|  |
| --- |
| 北京航空航天大学 |
| 软件需求说明书 |
| **Redis** |
|  |
| SY1406108 陈志伟 SY1406112 王珊珊 SY1406311 林 璐 SY1406117 王志鹏 |
| **2015/04/10** |

**版本变更历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 提交日期 | 编制人 | 说明 |
| Version 1.0 | 2015-03-31 | 全体组员 | 初步完成需求规格说明书的第一版 |
| Version 1.1 | 2015-04-01 | 陈志伟 | 完成1.1版本的修改 |
| Version 1.2 | 2015-04-01 | 林璐 | 完成1.2版本的修改 |
| Version 1.3 | 2015-04-01 | 王志鹏 | 完成1.3版本的修改 |
| Version 1.4 | 2015-04-02 | 王珊珊 | 完成1.4版本的修改 |
| Version 2.0 | 2015-04-04 | 全体组员 | 完成2.0正式本版的修改 |
| Version 3.0 | 2015-04-10 | 陈志伟 | 完成3.0版本的修改 |
| Version 3.1 | 2015-04-11 | 王志鹏 | 完成3.1版本的修改 |
| Version 3.2 | 2015-04-11 | 林璐 | 完成3.2版本的修改 |
| Version 3.3 | 2015-04-11 | 王珊珊 | 完成3.3版本的修改 |
| Version4.4 | 2015-04-17 | 陈志伟 | 完成4.4版本的修改 |
| Version4.5 | 2015-04-17 | 王志鹏 | 完成4.5版本的修改 |
| Version4.6 | 2015-04-18 | 林璐 | 完成4.6版本的修改 |
| Version4.7 | 2015-04-18 | 王珊珊 | 完成4.7版本的修改 |

目录

[1范围 1](#_Toc416508898)

[1.1 标识 1](#_Toc416508899)

[1.2 系统概述 1](#_Toc416508900)

[1.3 数据字典 2](#_Toc416508901)

[1.4 文档概述 5](#_Toc416508902)

[2 项目描述 5](#_Toc416508903)

[2.1 模块概述 5](#_Toc416508904)

[2.2 典型应用场景描述 7](#_Toc416508905)

[3 功能需求 8](#_Toc416508906)

[3.1 服务器模块 8](#_Toc416508907)

[3.1.1 启动服务 9](#_Toc416508908)

[3.1.2 自定义服务器配置 10](#_Toc416508909)

[3.1.3 处理事件 10](#_Toc416508910)

[3.1.4 关闭服务器 13](#_Toc416508911)

[3.2 客户端模块 13](#_Toc416508912)

[3.2.1 发送命令请求 14](#_Toc416508913)

[3.2.2 读取命令请求 15](#_Toc416508914)

[3.2.3 执行命令请求 16](#_Toc416508915)

[3.2.4 回复命令 17](#_Toc416508916)

[3.3 RDB持久化模块 17](#_Toc416508917)

[3.3.1 保存快照 18](#_Toc416508918)

[3.3.2 同步回写SAVE 19](#_Toc416508919)

[3.3.3 异步回写BGSAVE 20](#_Toc416508920)

[3.3.4 载入数据 20](#_Toc416508921)

[3.4 AOF持久化模块 21](#_Toc416508922)

[3.4.2 命令传播 23](#_Toc416508923)

[3.4.3 缓存追加 23](#_Toc416508924)

[3.4.4 文件写入和保存 24](#_Toc416508925)

[3.4.5 AOF文件读取和数据还原 25](#_Toc416508926)

[3.4.6 AOF后台重写 25](#_Toc416508927)

[4 非功能性需求分析 26](#_Toc416508928)

[4.1 鲁棒性 26](#_Toc416508929)

[4.2 安全性 27](#_Toc416508930)

[4.3 高效率 28](#_Toc416508931)

[5 参考文献 29](#_Toc416508932)

# 1范围

## 1.1 标识

Redis版本号： redis-3.0.0-rc5

模 块 名 称： Redis服务器、客户端、RDB持久化、AOF持久化模块

需求报告版本：V4.0

## 1.2 系统概述

Redis是NoSQL系列中的一种key-value类型的轻量级内存数据库，全名为远程字典服务(REmote DIctionary Server)。它是由Salvatore Sanfilippo使用ANSI C语言编写的，支持网络、可基于内存亦可持久化的开源日志型、Key-Value数据库，并提供多种语言的API。从2013年3月15日起，Redis的开发工作由VMware主持。从2013年5月开始，Redis的开发由Pivotal赞助。

Redis定位于一个内存数据库，但事实是其并不是将所有的数据都存储在内存中，其持久性体现在在硬盘上进行写操作，它不仅仅是一种简单的key-value存储。Redis支持存储的value类型有string(字符串)、lists(链表)、sets (集合)、zsets(有序集合)和hash等。这些数据类型均支持push、pop、add、remove，取交集并集和差集及更丰富的操作，且这些操作都是原子性的。它支持各种不同方式的排序，为保证效率，它的数据都保存在内存中,但是Redis会周期性地把更新的数据写入磁盘或把修改操作写入追加的记录文件,并且在此基础上实现了master-slave同步。因为是纯内存操作，Redis的性能非常出色,每秒可以处理超10万次读写操作，是已知性能最快的Key-Value数据库。

Redis具有许多优秀的特性，如支持多种数据结构、支持简单事务控制、支持持久化、支持主从复制、Virtual Memory功能等。

## 1.3 数据字典

表格 1 Reids需求分析数据字典

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 术语 | 英文 | 说明 |
|  | 非关系型数据库 | NoSQL | NoSQL数据库是Not Only SQL的简称，即不仅仅是SQL，是对SQL应用不足的一种补充，中文翻译过来也称为非关系型数据库。它是一类通常强调非关系性、分布式、开源和并行性的数据库。CAP理论，BASE思想(BASE模型是反ACID模型的，该模型牺牲高一致性,获得可用性)和最终一致性三者为NoSQL的三大理论基础。 |
|  | key-value 数据库 | Key-value DataBase | key-value是NoSQL数据库中最简单,也是发展最快的一种数据模型,其遵循的最基础的原理即CAP理论。每一个key值对应任意的数据值, 这个任意的数据值不用特别去管它到底是什么。key-value的数据是通过key值来进行相关存储和检索的,唯一键key可以识别一个条目和所有通过此键进行的删除、修改以及插入新key的操作。它可被看作是一个大的、分布式的、持久的、容错的哈希表。 |
|  | 持久化 | Persistence | 数据持久化就是将内存中的数据模型转换为存储模型，以及将存储模型转换为内存中的数据模型的统称。 数据模型可以是任何数据结构或对象模型,存储模型可以是关系模型、XML、二进制流等。Redis是将数据存储在内存中的,或者使用虚拟内存，但是会定期将内存中的数据写入磁盘，或者用日志方式记录每次更新的日志。前者的性能较高,但是可能会引起一定程度的数据丢失,而后者相反。 |
|  | 主从同步 | Master-slave Synchronization | Redis支持将数据同步到多台从数据库上， 这样可以显著提高数据读取性能。 |
|  | 事件 | Event | 事件是Redis 服务器的核心，它处理两项重要的任务：  1. 处理文件事件：在多个客户端中实现多路复用，接受它们发来的命令请求，并将命令的执行结果返回给客户端。  2. 时间事件：实现服务器常规操作（server cron job）。 |
|  | 事务 | Transaction | 事务提供了一种“将多个命令打包，然后一次性、按顺序地执行”的机制，并且事务在执行的期间不会主动中断——服务器在执行完事务中的所有命令之后，才会继续处理其他客户端的其他  命令。 |
|  | 虚拟内存 | Virtual Memory | 虚拟内存是计算机系统内存管理的一种技术。它使得应用程序认为它拥有连续的可用的内存（一个连续完整的地址空间），而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。Redis 的Virtual Memory ( 虚拟内存) 通过配置可以让用户设置最大使用内存, 当超出这个内存时，通过LRU类似算法, 将一部分数据存入文件中，在内存中只保存使用频率高的数据来提高Redis 性能。 |
|  | AOF | Append Only File | AOF将“操作 + 数据”以格式化指令的方式追加到操作日志文件的尾部，在append操作返回后(已经写入到文件或者即将写入)，才进行实际的数据变更，“日志文件”保存了历史所有的操作过程；当server需要数据恢复时，可以直接replay此日志文件，即可还原所有的操作过程。AOF文件内容是字符串，非常容易阅读和解析，它和redis-protocol具有一样的格式约束 |
|  | 重写 | Rewrite | 重写即创建一个新的 AOF 文件来代替原有的 AOF 文件，新 AOF 文件和原有 AOF 文件保存的数据库状态完全一样，但新AOF 文件的体积小于等于原有 AOF 文件的体积。 |
|  | 数据还原 | Data Reduction | 数据还原即只要根据 AOF 文件里的协议，重新执行一遍里面指示的所有命令，就可以还原Redis 的数据库状态。 |
|  | I/O多路复用 | I/O Multiplexing | IO多路复用是指一旦发现进程指定的一个或者多个描述符可进行无阻塞IO访问时，它就通知该进程。 |
|  | 命令执行器 | Command Actuator | 命令执行器是真正执行命令的程序，由Redis服务器来调用。 |
|  | 快照 | Snapshot | 快照是Redis默认的持久化方式。这种方式是就是将内存中数据以快照的方式写入到二进制文件中,默认的文件名为dump.rdb。用户可以通过配置设置自动做快照持久化的方式，比如配置redis在n秒内如果超过m个key被修改就自动做快照。 |
|  | 过期键 | Expire | Redis的一大特性之一是可以为数据库中的每一个键关联一个过期时间，当到达指定时间后Redis就会把该键从数据库中删除。Redis在RedisDb中维护了一个expires字典，其中的键就是那些已经设置了过期时间的键，而键值就是该键的过期时间（毫秒）。 |
|  | 同步/异步保存 | Synchronization/ Asynchronization | Redis在将内存中的数据库数据保存到磁盘时，有两种方式，同步或异步。同步保存直接在主进程中调用保存函数，阻塞Redis主进程，直到保存完成为止；而异步保存将fork出一个子进程，子进程调用保存函数，并在保存完成后向主进程发送信号通知。 |

## 1.4 文档概述

本文档是对Redis的部分模块的需求分析和规格说明书，主要借助RUCM结构化模板，采用用例图等形式进行分析。

# 2 项目描述

## 2.1 模块概述

本次实验中所选择的四个模块为：

**(1) 服务器模块**

Redis服务器模块主要实现了Redis从启动服务器，到服务器可以接受外来客户端的网络连接这段时间，需要执行的一系列初始化操作及初始化完成后，Redis进入事件轮询处理来自客户端的请求。整个过程主要包括初始化服务器全局状态，程序创建一个redisServer结构的实例变量server用作服务器的全局状态，并将server的各个属性初始化为默认值。接着，程序进入服务器初始化的下一步读入配置文件，并创建daemon进程来运行Redis，并创建相应的pid文件。然后，程序初始化服务器的功能模块，完成这一步之后，服务器打印出Redis的ASCII LOGO、服务器版本等信息，表示所有功能模块已经就绪，但这时服务器的数据库还是一片空白，程序还需要将持久化在RDB或者AOF文件里的数据，载入到服务器进程里面。最后程序打开事件循环，开始接受客户端连接，处理来自客户端的请求。

在服务器模块中选择了四个用例分别为：启动服务、自定义服务器配置、处理事件和关闭服务器。其中启动服务、自定义服务器配置主要来自redis 3.0.0源码中的redis.c文件中的int main()函数；处理事件主要来自ae.c文件中的void aeSetBeforeSleepProc(…)和void aeMain(…)函数，此函数由redis.c文件中的int main()函数调用；关闭服务主要来ae.c文件中的aeDeleteEventLoop(…)函数及其相关的调用函数。

**(2) 客户端模块**

每个客户端可以向服务器发送命令请求，而对于每个与服务器进行连接的客户端，都有相应的redisClient结构，这个结构保存了客户端当前的状态信息，以及执行相关功能时需要用到的数据结构。客户端状态包含许多属性，可以分为两类，一类是比较通用的属性，另一类是与特定功能相关的属性。多个客户端可以连接到同一个服务器，服务器会将新的客户端状态添加到服务器状态结构clients链表的末尾。

客户端模块中的用例选择是依据redis源码中的networking.c以及参考《redis设计与实现》这本书来确定的。书中介绍客户端与服务器端的通信主要可以分为：客户端发送命令请求，服务器读取命令请求，服务器执行命令请求，服务器回复命令四个部分。

**(3) RDB持久化模块**

RDB模块主要将当前内存中的数据库快照保存到磁盘文件中，在Redis重启动时，RDB程序可以通过载入RDB文件来还原数据库的状态。RDB持久化可以在指定的时间间隔内生成数据集的时间点快照。这种不定期的保存方式称为“半持久化模式”，包括同步回收SAVE或者异步回收BGSAVE。SAVE方式直接调用rdbSave函数将内存中的数据库数据以RDB格式保存到磁盘中，若RDB文件已存在，则新的RDB文件将替换已有的RDB文件，阻塞Redis主进程，直到保存完成为止，在主进程阻塞期间服务器不能处理客户端的任何请求；BGSAVE方式则fork出一个子进程调用rdbSave函数，并在保存完成后向主进程发送信号，通知保存完成，此期间Redis服务器主程序仍可继续处理客户端请求。

RDB持久化模块选择四个用例分别为：保存快照、同步回写SAVE、异步回写BGSAVE和载入数据。选择依据是通过阅读Redis-3.0.0源码rdb.c文件并参考其文档：用户通过配置文件或命令行的方式，选择保存方式；Redis服务器通过saveCommand函数和bgsaveCommand函数解析用户通过客户端发出的保存命令，并调用rdbSave函数进行相应保存操作；服务器再次启动时自动调用rdbLoad函数，检查环境、载入并校验数据。

**(4) AOF持久化模块**

AOF持久化模块实现了AOF 以协议文本的方式，将所有对数据库进行过写入的命令（及其参数）记录到 AOF文件，以此达到记录数据库状态的目的。整个模块过程包括Redis 将所有对数据库进行过写入的命令（及其参数）记录到 AOF 文件，以此达到记录数据库状态的目的，为了方便起见，我们称呼这种记录过程为同步；可以实现AOF 文件的读取和数据还原，AOF 文件保存了 Redis 的数据库状态，而文件里面包含的都是符合 Redis 通讯协议格式的命令文本，根据 AOF 文件里的协议，重新执行一遍里面指示的所有命令，就可以还原Redis 的数据库状态；AOF 文件通过同步 Redis 服务器所执行的命令，从而实现了数据库状态的记录。

AOF持久化模块选择的六个用例分别为：命令同步、AOF文件读取和数据还原、AOF后台重写、命令传播、缓存追加、文件写入和保存。选择依据是根据《redis设计与实现》这本书的第五部分内部运作机制中关于AOF的内容，以及Redis-3.0.0源码rdb.c文件。命令同步包括命令传播(propagate\_aof函数)、缓存追加(aofRewriteBufferAppend函数)、文件写入和保存(flushAppendOnlyFile函数)。Redis服务器还实现AOF后台重写(rewriteAppendOnlyFileBackground函数)、文件读取和数据还原（loadAppendOnlyFile函数）。

## 2.2 典型应用场景描述

Redis的一个典型应用场景就是在高并发环境。

由于集群方案即采用分布式Memcache结合Mysql集群，随着用户人群的增加无法应付日益增大的访问压力。此时，把DBMS作为主库将数据全部存放在磁盘中，主要负责写操作；而Redis作为高速数据查询从库，主要负责查操作，最大限度的存放近期热点数据，在节约成本的同时尽最大量提高效率。基于Redis的信息存储图如图1所示：

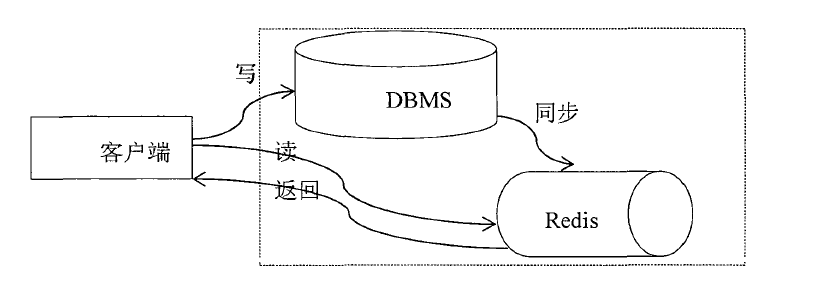


图1 基于Redis信息存储图

总体流程如下：用户发送http请求，如果是写操作将直接操作关系型数据库。如果是读操作则直接访问Redis，并由Redis返回http请求结果。数据平台内部主要实现关系型数据库与Redis之间的数据同步一致性。

下面介绍一下Redis在实际中的使用情况，即新浪微博平台。每天超过2200亿命令操作、超过5000亿的读操作、500亿的写操作、超过18TB内存、有6个网络中心超过500服务器提供服务、同时运行超过2000个实例，是相当大的Redis使用平台。新浪微博采用的处理方式就是将Memcache和MySql全部替换为Redis。Redis作为存储替代Mysql，这样可以解决数据多份之间的一致性问题。同时可以通过修改Redis源码，一方面来满足自己的业务需求；另一方面,完善Redis本身的缺陷。

# 3 功能需求

## 3.1 服务器模块

服务器模块主要实现了服务器启动时的初始化，接着进入事件轮询处理来自客户端的请求，最后关闭服务器。用例图如下：

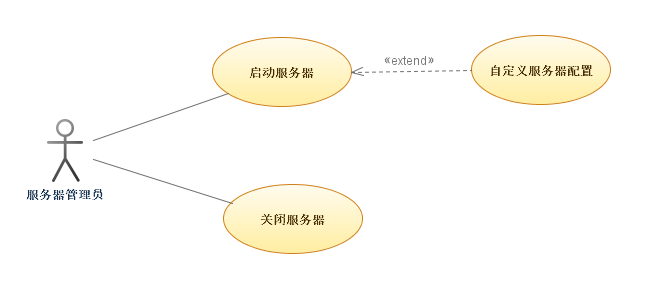


图2 服务器模块用例图

这里的actor分别为服务器管理员，因为启动和关闭服务器不应该是一个普通用户所拥有的职权，否则会对系统安全造成威胁，应该交给专门的服务器管理员；在3.1.3中还增加了一个处理事件的流程图，用来描述服务器内部运行过程，它是服务器的一个主要的组成部分。

### 3.1.1 启动服务

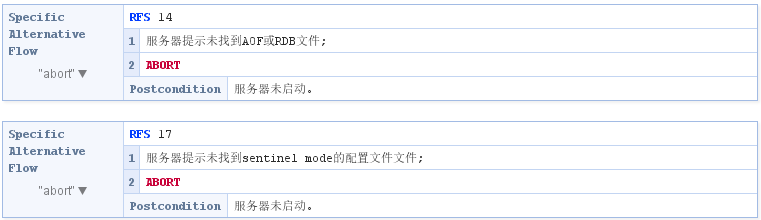


图 3 启动服务用例规格

### 3.1.2 自定义服务器配置

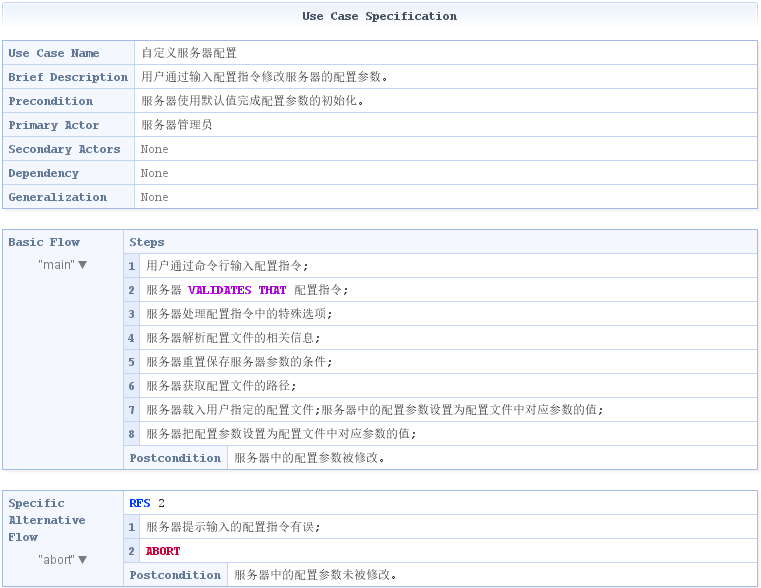


图 4 自定义服务器配置用例规格

### 3.1.3 处理事件

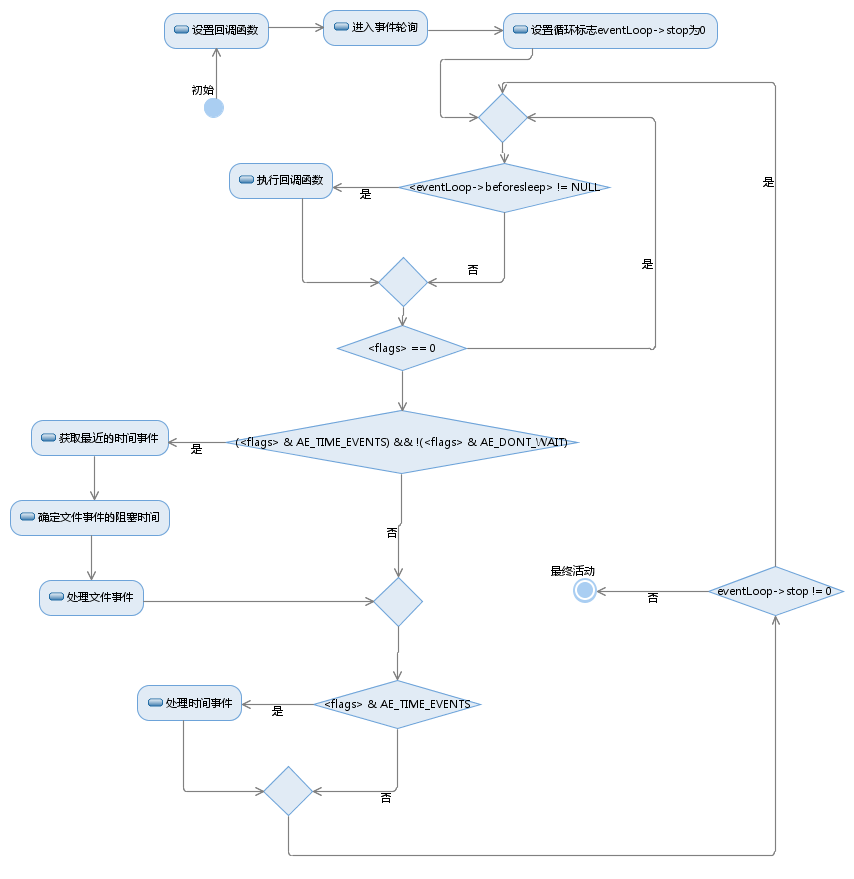


图 5 处理事件活动图

### 3.1.4 关闭服务器

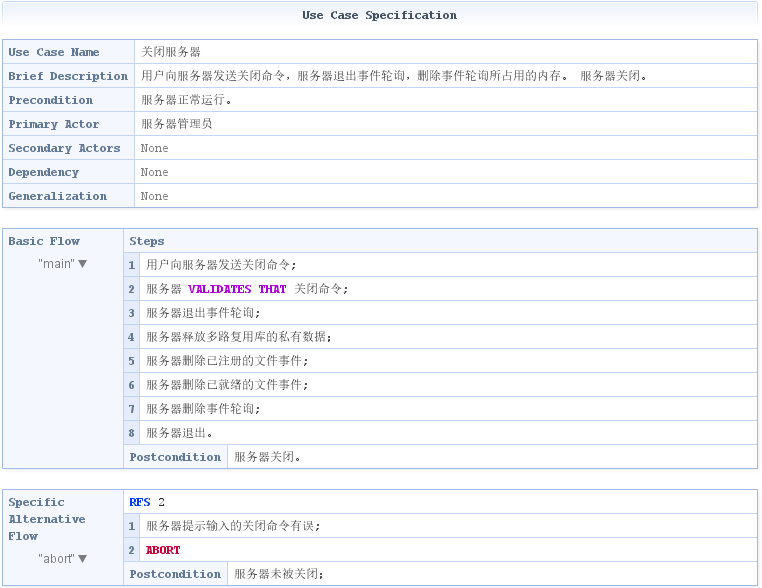


图 6 关闭服务器用例规格

## 3.2 客户端模块

Redis服务器负责与客户端建立网络连接，处理客户端发送的命令请求。客户端首先发送命令请求给服务器端，服务器读取命令请求并调用命令执行器执行命令请求，最后将请求结果返回给客户端，完成与客户端的网络连接与通信。

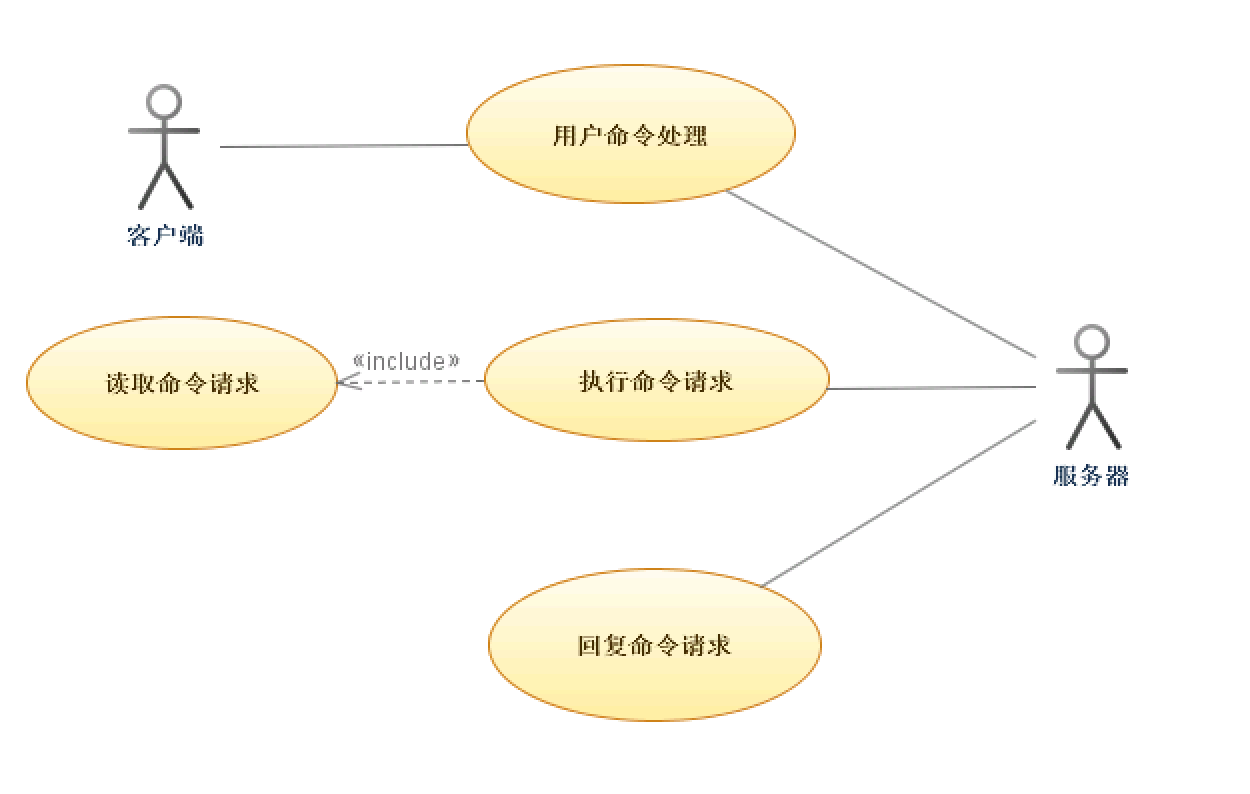


图8 客户端模块用例图

因为用户命令处理是redis客户端向服务器端发起的，所以actor选择为redis客户端，而执行命令请求是redis服务器端向命令执行器发起的，所以actor选择为redis服务器，回复命令是redis服务器向客户端发起的，所以actor选择为redis服务器。

### 3.2.1 用户命令处理处理



图 9 用户命令处理用例规格

### 3.2.2 读取命令请求



图 10 读取命令请求用例规格

### 3.2.3 执行命令请求

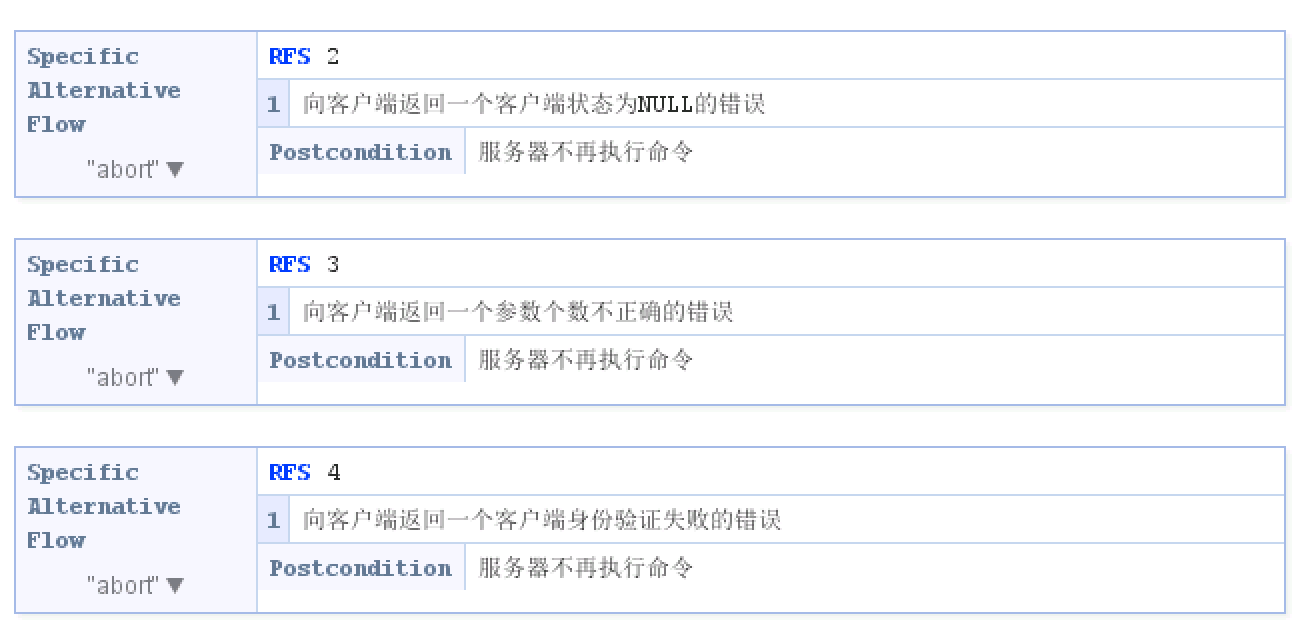


图 11 执行命令请求用例规格

### 3.2.4 回复命令



图 12 回复命令用例规格

## 3.3 RDB持久化模块

RDB持久化主要负责，在Redis 运行时，将当前内存中的数据库快照保存到磁盘文件中；在Redis 重启动时，RDB 程序可以通过载入RDB 文件来还原数据库的状态。

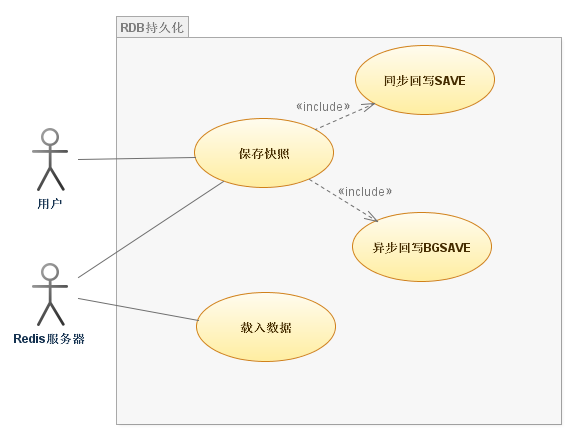


图13 RDB持久化用例图

这里的两个actor分别为用户和Redis服务器，其中用户包括系统管理员和普通用户，用户通过配置文件保存快照；Redis服务器作为保存命令的执行者，将解析配置文件，并选择相应的保存方式，并在重启时自动载入数据到内存中。

### 3.3.1 保存快照

快照是Redis默认的持久化方式。当用户在配置文件中配置保存快照命令，Redis服务器将按照配置命令，在运行过程中自动将内存中数据以快照的方式写入到二进制文件中,默认的文件名为dump.rdb。

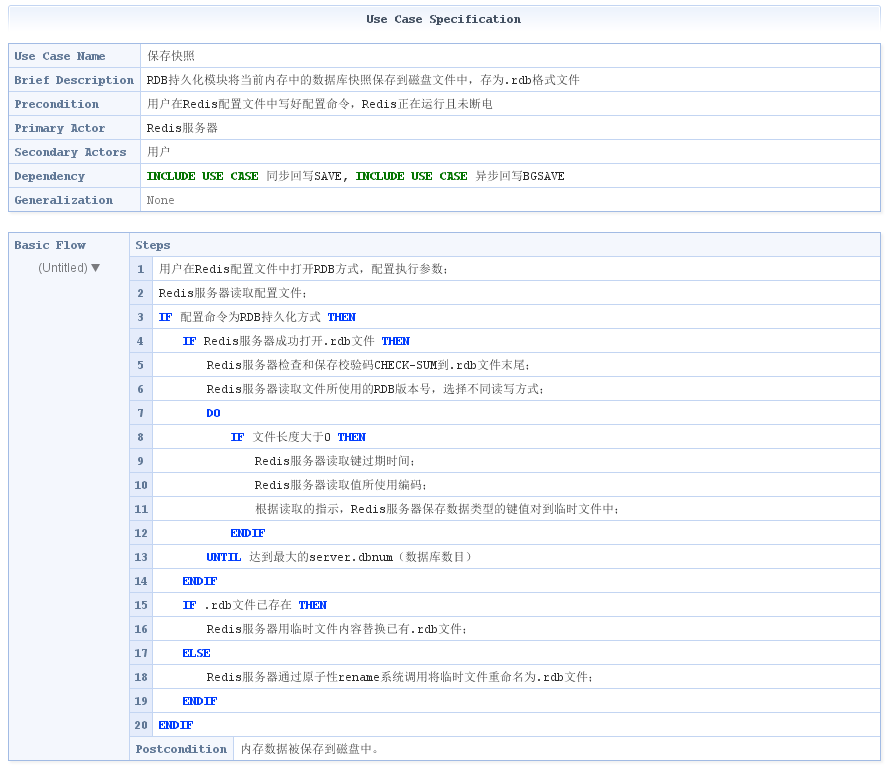


图 14 保存快照用例规格

### 3.3.2 同步回写SAVE



图 15 同步回写SAVE用例规格

### 3.3.3 异步回写BGSAVE



图 16 异步回写BGSAVE用例规格

### 3.3.4 载入数据

当Redis服务器启动时，执行载入函数rdbLoad，读取RDB文件，并将文件中的数据库数据载入到内存中。



图 17 载入数据用例规格

## 3.4 AOF持久化模块

AOF持久化模块包括命令同步、AOF文件读取和数据还原、AOF后台重写等功能。而命令同步包含命令传播、缓存追究、文件写入和保存等过程。

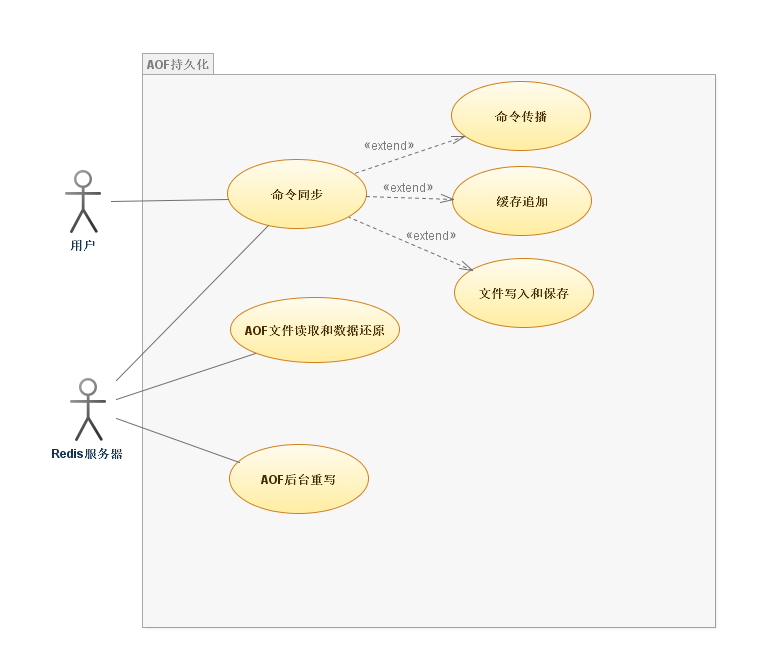


图18 AOF持久化用例图

这里的两个actor分别为用户和服务器，其中用户包括了服务器维护人员、客户端及其需要服务器提供服务的外部系统等；Redis服务器则以协议文本的方式，将所有用户对数据库进行过写入的命令（及其参数）记录到 AOF文件，以此达到AOF持久化的目的。

**3.4.1 命令同步**



图 19 命令同步用例规格

### 3.4.2 命令传播

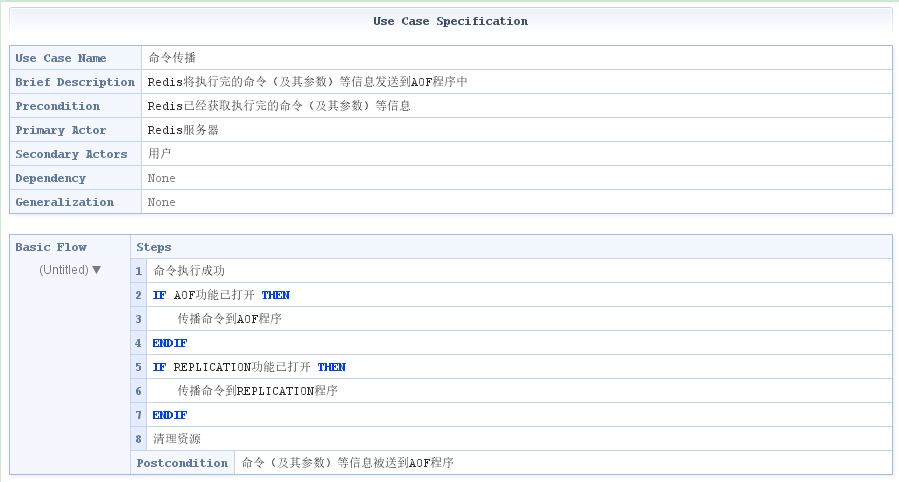


图 20 命令传播用例规格

### 3.4.3 缓存追加



图 21 缓存追加用例规格

### 3.4.4 文件写入和保存

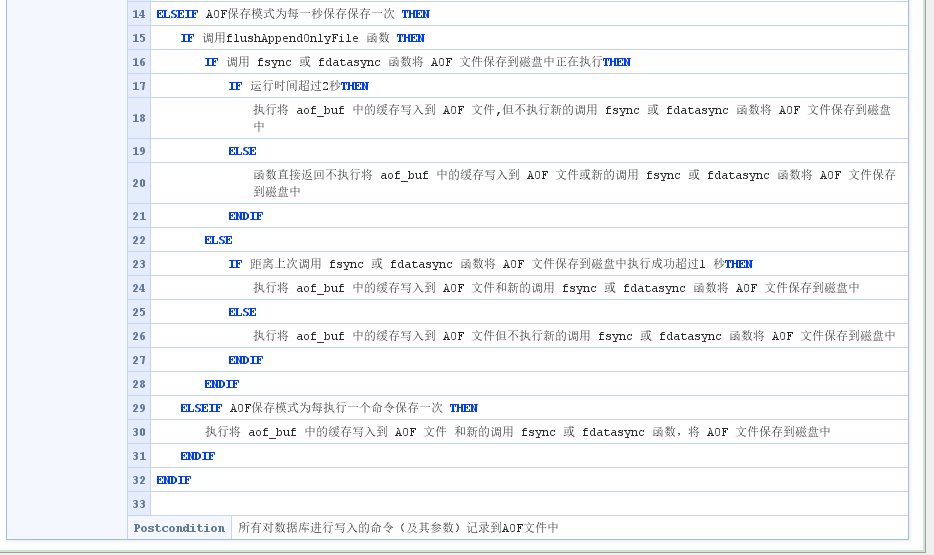


图 22 文件写入和保存用例规格

### 3.4.5 AOF文件读取和数据还原



图 23 AOF文件读取和数据还原用例规格

### 3.4.6 AOF后台重写



图 24 AOF后台重写用例规格

# 4 非功能性需求分析

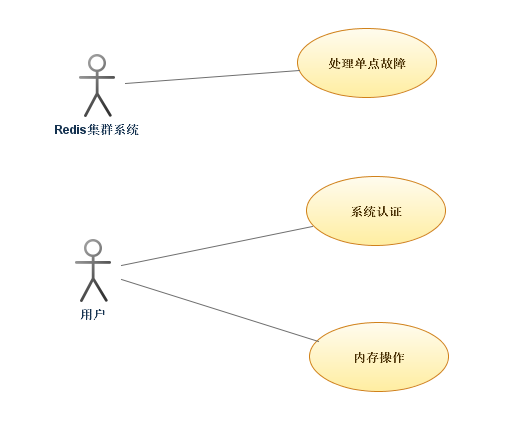


图25 非功能特性用例图

非功能特性的用例图中actor包括Redis集群系统和用户，Redis集群系统需要时刻监视整个网络并处理网络的单点故障问题，所以其为actor；用户指对Redis系统提出请求的角色，包括客户端、外界系统接口等。

## 4.1 鲁棒性

鲁棒性就是系统的健壮性，指系统或组件在面对非法输入、相连接的系统或组件故障、或非预期的运行条件下能够持续正确运行的程度。这里我们以Redis系统处理单点故障的场景为例来描述鲁棒性。

Redis系统实现了分布式并且允许单点故障，当网络故障和节点发生故障时集群系统会尽力去保证数据的一致性和有效性。

Redis系统主要使用主从同步的方式解决单点故障问题。此时，我们同时需要masters和 slaves。 即使主节点(master)和从节点(slave)在功能上是一致的，甚至说他们部署在同一台服务器上，从节点也仅用以替代故障的主节点。实际上如果对从节点没有read-after-write（写并立即读取数据以免在数据同步过程中无法获取数据）的需求，那么从节点仅接受只读操作。

总之，在Redis系统中，节点有责任/义务保存数据和自身状态，这其中包括把数据（key）映射到正确的节点，所有节点都应该自动探测系统中的其它节点，并且在发现故障节点之后把故障节点的从节点更改为主节点。规格说明如下：



图 26 处理单点故障用例规格

## 4.2 安全性

Redi系统提供了一个轻量级的认证方式来保证系统的安全性，可以编辑redis.conf配置来启用认证。

当授权方式启用时，redis系统将会拒绝来自非认证用户的任何查询。用户可以通过发送AUTH命令并带上密码给redis服务器来给自己授权。系统管理员在redis.conf文件里以明文方式设置密码。而且密码必须足够长来抵御暴力破解密码方式的攻击。认证层的目标是提供多一层的保护。如果防火墙或者用来保护redi的系统防御外部攻击失败的话，外部用户如果没有通过密码认证还是无法访问redis系统的。规格说明如下：

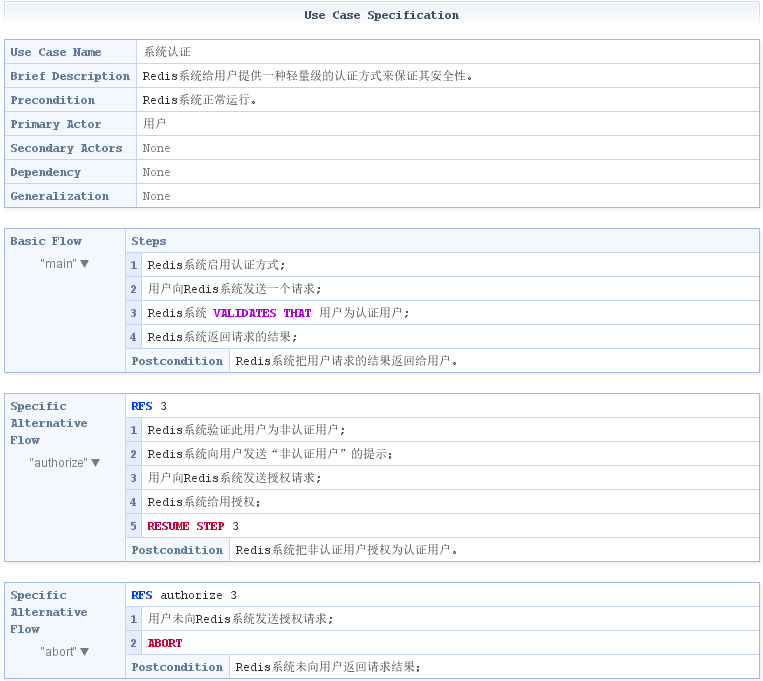


图 27 系统认证用例规格

## 4.3 高效率

Redis系统的高效率主要是因为以下三个方面，即自定义的数据结构、Redis自己实现的事件分离器和绝大部分请求是纯粹的内存操作。这里我们以Redis系统对绝大部分请求使用内存操作的场景为例来描述高效率。

和大多NoSQL数据库一样，Redis同样遵循了Key/Value数据存储模型。在有些情况下，Redis会将Keys/Values保存在内存中以提高数据查询和数据修改的效率，然而这样的做法并非总是很好的选择。鉴于此，Redis将之进一步优化，即尽量在内存中只保留Keys的数据，这样可以保证数据检索的效率，而Values数据在很少使用的时候则可以被换出到磁盘。规格说明如下：



图 28 内存操作用例规格

# 5 参考文献

[1] http://redis.io/

[2] 黄健宏. Redis设计与实现. 机械工业出版社[M]. 2014-06.

[3] 张景云. 基于Redis的矢量数据组织研究[D]. 南京师范大学. 2013.

[4] 白鑫. 基于Redis的信息存储优化技术研究与应用[D]. 北方工业大学. 2011.

[5] 曾超宇，李金香. Redis 在高速缓存系统中的应用[J]. 微型机与应用，2013， 32(12).