格式,并调用对应的插件进行解码,得到原始图像 A;

③eturn A

数据发送方通过 writeImage( )方法将图像序列化,然后将序列化的结果作为 value 存入 Redis,数据接收方先根据数据发送方存入的 key 取出相应的二进制流,然后通过 byte2image( )方法,将二进制流反序列化为图像,直接进行数据处理。

2.2　半结构化数据的序列化算法

半结构化的数据有一定的结构性,但结构变化很大,OEM[9](Object Exchange Model)是它的典型代表。由于其结构性,需要了解数据内部细节,因而不能将其看成非结构化数据,像 2.1 节那样将数据简单组织成一个文件;又因其数据结构变化大,也不能将其按照结构化数据处理方式存入二维表。设计了两种方法将其序列化后存入 Redis 数据库,一种是采用 JSON(JavaScript Object Notation)数据交换格式,另一种是利用 Java 对象序列化技术。

1)采用 JSON 序列化半结构化数据。

JSON 是一种轻量级的数据交换格式,是基于 JavaScript Programming Language, Standard ECMA-262 3rd Edition-December 1999 的一个子集。它易于机器解析和生成,因而用于在不同的编程语言之间交换数据,比如 JavaScript 和 Java、C#间交互。

JSON 主要有两种结构:对象和数组。

对象:用“{}”括起来的内容,数据结构为{key:value,key:value…}的键值对,key 为对象属性,value 为对应的属性值,通过“对象. key”来获取属性值,而属性值的类型可以是数字、字符串、数组或者对象。

数组:用“[ ]”括起来的内容,数据结构为[“java”,“c#”,“javascript”,“redis”…],通过索引进行取值,字段值的类型同样可以是数字、字符串、数组或者对象。

将半结构化的数据序列化为 JSON 字符串的方法是 object2json( ),具体的序列化过程如下:

object2json( ):将半结构化数据序列化,转换成 JSON 字符串。

输入:实时更新的半结构化数据 A

输出:JSON 字符串 S

步骤为:

①while(A 更新一次)

②if(A 为空)

③S. append(“”);

④else if(A 是 String、Integer、Boolean、Byte 等基本类型)

⑤S. append(“A”);

⑥else if(A 是 Object[ ]、List、Map、Set 类型数据)

⑦调用类似于 S. append(array2json((Object[ ])A))的方法进行解析;

⑧else S. append(bean2json)格式化输出;

⑨return S. toString( )

在 Redis 中,数据发送方先将半结构化数据通过 object2json 方法进行序列化,转换成 JSON 字符串,然后将该 JSON 字符串作为值写入特定的键中,即以(key,value)的形式写入 Redis。数据接收方通过 key 取出相应的 value-JSON 字符串,再将 JSON 字符串通过 fromObject 方法转换成 JSONObject,将对原始数据的解析转换成对 JSONObject 的解析。

将半结构化的数据序列化为 JSON 数据,优点是可以支持多种编程语言,并且被序列化的对象可以继续添加或者删除成员变量而不用变更 object2json 方法。但是,被序列化的对象必须要有无参数的构造方法和所有变量的 getter 和 setter 方法。数据接收方须知道被序列化的对象的“key”和该对象所有的成员变量,才能完全地解析 JSONString。

2)采用 Java 对象序列化技术序列化半结构化数据。

利用 Java 对象序列化技术将半结构化数据序列转化为二进制字符串的方法是:writeObject( )和 readObject( )。

(1)writeObject( ):由于 Java 中所有的对象都继承自 Object 类,考虑到代码的可重用性,可以利用父类 Object 的 writeObject 方法将所有不同类型的对象转换成字节数组,具体的序列化过程如下:

输入:实时更新的半结构化数据 A

输出:字节数组

步骤为:

①while(A 更新一次)

②A 向上转型成 Object 类 A′;

③从对象流中读取对象 A′,写入内存;

④捕获内存缓冲区数据 A′,转换成字节数组 B;

⑤end while

⑥return B

(2)readObject( )方法:利用抽象类 Object 类的 readObject 方法将字节数组转化为 Object 类,具体的反序列化过程如下:

输入:字节数组

输出:原始数据 A′

步骤为:

①将字节数组 B 转化为输入流 B′;

②将输入流中的数据 B′输入对象输入流 B″;

③从流中读取对象,恢复对象状态,得到 A′;

④return A′

数据发送方通过 Java 对象序列化技术,先将对象向上转型成父类 Object 类,然后通过 writeObject( )方法将对象序列化成二进制流,数据接收方经由 readObject( )方法将接收到的二进制流反序列化为抽象类 Object 类,然后再向下转型为原始数据类型。

2.3 对敏感数据字段进行加密

一些数据中可能存在敏感字段,比如年龄、性别、电话号码、密码等等,图像也可能涉及机密或隐私,序列化的优点是序列化之后的数据格式比较简单且统一,可以对其运用 DES[10]、AES[11]、RSA[12]、MD5[13]等经典加密算法或者自定义加密算法进行加密。数据发送方将加密后的二进制流写入 Redis,数据接收方需先将接收到的二进制流进行解密,再进行反序列化。即使 key 值被泄露,value 值还需先解密才能被成功的反序列化,对于无密钥的数据拦截者而言无法获取原始数据,从而保证了数据的安全。

3 实 验

3.1 实验环境

设计了基于流计算平台 Storm 的实验来测试所设计的序列化算法的性能。

Storm 是 Twitter 支持开发的一款分布式的、实时的、主从式大数据流式计算系统[14]。实时性主要体现在其可以处理流数据而非静态数据,并实时更新计算结果[6],主从架构由一个主节点 Nimbus 和多个工作节点 Supervisor 组成。Nimbus 负责在集群中分发代码,分配计算任务给机器,并且监控集群状态。Supervisor 负责监听分配给它的机器的工作,根据需要启动或关闭工作进程。

实验的硬件环境:内存 8 GB,CPU 为主频 2.7 GHz 的 i7 处理器,1 个 Nimbus 节点、2 个 Supervisor 节点的 Storm 集群。

软件环境:Storm0.9.1,JRE1.7,Zookeeper-3.4.6、redis-2.4.5。

操作系统:Centos6.4。

编程语言:Java。

3.2 序列化算法的正确性测试

(1)采用文件流序列化非结构化数据的测试结果。

将需要测试的商品号(1417,2227,3967,7237,8467,10477,10777…)序列化后存入 Redis,然后反序列化写入路径 Pubic/sundujing 下的 test_items. txt 中,如图 2(a)所示,文件内容如图 2(b)所示,与原始数据一致。

```
[root@node1 sundujing]# ls
test_items.txt
```

(a)反序列化结果

```
[root@node1 sundujing]# more test_items.txt
1417
2227
3967
7237
8467
10477
10777
```

(b)反序列化后的文件内容

图 2　采用文件流序列化非结构化数据

(2)图数据序列化测试结果。

图数据序列化的测试情况如图 3 所示。对图 3(a)的序列化结果为[B@17ee8b8,反序列化结果为图 3(b)。

(a) 原始图像　　(b) 反序列化后的图像

图 3　图数据序列化

(3)半结构化数据序列化测试结果。

半结构化数据的序列化与反序列化以简单的商品信息 ITEM 为例,ITEM 有商品号 item_id、商品所属类目号 cat_id、商品标题分词后的结果 terms 这三个属性。一个简单的实例 item:商品号 29、商品所属类目号 155、商品标题分词为 123950,53517,106068,59598,7503,171811,25618,147905。JSON 序列化结果为 item----{"item_id":"29","cat_id":"155","terms":"123950,53517,106068,59598,7503,171811,25618,147905"};Java 对象序列化结果为[B@fa3ac1,经过反序列化后,能得到原始数据实例 item。

3.3　序列化算法的效率测试

以下通过 Redis 存取序列化数据的性能来体现序列化算法的效率。采用用户购买记录作为数据集,如图 4 所示,三个字段分别为用户 id、商品 id、用户购买时间。将用户 id 作为 key 值,将用户购买的商品 id 作为 value 值存入对应的用户 id 中,用 50 万、100 万、150 万条购买记录进行测试,Redis 测试所需时间如图 5 所示。

Redis 官方声明过,Redis 在极佳的情况下能达到每秒 10 万次读写,而在该次实验中 50 万条数据序列化后存入只需 40 s,并且随着需处理的数据条数的增长,时间呈线性增长,不会因此导致内存激增而影响其他线程的执行,由此可以看出 Redis 存取序列化后的数据的性能很稳定。


```
1585376 12 20140903
7861561 12 20140920
11060852 12 20141023
11042247 12 20140919
9012005 12 20141017
12161339 12 20140903
3200706 12 20140903
159108 12 20141025
1388689 12 20140903
2928808 12 20140903
```

图 4　测试数据集截图

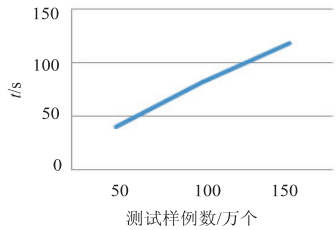

图 5　Redis 时效测试结果

4　结束语

基于 Redis 的特点和流式实时计算平台对半结构化和非结构化数据的处理需求,设计了面向 Redis 的数据序列化算法,借助文件序列化、图像序列化、JSON 序列化和 Java 对象序列化技术,解决了 Redis 无法直接存储半结构化与非结构化数据的问题,并在流数据处理平台 Storm 上通过实验证明了该算法能有效解决实时计算中半结构化与非结构化数据的存储和实时读取问题。

在海量数据实时计算中,无论使用哪种开发语言,使用 Redis 作为数据库并采用设计的序列化算法,一方面,当需要被处理的数据很大时,能有效降低系统的内存消耗和网络传输;另一方面,不仅可以利用 Redis 带来的高性能读写效率,而且可以存储任何半结构化数据甚至非结构化数据对象而无须重复开发代码,所设计的序列化算法能高效地解决半结构化和非结构化数据的存储问题。

参考文献:

[1] 蔡金花. 浅析 NOSQL 及使用[J]. 电脑知识与技术,2011,7(12):2757-2758.

[2] 宗　平,吴秀娟. 基于 NoSQL 系统的组合索引技术研究[J]. 计算机技术与发展,2014,24(12):53-56.

[3] 曾泉匀. 基于 Redis 的分布式消息服务的设计与实现[D]. 北京:北京邮电大学,2014.

[4] 苏翔宇. Key-Value 数据库及其应用研究[C]//中国职协 2013 年度优秀科研成果获奖论文集(下册). 出版地不详:出版者不详,2013.

(下转第 86 页)

nication,2010,52(1):12-40.
[2] Samantaray A K, Mahapatra K, Kabi B, et al. A novel approach of speech emotion recognition with prosody, quality and derived features using SVM classifier for a class of north-eastern languages[C]//2nd international conference on recent trends in information systems. [s. l.]:IEEE,2015:372-377.
[3] Sun Y, Wen G, Wang J. Weighted spectral features based on local Hu moments for speech emotion recognition[J]. Biomedical Signal Processing and Control,2015,18:80-90.
[4] Tuerxun M, Zhang S, Bao Y, et al. Improvements on bottleneck feature for large vocabulary continuous speech recognition [C]//12th international conference on signal processing. [s. l.]:IEEE,2014:516-520.
[5] Karafiát M, Grézl F, Hannemann M, et al. BUT BABEL system for spontaneous Cantonese[C]//Proceedings of Interspeech. [s. l.]:[s. n.],2013:2589-2593.
[6] Zhang Y, Chuangsuwanich E, Glass J R. Extracting deep neural network bottleneck features using low-rank matrix factorization[C]//ICASSP. [s. l.]:[s. n.],2014:185-189.
[7] Liu Y, Qian Y, Chen N, et al. Deep feature for text-dependent speaker verification[J]. Speech Communication,2015,73:1-13.
[8] Safari P, Ghahabi O, Hernando J. Feature classification by means of deep belief networks for speaker recognition[C]//23rd European signal processing conference. [s. l.]:IEEE,2015:2117-2121.
[9] Pal A, Baskar S. Speech emotion recognition using deep dropout autoencoders[C]//International conference on engineering and technology. [s. l.]:IEEE,2015:1-6.
[10] Zhang W, Zhao D, Chen X, et al. Deep learning based emotion recognition from Chinese speech[M]//Inclusive smart cities and digital health. [s. l.]:International Publishing,2016:49-58.
[11] 王 一,杨俊安,刘 辉,等. 基于层次稀疏 DBN 的瓶颈特征提取方法[J]. 模式识别与人工智能,2015,28(2):173-180.
[12] 李晋徽,杨俊安,王 一. 一种新的基于瓶颈深度信念网络的特征提取方法及其在语种识别中的应用[J]. 计算机科学,2014,41(3):263-266.
[13] 陈 雷,杨俊安,王 一,等. LVCSR 系统中一种基于区分性和自适应瓶颈深度置信网络的特征提取方法[J]. 信号处理,2015,31(3):290-298.
[14] Grézl F, Karafiát M, Kontár S, et al. Probabilistic and bottleneck features for LVCSR of meetings[C]//Proceedings of the IEEE international conference on acoustics, speech, and signal processing. Honolulu, USA:IEEE,2007:757-760.
[15] Gehring J, Miao Y, Metze F, et al. Extracting deep bottleneck features using stacked auto-encoders[C]//IEEE international conference on acoustics, speech and signal processing. [s. l.]:IEEE,2013:3377-3381.
[16] 张春霞,姬楠楠,王冠伟. 受限波尔兹曼机[J]. 工程数学学报,2015,32(2):159-173.
[17] You Y, Qian Y, He T, et al. An investigation on DNN-derived bottleneck features for GMM-HMM based robust speech recognition[C]//China summit and international conference on signal and information processing. [s. l.]:IEEE,2015:30-34.
[18] 陶华伟,查 诚,梁瑞宇,等. 面向语音情感识别的语谱图特征提取算法[J]. 东南大学学报:自然科学版,2015,45(5):817-821.
[19] Burkhardt F, Paeschke A, Rolfes M, et al. A database of German emotional speech[C]//Proceedings of Interspeech. [s. l.]:[s. n.],2005:1517-1520.
[20] Anagnostopoulos C N, Iliou T, Giannoukos I. Features and classifiers for emotion recognition from speech: a survey from 2000 to 2011[J]. Artificial Intelligence Review, 2015, 43(2):155-177.
[21] Hinton G E. A practical guide to training restricted Boltzmann machines[J]. Momentum,2010,9(1):599-616.
[22] Mariooryad S, Busso C. Compensating for speaker or lexical variabilities in speech for emotion recognition[J]. Speech Communication,2014,57:1-12.
[23] Koolagudi S G, Rao K S. Emotion recognition from speech: a review[J]. International Journal of Speech Technology,2012,15(2):99-117.

(上接第 81 页)

[5] 罗 军,陈席林,李文生. 高效 Key-Value 持久化缓存系统的实现[J]. 计算机工程,2014,40(3):33-38.
[6] Anderson Q. Storm real-time processing cookbook[M]. Birmingham:Packt Publishing,2013.
[7] 张 涛,黄 强,毛磊雅,等. 一个基于 JSON 的对象序列化算法[J]. 计算机工程与应用,2007,43(15):98-100.
[8] 袁晓铭,林 安. 几种主流快照技术的分析比较[J]. 微处理机,2008,29(1):127-130.
[9] Surhone L M, Tennoe M T, Henssonow S F, et al. Object exchange model[M]. [s. l.]:Betascript Publishing,2010.
[10] 李少芳. DES 算法加密过程的探讨[J]. 计算机与现代化,2006(8):102-104.
[11] 何明星,林 昊. AES 算法原理及其实现[J]. 计算机应用研究,2002,19(12):61-63.
[12] 陈传波,祝中涛. RSA 算法应用及实现细节[J]. 计算机工程与科学,2006,28(9):13-14.
[13] 张裔智,赵 毅,汤小斌. MD5 算法研究[J]. 计算机科学,2008,35(7):295-297.
[14] 李 浩,罗云彬,王志军,等. 基于分布式流式计算系统的任务处理方法、系统及节点:CN, CN 103763378A[P]. 2014.