分类号_____ 学号 M201576091
学校代码 10487 密级_____

华中科技大学

硕士学位论文

基于内存的键值对缓存系统设计与实现

学位申请人：刘景超

学 科 专 业：软件工程

指 导 教 师：黄立群　副教授

答 辩 日 期：2017.12.29

万方数据

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree for the Master of Engineering

Design and Implementation of Key-Value Cache System Based on Memory

Candidate : Liu Jingchao

Major : Software Engineering

Supervisor : Assoc. Prof. Huang Liqun

Huazhong University of Science & Technology

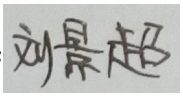
Wuhan 430074, P.R.China

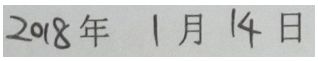
December, 2017

万方数据

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名： 

日期： 

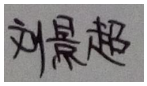
学位论文版权使用授权书

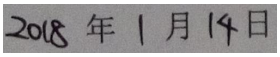
本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

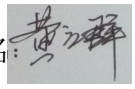
本论文属于 保密□，　　在_____年解密后适用本授权书。

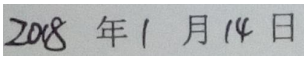
不保密  。

（请在以上方框内打“√”）

学位论文作者签名： 

日期： 

指导教师签名： 

日期： 

万方数据

摘 要

随着网民人数的快速增长和移动互联网的蓬勃发展，传统的数据库产品由于磁盘较慢的读写速度，当其面对高并发的请求和海量的格式复杂多变的数据时越来越力不从心，特别是秒杀、一元抢购等新的业务场景的出现更是加剧了数据库在短时间内的负担，如何快速的响应用户的请求并降低数据库的访问压力是服务器后端业务中不得不面对的问题。

针对服务器后端业务中瞬时高并发和海量数据查询时间较长等应用场景，设计并实现了基于内存存储的Armory键值对缓存系统，作为后端服务中应用层和数据层之间的缓冲层，对于用户查询的数据信息，如果缓存中存在则直接返回结果给用户，如果缓存中不存在才到数据库中查找，以此来提高数据查询的速度和效率。根据实际应用的需求可以将Armory缓存系统分为系统服务模块、数据持久化模块、主从复制模块三个部分，系统服务模块提供系统初始化、命令解析与执行、周期性任务和内存空间清理等基础的系统服务，数据持久化模块通过将缓存中的全部数据写入磁盘文件或者将对缓存数据造成改变的所有命令写入日志文件的方式来备份和恢复数据，主从复制模块通过将多台服务器节点组合在一起共同工作来提高系统的可用性和数据的安全性。

由于采用内存作为存储介质，其读写速度比传统的磁盘高出至少一个数量级，大大的提高了数据量非常庞大时的查询速度，而且数据持久化的功能也解决了内存作为一种易失性存储介质断电后数据会全部丢失的问题。采用Armory缓存系统解决了后端服务中数据库导致的性能瓶颈问题，明显缩短了数据查询请求的平均响应时间，提高了整个后端服务的性能。

关键词：缓存系统　　键值对　　内存存储　　数据持久化

万方数据

Abstract

With the swift growth of the number of Internet users and the flourish of the mobile Internet, due to the slower disk read and write speed, the traditional database products is increasingly inadequate when faced with the high concurrent requests and massive complex data formats, especially the appear of some new business scene such as seckill and panic buying exacerbate the burden of the database in a short time, how to response to the user's request faster and reduce the access pressure of the database server is a problem what developer have to face in the back-end business.

According to the high concurrency request instantaneous and long time of mass data query and other application scenarios in back-end business of server, Armory key-value caching system which based on memory is designed and implemented as a buffer layer between the application layer and the data layer in back-end service, for users to query data, if the cache exists directly returns results to user, or go to find in database if not exists in caching system, which can improve the speed and efficiency of query data. Armory caching system can be divided into three parts which include system service module, data persistence module and master-slave replication module according to the actual application requirements: system service module provide some base service such as system initialization, command parsing and execution, periodic tasks and memory space cleaning; data persistence module backup and restore data through write all data in the cache system into a disk file or record all the commands what can cause changes to the cache data into log file; master-slave replication module improve the safety and availability of the data system through multiple server nodes work together.

Due to the use of memory as storage medium, the speed of reading and writing at least one order of magnitude higher than the traditional disk, the query speed of the very large amount of data have a greatly improve, and data persistence function also solves

万方数据

the problem that data will be lost after powrefailure because memory is a non-volatile storage medium. The Armory cache system solves the performance bottleneck problem caused by the database in the back-end service, significantly shortens the average response time of the user's request and improves the performance of the whole back-end service.

Key words: Cache system Key-Value model Memory storage Data persistence

万方数据

目 录

摘 要..... I

Abstract..... II

1 绪论

1.1 研究背景.....(1)

1.2 研究目的和意义.....(2)

1.3 国内外发展现状.....(4)

1.4 主要研究内容.....(6)

2 关键技术介绍

2.1 I/O 多路复用..... (7)

2.2 简单动态字符串.....(8)

2.3 跳跃表.....(10)

2.4 本章小结.....(11)

3 Armory 缓存系统的分析与设计

3.1 Armory 缓存系统需求分析.....(12)

3.2 Armory 缓存系统总体设计.....(15)

3.3 系统服务模块设计.....(17)

3.4 数据持久化模块设计..... (21)

3.5 主从复制模块设计.....(27)

3.6 本章小结.....(33)

4 Armory 缓存系统的实现

4.1 开发环境.....(34)

4.2 系统服务模块实现.....(34)

万方数据

4.3 数据持久化模块实现..... (40)
4.4 主从复制模块实现.....(43)
4.5 系统测试.....(47)
4.6 本章小结.....(52)
5 总结与展望
5.1 全文总结.....(53)
5.2 展望.....(54)
致 谢.....(55)
参考文献.....(56)

万方数据

1 绪论

1.1 研究背景

随着移动互联网的蓬勃发展，人们通过手机便可以方便快捷的访问网络，社交、购物、支付、娱乐等各种类型手机应用的出现一方面丰富了人们的网络生活，另一方面也极大的促进了互联网技术的发展。截止 2017 年 6 月，中国的网民人数高达 7.51 亿人，数量如此庞大的网民用户带来的是规模呈指数增长的用户数据，如何高效准确的传输、存储和使用这些用户数据并保证用户良好的使用体验是很多大型互联网公司面临的一个很现实的问题。

因为其高效便捷、安全稳定、易于维护、易于拓展、冗余度低等特性，数据库系统是目前管理和存储数据最常用的技术手段，很多软件公司的大型信息系统都是构建在数据库系统之上的，可以毫不夸张的说，数据库管理系统已经成为了整个社会信息化的基石[1]。

基于表格、行、列、字段等关系模型的关系型数据库系统是目前最重要也是使用最广泛的数据库系统之一 [2-3]，它通过借助集合代数等数学方法和概念来处理数据[4]，人们可以使用结构化查询语言 SQL 方便的管理和维护关系型数据库系统中的数据[5]。当前主流的关系型数据库管理系统有微软公司出品的 Microsoft SQL Server、甲骨文公司出品的 Oracle 和 MySQL 等，其中 MySQL 由于其开源、免费、性能高、稳定性好等优点逐渐被很多大型的互联网公司采用，比如阿里巴巴集团旗下的淘宝和天猫网站、腾讯的微信、美团的手机应用、谷歌的搜索引擎等产品在后台服务中都有使用到 MySQL 来存储数据[6]。

关系型数据库产品的高性能保证了其在数据量和访问量不太大的环境下只需要单台节点就可以提供稳定的服务，然而随着大数据时代的来临，数据的规模呈指数级增长，数据的价值也变的越来越重要，在单点系统中一旦软件平台或者硬件平台发生故障，那么整个系统服务都将处于失效状态[7-8]。为了减少单点系统故障带来的影响以及缩短海量数据查询时的平均响应时间，各个数据库系统厂家采用了主从

万方数据

复制、中间件、数据分片等技术手段来提高产品的性能、可靠性和可扩展性[9-11]。缓存技术作为中间件的一种，其为关系型数据库的水平扩展提供了技术支持，这在大数据时代显得尤为必要，不同的缓存产品在负载均衡、读写分离、服务容错、稳定性、数据安全、是否支持数据库事务等方面的表现和性能也各不相同[12]。

虽然近年来关系型数据库在性能和稳定性方面都有了巨大的提升，但是随着云计算技术的发展和搜索引擎、社交网络、共享经济等不同类型应用的出现，传统的关系型数据库在读写速度、容量拓展、维护成本等方面都面临着巨大的挑战，NoSQL数据库凭借其高性能、高扩展性和维护成本低的优点在许多企业中得到了广泛的使用[13]。与传统的关系型数据库系统基于关系模型不同，在 NoSQL 数据库领域，Key-Value 模型才是其主要的数据存储模型[14]。Key-Value 模型是指数据库中的数据都是通过一个唯一的键来存储，数据的查询和修改也是通过键来完成，由于键值对模型结构简单，所以 NoSQL 数据库的读写速度要远远高于关系型数据库，而且海量数据的存储和对高并发的支持也更简单方便[15]。

为了高效便捷的存储和管理数据，人们发明了基于关系模型的关系数据库；随着数据规模的日益增长和数据格式的日益复杂，人们不得不面对传统的关系型数据库在读写速度、扩展性等方面的不足，为解决此问题人们又发明了基于 Key-Value 模型的 NoSQL 数据库[16]。Key-Value 模型让 NoSQL 数据库可以简单高效的存储互联网环境中产生的越来越多的非结构化数据，而且其读写速度快、易于扩展、能支持更大容量数据的存储以及维护成本低等优点极大的促进了互联网应用的发展[17]。然而不论是传统的关系型数据库还是 NoSQL 数据库，其存储数据的物理介质都还是以磁盘为主，当其面对越来越多的类似秒杀和商品抢购等访问并发量和数据量瞬间爆发的场景，磁盘存储器的读写速度依然无法满足人们对响应时间的要求，于是基于内存存储数据的内存数据库和数据缓存系统成为了人们新的研究方向[18]。

1.2 研究目的和意义

传统的后端服务架构中，对于每一个用户的数据访问请求，后端服务都会进行至少一次的数据库查询，并将查询结果直接返回给用户，伴随着互联网的发展和网

万方数据

络用户的增加，数据库系统的访问压力也越来越大，而且数据量的增大导致数据库查询的响应时间也迅速增大。近年来移动互联网在国内迅猛发展，中国已经成为继美国之后的互联网应用和规模第二大的国家，淘宝、天猫、微信、支付宝、美团、滴滴等无数手机应用的出现一方面极大的方便和丰富了人们的生活，另一方面大量不同类型的应用也导致互联网的数据规模日益膨胀，数据的格式也越来越复杂多样，这更是加重了数据库系统的负担。

2017 年双十一期间阿里巴巴的全天总成交额创下历史新高，达到了 1682 亿元，支付宝作为阿里巴巴集团蚂蚁金服旗下的支付产品在这次双十一盛宴中经受了前所未有的挑战，当天全球消费者一共通过支付宝支付了 14.8 亿笔，支付的最高峰值是 25.6 万笔每秒，数据库处理峰值高达 4200 万笔每秒。不仅仅是购物领域，在目前最热门的直播领域也是如此，最热门的直播间最多能有几千万人同时观看，用户每秒钟发送的弹幕条数能高达几十万条。像这种总体数据量巨大、短时间内访问请求呈爆炸性增长的应用场景目前已经成为了互联网应用领域的常态，这给本来就不堪重负的数据库系统带来了前所未有的严峻挑战。

统计资料显示，在所有用户的数据访问请求中，大部分请求返回的结果都是相同的，比如对于一个外卖应用来说，用户更多的是关心某一个商家他的外卖种类、商品价格、配送时间等信息，而很少有用户关注商家的负责人姓名和联系方式等信息。因此我们可以在后台应用层服务和数据库系统之间添加一层缓存系统，对于所有的用户数据访问请求都先到缓存系统中查询结果，如果查询到结果就直接返回给用户，如果没有查询到结果才到数据库系统中查询，然后将从数据库系统中查询到的结果返回给用户并保存一份结果到缓存系统中，于是下一个用户同样请求该数据结果时就可以直接从缓存系统中返回。

在服务层和数据层之间添加缓存系统，并结合数据库集群和科学合理的调度算法一起使用，可以极大的提高整个后端系统的性能，避免无意义的重复查询操作，降低了数据库系统的负载压力。为了进一步提高缓存系统的性能、降低后端服务的响应时间，人们采用读写速度比传统磁盘存储器高出一个数量级的内存来构建缓存系统，如 Memcached、Couchbase 等基于内存存储的缓存系统如雨后春笋般的出现

万方数据

在人们的视野中。由于内存是易失性存储设备，一旦断电则存储在内存中的所有数据都会丢失，因此基于内存的缓存系统还需要有数据备份和数据持久化的功能，为了增加多个节点数据的安全性，节点之间还应有数据同步功能，而这些正是本文的研究目标和方向。

1.3 国内外发展现状

随着用户数量的逐渐增长、数据规模的迅速膨胀以及短时间内瞬间聚集大量并发访问请求场景的常态化，传统的单点数据库系统以及数据库集群方案已经成为了后端服务中最薄弱的一个环节，也越来越无法应对当前复杂的网络请求环境[19-20]。为了解决这一问题，各大数据库系统软件公司和大型互联网公司相继都加入了缓存系统的研究行列，并取得了一定的研究成果。

1.3.1 国外研究现状

作为传统的关系型数据库，MySQL 采用的方案是在物理磁盘或内存中开辟一块空间作为缓存，当客户端向数据库发起访问请求时，数据库先将请求的命令解析为 SQL 语句，并将该 SQL 语句作为 Key 在缓存中查找结果，如果命中缓存则直接将结果返回给客户端，如果未命中缓存，则启动脚本解释器执行该 SQL 语句，并将得到的结果返回给客户端，同时将该结果添加入缓存[21-23]。该方案在一定程度上解决了数据库系统负载过高和大量重复查询的问题，但是缓存和数据库系统必须位于同一个节点，这大大的限制了该缓存系统的可扩展性，同时该缓存系统不能存储非结构化数据也限制了其实际应用场景[24-25]。

Memcached 是一个自由开源、高性能的分布式 Key-Value 对象缓存系统，基于内存存储和基于 libevent 事件处理的特性使其单台节点最高可以承受每秒几十万的并发访问量，大大高于传统的数据库系统[26-27]。Memcached 使用 32 位元的循环冗余检验来计算键值，根据键值将缓存数据分散在不同的节点上，如果缓存的数据量达到了设定的内存容量上限则使用最近最少使用(LRU)算法进行替换[28]。Memcached 通过缓存一部分常用数据减少了数据库的访问次数，提高了 Web 应用的响应速度和

万方数据

可扩展性，不过其也有一些缺点需要解决，比如只支持缓存简单的字符串对象、在分布式架构下不同节点之间不能直接相互通信、不支持数据持久化等[29-30]。

和 Memcached 类似，Redis 也是一个开源的基于内存存储的 Key-Value 存储系统，不过其比 Memcached 功能更加强大、支持的操作类型更多、同时也更加复杂，因此准确来说 Redis 应该算是一个内存数据库产品。Redis 支持存储的类型包括 string、list、dict、set、zset 等，这些数据类型还支持 push、pop、delete、取交集并集差集等原子性操作。为了防止断电和硬件设备故障等因素导致内存中存储的数据丢失，Redis 支持定时将内存中的数据备份到硬盘，或者在日志文件中记录下所有数据的操作语句，以此来备份和恢复数据[31-32]。Redis 的主从模式支持主节点通过 Replication 功能将数据复制到任意数量的从节点，该从节点还可以是关联其他从节点的主节点，即多级复制模式[33]。Redis 只能支持单线程模式，无法发挥服务器多核的优势，实际性能受限于机器的配置和 CPU 性能，而且由于支持的功能更多实现更复杂，因此单个 Redis 节点的每秒最大并发量要比 Memcached 低。

1.3.2 国内研究现状

国内在数据库系统设计和研发领域起步较晚，数据库基础理论和应用方向研究相对不如国外先进，然而得益于国内数量庞大的网民用户基础和国家大力发展“互联网+”、大数据、人工智能产业的政策支持，在需求的刺激下，国内的各大互联网公司投入了巨额的资金和大量的人员进行数据库和缓存系统方面的研究，也获得了可观的技术产出。

Tair 是阿里巴巴公司开源的一个分布式 Key-Value 数据存储解决方案，它既可以基于磁盘存储用作数据库，也可以基于内存存储用作缓存[34]。Tair 集群主要包括客户端、配置服务节点和不定量的数据服务节点，数据服务节点用来存储实际的应用数据，配置服务节点通过心跳检测的方式确定集群中数据服务节点的分布，对于客户端的数据访问请求，配置节点会根据集群中数据节点的分布信息进行分配和重定向[35-36]。Tair 还支持 Version、原子计数器和 Item 特性：Version 特性是指 Tair 中存储的每个数据都有一个唯一的版本号，每次对数据进行更新后版本号都会增加，

万方数据

此特性可以用来解决传统数据库中数据脏读和幻读的问题；Tair 支持原子计数操作，可以用来作为分布式计数器；Item 特性让 Tair 可以将一条记录中存储的值作为一个整体并取其中的一部分[37]。实践证明 Tair 已经取得了很大的成绩，最近几年的双十一活动中总交易额和每秒最大交易笔数履创新高，服务器也一次次的经受住了考验。

1.4 主要研究内容

本文的主要研究内容是设计与实现一个基于内存存储的键值对缓存系统，该系统即有基于内存存储数据的高速读写优势，又引入了 Key-Value 模型，该模型具有底层实现简单、读写效率高、方便拓展、占用资源少等优势，可以进一步提高缓存系统的性能和效率。

本文一共分为五个章节，其组织结构如下：

第一章（本章）为绪论，介绍了该课题的研究背景、研究目的与意义、国内外关于后台服务中数据缓存系统的研究现状，并给出了本文的整体组织结构。

第二章主要介绍了 IO 多路复用、简单动态字符串、跳跃表等相关技术。

第三章对键值对缓存系统 Armory 进行了需求分析和整体设计，给出了 Armory 缓存系统的架构框架，并对各个模块进行了详细设计。

第四章主要阐述了 Armory 缓存系统的系统服务模块、数据持久化模块、主从复制模块的具体实现并对系统进行了简单的功能测试。

第五章对全文做了总结并对下一步工作进行了分析与展望。

万方数据

2 关键技术介绍

本章将简单介绍一下在系统实现过程中需要用到的一些重要的数据结构和关键技术，包括极大的提高了网络请求处理效率的 I/O 多路复用技术和安全高效的简单动态字符串、作为集合类型底层实现的跳跃表两种数据结构，以此作为后续研究的理论和实践基础。

2.1 I/O 多路复用

传统的 I/O 模型包括阻塞 I/O 模型和非阻塞 I/O 模型，如图 2-1 所示，对于阻塞 I/O 模型，如果请求的数据尚未到达，则系统会阻塞在那里，一直等到数据到达后内核将数据拷贝到缓存中然后返回；对于非阻塞 I/O 模型，如果请求的数据尚未到达则会立即返回一个标识信息，用户进程会检查该标识信息进而判断数据是否就绪，如果数据未就绪系统会继续发送数据请求，直到数据到达后内核拷贝数据并返回成功标识[38-39]。

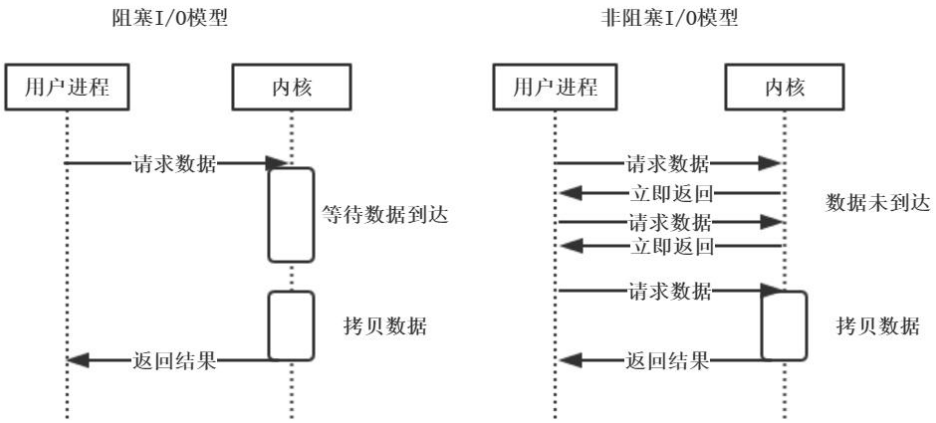


图 2-1 传统 I/O 模型

在传统的 I/O 模型中，如果要同时监听多个连接请求就需要创建对应个数的进程，每个进程负责监听一个连接请求，当请求的并发量非常大时，这样做的缺点非常明显：

万方数据

（1）由于每一个进程都需要系统为其分配资源，假如同时有多个连接请求，则需要为这些进程分配大量内存，这将给很多配置不高的服务器带来沉重的负担。

（2）进程在切换时需要保存堆栈空间、上下文等信息，如此大量的进程频繁切换不仅会占用过多的系统资源，而且由于操作系统是采用时间片轮转的方式调度进程，当进程数越多时平均分配给每个进程的时间片就越少，这样会导致系统的执行效率极其低下[40]。

为了解决传统 I/O 模型在高并发的情况下效率低下的问题，操作系统引入了 I/O 多路复用模型（IO Multiplexing）[41]。如图 2-2 所示，I/O 多路复用模型中有一个类似于 select、poll、epoll 的系统函数负责维护和轮询所有的文件句柄，当有文件句柄就绪后才会通知用户进程，这样维护所有的连接请求只需要一个进程而不是多个，当请求数量较多时可以明显的降低所需要的内存资源，同时系统调度和执行的效率也更高[42]。

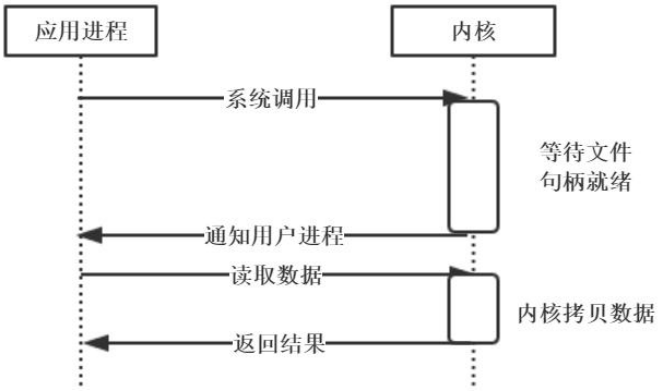


图 2-2　I/O 多路复用模型

2.2　简单动态字符串

在 C 语言中，传统的字符串采用字符数组存储，字符串结尾用空白字符“\0”标识，这种简单的字符串结构并不能满足 Armory 缓存系统在效率、安全性和便捷性等方面的需求，所以 Armory 缓存系统采用了简单动态字符串（Simple Dynamic String）类型来存储字符串。

如图 2-3 所示，简单动态字符串类型结构中一共有三个字段，free 字段用来记

万方数据

录剩余的可用内存字节数，length 字段用来记录已经使用的内存字节数，buffer 是一个指针，其指向存储有字符串内容的内存地址。

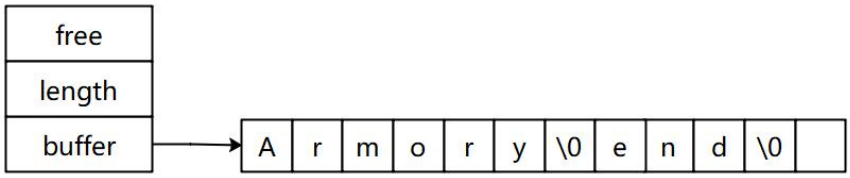


图 2-3　简单动态字符串

和传统的 C 语言风格字符串相比，简单动态字符串具有如下四个优势：

（1）获取字符串长度的时间复杂度为常数

传统的 C 语言风格字符串要想获取长度只能遍历字符串数组，直到遇见空白符标识的结尾为止，该算法的时间复杂度为 O(N)，当经常需要获取字符串长度时，C 语言风格字符串的效率显然不高。对于简单动态字符串，要想获取其长度，只需要取出结构体中 length 字段的值即可，该算法时间复杂度为 O(1)，这保证了在 Armory 缓存系统中获取字符串长度不会成为性能瓶颈。

（2）避免缓冲区溢出

C 语言风格的字符串由于不记录长度信息，除了获取字符串长度的时间复杂度较高外，还很有可能导致缓冲区溢出。在 C 语言中，字符串操作函数中需要用到字符串对象的函数参数都是指针类型，由于指针类型只给出了内存地址信息而没有给出长度信息，所以在复制字符串、字符串拼接的时候很有可能因为目标字符串长度不够而导致内存越界行为[43]。

在简单动态字符串中，当需要对字符串的内容做复制、拼接等修改操作时，会首先检查目标字符串的剩余空间是否足够容纳新生成的字符串，如果空间不足则分配内存将目标字符串的容量提升到满足需求，然后才执行实际的修改操作，这样就避免了缓冲区的溢出问题。

（3）减少内存分配次数

对于 C 语言风格的字符串，当需要增长或者缩短字符串的长度时，都需要重新

万方数据

分配或者回收字符串数据占用的内存空间，比如当执行字符串复制、字符串拼接等扩展字符串的操作时需要为目标字符串分配更大的存储空间，当执行字符串截断等缩短字符串的操作时需要把字符串数组返还的存储空间回收。内存的分配和回收涉及到系统调用，频繁的系统调用会占用大量的系统资源。

简单动态字符串采用了内存空间延后释放和预先分配足够空间的方式来避免频繁的进行内存资源分配和释放的系统调用。内存延后释放适用于当需要缩小简单动态字符串占用的内存空间时，此时并不释放字符串占用的内存，而是只修改字符串的 length 属性，并将多出来的内存空间大小记录进 free 属性中，当需要扩充该字符串时就可以不执行内存分配操作而直接存储。预先分配内存适用于需要扩大字符串占用的内存空间时，此时不仅为字符串分配所需的空间大小，还为其额外的分配内存空间并将大小信息记录在 free 字段中，当下次字符串又需要扩展空间时就可以不再执行内存分配。

（4）二进制安全

由于 C 语言字符串是利用空白字符“\0”来判断字符串的结尾，假如字符串的内容本身就含有这个空白字符，则 C 语言字符串只能正确获取空白字符前的字符串，而无法正确的获取完整的字符串，这就导致了二进制数据的不安全，简单动态字符串利用 length 字段来记录字符串的长度就避免了这一问题[44]。

2.3 跳跃表

跳跃表是一种类似于链表的有序数据结构，它通过在每个节点中存储多个指向其它节点的指针和跨度信息来快速的访问表中的节点[45]。跳跃表的平均时间复杂度为 O(logN)，最坏时间复杂度为 O(N)，在大多数情况下效率可以和平衡树相媲美，而且实现也更为简单，因此在 Armory 缓存系统中使用跳跃表来作为有序集合的底层存储数据结构。

如图 2-4 所示，位于图片最左边的是跳跃表的表头节点，其包含了四个字段，header 字段是一个指向表中表头节点的指针，tail 字段是一个指向表中表尾节点的指针，level 字段存储了当前表中最大的节点层数，length 字段存储了当前表中的节

万方数据

点个数。跳跃表中的节点中还存有跨度信息和后退指针，通过这些信息可以快速的遍历和查找整个表中的数据。

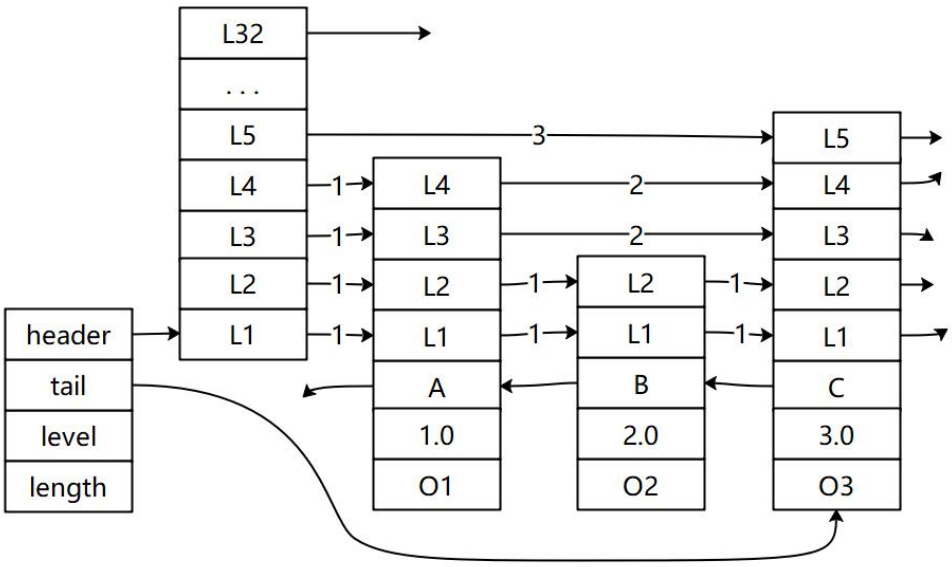


图 2-4　跳跃表

2.4　本章小结

本章主要介绍了 I/O 阻塞模型、I/O 非阻塞模型、I/O 多路复用三种常用的 I/O 模型具体的请求和应答流程，详细分析了简单动态字符串相比传统的 C 语言字符串的在二进制安全和系统性能方面的优势，并简单介绍了实现相对简单但时间复杂度较低的跳跃表数据结构。

万方数据

3 Armory 缓存系统的分析与设计

上一章中简单的介绍了 I/O 多路复用模型和简单动态字符串、跳跃表两种数据结构，本章将从基于内存的 Armory 缓存系统的需求分析入手，首先讨论设计该缓存系统的意义和需要实现的目标，然后从总体上对该缓存系统进行设计并划分重要模块，最后对重要模块进行详细设计，给出整个缓存系统的整体框架和结构。

3.1 Armory 缓存系统需求分析

几年前智能手机的出现是互联网发展的一个里程碑，它改变了整个互联网行业发展的进程，移动互联网的浪潮迅速席卷全球，各种手机应用如雨后春笋般层出不穷，例如国内的微信、淘宝、支付宝、美团以及国外的 Google、Fackbook 等，这些应用带来了海量的用户数据、极大的并发请求量和更加复杂的数据格式，这些都给传统的服务器后端数据服务带来了很大的挑战。

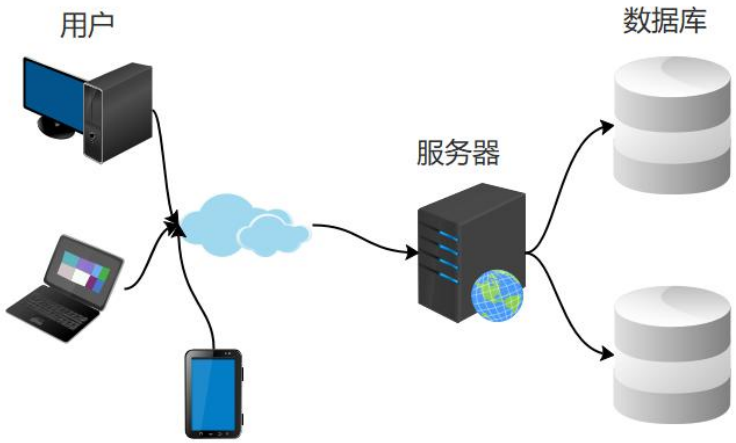


图 3-1　传统的后台服务架构

如图 3-1 所示，在传统的后端服务架构中，用户通过互联网连接服务器，后台的应用服务器直接连接数据库系统，对于每一个用户请求服务器都会直接去查询数据库中的数据。这种架构的弊端在于，当数据量非常庞大时查询操作也非常耗时，

万方数据

进而导致对于用户请求的响应时间变的很长，而且当同时有非常多的用户访问时会给数据库系统带来沉重的负担。

经过分析用户的请求数据发现，在 Mobogenie 应用市场软件中，用户最常获取的数据信息是当前的应用分类、热门应用、应用大小等，只有很少情况下用户需要获取应用开发者的联系方式等信息，于是我们可以设计一个缓存系统来缓存常用的数据，以此来提高整个后台服务的效率。

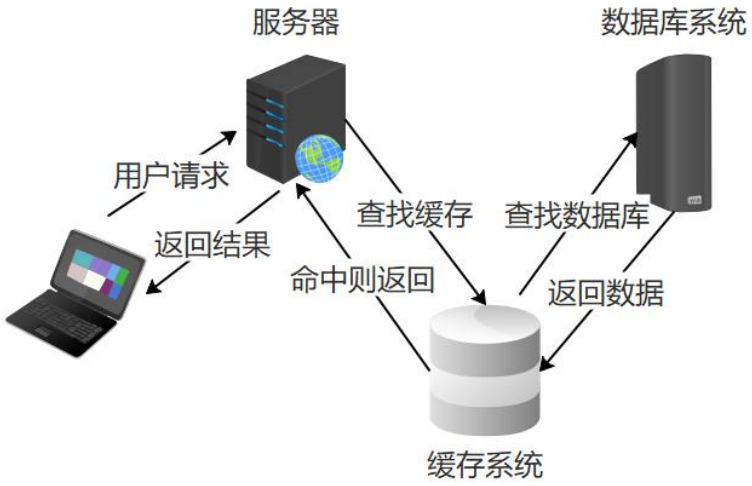


图 3-2　带有缓存系统的后台服务架构

如图 3-2 所示，在带有缓存系统的 Mobogenie 后台服务架构中，对于用户的数据查询请求应用服务器首先去查询缓存系统，如果缓存系统里有需要的数据则直接返回结果，如果缓存系统中没有则继续查询数据库系统，找到需要的数据后将结果返回给用户的同时保存一份到缓存系统中，这样下次有别的用户再次查询这个数据时就可以从缓存系统中直接返回。

3.1.1　功能性需求分析

为了减少重复的数据查询请求、提高数据查询效率并能同时支持庞大的用户并发请求，经过分析总结，设计的 Armory 缓存系统应该满足以下几个功能：

（1）使用内存作为存储介质

传统的数据库系统存储介质是普通的磁盘存储器，虽然近年来出现了随机读写

万方数据

速度更快的固态硬盘，但是总体来说磁盘存储器的读写速度还是比较缓慢，特别是Mobogenie项目中数据量非常庞大，磁盘的读写速度会明显的影响整个数据库系统的查询和写入速度，因此，为了保证较快的响应速度，Armory缓存系统采用读写速度更快的内存作为存储数据的介质。

（2）使用键值对模型存储数据

传统的关系型数据库只能存储结构化数据，而在Mobogenie项目中既用到了结构化数据，也用到了JSON序列化的非结构化数据，为了让Armory缓存系统尽可能的保持通用，其采用键值对模型来存储数据，这样既可以存储结构化数据也可以存储非结构化数据。

（3）采用I/O多路复用提高对高并发的支持

根据统计信息显示，Mobogenie应用的平均日活跃用户达到了一千多万人，用户请求最高能达到每秒二十多万次，为了提高Armory缓存系统应对用户大量并发请求的能力并降低对操作系统资源的消耗，其应该采用I/O多路复用模型。

（4）具有数据持久化功能

由于采用易失性存储介质的内存来存储数据，一旦服务器意外断电或者由于软硬件故障而导致进程崩溃，则内存中的数据会全部丢失，为了防止这种意外情况导致数据不可恢复性的丢失，Armory缓存系统还应该有数据持久化和数据备份的功能，一旦发生意外还可以将数据恢复。

（5）具有一定的错误检测和处理能力

Mobogenie的用户涵盖了多种用户群体和用户使用习惯，Armory缓存系统还应该保持健壮性，当用户进行了错误的操作或者发送了错误的请求指令，系统应该能准确的识别并做出合理的应答；如果因为外部原因发生了故障，系统还应该能够正确的恢复到正常状态并保证数据的安全。

3.1.2 非功能性需求分析

Armory缓存系统除了应该满足指定的功能性需求之外，还应该满足以下的非功能性需求：

万方数据

（1）易用性

Armory 缓存系统应该结构清晰、简单易用，便于运维人员进行维护，并提供完善的提示和帮助系统以此来进一步降低用户的使用门槛和难度。

（2）可重用性

Armory 缓存系统应该采用模块化设计，各模块之间的耦合性要小，方便在开发过程中重复利用底层模块，也能方便用户在现有代码的基础上根据新的业务需求进行功能的定制化开发。

3.2 Armory 缓存系统总体设计

根据上一节的需求分析结果，为了使设计的缓存系统能满足高性能、通用性强、数据安全等要求，接下来将从总体架构的角度设计系统的各个模块和功能。

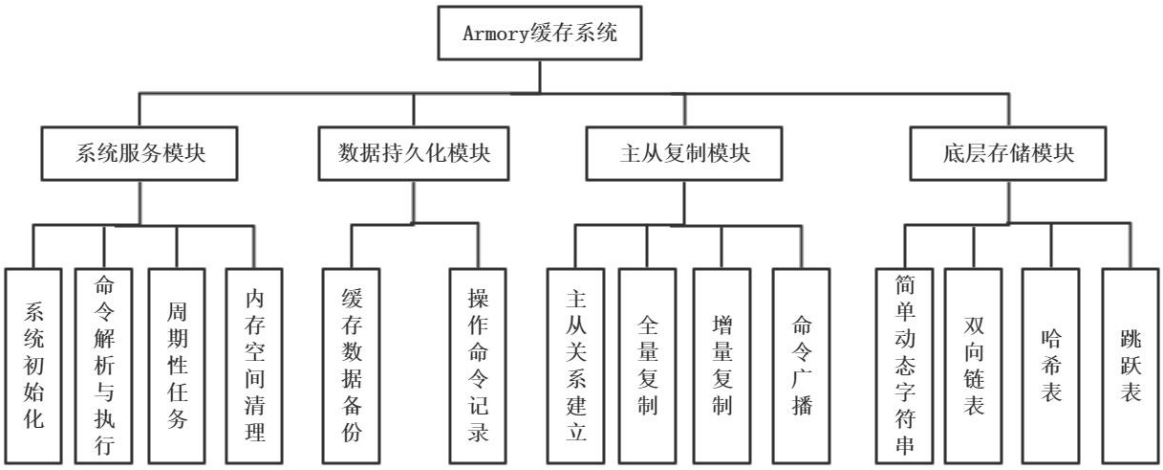


图 3-3　Armory 缓存系统总体结构图

如图 3-3 所示，Armory 缓存系统的功能可以按照图中所示的模块进行划分，其主要包括四大功能模块：

（1）系统服务模块

系统服务模块主要为 Armory 缓存系统提供关键的系统服务，比如启动服务器时初始化环境设置并监听网络端口，当接收到客户端发送来的命令请求后解析该命令，如果命令非法则提示客户端并报告错误，如果命令合法则执行该命令并将执行

万方数据

结果返回给客户端。该模块还包括系统会定期执行的检测文件事件和时间事件的周期性函数，且当系统使用的内存容量超过了设置的上限时负责清理内存使用空间。

（2）数据持久化模块

由于 Armory 缓存系统采用内存作为存储数据的介质，一旦发生意外断电或者系统错误导致进程崩溃的情况，内存中的所有数据都会丢失，这将给缓存系统的数据安全带来极大的隐患。数据持久化模块正是为了保障数据的安全性而设计，其主要包含两种数据持久化方式：缓存数据备份（Armory Data Backup）和操作命令记录（Operator Command Record）。

缓存数据备份是指 Armory 缓存系统会自动或者手动的将内存中的所有数据备份到二进制文件中，一旦系统发生故障就可以从这些二进制文件中恢复数据。操作命令记录是指 Armory 缓存系统会将所有的数据操作命令都记录在日志中，当系统发生故障后重新解释执行日志中的命令就可以恢复数据。两种方式各有优点和缺点，可以单独使用也可以配合一起使用。

（3）主从复制模块

为了进一步提高 Armory 缓存系统的数据安全性和能同时处理的请求并发量，该缓存系统支持多个节点组成集群同时工作，集群中包括一个主节点和至少一个从节点，主节点和从节点之间可以通过复制的方式来保持数据的同步性。主从复制模块主要包括建立主节点和从节点之间的关系、第一次建立主从关系时全量复制数据、主从关系建立后增量复制数据以及主节点向从节点广播造成缓存系统中数据被修改的命令等功能。

（4）底层存储模块

为了提高系统的通用性，Armory 缓存系统采用键值对模型来存储数据，除了可以存储字符串类型数据之外，还可以存储列表、集合、有序集合和字典类型的数据。底层存储模块主要包含简单动态字符换、双向链表、跳跃表和哈希表这四种数据结构，其中简单动态字符串用于存储字符串类型数据，双向链表用于存储列表类型数据，跳跃表用于存储集合类型数据，字典作为整个缓存系统的基础用于存储缓存数据、有序集合类型数据以及系统的一些参数。

万方数据

3.3 系统服务模块设计

系统服务模块主要包括启动服务器、维护服务器的一些参数信息、建立服务器与客户端之间的连接、接收客户端的命令请求执行后返回结果、周期性的响应文件事件和时间事件、根据需要清理已使用的内存空间等功能，通过对内存资源、文件资源、网络资源的合理管理和维护来保证服务器的正常运行。

其主要包含如图 3-4 所示的系统初始化、命令解析与执行、周期性任务和内存空间清理四个子模块：

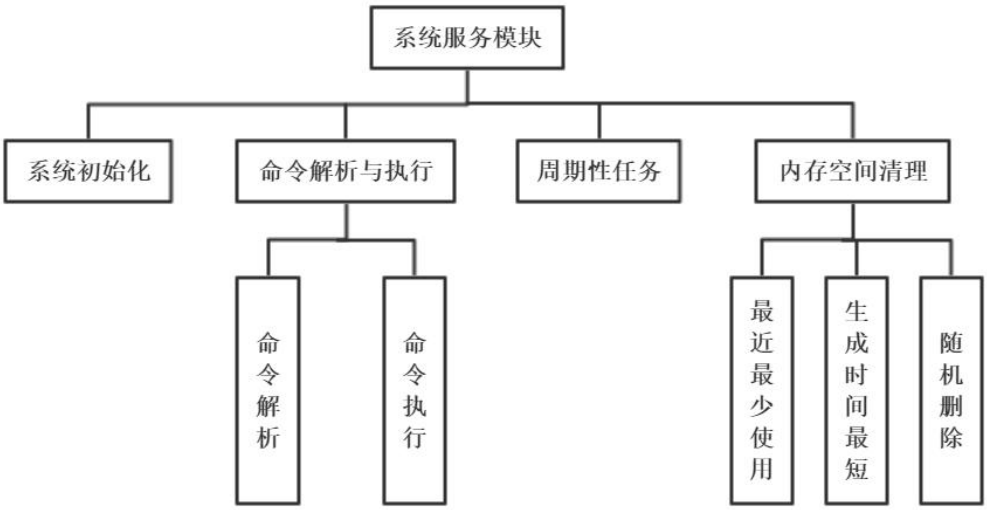


图 3-4　系统服务模块总体结构图

3.3.1 系统初始化

系统初始化子模块主要负责在启动服务器时初始化服务器状态，这是整个 Armory 缓存系统的执行入口，其执行流程如图 3-5 所示。

（1）首先设置线程安全函数、内存溢出处理函数、随机数发生器、哈希种子等初始状态信息。

（2）分析命令行读入的参数，检查参数是否合理正确并根据其指定的配置文件来设置服务器状态。

（3）使用默认值初始化程序在执行过程中需要用到的一些全局变量，

（4）检查指定的目录下是否有数据持久化信息，如果有操作命令记录的日志

万方数据

文件则用该日志文件恢复缓存数据，如果没有操作记录日志文件则检查是否有数据备份文件，如果有则用其恢复缓存数据。

（5）开启事件循环监听网络端口等待客户端的连接请求并处理接收到的客户端消息，直到事件循环被关闭为止。

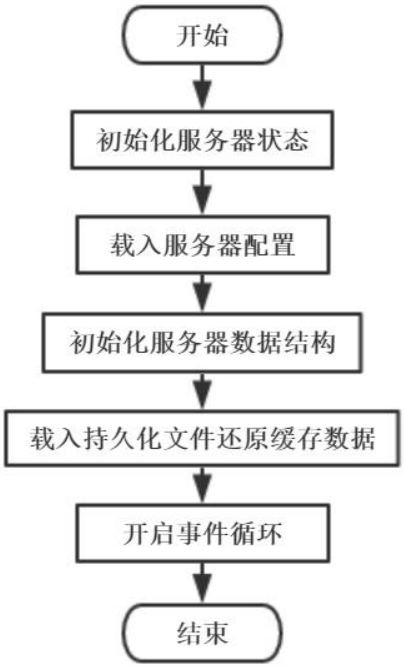


图 3-5　系统初始化流程图

3.3.2　命令解析与执行

在 Armory 缓存系统中，一条命令的执行过程包括以下四个步骤：客户端将命令发送给服务端、服务端接收并解析命令、服务端执行命令、服务端将命令执行结果返回给客户端，其执行过程如图 3-6 所示。

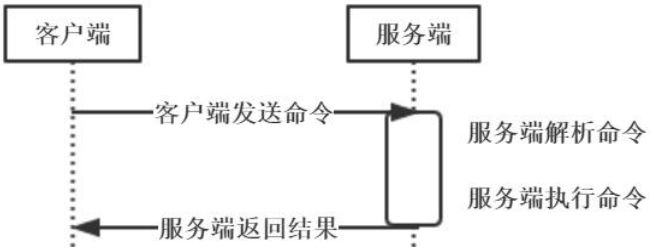


图 3-6　命令执行时序图

万方数据

服务端接收到来自客户端的命令后，首先会在命令列表中查找是否有该命令，如果有该命令则继续检查命令参数的个数是否符合要求，然后检查服务器是否有最大内存限制并按需要释放一部分内存，最后调用和命令相关联的函数并把执行结果返回给客户端，其执行流程如图 3-7 所示。

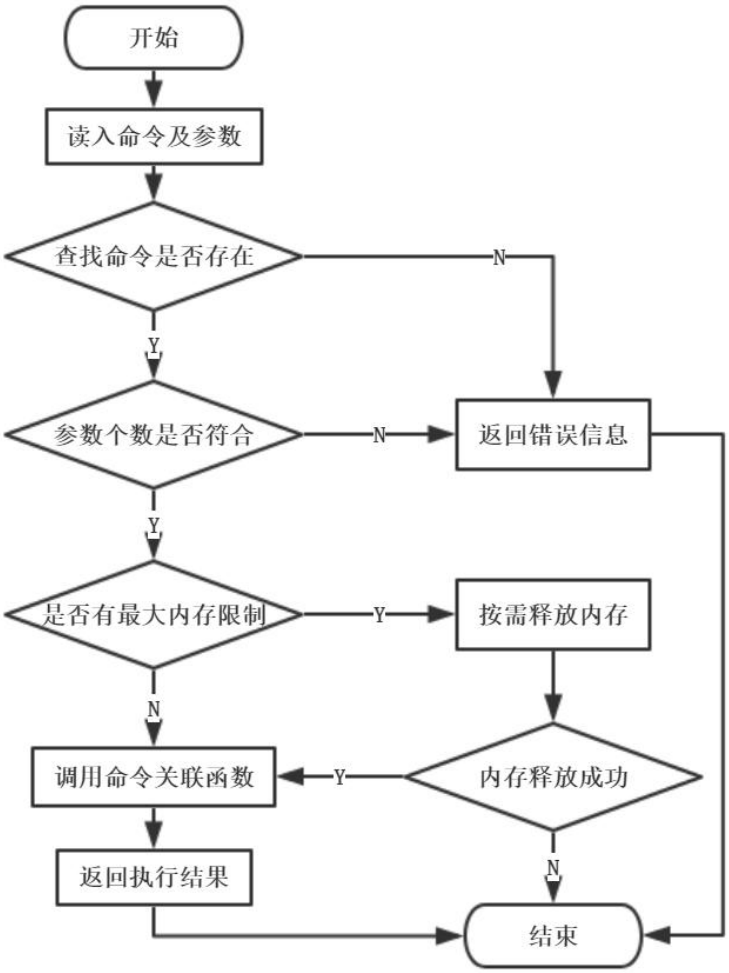


图 3-7　命令解析与执行流程图

3.3.3　周期性任务

Armory 缓存系统中有一个函数 ServerCrontab 主要用来周期性的执行一些文件事件和消息事件。如图 3-8 所示，首先设置服务器当前时间，更新命令执行的次数、从网络中接受到的字节数、向网络中发送的字节数、最大内存使用量等统计信息，并计算常驻内存的空间大小；然后检测关闭服务器标志位是否为 1，如果不为 1 就

万方数据

执行客户端周期性任务和服务器端周期端任务；再检测缓存数据备份标志位是否为1，如果为1就像当前缓存中的有效数据写入二进制备份文件；再检测操作命令记录标志位是否为1，如果为1就将缓冲区中的操作命令写入日志文件；最后清理已经失效的客户端连接并返回。

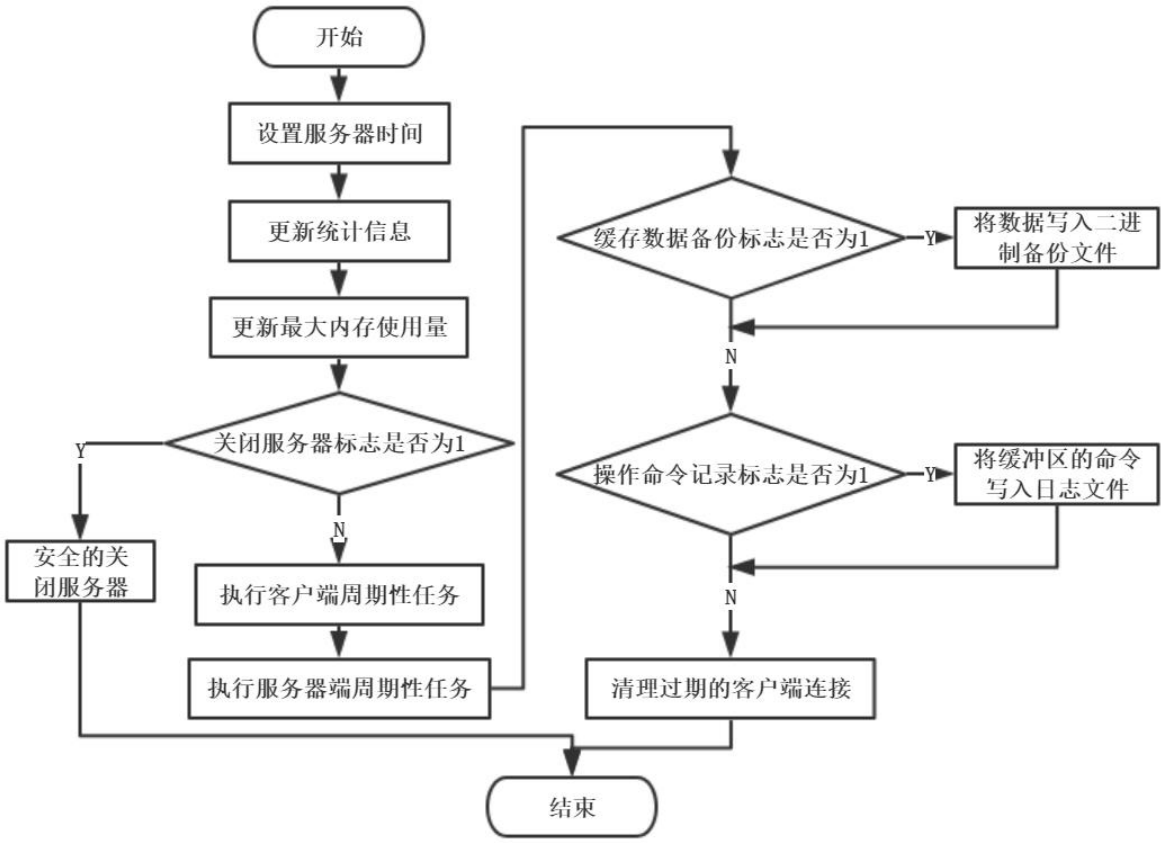


图 3-8　周期性任务流程图

3.3.4　内存空间清理

Armory 服务器在启动时会设置一个能够使用的内存容量上限值，如果超过了这个上限，则可以通过清理缓存中的键值对来释放空间。如图 3-9 所示，首先计算当前已用内存空间大小，如果有从节点或者开启了操作命令记录模式则减去这些缓冲区占用的内存大小得到实际的缓存数据占用的内存空间，然后根据指定的回收策略选择要释放的键并释放其所占用的内存空间，如果当前节点有从节点则还需要将删除命令发送给从节点，以此来保证主从节点之间数据的一致性。

万方数据

根据删除目标对象和选择键的算法不同，数据的回收策略一共包括六种模式：删除最近最少使用的失效键、删除生存时间最短的失效键、随机删除一个失效键、删除最近最少使用的有效键、随机删除一个有效键、不删除任何键。

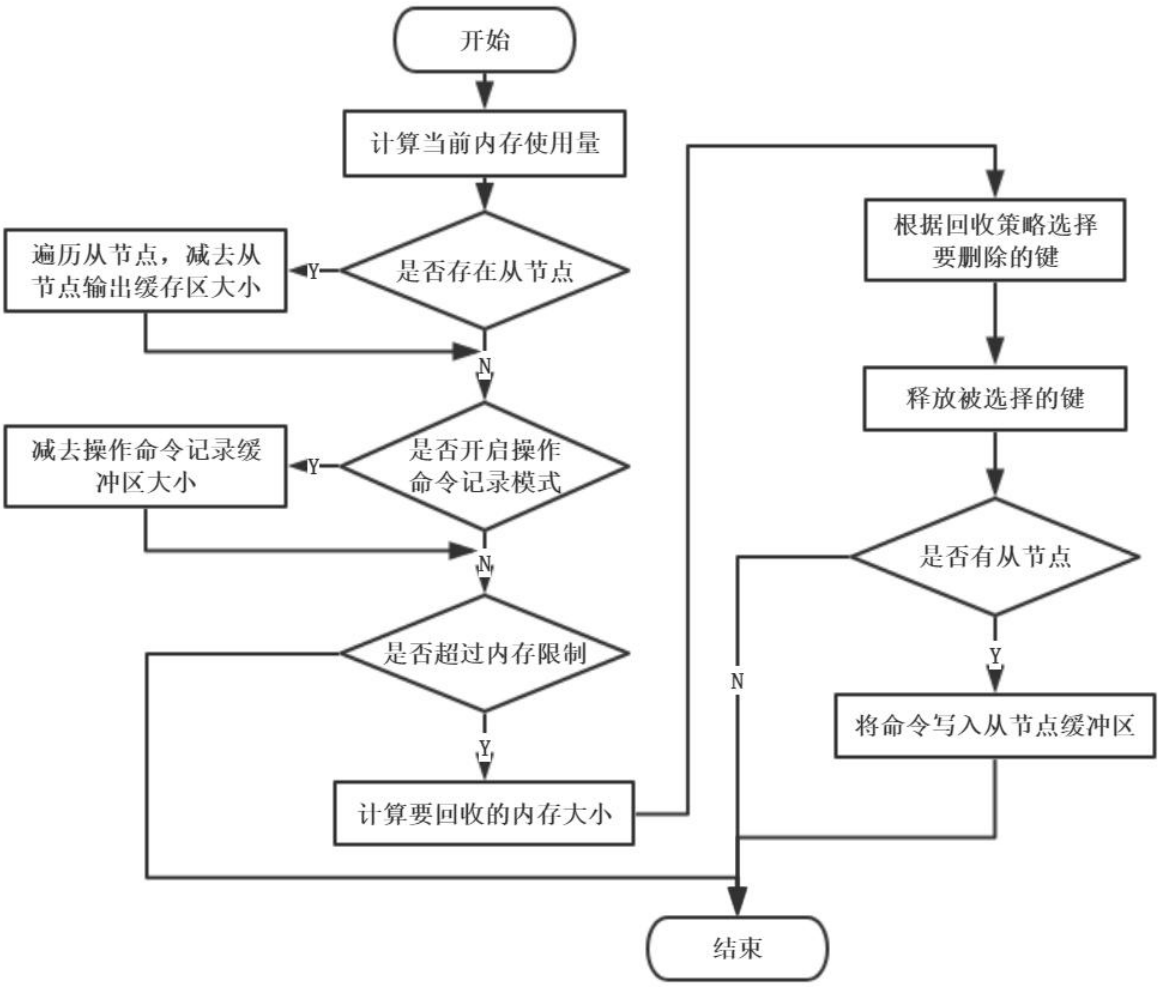


图 3-9　内存空间清理流程图

3.4　数据持久化模块设计

Armory 缓存系统采用内存存储数据，由于内存是易失性存储介质，一旦发生意外导致进程结束，则内存中的所有数据都会丢失，这给缓存系统的数据安全带来了极大的隐患。为了解决这个问题，Armory 缓存系统提供了数据持久化功能，如图 3-10 所示，其主要包含缓存数据备份（ADB，Armory Data Backup）和操作命令

万方数据

记录（Operator Command Record）两种方式，以此保证即使缓存系统发生了意外故障后还可以从备份文件中恢复数据。

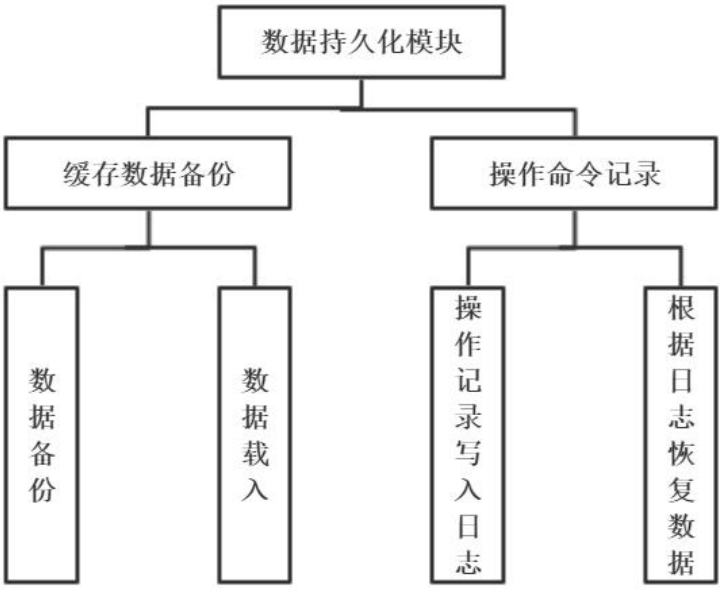


图 3-10　数据持久化模块总结结构图

3.4.1　缓存数据备份

缓存数据备份（ADB，Armory Data Backup）功能是将 Armory 缓存系统中当前时间节点的数据以二进制文件的形式写入磁盘中，避免因为意外原因导致数据丢失，其可以根据系统启动时指定的配置条件自动执行，也可以根据客户端的命令手动执行。

（1）ADB 文件结构

ADB 文件的结构如图 3-11 所示，第一部分长度为 6 个字节，保存着“ARMORY”六个字符，程序在载入文件时可以通过这六个字符快速判断该文件是否为 ADB 文件；第二部分长度为 4 个字节，用来存储表示 Armory 系统版本的一个整数；第三部分是存储的实际数据，其大小根据由备份的数据大小决定；第四部分长度为 1 个字节，用来存储文件结束符 EOF 标志；第五部分是一个 8 个字节长的无符号整数，表示该 ADB 文件的校验和，可以通过计算当前文件的实际校验和与文件中存储的校验和是否匹配来判断文件是否损坏。

万方数据

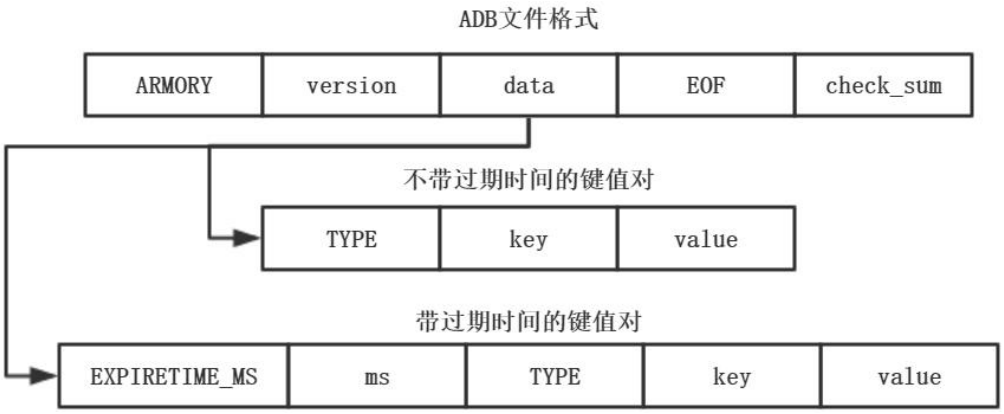


图 3-11　ADB 文件结构

ADB 文件中的数据部分用来存储缓存中的所有键值对信息，共有两种存储格式：不带过期时间的键值对和带过期时间的键值对。对于带过期时间的键值对，第一部分 1 个字节用来存储 EXPIRETIME_MS 标志，表示接下来的数据带有过期时间；第二部分 8 个字节存储着一个带符号类型的整数，记录着一个以毫秒为单位的 UNIX 时间戳，表示该键值对的过期时间；第三部分的 TYPE 为 1 个字节，表示该键值对中值的类型。Armory 缓存系统一共支持五种值类型，其对应的类型标记如表 3-1 所示。

表 3-1　类型标记信息

类型	标记符号	标记值
字符串	ARMORY_ADB_TYPE_STRING	0
列表	ARMORY_ADB_TYPE_LIST	1
集合	ARMORY_ADB_TYPE_SET	2
有序集合	ARMORY_ADB_TYPE_SORTED_SET	3
哈希字典	ARMORY_ADB_TYPE_HASH	4

（2）ADB 数据备份

ADB 数据备份功能是把缓存中的数据写入 ADB 文件，包括阻塞模式和非阻塞模式两种执行方式，阻塞模式是指系统暂停所有操作直到 ADB 文件写入完成为止，非阻塞模式是指当前的进程正常的处理消息请求，同时新创建一个子进程用来向 ADB 文件中写入数据。

万方数据

ADB 数据备份的执行流程如图 3-12 所示，首先检查当前系统是否正在进行 RDB 持久化或 OCR 持久化，如果是则直接退出，如果不是则更新相关的统计信息值，接着创建一个空白的临时文件并将缓存中的数据按照 ADB 文件结构以二进制的方式写入其中，然后刷新缓冲区以确保所有的数据都已经写入了该临时文件，最后将该临时文件按照系统配置更改为指定的名称并替换原有的 ADB 备份文件。

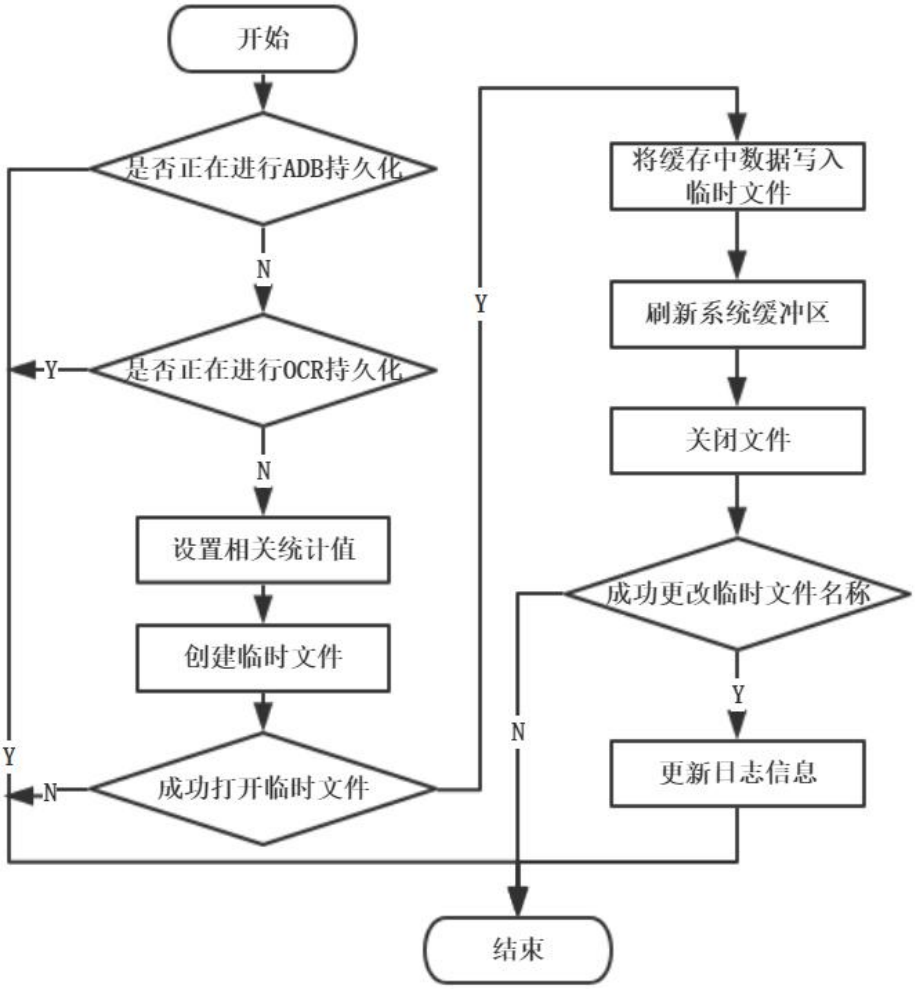


图 3-12　ADB 数据备份流程图

（3）ADB 数据载入

ADB 数据载入是 ADB 数据备份的逆过程，其执行流程如图 3-13 所示，首先根据前六个字符是否是“ARMORY”来判断该文件是否是 ADB 文件，如果不是则退出，

万方数据

如果是则接着检查版本号信息和校验和是否合法，如果都合法则循环遍历文件并读入数据，检查数据是否过期，如果没有过期则将其存入缓存系统，如果遇到文件结束符 EOF 则表示所有数据都已经全部读取完成。

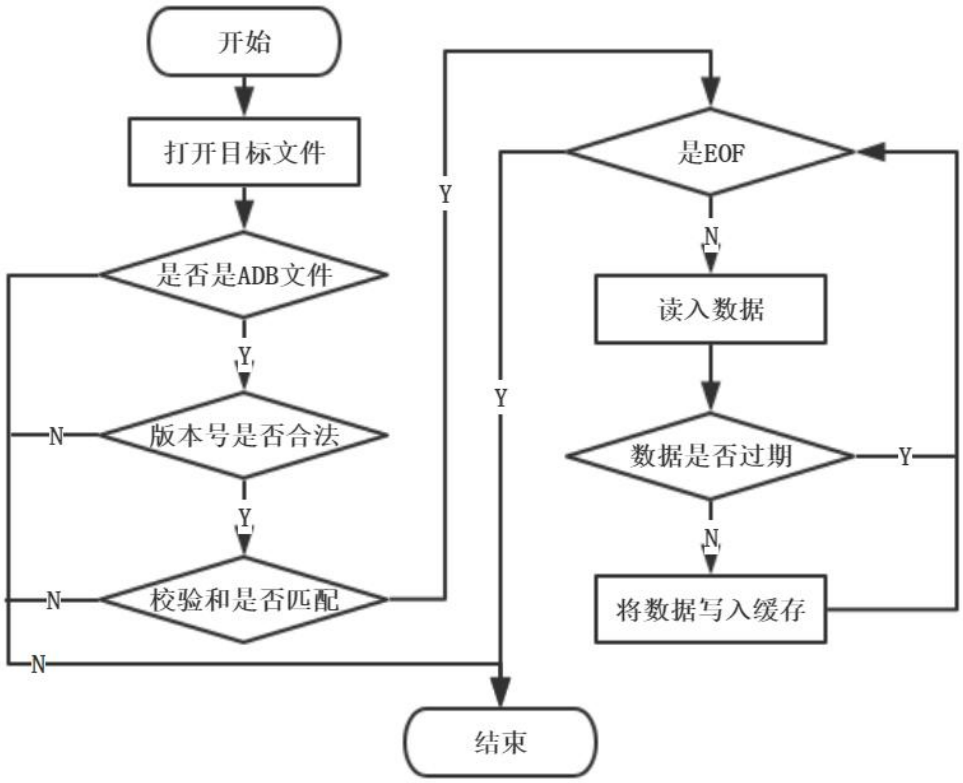


图 3-13　ADB 数据载入流程图

3.4.2　操作命令记录

当缓存系统中的数据量非常大时，采用 ADB 持久化将这些数据转换成二进制格式并写入磁盘是一件很耗时的事情，如果采用阻塞模式则系统可能会在很长一段时间内无法响应，如果采用非阻塞的方式则无法备份开启 ADB 持久化的那个时间节点之后的数据，即 ADB 持久化方法无法做到有效的实时备份数据。

与 ADB 持久化不同，Armory 缓存系统提供的操作命令记录（OCR，Operator Command Record）持久化方式将每次接收到的会更改缓存中数据的命令写入日志文件中，系统发生故障重启后可以从日志文件里依次读出所有的命令并执行，这样就可以将缓存中的数据恢复。

万方数据

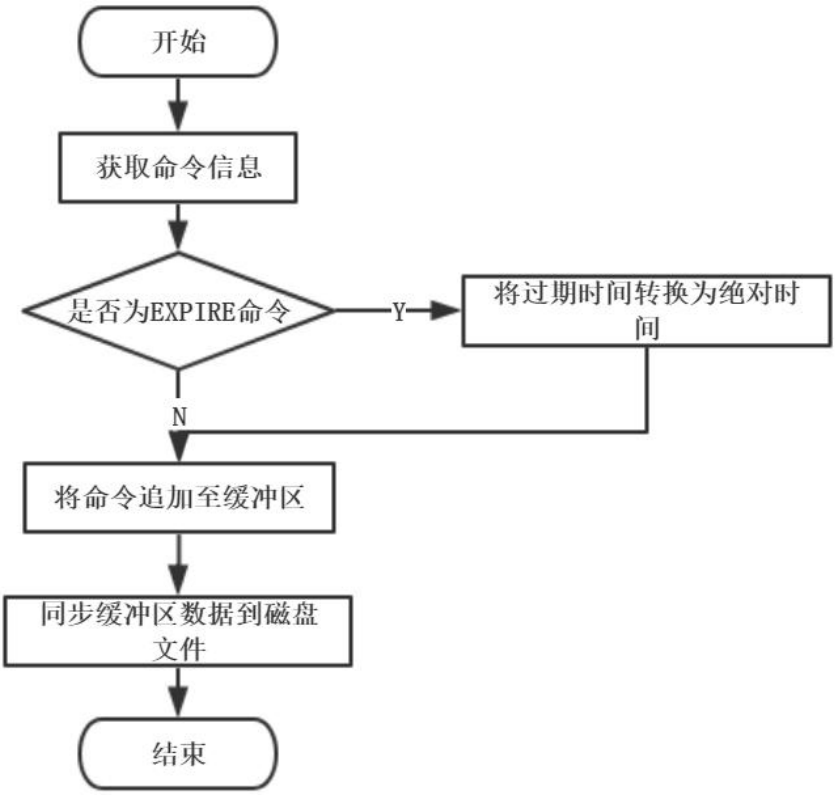


图 3-14　OCR 持久化流程图

如图 3-14 所示，OCR 持久化过程中先检查命令是否为 EXPIRE，如果是则将设置的键失效时间由相对时间转换为绝对时间，否则恢复数据时会造成数据错误，然后将命令添加到缓冲区并最终同步至磁盘日志文件中。

由于内存和磁盘读写速度的差异较大，因此为了提高效率，在默认情况下操作系统会先将数据写入自己的缓存区域，等到缓存区域空间满了或者到达了一定之间之后才会一次性将操作系统缓存区域中的数据写入磁盘。根据同步策略的不同，OCR 持久化一共有三种同步方式：

（1）OCR_FILESYNC_ALWAYS 表示每条命令都立即强制写入磁盘文件中。

（2）OCR_FILESYNC_EVERYSEC 表示先将命令写入系统缓存，然后每秒将系统缓存强制同步到磁盘文件中。

（3）OCR_FILESYNC_NO 表示将命令写入缓存而不指定何时同步，具体的同步时间由操作系统根据系统调度来决定。

万方数据

OCR 文件载入是和 OCR 命令记录相反的过程，其执行流程如图 3-15 所示，首先以只读的方式打开一个 OCR 日志文件，如果文件不为空则循环遍历文件每次读入一行，然后检查该行记录的命令和命令参数是否合法，如果合法就执行该条命令，如果不合法就关闭日志文件并退出。

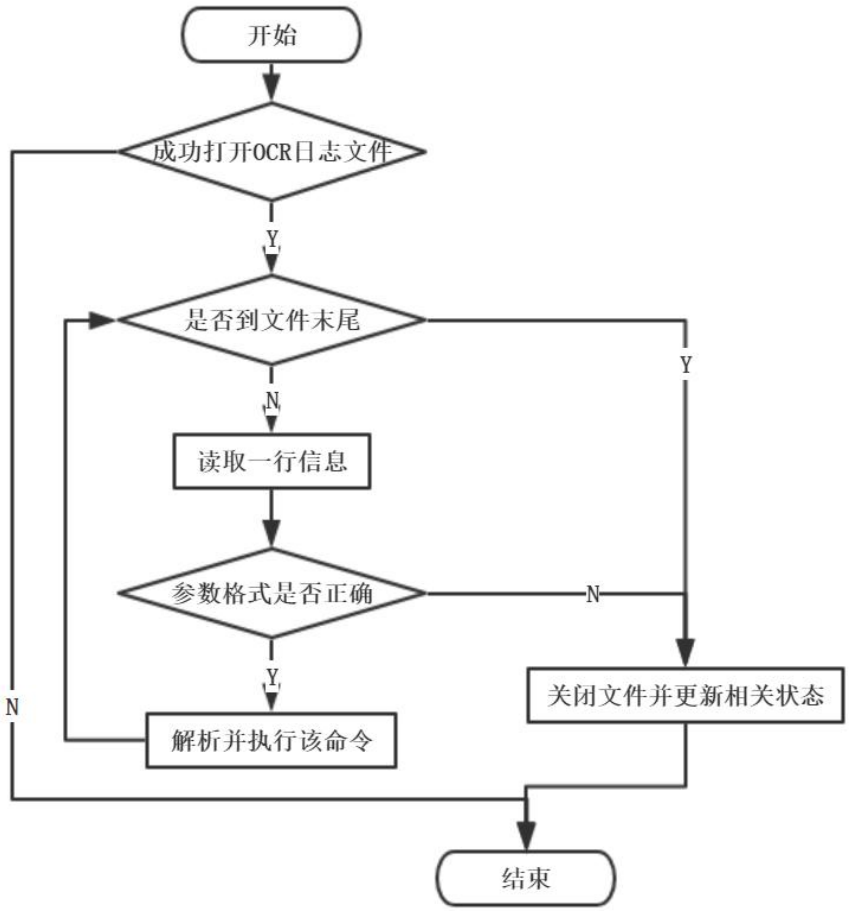


图 3-15　OCR 恢复数据流程图

3.5　主从复制模块设计

在单节点环境中，一旦系统出现故障则整个后台服务都无法正常工作，为了提高服务的可用性，Armory 缓存系统支持主从复制模式，即一个主节点和至少一个从节点一起组成集群共同工作，主从节点之间会自动备份和同步数据来保证数据的一致性，且主节点支持读数据和写数据而从节点只支持读数据，通过读写分离的方式来分散数据请求降低单台节点的负载，主从复制架构如图 3-16 所示。

万方数据

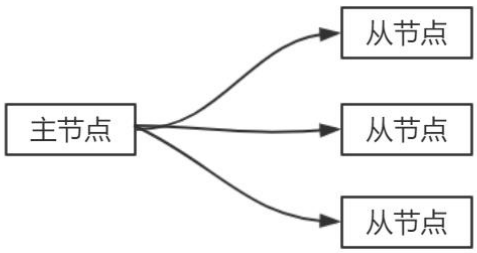


图 3-16　主从复制节点架构

Armory 缓存系统中主从复制模块是采用异步的方式实现，对于指定的事件可以提前注册其回调函数，当系统状态发生变化时操作系统会自动调用对应的函数，再根据全局变量中设置的标识信息进行处理，主从复制中相关的状态标识和意义如表 3-2 所示。

表 3-2　主从复制状态标识和意义

标识名称	意义
ARMORY_REPL_NONE	没有建立主从复制关系
ARMORY_REPL_ CONNECT	从节点等待与主节点建立连接
ARMORY_REPL_ CONNECTING	从节点正在与主节点建立连接
ARMORY_REPL_ RECEIVE_PONG	从节点等待主节点回复 PING 命令
ARMORY_REPL_ TRANSFER	从节点正在从主节点接收数据
ARMORY_REPL_ CONNECTED	从节点与主节点已经建立连接
ARMORY_REPL_ WAIT_BGSAVE_START	主节点等待后台备份数据开始
ARMORY_REPL_ WAIT_BGSAVE_END	主节点等待后台备份数据结束
ARMORY_REPL_ SEND_BULK	主节点将备份数据发送给从节点
ARMORY_REPL_ ONLINE	主节点备份数据传输完成

主从复制模块的功能主要包括建立主节点和从节点之间的关系、第一次同步或者长时间断开连接后的全量同步、少量数据的增量同步、主节点向从节点广播新的命令等。

3.5.1　主从关系建立

Armory 缓存系统中建立主从关系的方法有三种：在 armory.conf 配置文件中添

万方数据

加 slaveof <master_ip> <master_port>选项并在启动时使用该配置文件，或在 Armory-Server 启动时加上-- slaveof <master_ip> <master_port>，或者直接在从节点执行 SLAVEOF <master_ip> <master_port>命令。

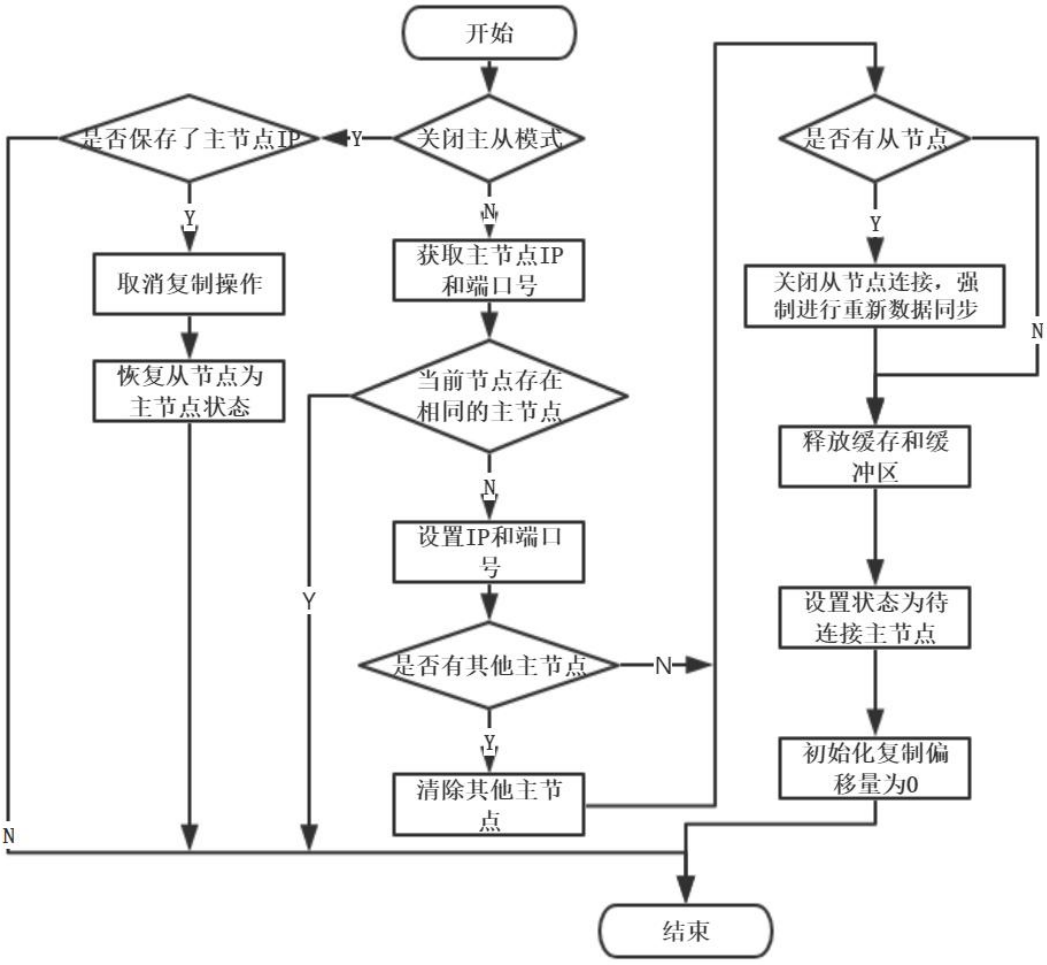


图 3-17　主从关系建立与关闭流程图

主从关系建立和关闭过程的流程图如图 3-17 所示，首先判断是否要取消主从关系，如果是则检查给定的 IP 是否是当前节点的主节点，然后断开主节点和从节点之间的连接并将从节点提升为主节点；如果不是则获取给定节点的 IP 和端口号，如果给定的节点 IP 和端口号和该节点目前的主节点信息完全符合则说明两个节点之间已经建立了主从关系，因为不需要再一次建立关系所以可以直接返回；否则清除和其他主节点之间的复制关系，并关闭和该节点从节点之间的连接，等该节点和

万方数据

其主节点数据同步完成后强制其从节点重新同步新的数据，最后将连接的状态信息设置为待连接主节点状态，并初始化复制操作的偏移量为 0。

3.5.2 全量复制

主从关系建立阶段只是在从节点中设置了主节点信息并初始化相关状态，实际上主节点和从节点之间并没有建立连接，当主从节点第一次执行数据复制时两者会先建立连接再同步数据。

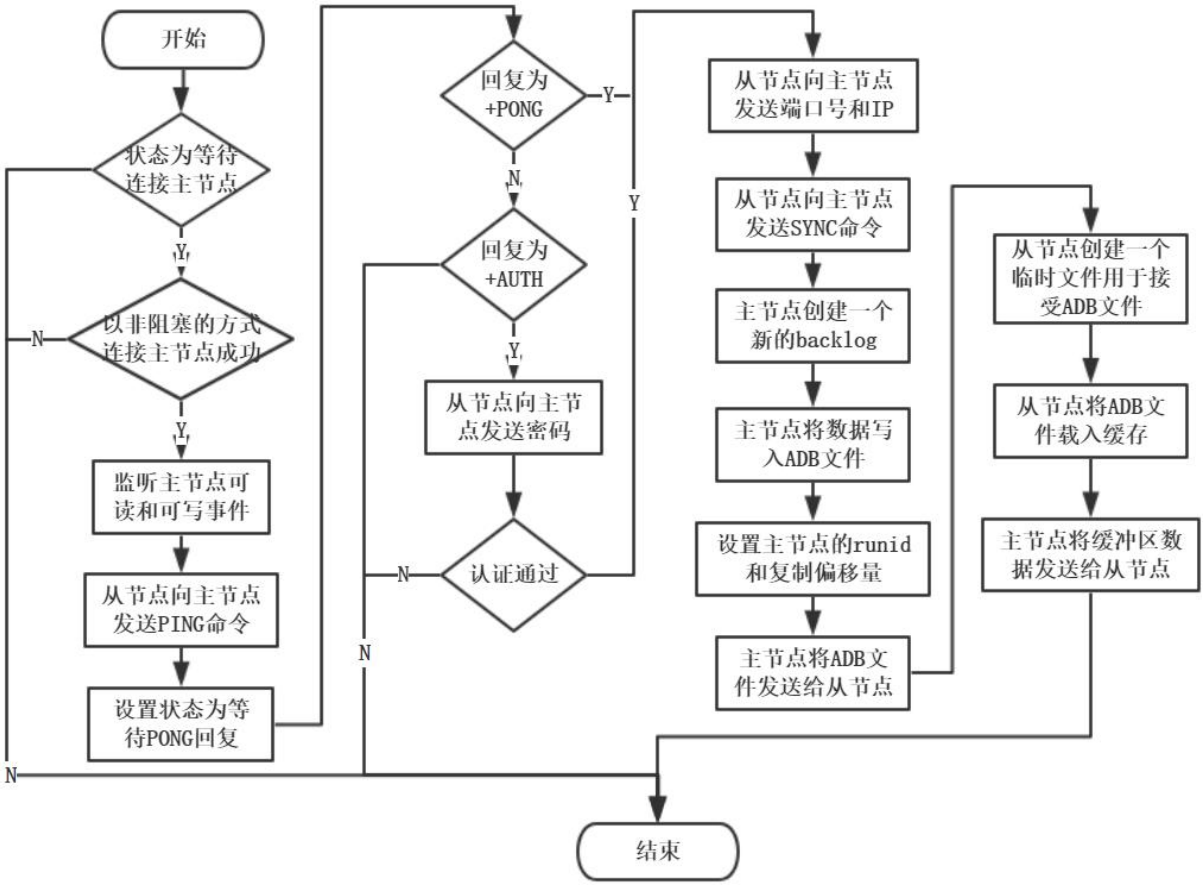


图 3-18　全量复制流程图

全量复制是指从节点复制主节点当前缓存中的全部数据，一般在主从节点之间第一次同步数据时或者在增量复制阶段主从节点之间断开太长时间时执行。全量复制的执行流程如图 3-18 所示，如果从节点的状态为等待连接主节点则以非阻塞的方式连接主节点，接着向主节点发送“PING”命令等待回复，如果主节点回复“+AUTH”

万方数据

则发送密码并进行验证，如果主节点回复“+PONG”表示主从节点之间连接正常，此时从节点将自己的 IP 地址和发送数据使用的端口号发送给主节点，然后向主节点发送一个全量复制的命令请求“SYNC”，主节点收到请求后将自己缓存中的数据写入 ADB 文件并发送给从节点，从节点以此文件来恢复自己缓存中的数据。由于主节点是以非阻塞的方式写入 ADB 文件，在开始执行写入 ADB 文件这个时间节点之后主节点接收到的所有新的数据都无法写入 ADB 文件中，因此主节点在写入 ADB 文件之前先创建一个 backlog 文件从来记录该阶段中新收到的数据，在将 ADB 文件发送给从节点之后将 backlog 文件中的内容也发送给从节点，以此来确保主节点和从节点之间数据的完全与执行。

3.5.3 增量复制

由于全量复制需要主节点先将数据备份至 ADB 文件再将其发送给从节点，当数据量比较大时将会消耗主服务器大量的 CPU、内存和网络资源，因此为了提高数据同步的效率和性能，Armory 缓存系统提供了增量复制机制。

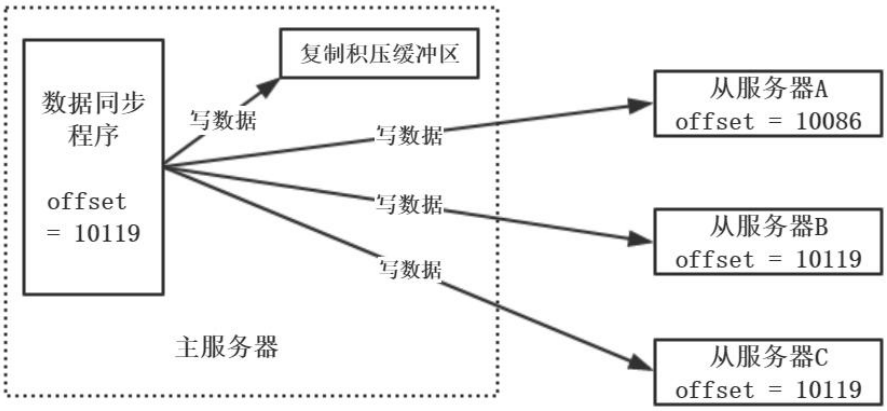


图 3-19　主节点向复制积压缓冲区和从节点写入数据

如图 3-19 所示，主节点和从节点分别维护一个复制偏移量来记录当前双方已经发送和接受了多少数据，主节点每次向从节点发送 N 个字节数据时将自己复制偏移量的值加上 N，从节点每次从主节点接收到 N 个字节数据后也将自己的复制偏移量值加上 N，通过对比主从节点之间复制偏移量的值是否相同就可以快速判断数据是否完全一致。主节点还会维护一个固定长度大小的队列作为复制积压缓冲区，当主

万方数据

节点向从节点发送数据时会写入一份到积压缓冲区中，即主节点的复制积压缓冲区中会保存着一部分最近传输的数据以及其对应的复制偏移量。一旦主从节点之间因为意外断开了连接，重新连接后首先检查从节点的复制偏移量是否在积压缓冲区存储的数据范围之内，如果在则只需要将这一小部门数据重新发送，如果不在则执行全量复制，这样就可以避免短时间断开连接之后重传所有的数据造成资源浪费。

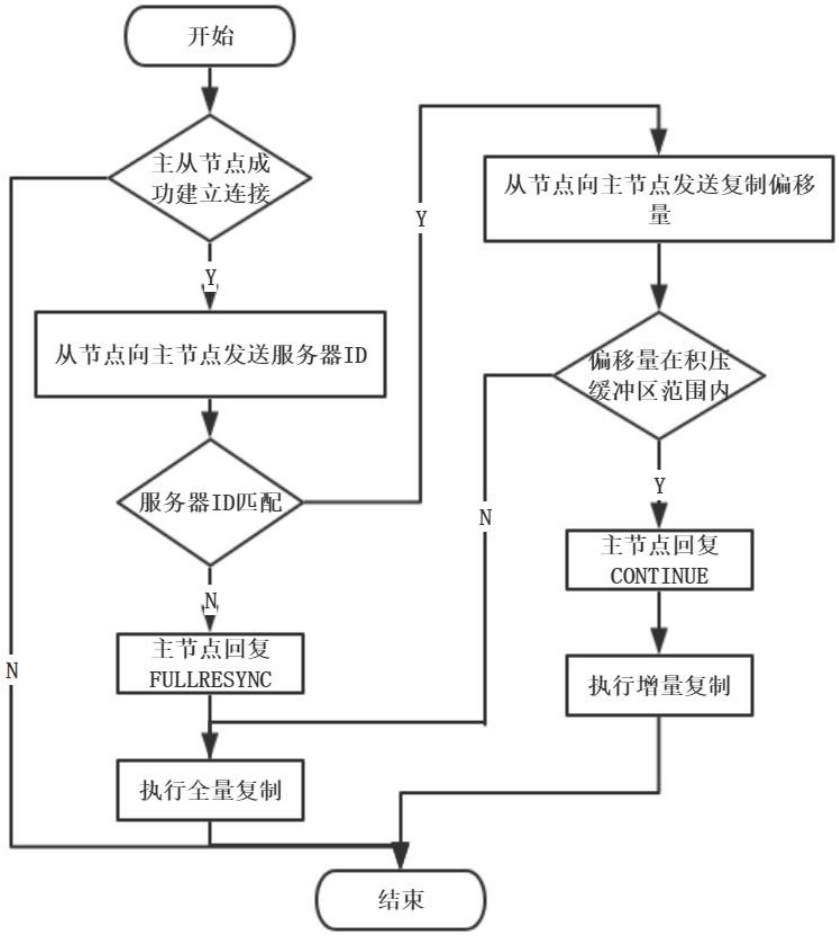


图 3-20　增量复制流程图

增量复制的执行流程图如图 3-20 所示，当主从节点重新连接后，从节点首先向主节点发送服务器运行 ID 和复制偏移量，如果服务器 ID 和主节点的 ID 不匹配则表示从节点重新连接上的服务器并不是之前连接的那台，所以需要重新复制全部数据；如果服务器 ID 匹配，则根据复制偏移量是否在积压缓冲区的范围内来决定是只发送缓冲区内的数据还是全部数据。

万方数据

3.5.4 命令广播

当完成了初始的数据同步之后，主从服务器就会进入命令广播阶段，当主节点接收到会对数据造成改变的命令后会将该命令广播给自己所有的从节点，从节点接收到命令后根据命令内容调用对应的函数执行，从而保证主从节点之间数据一直保持一致。

在命令广播阶段，从节点会以每秒一次的频率向主节点发送一个确认信号和当前从节点中存储的复制偏移量值作为心跳检测，心跳检测的作用有两个：

（1）心跳检测可以判断主从节点之间的连接是否正常，如果主节点超过一秒钟没有收到从节点的确认信号则表明连接可能出现了问题。

（2）如果主节点向从节点发送的命令在网络中丢失，则从节点的复制偏移量肯定小于主节点的复制偏移量，因此主节点可以根据从节点发送过来的复制偏移量判断有没有命令丢失，如果命令丢失了则将积压缓冲区中存储的命令重新发送一次，直到从节点正确收到所有命令为止。

3.6 本章小结

本章首先从功能性和非功能性两个方面对 Armory 缓存系统做了需求分析，然后根据需求分析的结果将系统的功能分为了系统服务模块、数据持久化模块、主从复制模块和底层存储模块共计四个模块，并且分别对前三个模块进行了详细的功能设计并给出了具体的执行流程图，其中底层存储模块由于在关键技术中已经有所阐述故不再重复介绍。

万方数据

4 Armory 缓存系统的实现

本章首先介绍了 Armory 缓存系统开发环境的选择，然后根据上一章中的详细设计来具体的实现系统的功能，并详细的阐述了一些关键功能的实现方法，最后对系统进行了功能性测试来验证实现的正确性。

4.1 开发环境

由于需要高性能和便捷的操作内存，因此 Armory 缓存系统采用高级语言中最为接近硬件的 C 语言作为开发语言，同时为了方便代码的部署和恢复，采用了 GitHub 作为代码的存储仓库和版本控制工具。开发环境的具体信息为：

（1）采用了 Ubuntu 16.04 LTS 作为开发用的操作系统。

（2）开发环境硬件配置为 Intel(R) Core(TM) i5-4210M CPU @ 2.60 GHZ 2.60GHZ，内存大小为 4G，硬盘大小为 1T。

（3）采用的编辑器为 VIM，版本号 7.4.1689。

（4）采用的编译器为 GCC，版本号为 5.4.0。

（5）采用的调试器为 GDB，版本号为 5.4.0。

4.2 系统服务模块实现

系统服务模块主要用来提供一些保证系统正常运行的关键功能，包括全局变量初始化、服务器状态初始化、网络端口监听的创建、接收客户端发送的命令、解析命令执行后返回结果、周期性的客户端任务和复制任务以及定期清理过期数据等。

4.2.1 系统初始化

系统初始化主要负责初始化服务器状态信息，并根据指定的或者默认的配置文件启动服务器，如果在设置中开启了 OCR 或者 ADB 持久化方式则从备份文件中恢复缓存数据，最后注册事件处理函数并运行事件循环，直到服务器关闭为止，其执行流程图如图 4-1 所示，主要实现了以下功能函数：

万方数据

（1）void ZMallocEnableThreadSafeness(void)：设置线程安全。

（2）void ZMallocSetOomHandler(void (*oom_handler)(size_t))：设置内存溢出的处理函数。

（3）void DictSetHashFunctionSeed(uint32_t seed)：设置哈希函数种子。

（4）void InitServerConfig(void)：以默认值初始化服务器的相关全局信息。

（5）void InitServer(void)：创建并初始化缓存数据结构、监听端口、创建时间时间和文件时间。

（6）void LoadDataFromDisk(void)：以 OCR 或 ADB 文件恢复缓存数据。

（7）void AmSetBeforeSleepProcess(AmEventLoop *loop, AmBeforeSleepProcess *before_sleep)：进入事前循环之前先设置有关信息。

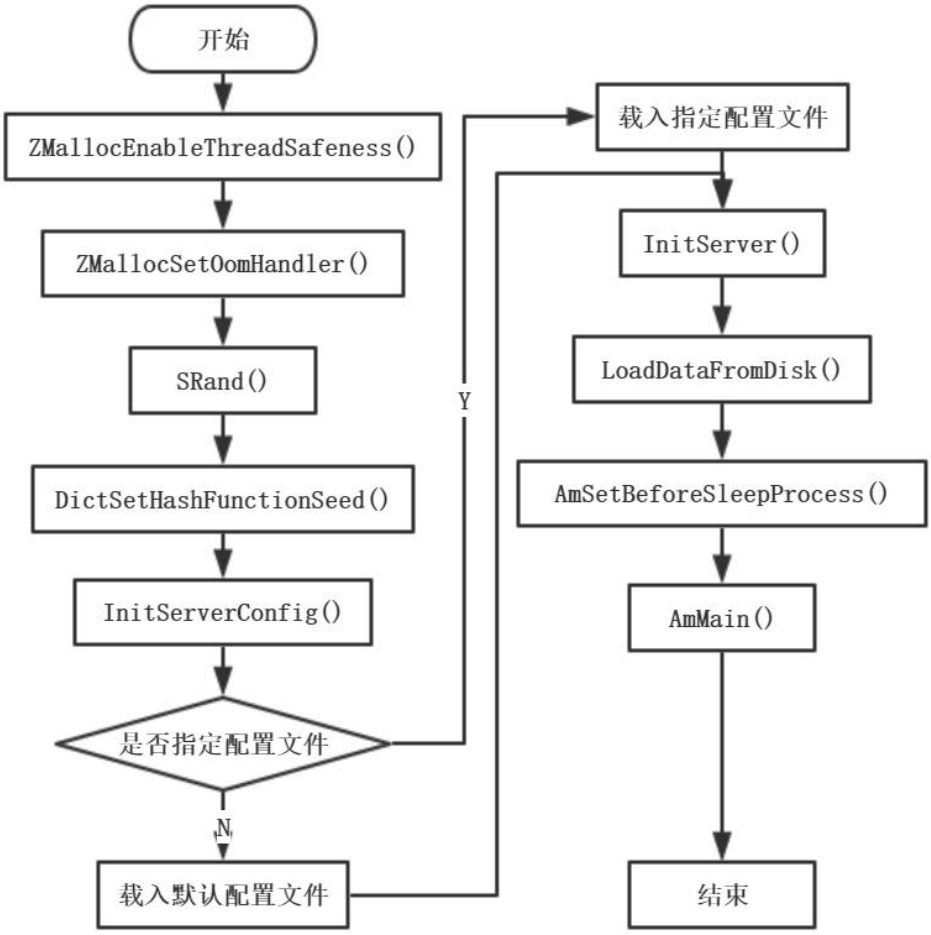


图 4-1　系统初始化流程图

万方数据

4.2.2 命令解析与执行

服务正常启动后系统会监听指定的端口等待客户端连接，当客户端成功连接后会向服务器发送命令进行指定的操作，服务器接收到命令后进行解析，其执行流程如图 4-2 所示。如果是 QUIT 命令则关闭与客户端的连接，否则在命令列表中查找该命令是否存在并检查参数格式是否正确，并根据系统当前是否有最大内存限制来选择性执行清理内存操作，最后检查当前节点是否是从节点并且命令是写命令，如果都不是则执行该命令对应的函数并返回结果，该功能中主要实现了以下函数：

（1）struct ArmoryCommand *LookupCommand(SdString name)：从命令列表中查找该命令，返回值是一个保存有该命令所有信息的结构体。

（2）int FreeMemoryIfNeeded(void)：计算当前使用的内存空间大小，如果大于系统设置的最大可用空间大小则释放一部分内存。

（3）void call(Client *c, int flags)：调用和命令相关联的执行函数并记录执行过程中的相关信息，如果命令执行后修改了缓存系统中的数据且该节点是主节点，则还需将该命令转发给所有的从节点。

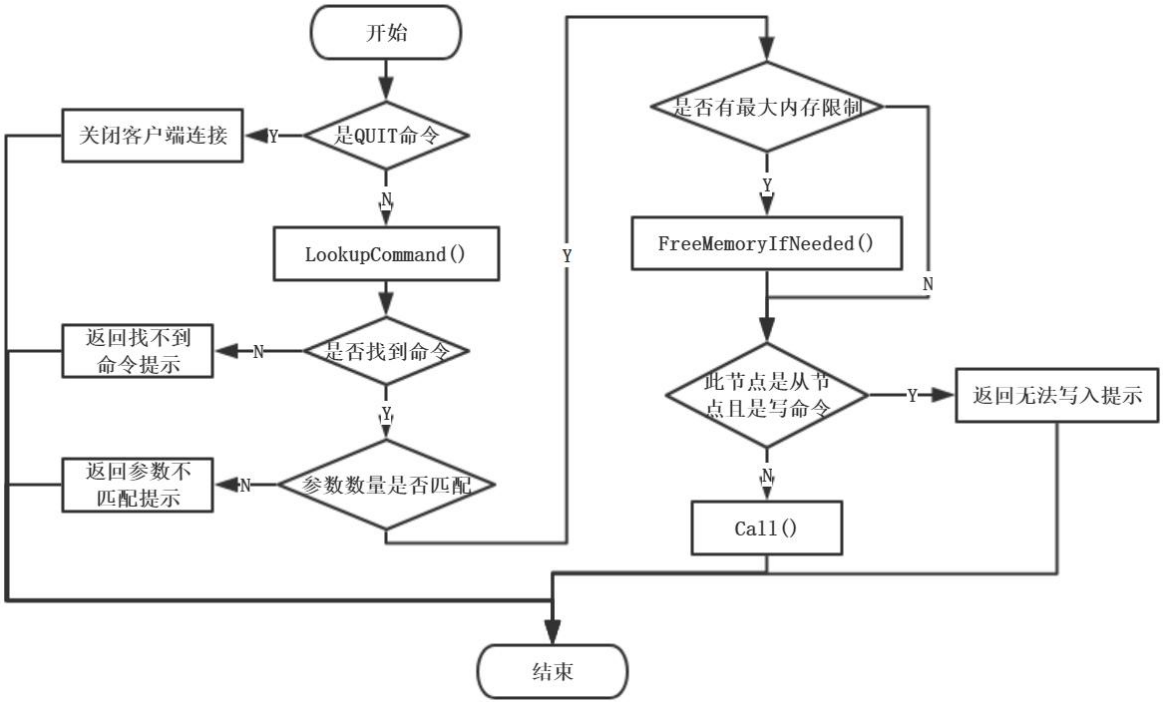


图 4-2　命令解析与执行流程图

万方数据

ArmoryCommand 是一个用来存储 Armory 缓存系统中命令信息的结构体类型，如表 4-1 所示，其包含了命令的名名称、相关联的执行函数、接受的参数个数、支持的读写属性等所有信息。

表 4-1　ArmoryCommand 结构体字段

字段名	类型	含义
name	char *	命令的名称
proc	ArmoryCommandProcess *	命令对应的执行函数
arity	int	命令接受的参数个数
sflags	char *	命令的读写属性等标记
flags	int	整数类型的命令读写属性标记
getkeys_proc	ArmoryGetKeysProcess *	从命令行获取键的参数
firstkey	int	指定第一个参数是 Key
lastkey	int	指定最后一个参数是 Key
keystep	int	指定参数的步长
microseconds	long long	执行该命令的总时长
calls	long long	该命令的总执行次数

4.2.3　周期性任务

周期性任务函数 ServerCrontab 是对 Armory 缓存系统中所有需要定期执行的任务的封装，每秒钟执行一次，主要包括更新服务器统计信息、如有需要则关闭服务器、执行客户端和缓存数据相关的周期任务、根据需要进行数据持久化、执行主从复制相关的周期任务等，其执行流程如图 4-3 所示，该功能中主要实现了以下函数：

（1）void UpdateCachedTime(void)：更新缓存系统中记录的系统时间。

（2）void TrackInstantaneousMetric(int metric, long long current_reading)：更新命令执行次数、从网络中读到的字节数、写入网络中的字节数等统计信息。

（3）size_t ZMallocUsedMemory(void)：更新服务器使用内存的最大峰值。

（4）size_t ZMallocGetRss(void)：更新常驻内存大小。

万方数据

（5）int PrepareForShutdown()：执行关闭服务器前的准备工作，包括将 ADB 和 OCR 文件保存到磁盘、杀死 ADB 和 OCR 数据持久化进程、刷新所有的输出缓冲区和关闭所有监听的连接。

（6）void ClientsCrontab(void)：执行检查客户端连接状态等客户端的周期性任务。

（7）void CachedDataCrontab(void)：执行删除过期的缓存数据等周期性任务。

（8）int RewriteOperatorCommandRecordBackground(void)：在后台重写 OCR 数据持久化文件。

（9）int DataSaveBackground(char *filename)：在后台进行 ADB 持久化操作。

（10）void FlushOperatorCommandRecord(int force)：将 OCR 缓存数据写入磁盘。

（11）void ReplicationCron(void)：执行与主从复制相关的周期性任务。

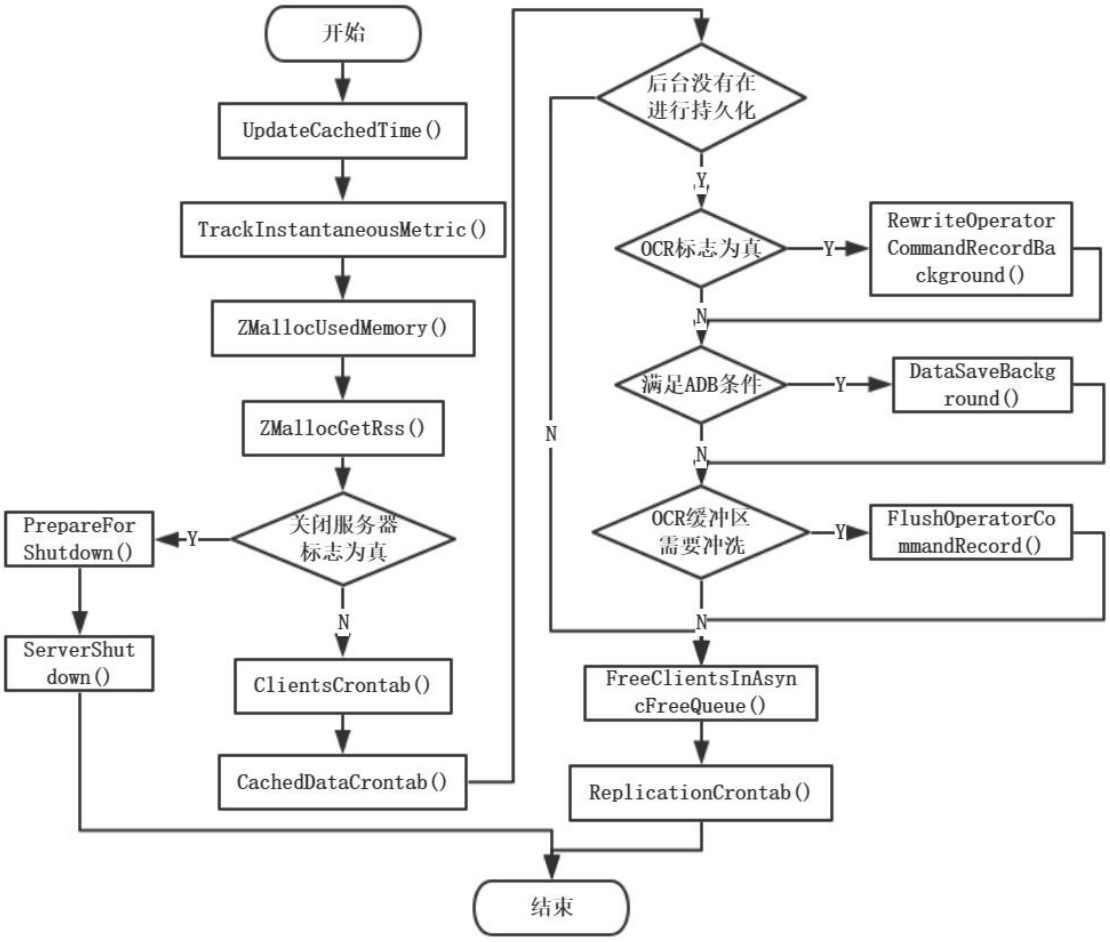


图 4-3　周期性任务流程图

万方数据

4.2.4 内存空间清理

Armory 缓存系统在启动时会设置能够使用的最大内存空间，当使用的内存大小超过这个上限后会自动删除一些键来释放空间，该功能由 FreeMemoryIfNeeded 函数实现。

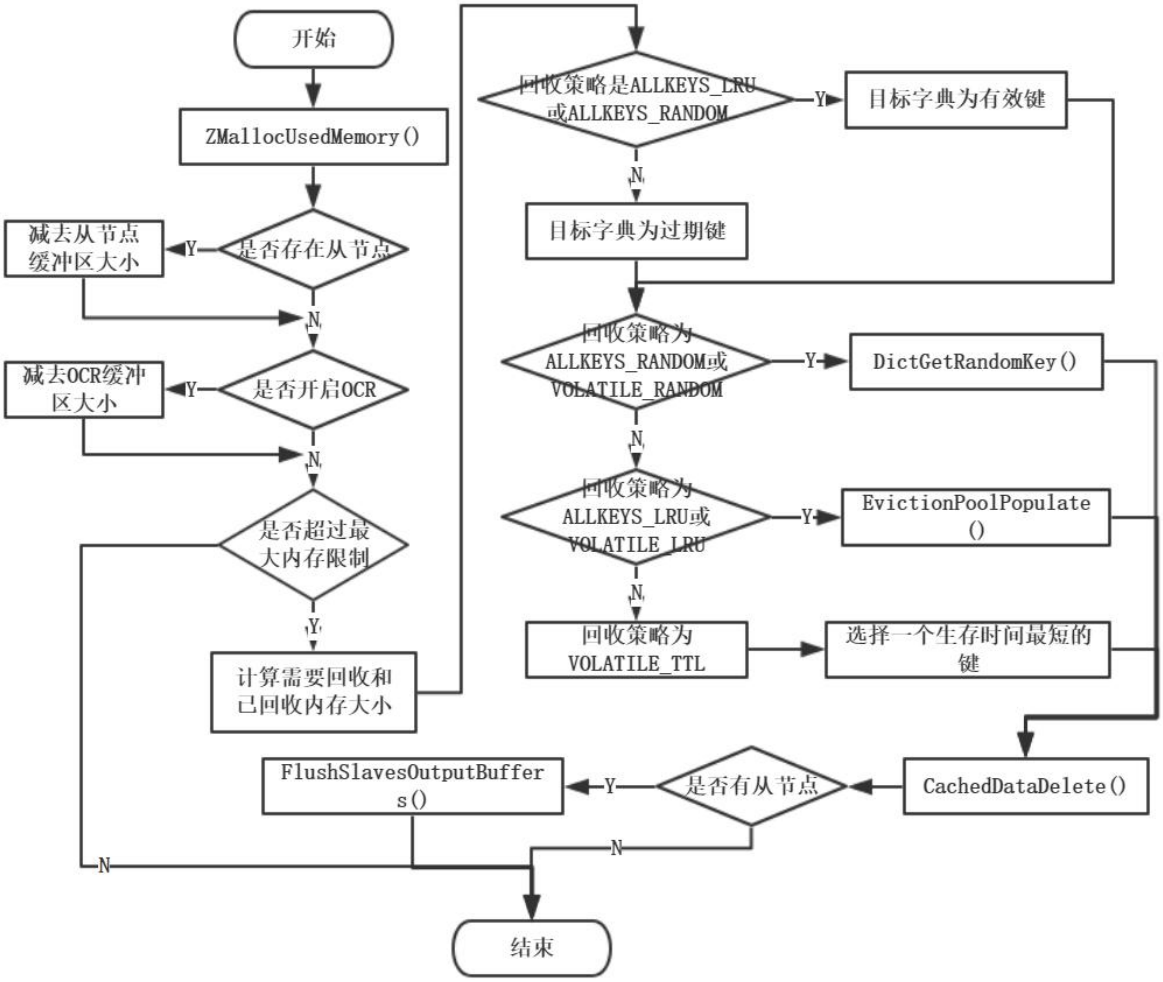


图 4-4　内存空间清理流程图

内存空间清理的执行流程如图 4-4 所示，首先统计目前使用的内存大小，然后减去从节点和 OCR 持久化的缓冲区大小，再根据配置的策略选择要删除键的类型并释放，如此循环直到内存空间大小满足要求为止，如果该节点有从节点，则每次删除键的同时将删除信息通知所有的从节点。该功能中主要实现了以下函数：

（1）size_t ZMallocUsedMemory(void)：统计当前已经使用的内存空间大小。

万方数据

（2）SdString * DictGetRandomKey(Dict *d)：从字典中随机的返回一个键。

（3）void EvictionPoolPopulate(Dict *s, dict *key, struct EvictionPoolEntry *pool)：从数据池中选择一个最近最少被使用的键返回。

（4）int CachedDataDelete(ArmoryData *data, ArmoryObj *key)：释放指定的键。

（5）void FlushSlavesOutputBuffers(void)：将相关数据写入所有从节点的缓冲区。

4.3 数据持久化模块实现

由于 Armory 缓存系统采用内存作为存储介质，一旦由于意外导致服务器断电或者系统发生故障导致进程退出，那么内存中存储的所有数据都会丢失，这样给数据安全带来极大的隐患。为了提高系统的安全性，Armory 缓存系统提供了两种可以将数据持久化到硬盘的方式，一种是将全部数据按照固定的格式写入二进制文件，另一种是将所有的操作命令都写入日志文件，一旦发生故障就可以使用二进制备份文件或日志文件恢复所有数据。

4.3.1 缓存数据备份

缓存数据备份功能（Armory Data Backup，ADB）会将当前时间节点内存中的所有数据按照固定格式写入磁盘文件，有阻塞和非阻塞两种写入方式，其执行流程如图 4-5 所示，如果系统没有正在进行 ADB 或 OCR 持久化则继续，首先设置一些相关的数据信息并关闭子进程中监听的所有连接，然后创建一个空白的临时文件将内存中存储的所有键值对写入文件中，最后将该临时文件重命名为 ADB 持久化指定的文件存储名称即可。该功能中主要实现了一下函数：

（1）void CloseListeningSockets(int unlink_unix_socket)：关闭监听的套接字连接。

（2）void ArmorySetProcessTitle(char *title)：将进程标题设置为指定值方便识别。

（3）void AioInitWithFile(AIO *r, FILE *fp)：以指定的文件初始化一个 IO 对象。

万方数据

（4）static int AdbWriteRaw(AIO *adb, void *p, size_t len)：将 Armory 版本标识和版本信息写入已初始化的 AIO 对象中。

（5）int AdbSaveAuxField(AIO *adb, void *key, size_t keylen, void *val, size_t vallen)：向 AIO 对象中写入一个特殊的辅助操作码字段。

（6）int AdbSaveKeyValuePair(AIO *adb, ArmoryObj *key, ArmoryObj *val, long long expire_time, long long now)：将键值信息、过期时间、类型写入 AIO 对象中。

（7）int FileFlush(FILE *fp)：将缓冲区中的数据强制同步到磁盘文件中。

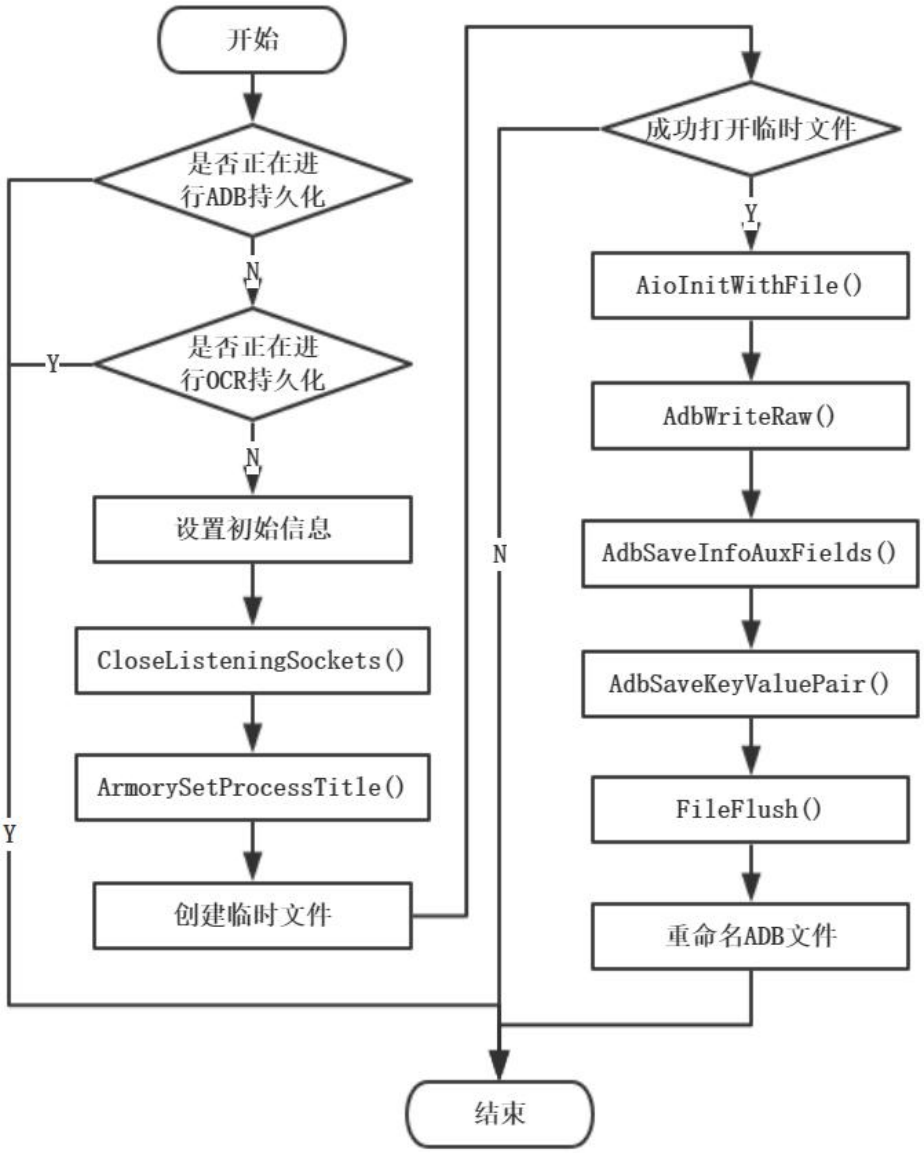


图 4-5　缓存数据备份流程图

万方数据

4.3.2 操作命令记录

由于当数据量较大时缓存数据备份功能不能做到秒级备份，为了防止在备份期间新增加的部分数据丢失，Armory 缓存系统提供了操作命令记录功能，对于所有会对缓存中数据做出改变的命令都将其写入日志文件中，一旦系统发生故障，只需要重新执行这些命令就可以恢复数据。其执行流程如图 4-6 所示，首先初始化一个空的简单字符串，如果是 EXPIRE 命令则将相对时间转换为绝对时间并追加到字符串中，如果字符串长度不为 0 且后台没有正在进行同步则设置延迟检测信息并将缓存的字符串写入文件中，如果写入文件的字节数小于实际缓存的字节数，则说明发生了错误需要将错误的信息截断并恢复磁盘文件之前的内容，最后更新当前 OCR 日志文件的大小。

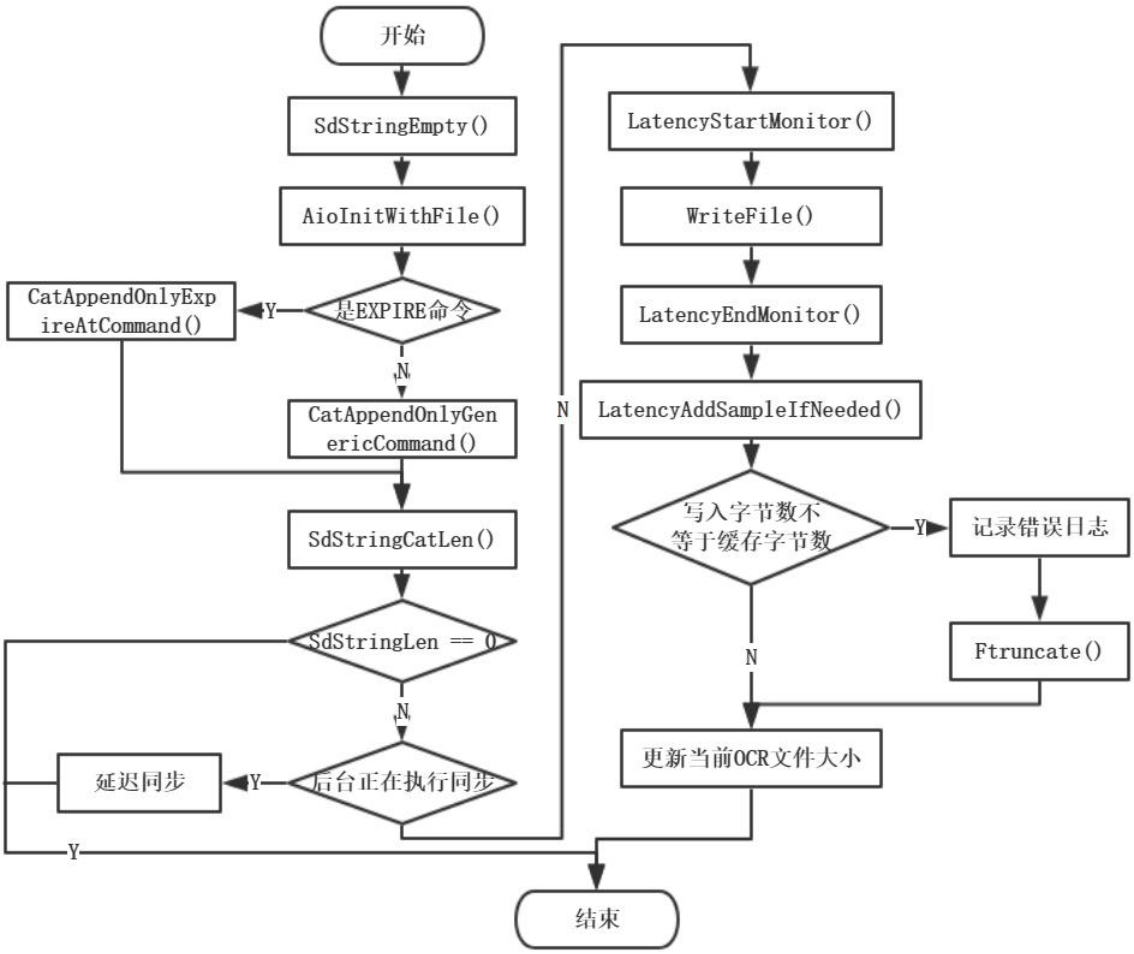


图 4-6　操作命令记录流程图

万方数据

操作命令记录功能中主要实现了以下函数：

（1）SdString SdStringEmpty(void)：初始化一个空的简单字符串。

（2）SdString CatAppendOnlyExpireAtCommand(SdString buf, ArmoryObj *key, ArmoryObj *seconds, struct ArmoryCommand *cmd)：将数据失效命令的相对时间转换成绝对时间。

（3）SdString CatAppendOnlyGenericCommand(SdString dst, int argc, ArmoryObj **argv)：将传入的命令和命令参数转换成协议格式。

（4）SdString SdStringCatLen(SdString s, const void *t, size_t len)：将字符串 t 的前 len 个字节追加到字符串 s 中。

（5）void LatencyStartMonitor(mstime_t time)：设置延迟检测开始的时间。

（6）void WriteFile(SdString buf)：将字符串缓存 buf 中的数据写入磁盘文件。

（7）void LatencyEndMonitor(mstime_t time)：设置延迟检测结束的时间。

（8）void LatencyAddSampleIfNeeded(SdString str, mstime_t time)：将相关信息添加进延迟诊断字典。

（9）int Ftruncate(FILE *fp, int len)：将文件 fp 中的后 len 个字节删除，以便将文件恢复到写入错误之前的状态。

4.4 主从复制模块实现

在单节点环境中，一旦服务器由于意外或者系统故障导致无法工作则会影响整个后台服务，为了提高系统的可用性，Armory 缓存系统支持将一台主节点和至少一台从节点组织成一个集群一起工作，主节点提供读写服务，从节点则只能读数据，主从节点之间会自动复制数据以保证数据的一致性，通过读写分离的方式来提高系统的安全性和稳定性。

4.4.1 主从关系建立

主从关系建立子模块主要负责设置从节点的主节点信息，其执行流程如图 4-7 所示，如果是取消主从关系命令则断开主从节点之间的连接并将从节点提升为主节

万方数据

点，否则获取提供的 IP 地址和端口号，检查指定的节点是否已经是当前节点的主节点，如果是则直接返回即可，如果不是则设置 IP 地址和端口号信息、释放当前节点的主节点、解除所有客户端的阻塞状态、断开该节点所有从节点的连接以便强制其从节点重新同步数据、释放主节点的缓存和积压缓冲区、取消正在执行的复制操作，最后初始化复制偏移量、清零连接断开的时长并将连接状态设置为等待连接主节点。

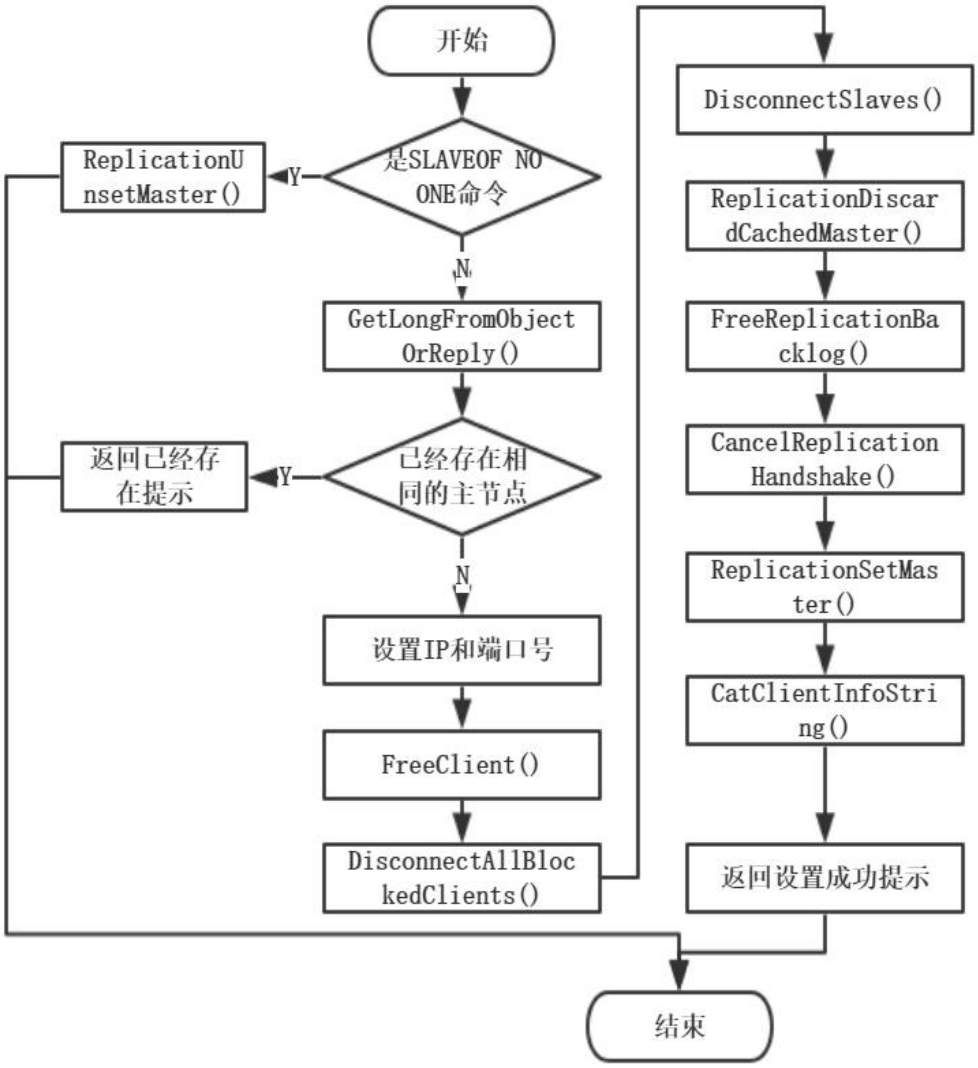


图 4-7　主从关系建立流程图

主从关系建立功能中主要包含了以下函数：

（1）void ReplicationUnsetMaster()：取消复制操作，将该节点提升为主节点。

万方数据

（2）int GetLongFromObjectOrReply(Client *c, ArmoryObj *o, long *target, const char *msg)：获取命令行中指定的端口号信息。

（3）void FreeClient(Client *c)：释放当前节点的主节点。

（4）void DisconnectAllBlockedClients(void)：取消所有客户端的阻塞状态。

（5）void DisconnectSlaves(void)：断开与所有从节点的连接。

（6）void ReplicationDiscardCachedMaster(void)：释放主节点的缓存结构。

（7）void FreeReplicationBacklog(void)：释放主节点的复制积压缓冲区。

（8）void CancelReplicationHandshake()：取消当前正在进行的复制操作。

（9）void ReplicationSetMaster(char *ip, int port)：设置复制偏移量、清零连接断开的时长并将连接状态设置为等待连接主节点。

（10）SdString CatClientInfoString(SdString s, Client *c)：获取客户端的信息。

4.4.2 全量复制

全量复制子模块的功能是将主节点缓存中的全部数据写入 ADB 文件后发送给从节点，从节点再根据接收到的二进制文件来恢复自己的数据，以此达到同步主从节点之间数据的目的。

其执行流程如图 4-8 所示，从节点首先以非阻塞的方式连接主节点，然后向主节点发送 PING 命令，如果收到的回复是“AUTH”则发送密码验证，如果收到的回复是“PONG”则将端口号和 IP 地址发送给主节点，然后发送 SYNC 命令表示开始同步数据，主节点创建一个积压缓冲区用来存储生成 ADB 文件期间其收到的新数据，然后将当前缓存中的数据写入 ADB 文件并发送给从节点，最后将积压缓冲区中的数据也发送给从节点完成全部数据的同步。该子模块中主要实现了以下函数：

（1）int connectWithMaster(void)：以非阻塞的方式连接主节点。

（2）void AmDeleteFileEvent(AmEventLoop *event_loop, int fd, int mask)：暂时取消监听 fd 的写时间，以等待接受主节点的回复信息。

（3）char *SendSynchronousCommand(int flags, int fd)：向主节点发送命令信息。

（4）SdString SdStringFromLongLong(long long v)：将端口信息转换成字符串。

万方数据

（5）void CreateReplicationBacklog(void)：创建主从复制操作的积压缓冲区，用来存储主节点备份数据期间新收到的数据。

（6）int StartBgsaveForReplication(int mincapa)：为主从复制操作在后台将缓存中数据写入 ADB 文件中。

（7）void ReplicationAllData(void)：将 ADB 文件和积压缓冲区中的所有数据都发送给从节点。

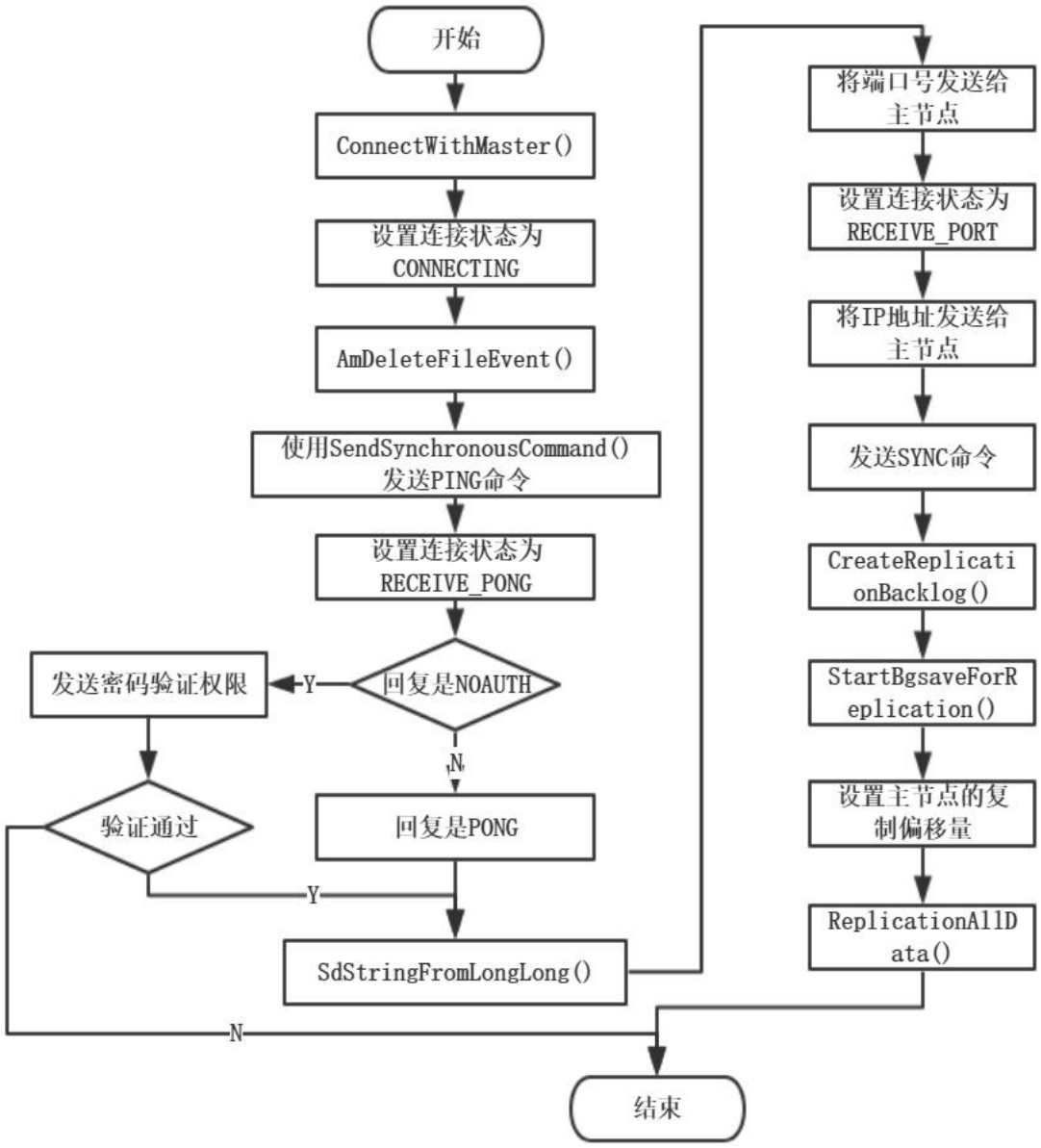


图 4-8　全量复制流程图

万方数据

4.5 系统测试

软件测试是软件开发流程中最重要的一个环节，通过测试可以验证软件的各个功能是否按照设计正常实现、是否有未知的潜在的问题、在复杂多变的运行环境下能否正常工作，根据测试结果可以对软件做进一步的修改和提高。本节通过设计多个测试内容并验证和预期结果是否相同来测试 Armory 缓存系统能否满足设计的要求，并最终给出总结性的测试结果。

4.5.1 测试环境与配置

本次测试使用了四台服务器，其中一台作为主节点，另外三台作为从节点，具体的配置信息如表 4-2 所示。

表 4-2 测试用机器配置信息

节点类型	节点 IP 地址	配置信息
主节点	192.168.0.101	操作系统：Ubuntu 16.04 LTS CPU：Intel(R) Core(TM) i5-4210M CPU @ 2.60 GHZ 2.60GHZ 内存大小：2G
从节点	192.168.0.102	
从节点	192.168.0.103	
从节点	192.168.0.104	

4.5.2 功能测试

（1）系统服务模块测试

系统服务模块为整个 Armory 缓存系统提供了基础服务，下面将重点测试命令解析与执行、周期性任务、内存空间清理等几个功能来验证该模块是否实现了设计要求。首先测试命令的解析与执行功能，其测试内容和测试用例如表 4-3 所示。

表 4-3 命令解析与执行测试内容

测试内容	测试用例	预期结果	测试结果
SET 命令	SET key1 “key_1”	返回 OK	测试通过
SET 命令	SET key2	提示参数错误	测试通过
SET 命令	SET key3 [a, b]	返回 OK	测试通过
SET 命令	SET key4 {“name”:”liu”}	返回 OK	测试通过
GET 命令	GET key1	返回“key_1”	测试通过
GET 命令	GET key2	返回 NULL	测试通过
DEL 命令	DEL key3	返回[a, b]	测试通过
EXPIRE 命令	EXPIRE key1 20	返回 OK	测试通过
EXPIREAT 命令	EXPIREAT key2 20171205 22:00	返回 OK	测试通过

万方数据

然后测试系统的周期性任务和内存空间清理功能，主要通过日志记录的信息分析周期性任务是否正常运行，并测试内存空间清理的五种不同的数据策略，其测试内容和测试用例如表 4-4 所示。

表 4-4　周期性任务和内存空间清理测试内容

测试内容	测试用例	预期结果	测试结果
客户端周期性任务	向日志文件中写入“ClientsCrontab”	日志中有“ClientsCrontab”标志	测试通过
缓存数据周期性任务	向日志文件中写入“CachedDataCrontab”	日志中有“CachedDataCrontab”标志	测试通过
复制相关周期性任务	向日志文件中写入“ReplicationCrontab”	日志中有“ReplicationCrontab”标志	测试通过
VOLATILE_LRU 清理策略	向缓存中大量随机写入即将失效的数据	最近使用较少的数据量变少	测试通过
VOLATILE_TTL 清理策略	向缓存中写入大量新的数据	新写入的数据量变少	测试通过
VOLATILE_RANDOM 清理策略	向缓存中随机写入即将失效的数据	失效数据总的大小变小	测试通过
ALLKEYS_LRU 清理策略	向缓存中随机写入大量数据	有效数据和失效数据最近未被使用的数据量都减少	测试通过
ALLKEYS_RANDOM 清理策略	向缓存中写入大量的随机数据	有效数据和失效数据的体积都变小	测试通过

（2）数据持久化模块测试

数据持久化模块是保障数据安全性的重要模块，下面将根据是否能正确的生成持久化文件并从持久化文件中恢复数据来测试数据持久化模块的功能，其测试内容和测试用例如表 4-5 所示。

表 4-5　数据持久化测试内容

测试内容	测试用例	预期结果	测试结果
ADB 数据备份	使用命令 SAVE	成功生成 ADB 文件	测试通过
ADB 数据备份	使用命令 BGSAVE	成功生成 ADB 文件	测试通过
ADB 数据载入	清空缓存系统、指定 ADB 文件后重启	数据成功恢复	测试通过
OCR 数据备份	关闭 ADB 模式并开启 OCR 模式	成功生成 OCR 日志文件	测试通过
OCR 数据载入	清空缓存系统、指定 OCR 文件后重启	数据成功恢复	测试通过

万方数据

（3）主从复制模块测试

通过读写分离，主从复制模块提高了 Armory 缓存系统的效率和可用性，下面将从主从关系建立、全量复制、增量复制等方面来测试主从复制模块，测试内容和测试用例如表 4-6 所示。

表 4-6　主从复制模块测试内容

测试内容	测试用例	预期结果	测试结果
主从关系建立	SLAVE 192.168.0.101	成功建立主从关系	测试通过
主从关系取消	SLAVEOF NO ONE	成功取消主从关系	测试通过
全量复制	建立主从关系后从节点向主节点发送 SYNC	成功同步数据	测试通过
增量复制	全量同步完成后从节点向主节点发送 PSYNC	成功开始增量复制阶段	测试通过

4.5.3　性能测试

为了对 Armory 缓存系统进行性能测试，可以使用 Python 脚本模拟用户的访问请求，同时为了避免由于网络延迟等因素导致额外的响应时长，可以在服务器本机上运行 Python 脚本进行测试。

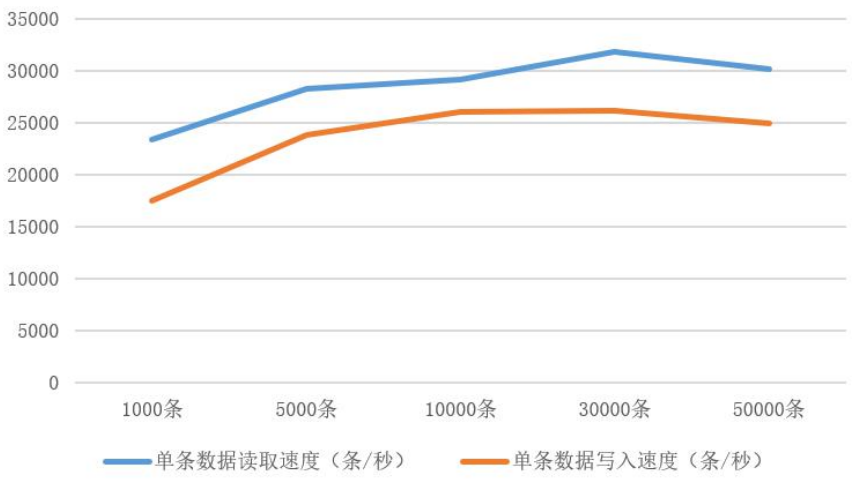


图 4-9　单条数据读写性能测试

万方数据

在单条数据读写性能测试中，每次只读取或写入一条数据，分别测试总计读取和写入 1000 条、5000 条、10000 条、30000 条、50000 条数据所需的时间，用数据量除以时间即可得到每秒平均的读写条数，其结果如图 4-9 所示，观察发现当数据量不太大时读写速度会随着数据量的增大而增大，当数据量大于一万条后平均读写速度会随着数据量规模的增大而降低。

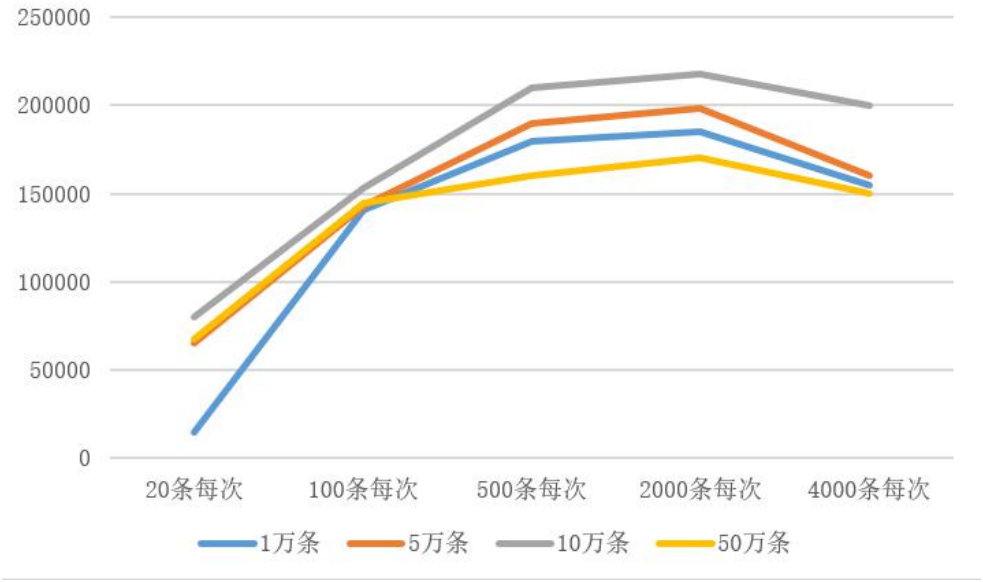


图 4-10　批量数据写入性能测试

在批量数据的写入性能测试中，每次分别写入 20 条、100 条、500 条、2000 条、4000 条数据，统计分别写入 1 万条、5 万条、10 万条、50 万条数据所需要的时间，用总的数据量除以时间即可得到平均每秒的写入条数，其测试结果如图 4-10 所示，数据总量为 10 万条时平均的写入速度最大，每次写入的条数较少时由于需要更多的请求和响应次数所以导致平均写入速度较小，随着每次写入条数的增加，平均写入的速度也随之增加，当写入条数超过 2000 条每次后由于需要写入的数据量太大导致系统的响应速度反而变慢。

批量数据的读取性能测试和写入性能测试类似，其测试结果如图 4-11 所示，随着每次读取条数的增加，平均读取速度会先增大后减小，当每次读取 500 条数据时平均速度达到最大值。

万方数据

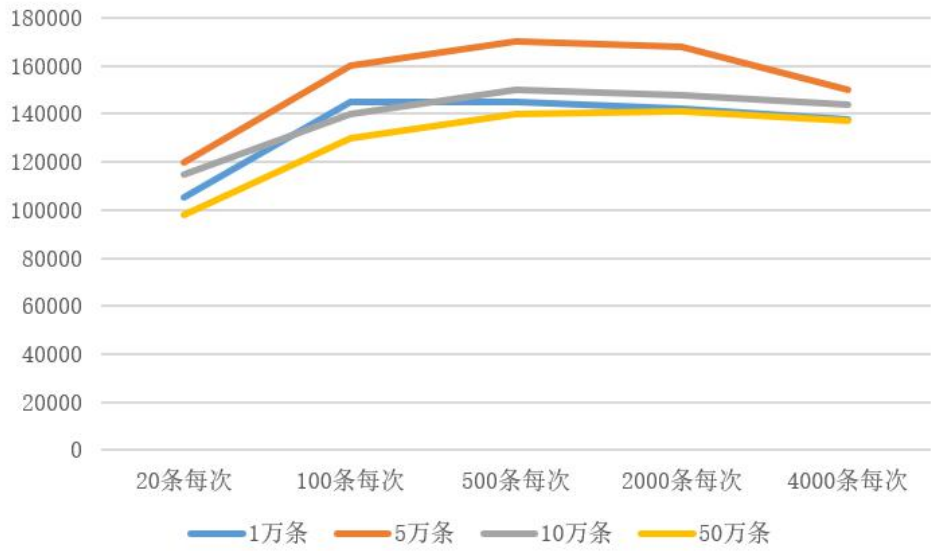


图 4-11　批量数据读取性能测试

使用脚本对 Armory 缓存系统和 MySQL 数据库进行单条数据请求的读写性能测试，测试结果如图 4-12 所示，Armory 缓存系统的平均读写速度均稳定在每秒 18000 次左右，而直接查询 MySQL 数据库，读取和更新数据的平均速度分别稳定在 5800 和 2800 次每秒左右，说明使用 Armory 缓存系统可以明显提高数据查询和更新的速度。

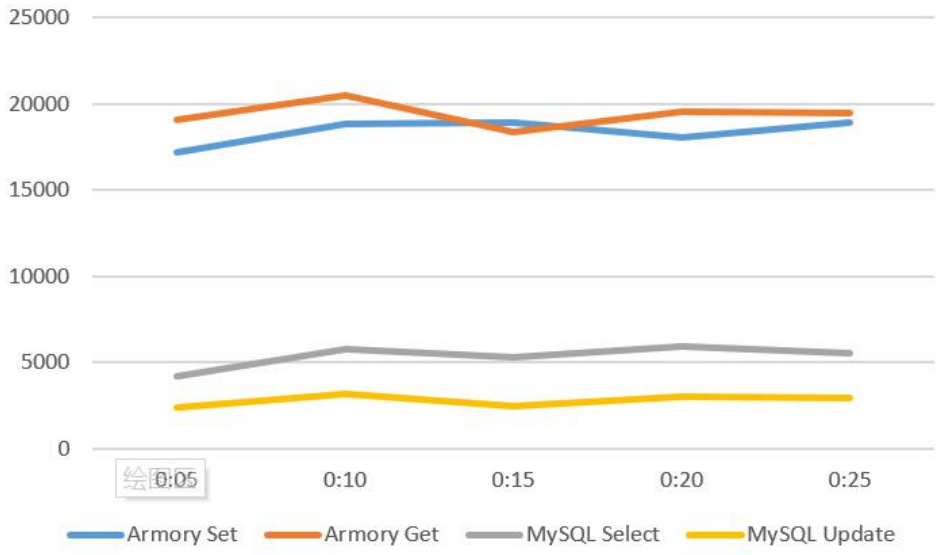


图 4-12　Armory 和 MySQL 读写性能对比

万方数据

4.5.4 测试结果

测试结果可以直观的反应一个软件是否符合设计的功能要求，本小节对命令解析与执行、周期性任务、内存空间清理、ADB 和 OCR 两种数据持久化方式和主从节点之间数据同步等子功能进行了简单的功能测试和性能测试，测试结果显示该软件已经实现了预定的功能和目标，满足我们的设计需求。

4.6 本章小结

本章首先介绍了 Armory 缓存系统的开发环境和开发工具，并根据第三章中的需求分析和模块设计，详细阐述了系统初始化、命令解析与执行、周期性任务、内存空间清理、数据缓存备份、操作命令记录、主从关系建立和全量复制子功能的具体实现，最后对实现的系统进行了详细的功能测试，测试结果显示该软件达到了预期的目的。

万方数据

5 总结与展望

5.1 全文总结

本章将对 Armory 缓存系统的设计与实现工作做一个全面的总结，文章首先阐述了目前在服务器后端业务中存储数据主要使用关系型数据库，随着数据的种类越来越多格式也越来越复杂，非关系型数据库也逐渐的被大量采用，然而由于采用磁盘作为存储介质限制了传统数据库的读写性能，进而提出了本文设计并实现一个基于内存存储的键值对缓存系统的研究目的，并阐述了目前国内外的研究现状。接着介绍了 Armory 缓存系统中使用到的 I/O 多路复用、简单动态字符串和跳跃表三个关键技术，然后根据实际业务需求进行了详细的需求分析和总体设计，将整个 Armory 缓存系统分为了系统服务模块、数据持久化模块和主从复制模块三大部分并对每个模块的功能进行了详细的设计，最后根据详细设计编码实现了该系统的主要功能并进行测试，测试结果显示系统功能符合预期的结果。

本文主要实现了以下几个方面：

（1）设计并实现了系统服务模块。系统服务模块主要提供系统的一些基础服务，包括系统初始化、命令解析与执行、周期性任务和内存空间清理四个主要的子模块，其主要功能是在系统启动时设置一些相关信息，然后监听端口等待客户端的连接并处理客户端的命令请求，同时每秒钟会定期执行一些客户端、缓存数据和复制相关的任务，当使用的内存空间超过设置的上限时还会根据指定的算法清理一部分缓存中的数据来释放空间。

（2）设计并实现了数据持久化模块。由于 Armory 缓存系统采用内存作为存储介质，一旦断电或者进程由于意外导致退出了，内存中的所有数据都会丢失，因此数据持久化功能可以大大提高数据的安全性。数据持久化模块包括缓存数据备份和操作命令记录两种持久化方式，二者各有优点，相辅相成。

（3）设计并实现了主从复制模块。Armory 缓存系统允许一台主节点和至少一台从节点一起组成集群共同工作，主节点和从节点之间会自动的同步数据，这种方

万方数据

式不仅提高了系统的性能和可用性，也可以间接的提高数据的安全性。主从复试模块主要包括主从关系建立、全量复制、增量复制和命令广播四个子功能。

（4）详细测试了 Armory 缓存系统的主要功能模块，测试结果显示该系统达到了设计的目标。

5.2 展望

由于个人能力和时间以及篇幅的限制，本文只针对 Armory 缓存系统的几个关键功能模块进行了详细的设计和介绍，有些地方由于篇幅原因没有进行说明。个人认为该系统还有以下几个不足的地方：

（1）没有事务的特性。事务是传统关系型数据库的一大特色，它可以让多条语句执行原子性操作，要么都执行要么都不执行，该特性可以在实际使用中保证数据在高并发情况下的一致性和正确性。

（2）主从复制模式下没有实现故障节点的自动恢复功能，该功能可以保证一旦主节点发生故障，系统会自动选择一个合适的从节点并将其提升为主节点继续工作，这可以大大提高系统的可靠性。

（3）没有真正的实现系统的集群工作方式。如果在集群中每个节点的身份和地位都是相同的，没有主节点和从节点的区别，系统会根据负载均衡的原则自动的分配客户端的访问，这样可以更好的发挥多台服务器节点的特性。

万方数据

致 谢

转眼间来到华中科技大学已经两年半了，回首刚来武汉的时候仿佛就在上个星期，不得不感慨真是时光飞逝如流水。经过半年多的前期准备和近一个月的写作，我的学位论文已经基本完成，而我也即将告别校园生活，在这里我非常感谢过去的这段时光中大家给予我的帮助。

首先感谢我的导师黄立群老师，来实验室两年多了也没能为黄老师做出些什么，每次想到这里我都羞愧万分。黄老师是一个很好的老师，性格温和、治学严谨、待人宽厚，平时对于我们的生活和学习都非常关心，而且教学风格不拘一格、因材施教，给予了我们充分的自由，让我们可以根据自己的兴趣爱好选择学习自己想学的知识。

还要感谢我的爸爸妈妈这么多年来对我无条件的支持，当初决定考研后是他们的鼓励和支持让我坚持到了最后，最终来到了自己梦寐以求的学校，是父母的关爱和理解陪伴我走过了过往的二十多年时光。

感谢实验室的黄婷婷同学，在论文写作的过程中不断的监督我、督促我、提醒我并能及时的解答我的各种困惑和疑问，让我能按时保质保量的完成论文。

感谢我的母校华中科技大学和软件学院为我们提供了如此良好的生活和学习环境，让我们能够安心的学习知识。

衷心的感谢每一个在生活、学习和工作上帮助过我的人，正是因为你们的帮助和支持，我才能不断的提高自己、突破自己！

万方数据

参考文献

[1] 萨师煊, 王珊. 数据库系统概论. 第 3 版. 北京: 高等教育出版, 2000: 13-32

[2] 张华强. 关系型数据库与 NoSQL 数据库. 电脑知识与技术, 2011, 7(20): 4802-4804

[3] C. Nance, T. Losser, R. Iype et al. NoSQL VS RDBMS - Why there is room for both. In: Proceedings of the Southern Association for Information Systems Conference. Savannah: AIS Electronic Library, 2013: 110-116

[4] E. C. Foster, S. Godbole. Database Systems. CA: Apress, Berkeley, 2016: 195-201

[5] M. Armbrust, R. S. Xin, C. Lian et al. Spark SQL: Relational Data Processing in Spark. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. Australia: ACM New York, 2015: 1383-1394

[6] 伍志聪. MySQL 数据库在中小型业务系统的应用. 数字技术与应用, 2011, 11: 122-124

[7] U. F. Minhas. RemusDB: transparent high availability for database systems. The VLDB Journal, 2013, 22(1): 29-45

[8] C. Franke, S. Morin, A. Chebotko et al. Distributed Semantic Web Data Management in HBase and MySQL Cluster. In: Cloud Computing(CLOUD), 2011 IEEE International Conference on. Washington: IEEE, 2011: 105-112

[9] Schwartz, Zaitsev, Tkachenko et al. High performance MySQL: Optimization, backups, and replication. American: O'Reilly Media, 2012: 27-125

[10] R. Han, L. Guo, M. M. Ghanem. Lightweight Resource Scaling for Cloud Applications. In: Cluster, Cloud and Grid Computing(CCGrid), 2012 12th IEEE/ACM International Symposium on. Canada: IEEE, 2012: 644-651

[11] E. Cecchet, G. Candea, A. Ailamaki. Middleware-based database replication: the gaps between theory and practice. In: Proceedings of the 2008 ACM SIGMOD

万方数据

international conference on Management of data. Canada: ACM New York, 2008: 739-752

[12] Z. Wang, Z. Wei, H. Liu. Research on High Availability Architecture of SQL and NoSQL. AIP Conference Proceedings, 2017, 1820(1): 1-4

[13] 马文龙, 朱妤晴, 蒋德钧. Key-Value 型 NoSQL 本地存储系统研究. 计算机学报, 2017, 40: 1-35

[14] M. Stonebraker. SQL databases v. NoSQL databases. Communications of the Acm, 2010, 53(4): 10-11

[15] J. Han, H. E, G. Le. Survey on NoSQL Database. In: International Conference on Pervasive Computing & Applications. South Africa: IEEE, 2011: 363-366

[16] 吕明育, 李小勇. NoSQL 数据库与关系数据库的比较分析. 微型电脑应用, 2011, 27(10): 55-58

[17] 蒋付彬, 王华军. NoSQL 数据库的应用及选型研究. 信息与电脑, 2016, 3: 141-142

[18] 吴丹丹, 王松. 内存数据库及其应用综述. 软件导刊, 2016, 15(6): 168-170

[19] 陈勇. 探析目前对于移动互联网数据研究及对策. 中国新通信, 2015, 17(11): 20-21

[20] 沈啸. 基于 Oracle 数据库海量数据的查询优化研究. 无线互联科技, 2016, 10: 106-107

[21] S. Ghandeharizadeh, J. Yap, H. Nguyen. Strong consistency in cache augmented SQL systems. In: Proceedings of the 15th International Middleware Conference. France: ACM New York, 2014: 181-192

[22] S. Ghandeharizadeh, J. Yap. Cache augmented database management systems. In: Proceedings of the ACM SIGMOD Workshop on Databases and Social Networks. New York: ACM New York, 2013: 31-36

[23] 谷伟, 陈莲君. 基于 MySql 的查询优化技术研究. 微型电脑应用, 2013, 30(7): 48-50

万方数据

[24] W. J. Starke, J. Stuecheli, D. M. Daly. The cache and memory subsystems of the IBM POWER8 processor. IBM Journal of Research and Development, 2015, 59(1): 1-3

[25] 姜承尧. 高性能网站 MySQL 数据库实践. 程序员, 2013(9): 49-53

[26] B. Fitzpatrick. Distributed caching with memcached. Linux Journal, 2004, 2004 (124): 72-76

[27] 曾里. 缓存系统在实际应用中的重要性. 信息通信, 2017(1): 231-232

[28] 李瑞. 基于 memcached 分布式缓存系统的内存利用优化研究: [硕士学位论文]. 武汉: 华中科技大学图书馆, 2015.

[29] 南轶，李先国. 基于.NET Cache+Memcached Web 缓存技术的研究与应用. 科学技术与工程，2011，11(31)：7808-7811

[30] J Petrovic. Using Memcached for Data Distribution in Industrial Environment. In: International Conference on Systems. Cancun: IEEE, 2008. 368-372

[31] S. Chen, X. Tang, H. Wang et al. Towards Scalable and Reliable In-Memory Storage System: A Case Study with Redis. In: Trustcom/BigDataSE/I SPA, 2016 IEEE. Tianjin: IEEE, 2016: 1660-1667

[32] 余景寰, 李贞昊. Redis 数据库可靠性与自适应持久化改进方案. 信息系统工程, 2017(2): 141-144

[33] 杨雪婵. 针对 Redis 主从复制. 网络安全和信息化, 2017(3): 69-71

[34] 赵璐. 阿里巴巴广告应用质量平台设计与实现: [硕士学位论文]. 南京: 南京大学图书馆, 2014.

[35] 高洪，郭斌. 淘宝 Tair 开源分析实践. 程序员, 2011(2): 107-109

[36] 韩君易. NoSQL 数据库解决方案 Tair 浅析. 电子商务, 2011(9): 54-58

[37] 刘磊. 淘宝开源 KV 结构数据存储系统 Tair 技术分析. 电子商务, 2016(3): 61-62

[38] W. R. Stevens, S. A. Rago. Advanced Programming in the UNIX Environment. 第 3 版. America: Addison-Wesley Professional, 2013: 1-78

万方数据

[39] 程远忠, 杜平安. Winsock 中的 I/O 模型探讨. 计算机工程, 2001(1): 178-179

[40] 玉柱. Linux 操作系统中进程调度策略及特点. 信息技术, 2001(11): 36-37

[41] 苑晓芳, 刘志广. Linux 下基于 TCP 传输组件的实现. 无线电通信技术, 2014(4): 46-49

[42] 常正超. 高并发访问量下网络 I/O 模型选择的研究. 电脑知识与技术: 学术交流, 2016, 12(7): 28-29

[43] 王雅文, 姚欣洪, 宫云战. 一种基于代码静态分析的缓冲区溢出检测算法. 计算机研究与发展, 2012, 49(4): 839-845

[44] 韩煜, 张济国, 张冬芳. 一种基于二进制程序的安全加固系统设计和实现. 信息安全与技术, 2015(7): 17-21

[45] 陈庆全, 黄文明, 崔亚楠. 基于改进跳跃表的数据检索系统应用. 计算机系统应用, 2008, 17(12): 73-76

万方数据