|  |
| --- |
| 北京航空航天大学 |
| 软件测试分析报告 |
| **Redis** |
|  |
| SY1406108 陈志伟 SY1406112 王珊珊 SY1406311 林 璐 SY1406117 王志鹏 |
| **2015/05/17** |

**版本变更历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 提交日期 | 编制人 | 说明 |
| Version 1.0 | 2015-05-17 | 陈志伟、林璐 | 初步完成测试分析报告的第一版 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 编写目的 1](#_Toc419703820)

[2 规范及对照表 1](#_Toc419703821)

[2.1 设计测试用例 1](#_Toc419703822)

[2.2 测试规格与测试用例对照表 2](#_Toc419703823)

[3 服务器模块 2](#_Toc419703824)

[3.1 启动服务器测试 2](#_Toc419703825)

[3.1.1 测试目标 2](#_Toc419703826)

[3.1.2 测试用例 2](#_Toc419703827)

[3.1.3 测试代码 3](#_Toc419703828)

[3.1.4 测试结果及分析 4](#_Toc419703829)

[3.2 自定义服务器配置正常测试 4](#_Toc419703830)

[3.2.1 测试目标 4](#_Toc419703831)

[3.2.2 测试用例 4](#_Toc419703832)

[3.2.3 测试代码 5](#_Toc419703833)

[3.2.4 测试结果及分析 6](#_Toc419703834)

[3.3 自定义服务器配置异常测试 6](#_Toc419703835)

[3.3.1 测试目标 6](#_Toc419703836)

[3.3.2 测试用例 6](#_Toc419703837)

[3.3.3 测试代码 7](#_Toc419703838)

[3.3.4 测试结果及分析 8](#_Toc419703839)

[3.4 服务器性能测试 8](#_Toc419703840)

[3.4.1 测试目标 8](#_Toc419703841)

[3.4.2 测试用例 8](#_Toc419703842)

[3.4.3 测试代码 10](#_Toc419703843)

[3.4.4 测试结果及分析 11](#_Toc419703844)

[4 RDB持久化模块 11](#_Toc419703845)

[4.1 同步回写SAVE测试 12](#_Toc419703846)

[4.1.1 测试目标 12](#_Toc419703847)

[4.1.2 测试用例 12](#_Toc419703848)

[4.1.3 测试代码 14](#_Toc419703849)

[4.1.4 测试结果及分析 19](#_Toc419703850)

[4.2 异步回写BGSAVE测试 20](#_Toc419703851)

[4.2.1 测试目标 20](#_Toc419703852)

[4.2.2 测试用例 21](#_Toc419703853)

[4.2.3 测试代码 22](#_Toc419703854)

[4.2.4 测试结果及分析 26](#_Toc419703855)

[4.3 载入数据测试 28](#_Toc419703856)

[4.3.1 测试目标 28](#_Toc419703857)

[4.3.2 测试用例 28](#_Toc419703858)

[4.3.3 测试代码 29](#_Toc419703859)

[4.3.4 测试结果及分析 29](#_Toc419703860)

[5 参考文献 30](#_Toc419703861)

# 1 编写目的

本文档主要说明测试阶段的工作内容和测试结果。首先叙述了本组对测试工作的分析过程，给出了决定撰写测试用例的需求用例与对应的测试用例的对照表，对每个测试用例按照RUCM4test的标准进行了说明，同时也给出了测试过程、测试数据和测试结果分析。

# 2 规范及对照表

## 2.1 设计测试用例

测试用例（Test Case）是将软件测试的行为活动做一个科学化的组织归纳，目的是能够将软件测试的行为转化成可管理的模式；同时测试用例也是将测试具体量化的方法之一，不同类别的软件，测试用例是不同的。不同于诸如系统、工具、控制、游戏软件，管理软件的用户需求更加不同的趋势。

要使最终用户对软件感到满意，最有力的举措就是对最终用户的期望加以明确阐述，以便对这些期望进行核实并确认其有效性。测试用例反映了要核实的需求。然而，核实这些需求可能通过不同的方式并由不同的测试员来实施。

测试用例很重要，主要有以下几个方面：

测试用例构成了设计和制定测试过程的基础。

测试的“深度”与测试用例的数量成比例。由于每个测试用例反映不同的场景、条件或经由产品的事件流，因而，随着测试用例数量的增加，我们对产品质量和测试流程也就越有信心。

判断测试是否完全的一个主要评测方法是基于需求的覆盖，而这又是以确定、实施和/或执行的测试用例的数量为依据的。类似下面这样的说明：“95 % 的关键测试用例已得以执行和验证”，远比“我们已完成 95 % 的测试”更有意义。

测试工作量与测试用例的数量成比例。根据全面且细化的测试用例，可以更准确地估计测试周期各连续阶段的时间安排。

测试设计和开发的类型以及所需的资源主要都受控于测试用例。

测试用例通常根据它们所关联关系的测试类型或测试需求来分类，而且将随类型和需求进行相应地改变。最佳方案是为每个测试需求至少编制两个测试用例，一个测试用例用于证明该需求已经满足，通常称作正面测试用例；另一个测试用例反映某个无法接受、反常或意外的条件或数据，用于论证只有在所需条件下才能够满足该需求，这个测试用例称作负面测试用例。

总之，测试用例是测试工作的指导，是软件测试的必须遵守的准则，更是软件测试质量稳定的根本保障。

## 2.2 测试规格与测试用例对照表

依据测试需求规格说明书，小组选择了各模块中具有代表性的测试规格进行了测试用例的详细设计。测试规格与测试用例的对照表如下：

|  |  |  |
| --- | --- | --- |
| 模块 | 测试规格 | 测试用例 |
| 服务器 | 启动服务器 | 启动服务器测试 |
| 自定义服务器配置 | 自定义服务器配置正常测试 |
| 自定义服务器配置异常测试 |
| 服务器性能 | 服务器性能测试 |
| RDB持久化 | 同步回写SAVE | 同步回写SAVE功能正常测试 |
| 同步回写SAVE功能阻塞测试 |
| 异步回写BGSAVE | 异步回写BGSAVE功能正常测试 |
| 异步回写BGSAVE功能异常测试 |
| 载入数据 | 载入数据测试 |

# 3 服务器模块

## 3.1 启动服务器测试

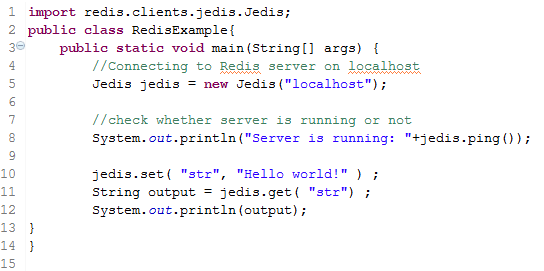
### 3.1.1 测试目标

本测试用例是对应启动服务器测试规格的，主要测试服务器正常启动时的处理方式。正常启动时，测试程序会直接输出服务器返回的ping值，并进行简单的字符串存储操作；但若测试程序抛出了异常，即可判断服务器启动异常。

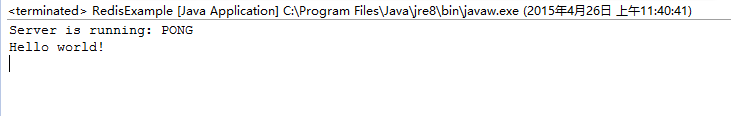
### 3.1.2 测试用例

|  |  |  |
| --- | --- | --- |
| **Test Case Specification** | | |
| Name | 启动服务器测试 | |
| Brief Description | 测试Redis服务器是否正常启动 | |
| Precondition | 系统已安装Redis数据库 | |
| Tester | 测试员 | |
| Dependency | None | |
| Test Setup | Name | 准备测试文件RedisExample.Java |
| Description | 测试员编写好用于测试的Java源文件 |
| Basic Flow  (Test Setup) | Steps | |
| 1 | 测试员在eclipse中配置好编写Redis测试程序所需的jar包； |
| 2 | 测试员编写Java测试源文件； |
| 3 | 测试员通过人工走查方式检查源文件； |
| Postcondition  (Test Oracle) | 指定目录的RedisExample.Java源文件已存在；  测试代码没有逻辑和语法错误； |
| Basic Flow  (Test Sequence) | Steps | |
| 1 | 测试员在eclipse中编译Java测试程序； |
| 2 | 测试员在eclipse中运行Java测试程序； |
| 3 | 测试程序使用指定的IP地址连接redis服务器； |
| 4 | 服务器返回ping值； |
| 5 | 测试程序VALIDATES THAT服务器正常启动； |
| 6 | 测试程序执行简单的字符串存储操作； |
| Postcondition  (Test Oracle) | 测试程序输出服务器正常运行的提示；  测试程序输出存储的字符串；  服务器正常启动的情况被测试； |
| Specific Alternative Flows  (Test Sequence) | RFS 5 | |
| 1 | 测试程序抛出服务器连接异常； |
| 2 | ABORT |
| Postcondition  (Test Sequence) | 服务器启动异常； |

### 3.1.3 测试代码



### 3.1.4 测试结果及分析



**预期结果：**

输出服务器正在运行的提醒和ping值，输出从服务器取回的字符串。

**执行结果分析：**

测试程序没有抛出异常，而是显示服务器正在运行，并输出服务器返回的ping值，同时测试程序还向服务器存储了一个简单的字符串，再用查询操作取得了这个字符串并进行了正确的输出。这些证明了服务器正常启动且正在运行，与预期结果一致，故测试通过。

## 3.2 自定义服务器配置正常测试

### 3.2.1 测试目标

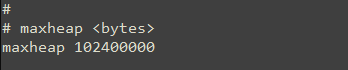
本测试用例是对应自定义服务器配置测试规格的，主要测试在自定义配置参数合理的情况下，即测试需求能满足时，服务器启动时的处理方式。在启动时，服务器读取配置文件中的参数设置自身参数，并显示开始界面，即可判断自定义的服务器配置文件正常。

### 3.2.2 测试用例

|  |  |  |
| --- | --- | --- |
| **Test Case Specification** | | |
| Name | 自定义服务器配置正常测试 | |
| Brief Description | 测试在用户定义的配置正常的情况下对服务器的影响 | |
| Precondition | 系统已安装Redis数据库 | |
| Tester | 测试员 | |
| Dependency | None | |
| Test Setup | Name | 准备自定义的配置文件 |
| Description | 测试员编写好用于测试的自定义配置文件 |
| Basic Flow  (Test Setup) | Steps | |
| 1 | 测试员编写Redis的配置文件； |
| 2 | 测试员给配置设置合理的参数； |
| 3 | 测试员将配置文件保存到指定目录； |
| 4 | 测试员通过人工走查方式检查配置文件； |
| Postcondition  (Test Oracle) | 指定目录的自定义配置文件已存在；  配置文件符合Redis的配置文件相关规范； |
| Basic Flow  (Test Sequence) | Steps | |
| 1 | 测试员在命令行中输入服务器可执行程序名和自定义的配置文件名； |
| 2 | 服务器读入配置文件中的相关配置参数的值； |
| 3 | 服务器VALIDATES THAT配置文件中的参数有效； |
| 4 | 服务器按配置文件中的参数的值设置自身的参数； |
| Postcondition  (Test Oracle) | 服务器开始运行，在命令行中输出启动界面；  配置文件中的参数有效的情况被测试； |

### 3.2.3 测试代码

* 自定义配置文件（显示将改变的参数）

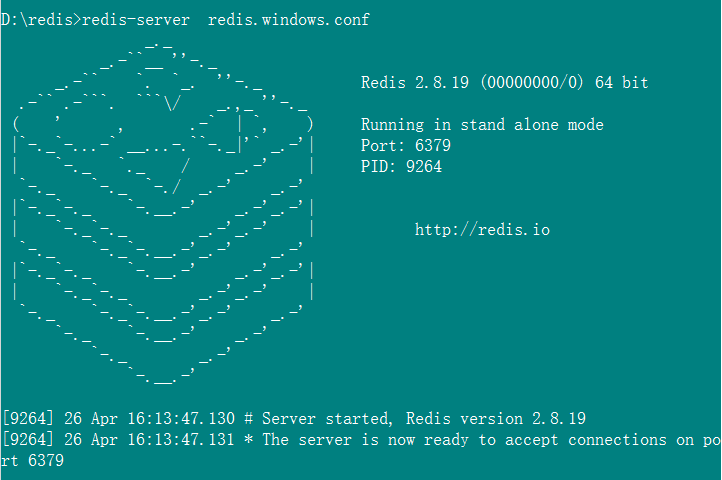


这是自定义配置文件的关键部分的截图。maxheap表示堆栈的最大内存，在自定义的配置文件中，将修改maxheap的值，这里取了一个较小的102,400,000byte。

* 测试指令



### 3.2.4 测试结果及分析



**预期结果：**

在配置参数设置合理的情况下，使用自定义的配置文件能让服务器正常启动运行。

**执行结果分析：**

由上图中的结果可知，服务器已经正常启动运行，等待着客户端的连接，所以测试通过。

## 3.3 自定义服务器配置异常测试

### 3.3.1 测试目标

本测试用例也是对应自定义服务器配置测试规格的，主要测试在自定义配置参数异常的情况下，即测试需求不能满足时，服务器启动时的处理方式。在启动时，服务器读取配置文件中的参数设置自身参数，由于测试设置异常导致服务器没法启动运行，故服务器退出给出出错的信息， 即可测试配置异常的情况。

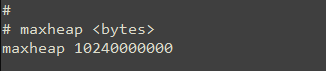
### 3.3.2 测试用例

|  |  |  |
| --- | --- | --- |
| **Test Case Specification** | | |
| Name | 自定义服务器配置异常测试 | |
| Brief Description | 测试在用户定义的配置异常的情况下对服务器的影响 | |
| Precondition | 系统已安装Redis数据库 | |
| Tester | None | |
| Dependency | None | |
| Test Setup | Name | 准备自定义的配置文件 |
| Description | 测试员编写好用于测试的自定义配置文件 |
| Basic Flow  (Test Setup) | Steps | |
| 1 | 测试员编写Redis的配置文件； |
|  | 测试员给配置设置异常的参数； |
| 2 | 测试员将配置文件保存到指定目录； |
| 3 | 测试员通过人工走查方式检查配置文件； |
| Postcondition  (Test Oracle) | 指定目录的自定义配置文件已存在；  配置文件符合Redis的配置文件相关规范； |
| Basic Flow  (Test Sequence) | Steps | |
| 1 | 测试员在命令行中输入服务器可执行程序名和自定义的配置文件名； |
| 2 | 服务器读入配置文件中的相关配置参数的值； |
| 3 | 服务器VALIDATES THAT配置文件中的参数无效； |
| 4 | 服务器启动失败并给出配置文件中参数设置错误； |
| 5 | ABORT |
| Postcondition  (Test Oracle) | 配置文件中参数设置异常的情况被测试； |

### 3.3.3 测试代码

（1）测试代码

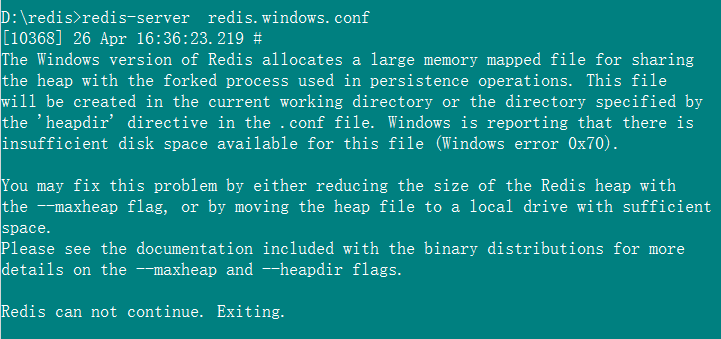
* 自定义配置文件（显示改变的参数）



* 测试指令



### 3.3.4 测试结果及分析



**预期结果：**

让服务器无法启动运行并退出，给出异常的原因。

**执行结果分析：**

由上图中的结果可知，服务器启动运行失败。这是因为把配置文件中的maxheap参数设置为一个较大的值，即10,240,000,000bytes，由于堆栈的最大内存分配过大，导致磁盘没有足够的容纳空间，所以服务器启动失败，故测试通过。

## 3.4 服务器性能测试

### 3.4.1 测试目标

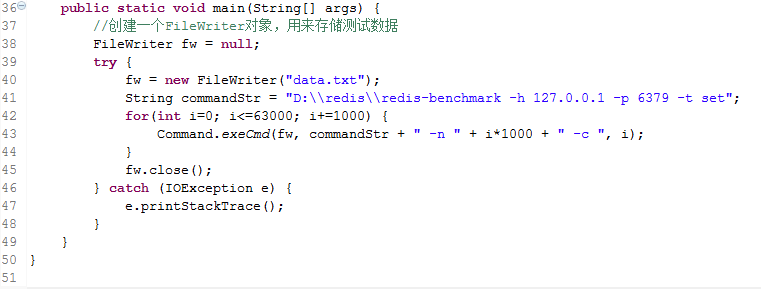
本测试用例是对应服务器性能测试规格的，主要测试在每个客户端对服务器请求数量一定的情况下，研究不同的数量的客户端，即并发数对服务器性能的影响。通过把每个客户端的请求数量设置为恒定的1000次，然后按一定的步长改变并发数，即增加客户端的数量，分别得到完成所有请求所用的时间，就可推算出对服务器性能的影响，从而完成测试过程。

### 3.4.2 测试用例

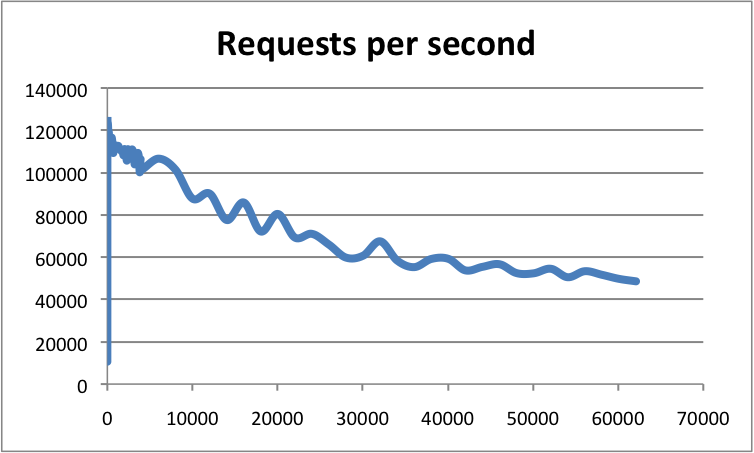
|  |  |  |
| --- | --- | --- |
| **Test Case Specification** | | |
| Name | 服务器性能测试 | |
| Brief Description | 模拟测试不同的并发数对服务器的性能影响 | |
| Precondition | 服务器已正常启动 | |
| Tester | 测试员 | |
| Dependency | None | |
| Test Setup | Name | 准备测试文件Command.Java |
| Description | 测试员编写好用于测试的Java源文件 |
| Basic Flow  (Test Setup) | Steps | |
| 1 | 测试员编写java测试源文件； |
| 2 | 测试员将源文件保存到指定目录； |
| 3 | 测试员通过人工走查方式检查源文件； |
| Postcondition  (Test Oracle) | 指定目录的Java源文件已存在；  Java代码没有逻辑和语法错误； |
| Basic Flow  (Test Sequence) | Steps | |
| 1 | 测试员设置测试的并发数N的最小值为0，最大值为63000，步进为1000； |
| 2 | 测试程序初始化并发数N为0； |
| 3 | DO |
| 4 | 测试程序调用性能测试工具向服务器以并发数N发送恒定数量的请求； |
| 5 | 测试工具给出测试结果； |
| 6 | 测试程序记录测试数据； |
| 7 | 测试程序把并发数N更改为N + 1000； |
| 8 | UNTIL N > 63000 |
| 9 | 测试员对测试程序所返回的测试数据进行分析； |
| Postcondition  (Test Oracle) | 在请求数量一定的情况下，服务器在不同的并发数下的性能得到测试 |

### 3.4.3 测试代码





### 3.4.4 测试结果及分析



**预期结果：**

服务器的性能曲线应该先上升，然后到达峰值后开始下降。

**执行结果分析：**

由上图可知，在并发数较小时服务器的性能上升得非常快，这主要是因为开始时客户端的个数较小，而每个客户端发送的请求数一定，故总的请求数也较小，此时服务器无法准确的估计出每秒处理请求数量的值。而到了客户端数量为100～150这个范围时，服务器每秒处理请求的数量达到峰值。最后，当客户端数量达到60000时，服务此时的性能只有峰值性能的一半，所以60000为Redis系统的阈值。在实际中，Redis每秒处理的请求数都是上万的，此时服务器一定不是运行于性能的峰值，但，虽然性能有所损害，每秒能服务的客户端却变得更多，这能显著的节省成本，性能和并发数的折中在实际中非常有意义，总之测试通过。

# 4 RDB持久化模块

Redis提供了两种持久化策略，RDB和AOF。RDB是默认的，它定时创建数据库的完整磁盘镜像，即dump.rdb文件。创建镜像的时间间隔是可以设置的，假如每5分钟创建一次镜像，那么当系统崩溃时用户可能会丢失5分钟的数据。因此，**RDB不是一个可靠性很高的方案，但是性能不错**。RDB非常容易备份，用户直接将dump.rdb文件复制即可。为了提供更好的可靠性，Redis支持AOF，即将操作写入日志中（appendonly.aof文件）。写日志的策略可以是每秒一次或每次操作一次，显然每秒一次意味着用户可能丢失1秒的数据，而**每次AOF操作的可靠性最高，但是性能最差**。日志文件可能会增长到非常大，因此Redis后台会执行rewrite操作整理日志。AOF不适合备份。

**Redis推荐使用RDB，以及在需要可靠性的时候用RDB+AOF，不推荐单独使用AOF。**Redis为了减少磁盘的负载，任何时刻都不会同时执行写镜像和写日志。

第四部分主要对RDB策略进行测试，第五部分主要对AOF持久化策略进行测试。

## 4.1 同步回写SAVE测试

### 4.1.1 测试目标

同步回写SAVE在Redis主进程中直接调用rdbSave()函数，阻塞主进程，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任何请求。

**一般来说，在生产环境中很少用SAVE操作**，因为会阻塞所有客户端的请求，保存数据库的任务通常由BGSAVE命令异步保存。然而，如果负责保存数据的后台子进程不幸出现问题时，SAVE可以作为保存数据的最后手段来使用。

本部分测试在window环境下：

1. 当Redis开启rdb方式，数据变更后，手动进行SAVE保存的功能。
2. 验证其主进程被阻塞时，其他客户端发送请求，服务器报出异常的功能。

### 测试用例

#### 4.1.2.1 同步回写SAVE功能正常测试

|  |  |  |
| --- | --- | --- |
| **Test Case Specification** | | |
| Name | 同步回写SAVE功能正常测试 | |
| Brief Description | 测试能否正确读取用户配置命令，服务器进行同步回写SAVE操作 | |
| Precondition | Redis在windows下正确安装 | |
| Tester | 测试员 | |
| Dependency | None | |
| Test Setup | Name | 配置redis.conf文件 |
| Description | 配置redis.conf文件，以选择RDB持久化方式，并调用SAVE命令 |
| Basic Flow  (Test Setup) | Steps | |
| 1 | 测试员打开redis.conf文件 |
| 2 | 测试员写入命令”**save “”** ”，不设置快照保存周期，而手动进行保存 |
| 3 | 写入命令”dbfilename dump.rdb”，设置快照保存文件的文件名 |
| 4 | 写入命令”dir ./”，设置备份文件放置路径 |
| 5 | 写入命令”appendonly no”，选择非AOF方式，也即RDB方式 |
| Postcondition  (Test Oracle) | 配置完成，准备启动Redis进行快照SAVE测试 |
| Basic Flow  (Test Sequence) | Steps | |
| 1 | 打开windows下的cmd运行窗口 |
| 2 | 输入”cd PATH-TO-REDIS”，切换到Redis文件夹下 |
| 3 | 输入”redis-server.exe redis.conf”，启动Redis |
| 4 | 输入”redis-cli.exe”，打开一个窗口运行客户端A |
| 5 | 创建Eclipse项目，引入jedis客户端包 |
| 6 | 编写测试程序A-test，实现对100条数据的修改 |
| 7 | 客户端A中输入”save”，手动选择save方式保存快照 |
| 7 | 客户端A输入命令”get xxx”，获取被修改键的最新值 |
| Postcondition  (Test Oracle) | 最新的保存值恰是测试程序逻辑所得，手动同步保存SAVE的功能被测试 |

#### 4.1.2.2 同步回写SAVE功能阻塞测试

|  |  |  |
| --- | --- | --- |
| **Test Case Specification** | | |
| Name | 同步回写SAVE功能阻塞测试 | |
| Brief Description | 测试进行同步回写操作时，主进程被阻塞，其他客户端发出请求时，服务器能否正常报错 | |
| Precondition | Redis在windows下可正常进行RDB持久化 | |
| Tester | 测试员 | |
| Dependency | None | |
| Test Setup | Name | 配置redis.conf文件 |
| Description | 配置redis.conf文件，以选择RDB持久化方式，并调用SAVE命令 |
| Basic Flow  (Test Setup) | Steps | |
| 1 | 测试员打开redis.conf文件 |
| 2 | 测试员写入命令”**save “”** ”， 不设置快照保存周期，而手动进行保存 |
| 3 | 写入命令”dbfilename dump.rdb”，设置快照保存文件的文件名 |
| 4 | 写入命令”dir ./”，设置备份文件放置路径 |
| 5 | 写入命令”appendonly no”，选择非AOF方式，也即RDB方式 |
| Postcondition  (Test Oracle) | 配置完成，准备启动Redis进行快照SAVE测试 |
| Basic Flow  (Test Sequence) | Steps | |
| 1 | 打开windows下的cmd运行窗口 |
| 2 | 输入”cd PATH-TO-REDIS”，切换到Redis文件夹下 |
| 3 | 输入”redis-server.exe redis.conf”，启动Redis |
| 4 | 创建Eclipse项目，引入jedis客户端包 |
| 5 | 编写测试程序A-test，实现对1000000条数据的修改 |
| 6 | 编写测试程序B-test，实现同时调用save命令和set命令，以观察阻塞现象 |
| Postcondition  (Test Oracle) | 观察客户端接收的错误信息，服务器在save阻塞过程中拒绝客户端其他请求的报错功能被测试 |

### 测试代码

1. 同步回写SAVE功能正常测试
   1. 创建Jedis类，连接redis服务器，实现数据的读写方法：

**package** SE.lin;

**import** redis.clients.jedis.Jedis;

**import** redis.clients.jedis.JedisPool;

**import** redis.clients.jedis.JedisPoolConfig;

**public** **class** RDBtester {

JedisPool pool;

Jedis jedis;

**public** **void** etup() {

pool = **new** JedisPool(**new** JedisPoolConfig(), “127.0.0.1”);

jedis = pool.getResource();

// jedis.auth(“password”);

}

**public** **void** testWriteData(String key, String value) {

// ---添加数据---

// jedis.set(“name1”, “value”);

// System.out.println(jedis.get(“name1”));

// ---修改数据---

// 1.在原来基础上追加

// jedis.append(“name1”, “b1”);

// System.out.println(jedis.get(“name1”));

// 2.直接覆盖

jedis.set(key, value);

// System.out.println(jedis.get(“name1”));

// 删除key对应记录

// jedis.del(“name1”);

// System.out.println(jedis.get(“name1”));

}

**public** String testReadData(String key) {

**return** jedis.get(key);

}

**public** **static** **void** main(String[] args) {

RDBtester tester1 = **new** RDBtester();

**int** I = 0;

**while** (I < 100) {

I = I + 1;

tester1.setUp();

tester1.testWriteData(“name” + I, “test” + i);

}

}

}

* 1. 实例化读数据线程，实现生成100个键值对：

**package** SE.lin;

**import** java.io.IOException;

**public** **class** BlockTester {

**int** changeID = 0;

**public** **static** **void** main(String args[]) {

System.*out*.println(“\*\*\*SAVE测试开始\*\*\*”);

// 创建thread1,写数据

ThreadUseExtends thread1 = **new** ThreadUseExtends(0, 10);

Thread1start();

System.*out*.println(“主线程将挂起1秒”);

**try** {

Thread.*sleep*(10000);

} **catch** (InterruptedException e) {

**return**;

}

System.*out*.println(“回到主线程”);

**if** (thread1.isAlive()) {

// 如果thread1还存在则杀掉他

thread1.~~stop~~();

System.*out*.println(“thread1杀掉，写数据完成”);

} **else** {

System.*out*.println(“主线程没发现thread1，则它已顺利结束”);

}

}// main

}// BlockTester

**class** ThreadUseExtends **extends** Thread {

**int** changeID;

**int** step;

ThreadUseExtends(**int** etupd, **int** stp) {

**this**.changeID = etupd;

**this**.step = stp;

} // 构造函数

**public** **void** run() {

System.*out*.println(“\t thread1:我是写数据线程”);

System.*out*.println(“\t thread1:我将不断写数据”);

RDBtester tester1 = **new** RDBtester();

**while** (changeID < 100) {

changeID = changeID + 1;

tester1.setUp();

tester1.testWriteData(“name” + changeID, “test” + changeID);

**try** {

*sleep*(step);

} **catch** (InterruptedException e) {

**return**;

}

}

}

}

1. 同步回写SAVE功能阻塞测试

一个线程调用save命令，一个线程循环调用set命令，二者同时进行10秒，则若save命令执行时间合适，set命令循环调用过程中很可能被阻塞，此时可观察服务器报错信息。

**Package** SE.lin;

**import** java.io.IOException;

**import** redis.clients.jedis.exceptions.JedisException;

**public** **class** SaveBlocker {

**public** **static** **void** main(String args[]) {

System.*out*.println(“\*\*\*测试开始\*\*\*”);

// 创建thread1,写数据

SaveThread thread1 = **new** SaveThread(0);

// 创建Thread2，读数据

SaveThread thread2 = **new** SaveThread(1);

// Thread thread2 = new Thread(new ThreadUseRunnable(), “SecondThread”);

// 同时启动

thread2.start();

thread1.start();

System.*out*.println(“主线程将挂起10秒”);

**try** {

Thread.*sleep*(10000);

} **catch** (InterruptedException e) {

**return**;

}

System.*out*.println(“回到主线程”);

**if** (thread1.isAlive()) {

// 如果thread1还存在则杀掉他

thread1.~~stop~~();

System.*out*.println(“thread1杀掉，写数据完成”);

} **else** {

System.*out*.println(“主线程没发现thread1，则它已顺利结束”);

}

**if** (thread2.isAlive()) {

thread2.~~stop~~();// 如果thread2还存在则杀掉他

System.*out*.println(“thread2杀掉，读数据完成”);

} **else**

System.*out*.println(“主线程没发现thread2,则它已顺利结束!”);

System.*out*.println(“程序结束按任意键继续!”);

**try** {

System.*in*.read();

} **catch** (IOException e) {

System.*out*.println(e.toString());

}

}

}

/\*\*\*

\* 不断写数据，以触发SAVE同步回写

\*

\* **@author** linlu

\*

\*/

**class** SaveThread **extends** Thread {

**int** func;

RDBtester tester1 = **new** RDBtester();

SaveThread(**int** funcid) {

**this**.func = funcid;

**this**.tester1.setUp();

} // 构造函数

**public** **void** run() {

**int** I = 0;

**if** (**this**.func == 0) {

System.*out*.println(“\t thread1:我是save线程”);

tester1.saveData();

} **else** {

System.*out*.println(“\t thread1:我是发送请求线程”);

**while** (**true**) {

I = I + 1;

**try** {

tester1.testWriteData(“name1”, “blocktest” + i);

} **catch** (JedisException e) {

System.*out*.println(e.getMessage());

}

}

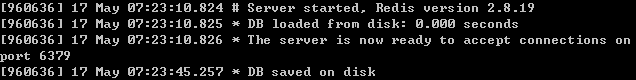
}

}

### 测试结果及分析

1. 同步回写SAVE功能正常测试

jedis测试程序修改100条键值对，在客户端手动输入SAVE命令，服务器保存成功信息如图所示：



客户端得到成功反馈：



开启客户端，随机读取添加的键，正是新修改的值，如图所示:



由以上结果可知，同步回写SAVE方式保存的功能已成功实现。

1. 同步回写SAVE功能阻塞测试

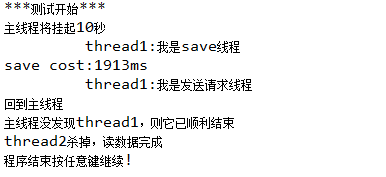
为考察save的执行时间，以便选择合适的快照数据大小，首先进行save 的性能测试：

由图可知，2秒左右时间，对于测试命令的并发比较合适，因此我们选择40M左右数据进行修改和保存。

当save命令执行时，由于主进程被阻塞，服务器将对客户端的其他请求报错，因此当调用”set name1 blocktestID”时，客户端收到错误信息：



当首先调用save命令的线程，并使发送其他请求（set命令）的线程休眠5秒使save命令先执行完，此时服务器可正常处理其他请求：





## 4.2 异步回写BGSAVE测试

### 4.2.1 测试目标

异步回写BGSAVE与SAVE的不同点在于，主进程会fork出一个子进程，子进程负责调用rdbSave()函数，并在保存完成后向主进程发送信号，通知保存已完成。因为rdbSave()执行期间，是被子进程调用的，所以Redis服务器在BGSAVE期间仍可以继续处理客户端的请求。

本部分测试在window环境下:

1. 当Redis开启rdb方式，配置每m秒发生n次变更进行rdb文件保存；
2. 验证其主进程被阻塞时，服务器仍能接受客户端命令，但是拒绝处理同时调用的SAVE或BGSAVE命令以避免竞争。

### 4.2.2 测试用例

#### 4.2.2.1 异步回写BGSAVE功能正常测试

|  |  |  |
| --- | --- | --- |
| **Test Case Specification** | | |
| Name | 异步回写BGSAVE功能正常测试 | |
| Brief Description | 测试能否正确读取用户配置命令，服务器进行异步回写BGSAVE操作 | |
| Precondition | Redis在windows下正确安装 | |
| Tester | 测试员 | |
| Dependency | None | |
| Test Setup | Name | 配置redis.conf文件 |
| Description | 配置redis.conf文件，以选择RDB持久化方式，并调用SAVE命令 |
| Basic Flow  (Test Setup) | Steps | |
| 1 | 测试员打开redis.conf文件 |
| 2 | 测试员写入命令”**save 2 1**”，设置快照保存的策略为每2秒发生1次变更则进行保存，默认为BGSAVE方式 |
| 3 | 写入命令”dbfilename dump.rdb”，设置快照保存文件的文件名 |
| 4 | 写入命令”dir ./”，设置备份文件放置路径 |
| 5 | 写入命令”appendonly no”，选择非AOF方式，也即RDB方式 |
| Postcondition  (Test Oracle) | 配置完成，准备启动Redis进行快照BGSAVE测试 |
| Basic Flow  (Test Sequence) | Steps | |
| 1 | 打开windows下的cmd运行窗口 |
| 2 | 输入”cd PATH-TO-REDIS”，切换到Redis文件夹下 |
| 3 | 输入”redis-server.exe redis.conf”，启动Redis |
| 4 | 输入”redis-cli.exe”，打开一个窗口运行客户端A |
| 5 | 创建Eclipse项目，引入jedis客户端包 |
| 6 | 编写测试程序A-test，实现每秒修改一次数据，运行30秒 |
| 7 | 测试程序A-test运行完成，在客户端A输入命令”get xxx”，获取被修改键的最新值 |
| Postcondition  (Test Oracle) | 最新的保存值恰是测试程序逻辑所得，异步保存BGSAVE的每2秒发生1次变更则保存的功能被测试 |

#### 4.1.2.2 异步回写BGSAVE功能异常测试

|  |  |  |
| --- | --- | --- |
| **Test Case Specification** | | |
| Name | 异步回写BGSAVE功能异常测试 | |
| Brief Description | 测试进行异步回写操作时，其他客户端发出请求，服务器能否正常处理 | |
| Precondition | Redis在windows下可正常进行RDB持久化 | |
| Tester | 测试员 | |
| Dependency | None | |
| Test Setup | Name | 配置redis.conf文件 |
| Description | 配置redis.conf文件，以选择RDB持久化方式，并调用BGSAVE命令 |
| Basic Flow  (Test Setup) | Steps | |
| 1 | 测试员打开redis.conf文件 |
| 2 | 测试员写入命令”**save “”** ”，设置手动快照保存 |
| 3 | 写入命令”dbfilename dump.rdb”，设置快照保存文件的文件名 |
| 4 | 写入命令”dir ./”，设置备份文件放置路径 |
| 5 | 写入命令”appendonly no”，选择非AOF方式，也即RDB方式 |
| Postcondition  (Test Oracle) | 配置完成，准备启动Redis进行快照BGSAVE测试 |
| Basic Flow  (Test Sequence) | Steps | |
| 1 | 打开windows下的cmd运行窗口 |
| 2 | 输入”cd PATH-TO-REDIS”，切换到Redis文件夹下 |
| 3 | 输入”redis-server.exe redis.conf”，启动Redis |
| 4 | 创建Eclipse项目，引入jedis客户端包 |
| 5 | 编写测试程序A-test，实现10000000L数据的修改（大概40M） |
| 6 | 编写测试程序B-test，同时运行两个线程，分别调用save命令和其他命令 |
| 7 | 编写测试程序C-test，同时运行两个线程，分别调用两个save命令 |
| Postcondition  (Test Oracle) | 测试程序B-test正常执行，而C-test报错，异步回写BGSAVE过程中服务器处理其他客户端请求的功能被测试 |

### 4.2.3 测试代码

1. 异步回写BGSAVE功能正常测试
   1. 创建Jedis类，连接redis服务器，实现数据的读写方法：

代码与4.1.3(1)(a)相同。

* 1. 实例化读数据线程，实现每秒修改一次数据；主线程中运行30秒：

**package** SE.lin;

**import** java.io.IOException;

**public** **class** BlockTester {

**public** **static** **void** main(String args[]) {

// 创建thread1,写数据

ThreadUseExtends thread1 = **new** ThreadUseExtends();

thread1.start();

System.*out*.println("主线程将挂起30秒");

**try** {

Thread.*sleep*(30000);

} **catch** (InterruptedException e) {

**return**;

}

System.*out*.println("回到主线程");

**if** (thread1.isAlive()) {

// 如果thread1还存在则杀掉他

thread1.~~stop~~();

System.*out*.println("thread1杀掉，写数据完成");

} **else** {

System.*out*.println("主线程没发现thread1，则它已顺利结束");

}

}// main

}// BlockTester

/\*\*\*

\* 不断写数据，以触发快照保存

\*

\* **@author** linlu

\*

\*/

**class** ThreadUseExtends **extends** Thread {

ThreadUseExtends() {

} // 构造函数

**public** **void** run() {

System.*out*.println("\t thread1:我是写数据线程");

System.*out*.println("\t thread1:我将不断写数据");

RDBtester tester1 = **new** RDBtester();

**int** i = 0;

**while** (**true**) {

// 每1秒写一次数据

i = i + 1;

tester1.setUp();

tester1.testWriteData("test" + i);

**try** {

*sleep*(1000);

} **catch** (InterruptedException e) {

**return**;

}

}

}

}

1. 异步回写BGSAVE功能异常测试

**package** SE.lin;

**import** java.io.IOException;

**import** redis.clients.jedis.exceptions.JedisException;

**public** **class** SaveBlocker {

**public** **static** **void** main(String args[]) {

System.*out*.println("\*\*\*测试开始\*\*\*");

// 创建thread1,写数据

SaveThread thread1 = **new** SaveThread(0);

// 创建Thread2，读数据

SaveThread thread2 = **new** SaveThread(0);

// Thread thread2 = new Thread(new ThreadUseRunnable(), "SecondThread");

// 同时启动

thread2.start();

thread1.start();

System.*out*.println("主线程将挂起10秒");

**try** {

Thread.*sleep*(10000);

} **catch** (InterruptedException e) {

**return**;

}

System.*out*.println("回到主线程");

**if** (thread1.isAlive()) {

// 如果thread1还存在则杀掉他

thread1.~~stop~~();

System.*out*.println("thread1杀掉，保存数据完成");

} **else** {

System.*out*.println("主线程没发现thread1，则它已顺利结束");

}

**if** (thread2.isAlive()) {

thread2.~~stop~~();// 如果thread2还存在则杀掉他

System.*out*.println("thread2杀掉，保存数据完成");

} **else**

System.*out*.println("主线程没发现thread2,则它已顺利结束!");

System.*out*.println("程序结束按任意键继续!");

**try** {

System.*in*.read();

} **catch** (IOException e) {

System.*out*.println(e.toString());

}

}

}

/\*\*\*

\* 向服务器发送save、或set请求

\*

\* **@author** linlu

\*

\*/

**class** SaveThread **extends** Thread {

**int** func;

RDBtester tester1 = **new** RDBtester();

SaveThread(**int** funcid) {

**this**.func = funcid;

**this**.tester1.setUp();

} // 构造函数

**public** **void** run() {

**int** i = 0;

**int** flag = 0;

**if** (**this**.func == 0) {

System.*out*.println("\t thread1:我是save线程");

tester1.saveData();

} **else** {

// try {

// sleep(5000);

// } catch (InterruptedException e1) {

// // **TODO** Auto-generated catch block

// e1.printStackTrace();

// }

System.*out*.println("\t thread1:我是发送请求线程");

System.*out*.println("\t \*服务器执行set命令之前：name1的值为"

+ tester1.testReadData("name1"));

**while** (**true**) {

i = i + 1;

**try** {

tester1.testWriteData("name1", "blocktest" + i);

**if** (flag == 0) {

System.*out*.println("\t \*服务器执行set命令之后：name1的值为"

+ tester1.testReadData("name1"));

flag = 1;

}

} **catch** (JedisException e) {

System.*out*.println(e.getMessage());

}

}

}

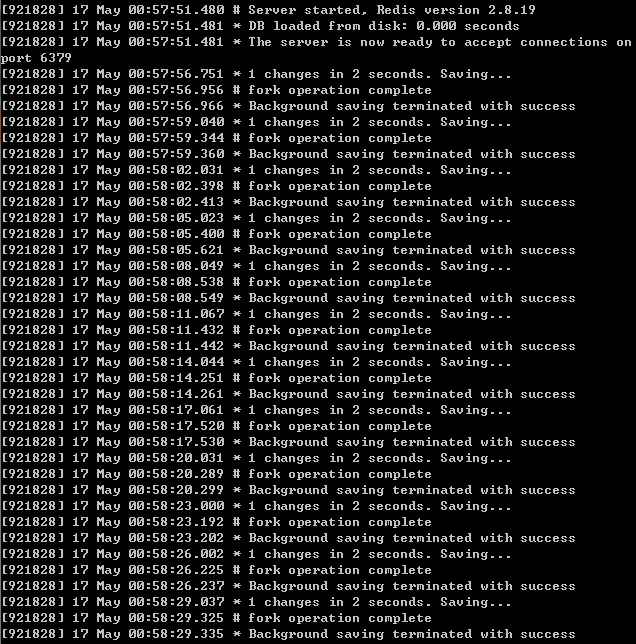
}

}

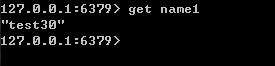
### 4.2.4 测试结果及分析

1. 异步回写BGSAVE功能正常测试

在jedis测试程序一共运行30秒，每秒修改一次数据，redis服务器端每两秒检查符合同步回写条件，则进行同步回写，服务器保存成功信息如图所示：



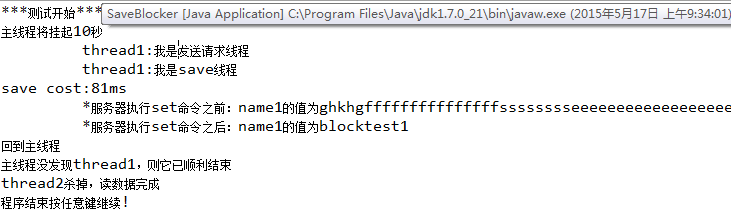
开启客户端，读取被修改的键”name1”，正是最新修改的值，如图所示:



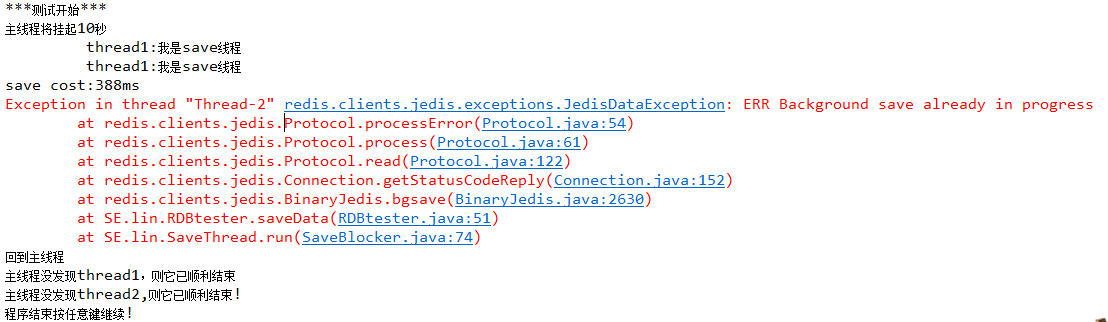
由以上结果可知，同步回写SAVE方式，可实现每m秒修改n次保存的功能。

1. 异步回写BGSAVE功能异常测试

对于非save和bgsave的普通客户端请求（如set命令），服务器能够正常响应：



对于其他save命令，服务器报错：



## 4.3 载入数据测试

### 4.3.1 测试目标

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

本部分测试当修改数据键值，并手动调用save命令将更改备份后，服务器关闭又重启后，是否能正确载入备份的数据到内存。

### 4.3.2 测试用例

|  |  |  |
| --- | --- | --- |
| **Test Case Specification** | | |
| Name | 载入数据测试 | |
| Brief Description | 测试Redis服务器是否能正确从.rdb备份文件中恢复数据到内存中 | |
| Precondition | Redis服务器已启动，正在运行并未断电，已自动进行快照保存 | |
| Tester | 测试员 | |
| Dependency | None | |
| Test Setup | Name | 修改数据库值 |
| Description | 修改某个数据库的值，以验证修改能被保存和恢复 |
| Basic Flow  (Test Setup) | Steps | |
| 1 | 测试员在运行窗口输入”redis-cli.exe”，打开一个窗口运行客户端A |
| 2 | 输入”set name1 10”和”set name2 20”，设为测试键值 |
| 3 | 输入”del name1”，删除测试键值对name1 |
| 4 | 输入”appen name2 100”，修改测试键值对name2 |
| 3 | 输入”save”，手动进行快照保存 |
| 4 | 输入”shutdown”，停止Redis服务器 |
| Postcondition  (Test Oracle) | 数据库的一对键值被删除 |
| Basic Flow  (Test Sequence) | Steps | |
| 1 | 输入“redis-server.exe redis.conf”，重启服务器 |
| 2 | 输入”redis-cli.exe”，再启动客户端 |
| 3 | 输入”get name1”和”get name2”，获取之前被修改的数据 |
| Postcondition  (Test Oracle) | name1为null，name2为20100，说明服务器重启后，重新从备份文件载入数据到内存成功 |

### 4.3.3 测试代码

本测试无需测试代码。

### 4.3.4 测试结果及分析

首先初始化键值，并修改之（name1被删除，name2被新值覆盖），并关闭服务器：

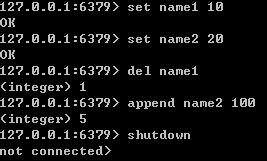


图 1 客户端修改、保存数据，关闭服务器

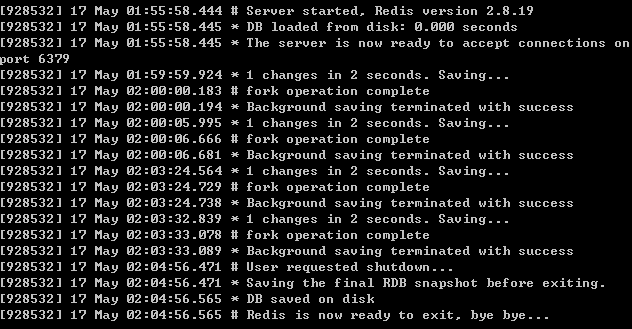


图 2 服务器在此期间的响应

重启服务器之后，再读取该键值，为修改之后的值（因为是删除，因此返回为null），说明数据在服务器重启时已正常载入：



# 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] 肖丁、吴建林等. 软件工程模型与方法[M]. 北京邮电大学出版社. 2008-03.

[4] http://my.oschina.net/hanruikai/blog/313724

[5] http://www.w3cschool.cc/redis/redis-benchmarks.html