



一、简介.....	2
1.1 简介.....	3
1.2 Redis 定义.....	3
1.3 作者.....	3
1.4 性能.....	3
1.4 支持语言.....	4
1.5 数据模型.....	4
1.6 redis 常见命令.....	4
1.7 redis 存储.....	4
二、Window 和 Linux 下安装 Redis 使用.....	5
2.1 Redis-Window 准备工作.....	5
2.2 Redis-Window 下载.....	5
2.4 Redis-Linux 安装.....	5
2.5 Redis-Window 安装与注意事项.....	9
三、Redis 服务器命令.....	10
3.1 启动 redis-cli 客户端.....	10
3.2 简单的命令.....	11
3.2 set 和 get 命令.....	12
3.3 sadd 命令.....	13
3.4 hset 命令.....	13
3.5 keys 命令.....	13
3.6 del 命令.....	14
3.7 exists 命令.....	14
3.8 move 和 select 命令.....	14
3.9 rename 命令.....	15
3.40 expire 命令.....	15
3.41 ttl 命令.....	15
3.42 persist 命令.....	16
3.43 type 命令.....	16
3.44 randomkey 命令.....	16
3.45 flushdb 命令.....	16
3.46 flushAll 命令.....	17
3.47 处理中文乱码.....	17
四、Redis 高级命令.....	17
4.1 数据备份和恢复.....	17
4.2 授权与安全.....	18
4.3 Redis 发布订阅.....	19
4.4 Redis 性能测试.....	20
4.5 Redis 分区.....	21

4.6Redis 管道.....	22
4.7 命令与 java 代码一起兼用.....	23
五、Redis 主从复制.....	24
5.1Redis 主从复制特点.....	24
5.2Redis 主从复制原理图.....	25
5.3Redis 主从机配置.....	25
六、Redis 在 Java 下使用.....	26
6.1Redis 开发准备工作.....	26
6.2Redis 字符串使用.....	28
6.3Redis 字符串使用.....	28
6.4Redis 数组使用.....	29
6.5Redis-Keys 使用.....	30
6.6 学习网.....	30
七、Redis 的 Cache 缓存.....	30
7.1Redis 准备工作.....	31
7.2 简单的 redis 缓存例子.....	31
7.3 实体.....	36
7.4Redis 测试类.....	38
八、Redis 集群.....	39
8.1 集群简介.....	39
8.2 单机集群.....	40
8.3 Keepalived 介绍.....	41
8.4 通过 keepalived 实现的高可用方案.....	42
8.5 使用 Twemproxy 实现集群方案.....	43
8.7 监控工具.....	43
8.6 多台机集群.....	44
8.7Spring+redis 整合.....	45
九、总结.....	54

## 一、简介

---

---

=====Author:jilongliang=====

=====Date:2015/12/16 编著=====

=====Email:[jilongliang@sina.com](mailto:jilongliang@sina.com)=====

---

## 1.1 简介

Redis 是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库，并提供多种语言的 API。从 2010 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

## 1.2 Redis 定义

redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和 hash（哈希类型）。这些数据类型都支持 push/pop、add/remove 及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis 支持各种不同方式的排序。与 memcached 一样，为了保证效率，数据都是缓存在内存中。区别的是 redis 会周期性的把更新的数据写入磁盘或者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(主从)同步。

Redis 是一个高性能的 key-value 数据库。redis 的出现，很大程度补偿了 memcached 这类 key/value 存储的不足，在部分场合可以对关系数据库起到很好的补充作用。它提供了 Java, C/C++, C#, PHP, JavaScript, Perl, Object-C, Python, Ruby, Erlang 等客户端，使用很方便。[1]

Redis 支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得 Redis 可执行单层树复制。存盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

redis 的官网地址，非常好记，是 redis.io。（特意查了一下，域名后缀 io 属于国家域名，是 british Indian Ocean territory，即英属印度洋领地）

目前，Vmware 在资助着 redis 项目的开发和维护。

## 1.3 作者

redis[2] 的作者，叫 Salvatore Sanfilippo，来自意大利的西西里岛，现在居住在卡塔尼亚。目前供职于 Pivotal 公司。他使用的网名是 antirez。

## 1.4 性能

下面是官方的 bench-mark 数据：测试完成了 50 个并发执行 100000 个请求。设置和获取的值是一个 256 字节字符串。Linux box 是运行 Linux 2.6,这是 X3320 Xeon 2.5 ghz。文本执行使用 loopback 接口(127.0.0.1)。结果:读的速度是 110000 次/s,写的速度是 81000 次/s 。

## 1.4 支持语言

许多语言都包含Redis支持，包括：<sup>[1]</sup>

<ul style="list-style-type: none"><li>• <a href="#">ActionScript</a></li><li>• <a href="#">C</a></li><li>• <a href="#">C++</a></li><li>• <a href="#">C#</a></li><li>• <a href="#">Clojure</a></li></ul>	<ul style="list-style-type: none"><li>• <a href="#">Common Lisp</a></li><li>• <a href="#">Dart</a></li><li>• <a href="#">Erlang</a></li><li>• <a href="#">Go</a></li><li>• <a href="#">Haskell</a></li></ul>	<ul style="list-style-type: none"><li>• <a href="#">Haxe</a></li><li>• <a href="#">Io</a></li><li>• <a href="#">Java</a></li><li>• <a href="#">Node.js</a></li><li>• <a href="#">Lua</a></li></ul>	<ul style="list-style-type: none"><li>• <a href="#">Objective-C</a></li><li>• <a href="#">Perl</a></li><li>• <a href="#">PHP</a></li><li>• <a href="#">Pure Data</a></li><li>• <a href="#">Python</a></li></ul>	<ul style="list-style-type: none"><li>• <a href="#">R</a></li><li>• <a href="#">Ruby</a></li><li>• <a href="#">Scala</a></li><li>• <a href="#">Smalltalk</a></li><li>• <a href="#">Tcl</a></li></ul>
---	--	--	---	--

## 1.5 数据模型

Redis 的外围由一个键、值映射的字典构成。与其他非关系型数据库主要不同在于：Redis 中值的类型<sup>[1]</sup> 不仅限于字符串，还支持如下抽象数据类型：

## 1.6 redis 常见命令

Redis 成绩已经很惊人了，且不说 memcachedb 和 Tokyo Cabinet 之流，就说原版的 memcached，速度似乎也只能达到这个级别。Redis 根本是使用内存存储，持久化的关键是这三条指令：SAVE BGSAVE LASTSAVE ...

当接收到 SAVE 指令的时候，Redis 就会 dump 数据到一个文件里面

## 1.7 redis 存储

redis 使用了两种文件格式：全量数据和增量请求。

全量数据格式是把内存中的数据写入磁盘，便于下次读取文件进行加载；

增量请求文件则是把内存中的数据序列化为操作请求，用于读取文件进行 replay 得到数据，序列化的操作包括 SET、RPUSH、SADD、ZADD。

redis 的存储分为内存存储、磁盘存储和 log 文件三部分，配置文件中有三个参数对其进行配置。

save seconds updates，save 配置，指出在多长时间內，有多少次更新操作，就将数据同步到数据文件。这个可以多个条件配合，比如默认配置文件中的设置，就设置了三个条件。

appendonly yes/no，appendonly 配置，指出是否在每次更新操作后进行日志记录，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为 redis 本身同步数据文件是按上面的 save 条件来同步的，所以有的数据会在一段时间內只存在于内存中。

appendfsync no/always/everysec，appendfsync 配置，no 表示等操作系统进行数据缓存同步到磁盘，always 表示每次更新操作后手动调用 fsync()将数据写到磁盘，everysec 表示每秒同步一次

## 二、Window 和 Linux 下安装 Redis 使用

### 2.1 Redis-Window 准备工作

1)官方 <http://redis.io>

### 2.2 Redis-Window 下载

github 下载

<https://github.com/MSOpenTech/redis/releases>

Downloads

6.4 MBRedis-x64-3.0.500.msi

5.58 MBRedis-x64-3.0.500.zip

Source code (zip)

Source code (tar.gz)

百度网盘: <http://pan.baidu.com/s/1HunuQ>

### 2.4 Redis-Linux 安装

- 1) 使用 putty 或 SSH Secure Shell Client 客户端登陆 linux
- 2) 使用 mkdir 命令在 linux 环境下面一个目录存放 reid 文件

这个命令和 window 命令同等

```
root@iZ94vo38mt6Z:/# ls
apps  bin  boot  dev  etc  home  initrd.img  lib  lib64  lost+found  media  mnt  opt  proc  redis  root  run  sbin  srv  sys  usr  var
```

3)使用 cd 命令进入 redis 这个目录

4)使用 wget 命令下载

<http://download.redis.io/releases/> 所有版本

wget <http://download.redis.io/releases/redis-2.8.22.tar.gz>

目前我是用 2.8 这个版本

5)解压文件

```
tar xzf redis-2.8.22.tar.gz
```

6)检查所有文件,你会找不到 redis-server 这个文件 里面都有文件都是 c 语言文件,此时我们得编译一下这些 C 语言文件,成功之后得到我们想要的文件

7)编译 C 语言文件

```
1) make
```

2) `yum install -y gcc g++ gcc-c++ make` //敲完这个命令在之后,如果有问题,再使用 `make MALLOC=libc`  
此时出现这个界面的信息,说明已经成功差不多了

```
INSTALL
CC redis-cli.o
LINK
CC redis-benchmark.o
LINK
CC redis-check-dump.o
LINK
CC redis-check-aof.o
LINK

Hint: It's a good idea to run 'make test' ;)
```

8) 进入 src 目录此时就看到我们想要的文件

使用 `ls` 命令查看所有文件

```
iz:/redis/redis-2.8.22# cd src/
iz:/redis/redis-2.8.22/src# ls
.c      crc64.o      hyperloglog.c  lzfP.h      notify.o      rdb.o          redis-cli.c    rio.o
.o      db.c         hyperloglog.o  Makefile     object.c      redisassert.h  redis-cli.o    scripting.c
aiologo.h db.o         intset.c       Makefile.dep pqsort.c      redis-benchmark.c  redis.h        scripting.o
.c      debug.c      intset.h       memtest.o    pqsort.h      redis-benchmark.o  redis.o        sds.c
.o      dict.c       latency.h      memtest.o    pqsort.o      redis.c          redis.o        sds.h
.ops.c  dict.h       latency.c      migrate.o    pubsub.c      redis-check-aof  redis-server   sds.o
.ops.o  dict.o       latency.o      migrate.o    pubsub.o      redis-check-aof.c release.c       sentinel.c
fig.c   endianconv.c lzf_c.c        mkreleashehdr.sh multi.c        redis-check-aof.o release.h       sentinel.o
fig.h   endianconv.h lzf_c.o        multi.c        rand.c         redis-check-aof.o release.o       setproctitle.c
fig.o   endianconv.o lzf_d.c        networking.c   rand.h         redis-check-dump.c replication.c   setproctitle.o
:64.c   fmacros.h    lzf_d.o        networking.o  rdb.c          redis-check-dump.o rio.c          sha1.c
:64.h   help.h       lzf.h          notify.c      rdb.h          redis-check-dump.o rio.h          sha1.h
:64.o   sha1.o       sha1.h         sha1.o        sha1.o         sha1.o          sha1.o
```

9) 在之前解压的 gz 文件存放, 编译之后想要的文件, 方便启动环境服务  
进入文件, 使用 `mkdir` 文件创建一个 `redis-server` 文件

```
redis-2.6.16 redis-2.6.16.tar.gz redis-2.8.22 redis-2.8.22.tar.gz redis-3.0.6 redis-3.0.6.tar.gz
root@iz94vo38mt6Z:/redis# cd redis-2.8.22/
root@iz94vo38mt6Z:/redis/redis-2.8.22# ls
00-RELEASENOTES CONTRIBUTING deps Makefile README runtest sentinel.conf tests
BUGS COPYING INSTALL MANIFESTO redis.conf runtest-sentinel src utils
root@iz94vo38mt6Z:/redis/redis-2.8.22# mkdir redis-server
root@iz94vo38mt6Z:/redis/redis-2.8.22#
```

A) 首先把 `redis.conf` 文件拷贝进去

`cp redis.conf redis-server`

此时为空看到第一个文件拷贝成功

```
root@iz94vo38mt6Z:/redis/redis-2.8.22# mkdir redis-server
root@iz94vo38mt6Z:/redis/redis-2.8.22# cp redis.conf redis-server/
root@iz94vo38mt6Z:/redis/redis-2.8.22# ls
00-RELEASENOTES CONTRIBUTING deps Makefile README redis-s
BUGS COPYING INSTALL MANIFESTO redis.conf runtest
root@iz94vo38mt6Z:/redis/redis-2.8.22# cd redis-server/
root@iz94vo38mt6Z:/redis/redis-2.8.22/redis-server# ls
redis.conf
```



B)依次把需要的文件拷贝进去

.o	rdb.o	redis-cli.c	r
.c	redisassert.h	redis-cli.o	s
.o	redis-benchmark	redis.h	s
.c	redis-benchmark.c	redis.o	s
.h	redis-benchmark.o	redis-sentinel	s
.o	redis.c	redis-server	s
.c	redis-check-aof	release.c	s
.o	redis-check-aof.c	release.h	s
	redis-check-aof.o	release.o	s
	redis-check-dump	replication.c	s
	redis-check-dump.c	replication.o	s
	redis-check-dump.o	rio.c	s
	redis-cli	rio.h	s

因为我们的这些文件都在 src 目录下，我们拷贝得注意路径啦,这时候要加../

```
cp redis-check-dump ../redis-server
cp redis-cli ../redis-server
cp redis-server ../redis-server
cp redis-benchmark ../redis-server
cp redis-check-aof ../redis-server
```

把这些文件都拷贝文件了,启动一下服务

命令是：`redis-server redis.conf`

安装文件命：`sudo apt-get install 文件名`

如：`sudo apt-get install redis-server`

`sudo apt-get install redis-cli`

.....

此时可能还不行

**FATAL CONFIG FILE ERROR \*\*\***

Reading the configuration file, at line 54

>>> 'tcp-backlog 511'

```
Processing triggers for ureadahead (0.100.0-16) ...
root@iz94vo38mt6z:/redis/redis-2.8.22/redis-server# redis-server redis.conf
*** FATAL CONFIG FILE ERROR ***
Reading the configuration file, at line 54
>>> 'tcp-backlog 511'
bad directive or wrong number of arguments
```

用 vi 编辑命令去参考这个 redis.conf 配置是不是有问题，打开这个文件的时候看到有这么一行代码。>>>'tcp-backlog 511'这些配置去掉或者用#注释掉。

指定配置

```
31 # include /path/to/other.conf
32
33 ##### GENERAL #####
34
35 # By default Redis does not run as a daemon. Use 'yes' if y
36 # Note that Redis will write a pid file in /var/run/redis.p
37 daemonize no
38
39 # When running daemonized, Redis writes a pid file in /var/
40 # default. You can specify a custom pid file location here.
41 pidfile /var/run/redis/redis-server.pid
42
43 # Accept connections on the specified port, default is 6379
44 # If port 0 is specified Redis will not listen on a TCP so
```

再使用 **redis-server redis.conf** 启动服务端

```
root@iz94vo38mt6Z:/redis/redis-2.8.22/redis-server1# redis-server
[18382] 23 Dec 14:47:19.385 # Warning: no config file specified, using the default
fig. In order to specify a config file use redis-server /path/to/redis.conf

Redis 2.8.4 (00000000/0) 64 bit

Running in stand alone mode
Port: 6379
PID: 18382

http://redis.io
```

再使用 **redis-cli** 启动客户端

```
File Edit View Window Help
[Icons]
Quick Connect Profiles

127.0.0.1:6379> set mytest hadoop
OK
127.0.0.1:6379> get mytest
"hadoop"
127.0.0.1:6379> █
```

10) 查看端口命令

netstat -tunpl | grep 6379



Remote Name	Size	Type
redis-benchmark	2,168,629	文件
redis-check-aof	29,267	文件
redis-check-dump	62,075	文件
redis-cli	2,316,414	文件
redis-sentinel	4,249,005	文件
redis-server	4,249,005	文件
redis.conf	37,077	CONF ..
sentinel.conf	7,109	CONF ..

## 2.5 Redis-Window 安装与注意事项

1)我下载的到本地的文件是 redis-windows-master

打开这个文件 D:\Server\redis\redis-windows-master\downloads\redis64-2.6.12.1.zip 这个目录下面有

redis.conf 配置是 window 下使用启动 redis 所需要的文件,因为我本机是 32bit 的.我在官方也没找到然后 redis64

把下面的文件拷贝过来,虽然上面是写着 64.拷贝过来也可以启动。当然了,你不能直接用 redis-windows-master

下面的所有

2)解压的 redis 包含的文件

[redis-benchmark.exe](#)

[redis-check-aof.exe](#)

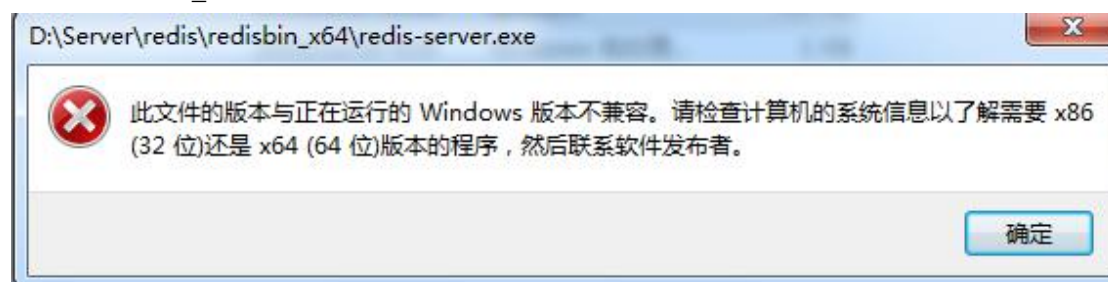
[redis-check-dump.exe](#)

[redis-cli.exe](#)

[redis-server.exe](#)

解压直接双击 redis-server.exe 文件启动 redis 系统会报一个错误,无法识别是 32bit 还是 64bit 的系统,从此验证

这个五个文件不能在 32bit 系统启动,当然这个不用担心,我已经下载好有 32bit 的 redis 的文件.redisbin\_x32



3) 自己创建 bat 后缀格式的批处理文件作为快捷键启动 window 服务,redis.conf 这个文

件可以自己随便定，有意义即可

并且必须要这个批处理能读取这个配置

A)start-redis.bat

`redis-server redis.conf`

B)service-install.bat

`redis-server.exe --service-install redis.conf --loglevel verbose`

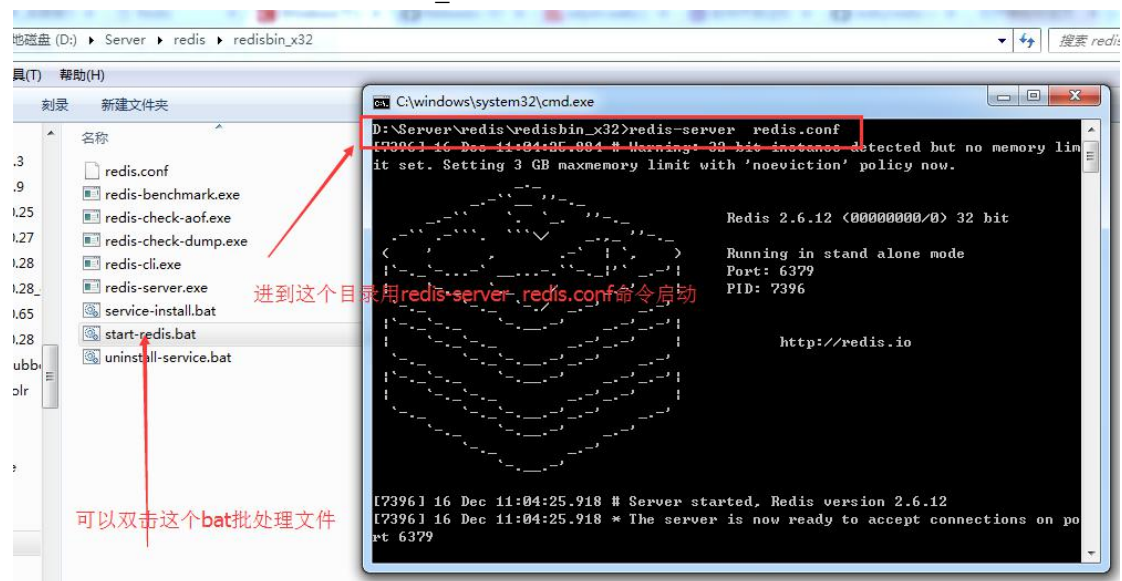
C)uninstall-service.bat

`redis-server --service-uninstall`

4) 使用 redis 命令启动。

A)首先进入 redis 的根目录下,加上我的文件目录放在，打开 cmd 命令窗体

D:\Server\redis>cd redisbin\_x32 出现如下截图表示成功：



```
$ wget http://download.redis.io/releases/redis-3.0.6.tar.gz
$ tar xzf redis-3.0.6.tar.gz
$ cd redis-3.0.6
$ make
```

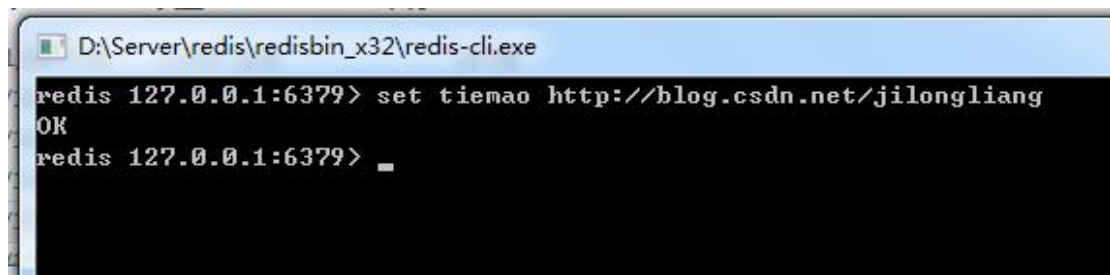
## 三、Redis 服务器命令

### 3.1 启动 redis-cli 客户端

此时出现上面那个界面就可以双击 **redis-cli.exe** 文件

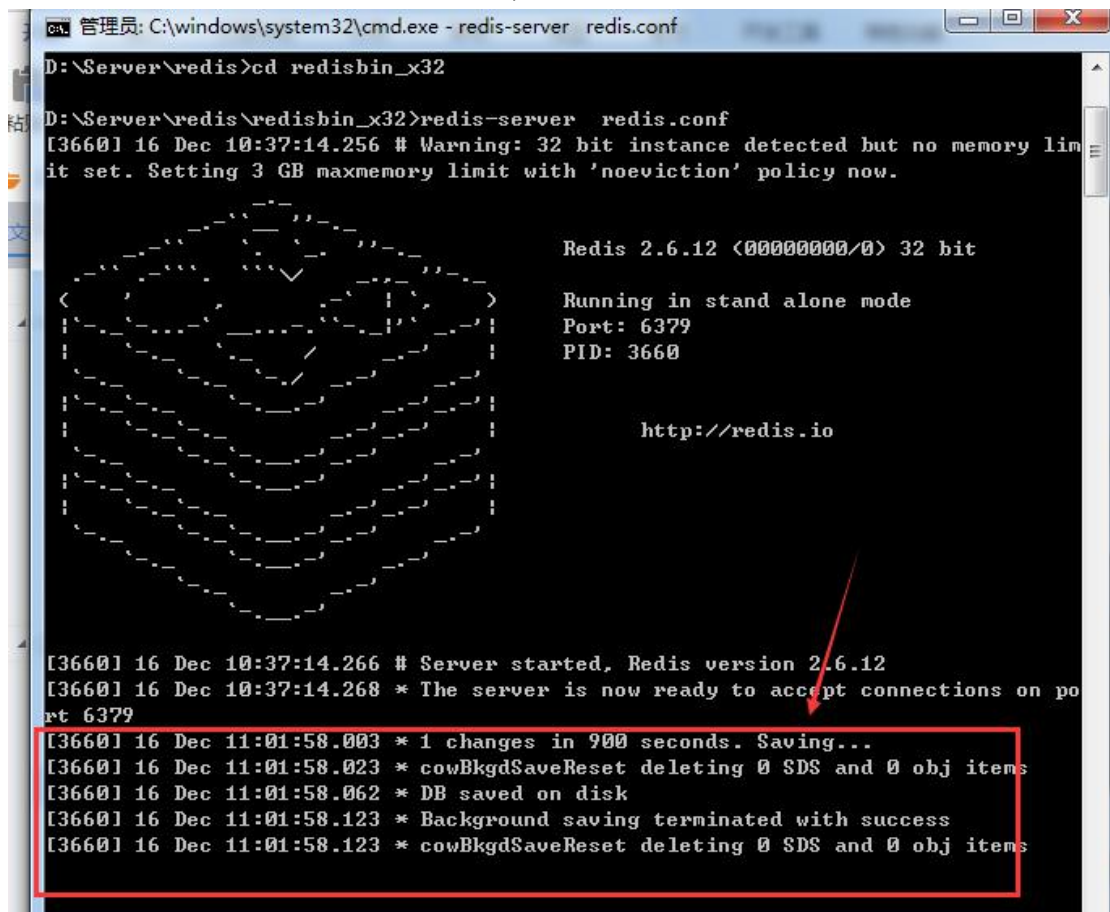
## 3.2 简单的命令

1) **set 命令**的使用，使用此命令必须保证服务是启动的



```
D:\Server\redis\redisbin_x32\redis-cli.exe
redis 127.0.0.1:6379> set tiemao http://blog.csdn.net/jilongliang
OK
redis 127.0.0.1:6379> _
```

一回车之后此时，数据向磁盘内存存数据,可以看到这个服务这个窗体相应



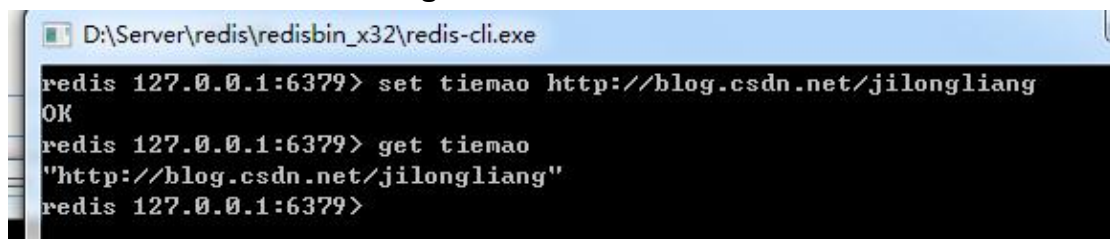
```
管理员: C:\windows\system32\cmd.exe - redis-server redis.conf
D:\Server\redis>cd redisbin_x32
D:\Server\redis\redisbin_x32>redis-server redis.conf
[3660] 16 Dec 10:37:14.256 # Warning: 32 bit instance detected but no memory limit set. Setting 3 GB maxmemory limit with 'noeviction' policy now.

Redis 2.6.12 <00000000/0> 32 bit
Running in stand alone mode
Port: 6379
PID: 3660

http://redis.io

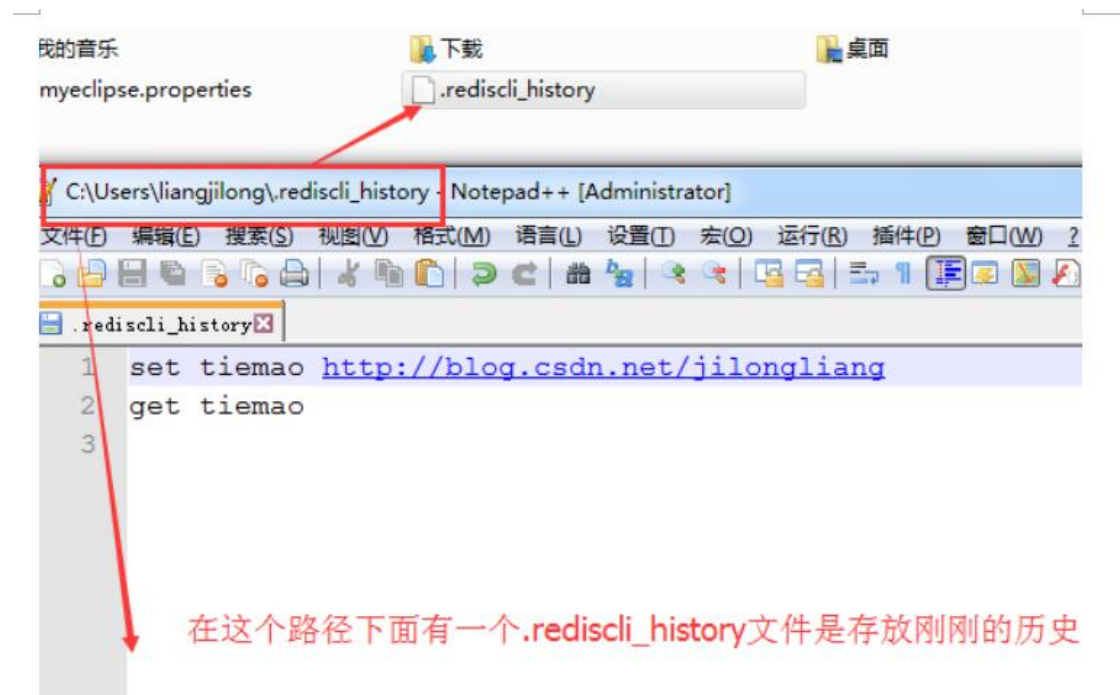
[3660] 16 Dec 10:37:14.266 # Server started, Redis version 2.6.12
[3660] 16 Dec 10:37:14.268 * The server is now ready to accept connections on port 6379
[3660] 16 Dec 11:01:58.003 * 1 changes in 900 seconds. Saving...
[3660] 16 Dec 11:01:58.023 * cowBkgdSaveReset deleting 0 SDS and 0 obj items
[3660] 16 Dec 11:01:58.062 * DB saved on disk
[3660] 16 Dec 11:01:58.123 * Background saving terminated with success
[3660] 16 Dec 11:01:58.123 * cowBkgdSaveReset deleting 0 SDS and 0 obj items
```

2) 设置了这个命令就可以用 **get tiemao**

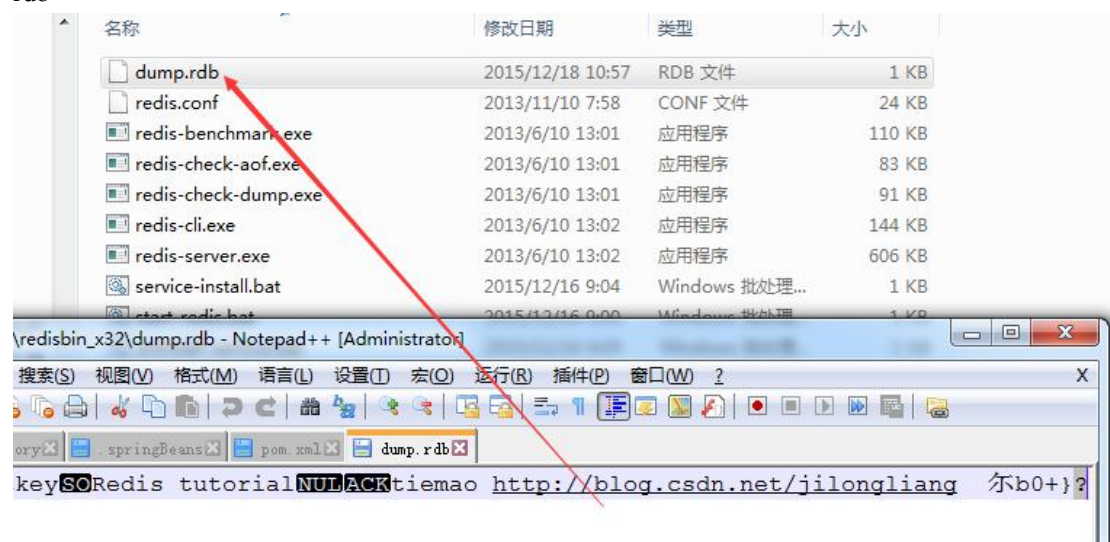


```
D:\Server\redis\redisbin_x32\redis-cli.exe
redis 127.0.0.1:6379> set tiemao http://blog.csdn.net/jilongliang
OK
redis 127.0.0.1:6379> get tiemao
"http://blog.csdn.net/jilongliang"
redis 127.0.0.1:6379>
```

3) 查看历史记录



rdb



### 3.2 set 和 get 命令

语法: set 定义的 key 名称 值

set mykey1 'redis' 这个命令就是说这种一个 key=mykey1,value=redis 这样在字符串

```
redis 127.0.0.1:6379> set mykey1 'redis'
OK
redis 127.0.0.1:6379>
```

语法: get 定义的 key 名

Get mykey1 这个命令的意思是说,获取 mykey1 的值,并输出的值一定是 redis

```
OK
redis 127.0.0.1:6379> get mykey1
"redis"
redis 127.0.0.1:6379>
```

### 3.3 sadd 命令

#添加 Set 类型的模拟数据。

语法: **sadd** 变量名 value1 value2 value3 .....

**sadd mykey2 1 2 3** 这里就说存了三个值并且的值是值 1 2 3 的值。

```
redis
redis 127.0.0.1:6379> sadd mykey2 1 2 3
(integer) 3
redis 127.0.0.1:6379> _
```

### 3.4 hset 命令

#添加 Hash 类型的模拟数据

语法: **hset** 变量名 username '参数值' 变量名不能重复, 也就说在你用过这个变量名的时候, 此时你回车键的时候他返回的值是为 0

```
(integer) 0
redis 127.0.0.1:6379> hset it username "hadoop"
(integer) 1
redis 127.0.0.1:6379> hset it1 k 'redis';
Invalid argument(s)
redis 127.0.0.1:6379> hset k username 'xiaoming'
(integer) 1
redis 127.0.0.1:6379> hset it username 'hadoop'
(integer) 0
redis 127.0.0.1:6379> hset bigdata username hadoop
(integer) 1
redis 127.0.0.1:6379> _
```

### 3.5 keys 命令

语法: **keys** my\*

Keys 是关键词, 这里不区分大小写, my\*是代表找到 my 开头多所有 key 变量.

#根据参数中的模式, 获取当前数据库中符合该模式的所有 key, 从输出可以看出, 该命令在执行时并不区分与 Key 关联的 Value 类型。

刚刚我是定义了两个 my 开头的 key 分别是如下



```
redis 127.0.0.1:6379> keys my*
1) "mykey1"
2) "mykey2"
redis 127.0.0.1:6379> Keys my*
1) "mykey1"
2) "mykey2"
redis 127.0.0.1:6379>
```

### 3.6 del 命令

语法: `del key 变量`

`Del mykey1 mykey2` 这句命令的意思, 就说删除两个 key 为 mykey1 mykey2

```
redis 127.0.0.1:6379> del mykey1 mykey2
(integer) 2
redis 127.0.0.1:6379>
```

### 3.7 exists 命令

语法: `exists key 变量`

`Exists mykey1` 此时把命令敲上, 返回的值是为 0, 说明确实已经删掉了.

```
(integer) 2
redis 127.0.0.1:6379> exists mykey1
(integer) 0
redis 127.0.0.1:6379>
```

### 3.8 move 和 select 命令

之前把 **mykey1 mykey2** 都删掉了, 此时我们要创建 2 个.

```
redis 127.0.0.1:6379> set mykey3 'SpringScurity'
OK
redis 127.0.0.1:6379>

redis 127.0.0.1:6379> set mykey4 'SpringShiro'
OK
redis 127.0.0.1:6379>
```

语法: `move 存在的 key 变量值`

*#将当前数据库中的mykey4 键移入到ID 为1 的数据库中, 从结果可以看出已经移动成功。*

```
redis 127.0.0.1:6379> move mykey4 1
(integer) 1
redis 127.0.0.1:6379>
```



语法: select 被移进去的值

```
(integer) 1
redis 127.0.0.1:6379> select 1
OK
redis 127.0.0.1:6379[1]> _
```

### 3.9 rename 命令

语法: rename 存在的 key 不存在的 key

*#将 mykey4 改名为 mykey1*

```
redis 127.0.0.1:6379> rename mykey4 mykey1
OK
redis 127.0.0.1:6379>
```

此时再用 **get mykey4** 去获取便返回,null

```
redis 127.0.0.1:6379> get mykey4
(nil)
redis 127.0.0.1:6379>
```

### 3.40 expire 命令

Expire 故名思议就是设置时间时效的命令

语法是: expire key 名 时间

```
redis 127.0.0.1:6379> expire mykey1 10
(integer) 1
redis 127.0.0.1:6379>
```

此时截图是说,设置一个 mykey1 的 key 在 10 秒就时效, 而且获取不了值

```
redis 127.0.0.1:6379> expire mykey1 10
(integer) 1
redis 127.0.0.1:6379> get mykey1
(nil)
redis 127.0.0.1:6379> _
```

### 3.41 ttl 命令

Ttl 命令是告诉我们时间离时效的时间还有多少

```
redis 127.0.0.1:6379> expire mykey1 1000
(integer) 1
```

我们再设置一下, 这时时间设置大点为 1000 秒  
用 ttl 命令跟踪一下时间

```
redis 127.0.0.1:6379> expire mykey1 1000
(integer) 1
redis 127.0.0.1:6379> ttl mykey1
(integer) 993
redis 127.0.0.1:6379>
```

### 3.42persist 命令

语法: persist key 名称

*#立刻执行 **persist** 命令, 该存在超时的键变成持久化的键, 即将该 **Key** 的超时去掉。*

```
redis 127.0.0.1:6379> persist mykey1
(integer) 1
redis 127.0.0.1:6379>
```

*##ttl 的返回值告诉我们, 该键已经没有超时了。*

```
(integer) 1
redis 127.0.0.1:6379> ttl mykey1
(integer) -1
redis 127.0.0.1:6379>
```

### 3.43type 命令

语法: type key *#由于 **hibernate** 键在数据库中不存在, 因此该命令返回 **none**。*

```
redis 127.0.0.1:6379> type hibernate
none
```

*#mykey1 的值是字符串类型, 因此返回 **string**。*

```
redis 127.0.0.1:6379> type mykey1
string
```

### 3.44randomkey 命令

语法: randomkey 随机 key

*#返回数据库中的任意键。*

```
redis 127.0.0.1:6379> randomkey
"bigdata"
```

### 3.45flushdb 命令

语法: flushdb 清空 此命令是清除数据库所有数据

```
redis 127.0.0.1:6379> flushdb
OK
redis 127.0.0.1:6379> get mykey1
(nil)
redis 127.0.0.1:6379>
```

### 3.46 flushAll 命令

语法: flushAll //此命令是清除数据库 Key 所有数据

```
redis 127.0.0.1:6379> flushall
OK
redis 127.0.0.1:6379> keys *
(empty list or set)
```

### 3.47 处理中文乱码

```
$ redis-cli --help
```

--raw [显示中文](#)，而不是"\xd6\xd0"

```
redis 127.0.0.1:6379> set aaa 中
```

```
OK
```

```
redis 127.0.0.1:6379> get aaa
```

```
中
```

如果你是用的 windows cmd，还是乱码，要设置窗口的编码

chcp 65001 就是换成 UTF-8 代码页，在命令行标题栏上点击右键，选择"属性"->"字体"，将字体修改为 True Type 字体"Lucida Console"，然后点击确定将属性应用到当前窗口

## 四、Redis 高级命令

### 4.1 数据备份和恢复

Redis **SAVE** 命令用于创建当前数据库的备份

```
redis 127.0.0.1:6379> save
OK
redis 127.0.0.1:6379>
```

此时命令回车之后，目录下面立刻生成一个 rdb 文件，此时就是数据库文件



创建 redis 备份文件也可以使用命令 **BGSAVE**，该命令在后台执行

```
redis 127.0.0.1:6379> bgsave
Background saving started
redis 127.0.0.1:6379>
```

恢复数据如果需要恢复数据，只需将备份文件 (dump.rdb) 移动到 redis 安装目录并启动服务即可。获取 redis 目录可以使用 CONFIG 命令，如下所示：

```
OK
redis 127.0.0.1:6379> config get dir
1) "dir"
2) "D:\\Server\\redis\\redisbin_x32"
redis 127.0.0.1:6379>
```

## 4.2 授权与安全

设置客户端连接后进行任何其他指定前需要使用的密码

注意：因为 redis 速度相对快，所有一太比较好的服务器下，一个外部连接的用户可以一秒进行 150k 次的密码尝试，这意味着您需要指定非常强大的密码来防止暴力破解

### 1) 登陆授权

进入 redis-cli 目录

Redis-cli -a 密码

Redis-cli -a 123456

### 2) 直接授权

语法：auth 密码

```
redis 127.0.0.1:6379> auth liangjilong
OK
```

### 3) 直接授权

我们可以通过 redis 的配置文件设置密码参数，这样客户端连接到 redis 服务就需要密码验证，这样可以让你的 redis 服务更安全。实例：我们可以通过以下命令查看是否设置了密码验证

语法：config set requirepass 密码字符串

```
(error) ERR unknown command
redis 127.0.0.1:6379> CONFIG set requirepass "123456"
OK
```

此时想用命令获取相关的信息就会报错了(error) ERR operation not permitted  
这句英文就说没权限进行操作。

```
OK
redis 127.0.0.1:6379> keys *
(error) ERR operation not permitted
redis 127.0.0.1:6379>
```

```
redis 127.0.0.1:6379> CONFIG get requirepass
(error) ERR operation not permitted
```

此时必须用 Auth 命令去授权

```
redis 127.0.0.1:6379> auth 123456
OK
```

授权完成之后就会输出 OK 的话，顾名思义就可以用命令去操作了。

```
(error) ERR operation not permitted
redis 127.0.0.1:6379> auth 123456
OK
redis 127.0.0.1:6379> keys *
1) "key1"
2) "tutorial-list"
3) "foo:rand:0000000000000"
4) "counter:rand:0000000000000"
5) "spring1"
6) "mylist"
redis 127.0.0.1:6379> .
```

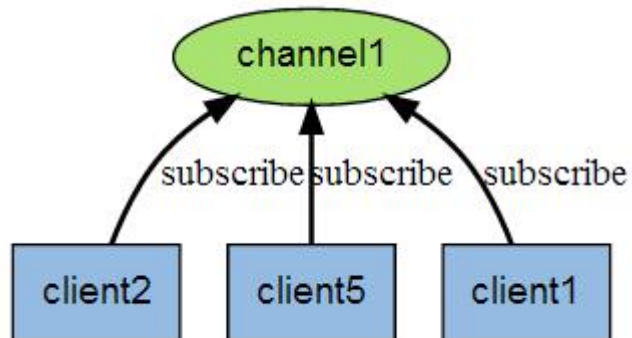
事问谷歌

### 4.3 Redis 发布订阅

Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

Redis 客户端可以订阅任意数量的频道。

下图展示了频道 channel1，以及订阅这个频道的三个客户端——client2、client5 和 client1 之间的关系



语法：subscribe 名称

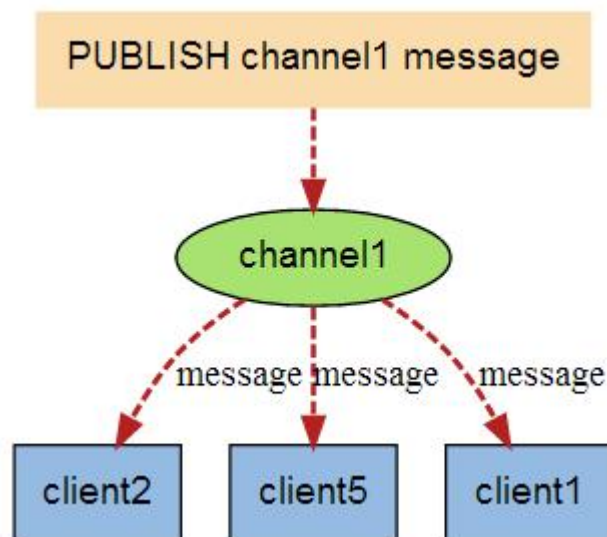
第一个客户端的订阅

```
redis 127.0.0.1:6379> subscribe test1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "test1"
3) (integer) 1
```

第二个客户端的订阅

```
redis 127.0.0.1:6379> subscribe test2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "test2"
3) (integer) 1
```

当有新消息通过 PUBLISH 命令发送给频道 channel1 时，这个消息就会被发送给订阅它的三个客户端：



推送信息给订阅的客户端

语法是： **PUBLISH** 客户端名 "内容"

```
redis 127.0.0.1:6379> auth k
(error) ERR Client sent AUTH, but no password is set
redis 127.0.0.1:6379> subscribe test1
(error) ERR unknown command 'subscribe'
redis 127.0.0.1:6379> subscribe test1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "test1"
3) (integer) 1
1) "message"
2) "test1"
3) "send to test1 message"
```

```
redis 127.0.0.1:6379> subscribe test2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "test2"
3) (integer) 1
```

```
D:\Server\redis\redis_32\redisbin_x32\redis-cli.exe
redis 127.0.0.1:6379> PUBLISH test1 "send to test1 message"
(integer) 1
redis 127.0.0.1:6379>
```

## 4.4 Redis 性能测试

启动 [Benchmark.exe](#) 文件可以看到如下信息



打印出来的信息说明，1 万个请求需要 0.38 完成

```
===== LPOP =====  
10000 requests completed in 0.38 seconds  
50 parallel clients  
3 bytes payload  
keep alive: 1
```

## 4.5 Redis 分区

分区是分割数据到多个 Redis 实例的处理过程，因此每个实例只保存 key 的一个子集。

### 分区的优势

通过利用多台计算机内存的和值，允许我们构造更大的数据库。

通过多核和多台计算机，允许我们扩展计算能力；通过多台计算机和网络适配器，允许我们扩展网络带宽。

### 分区的不足

redis 的一些特性在分区方面表现的不是很好：

涉及多个 key 的操作通常是不被支持的。举例来说，当两个 set 映射到不同的 redis 实例上时，你就不能对这两个 set 执行交集操作。

涉及多个 key 的 redis 事务不能使用。

当使用分区时，数据处理较为复杂，比如你需要处理多个 rdb/aof 文件，并且从多个实例和主机备份持久化文件。增加或删除容量也比较复杂。redis 集群大多数支持在运行时增加、删除节点的透明数据平衡的能力，但是类似于客户端分区、代理等其他系统则不支持这项特性。然而，一种叫做 presharding 的技术对此是有帮助的。

### 分区类型

**Redis 有两种类型分区。**

假设有 4 个 Redis 实例 R0, R1, R2, R3，和类似 `user:1`，`user:2` 这样的表示用户的多个 key，对既定的 key 有多种不同方式来选择这个 key 存放在哪个实例中。也就是说，有不同的系统来映射某个 key 到某个 Redis 服务。

#### 范围分区

最简单的分区方式是按范围分区，就是映射一定范围的对象到特定的 Redis 实例。

比如，ID 从 0 到 10000 的用户会保存到实例 R0，ID 从 10001 到 20000 的用户会保存到 R1，以此类推。

这种方式是可行的，并且在实际中使用，不足就是要有一个区间范围到实例的映射表。这个表要被管理，同时还需要各种对象的映射表，通常对 Redis 来说并非是好的方法。

#### 哈希分区

另外一种分区方法是 hash 分区。这对任何 key 都适用，也无需是 `object_name` 这种形式，像下面描述的一样简单：

用一个 hash 函数将 key 转换为一个数字，比如使用 `crc32` hash 函数。对 key `foobar` 执行 `crc32(foobar)` 会输出类似 93024922 的整数。

对这个整数取模，将其转化为 0-3 之间的数字，就可以将这个整数映射到 4 个 Redis 实例中的一个了。 $93024922 \% 4 = 2$ ，就是说 key `foobar` 应该被存到 R2 实例中。注意：取模操作是取除的余数，通常在多种编程语言中用 `%` 操作符实现

## 4.6 Redis 管道

Redis 是一种基于客户端-服务端模型以及请求/响应协议的 TCP 服务。这意味着通常情况下一个请求会遵循以下步骤：

客户端向服务端发送一个查询请求，并监听 Socket 返回，通常是以阻塞模式，等待服务端响应。

服务端处理命令，并将结果返回给客户端。

Redis 管道技术 Redis 管道技术可以在服务端未响应时，客户端可以继续向服务端发送请求，并最终一次性读取所有服务端的响应。

实例

查看 redis 管道，只需要启动 redis 实例并输入以下命令：`$(echo -en "PING\r\n SET w3ckey redis\r\nGET w3ckey\r\nINCR visitor\r\nINCR visitor\r\nINCR visitor\r\n"; sleep 10) | nc localhost 6379`

+PONG+OK

redis:1:2 以上实例中我们通过使用 PING 命令查看 redis 服务是否可用，之后我们设置了 w3ckey 的值为 redis，然后我们获取 w3ckey 的值并使得 visitor 自增 3 次。

在返回的结果中我们可以看到这些命令一次性向 redis 服务提交，并最终一次性读取所有服务端的响应

管道技术的优势

管道技术最显著的优势是提高了 redis 服务的性能。

一些测试数据

在下面的测试中，我们将使用 Redis 的 Ruby 客户端，支持管道技术特性，测试管道技术对速度的提升效果。

```
require 'rubygems' require 'redis' def bench(descr)

start = Time.now yield

puts "#{descr} #{Time.now-start} seconds" enddef without_pipelining

r = Redis.new 10000.times {

  r.ping } enddef with_pipelining

r = Redis.new

r.pipelined {
```

```
10000.times {  
  
    r.ping  
  
} } end  
  
bench("without pipelining") {  
  
    without_pipelining }  
  
bench("with pipelining") {  
  
    with_pipelining }
```

从处于局域网中的 Mac OS X 系统上执行上面这个简单脚本的数据表明，开启了管道操作后，往返时延已经被改善得相当低了。

```
without pipelining 1.185238 seconds with pipelining 0.250783 seconds
```

如你所见，开启管道后，我们的速度效率提升了 5 倍。

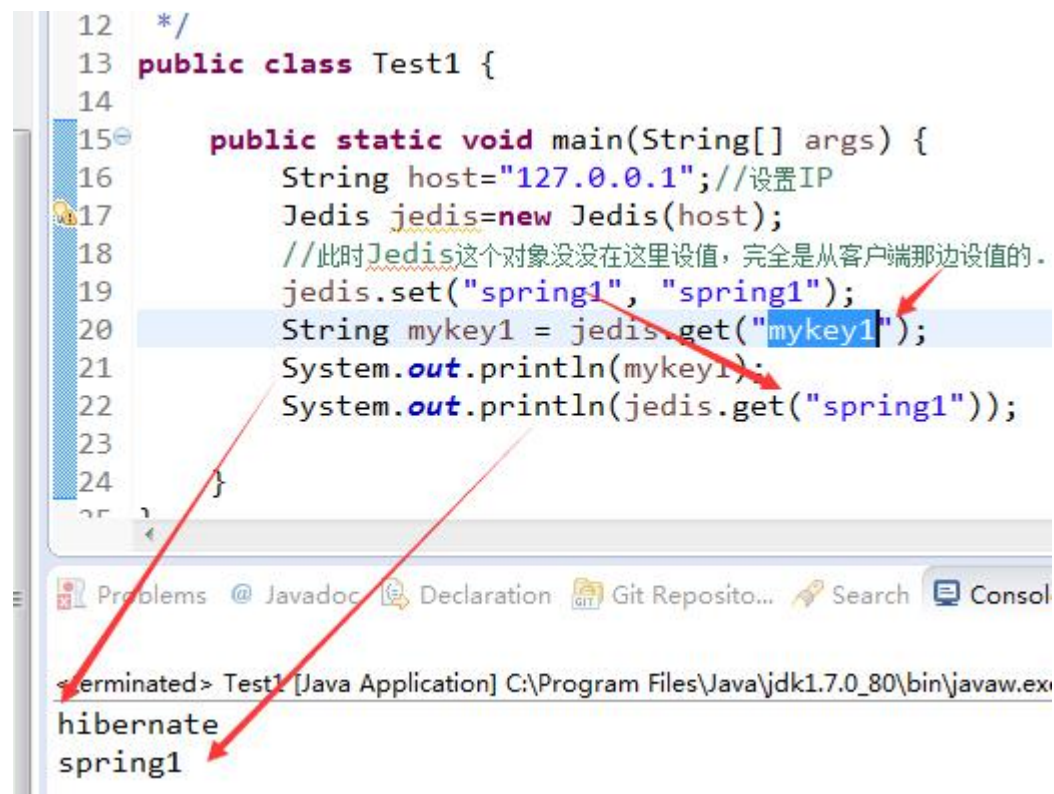
#### 4.7 命令与 java 代码一起兼用

好啦，此时好多人都会很烦，敲了那么多命令，也学了那么多，到头不是为了在 java 一样开发吗？好马上看到了，大家都很期待在时光到了。

```
12  */
13  public class Test1 {
14
15      public static void main(String[] args) {
16          String host="127.0.0.1";//设置IP
17          Jedis jedis=new Jedis(host);
18          //此时Jedis这个对象没在这里设值，完全是从客户端那边设值的。
19          jedis.set("spring1", "spring1");
20          String mykey1 = jedis.get("mykey1");
21          System.out.println(mykey1);
22          System.out.println(jedis.get("spring1"));
23      }
24  }
```

Problems @ Javadoc Declaration Git Reposito... Search Console

<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.7.0\_80\bin\javaw.exe  
hibernate  
spring1

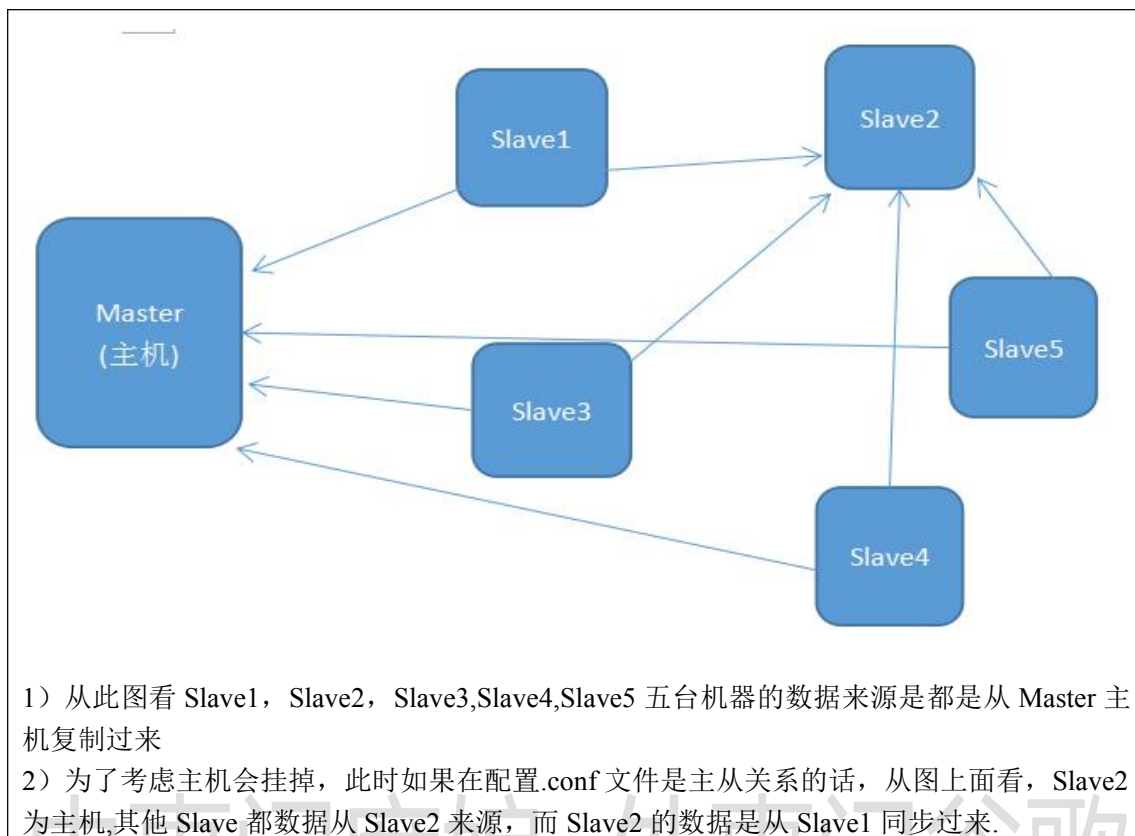


## 五、Redis 主从复制

### 5.1 Redis 主从复制特点

- 1) Master 可以拥有多个 Slave
- 2) 多个 Slave 可以连接同一个 Master 外，还可以连接其他 Slave
- 3) 主从复制不会阻塞 Master,在同步数据时 Master 可以继续 Client 请求
- 4) 提高系统的伸缩性

## 5.2 Redis 主从复制原理图



## 5.3 Redis 主从机配置

### 1) 主机 conf 配置

Redis 的主从复制功能非常强大, 一个 master 可以拥有多个 slave, 而一个 slave 又可以拥有多个 slave, 如此下去, 形成了强大的多级服务器集群架构。下面楼主简单的进行一下配置。

1、上面安装好的一个 Redis 作为 master, 然后使用 VirtualBox 的虚拟机克隆功能将刚刚那个 linux 系统克隆一份作为 slave, 并修改其 IP 为 192.168.0.110。

2、修改 slave 的 redis 配置文件:

`slaveof 192.168.0.100 6379` (映射到主服务器上)如果 master 设置了验证密码, 还需配置 `masterauth`。楼主的 master 设置了验证密码为 `admin`, 所以配置 `masterauth admin`。

配置完之后启动 slave 的 Redis 服务, OK, 主从配置完成。下面测试一下:

在 master 和 slave 分别执行 `info` 命令

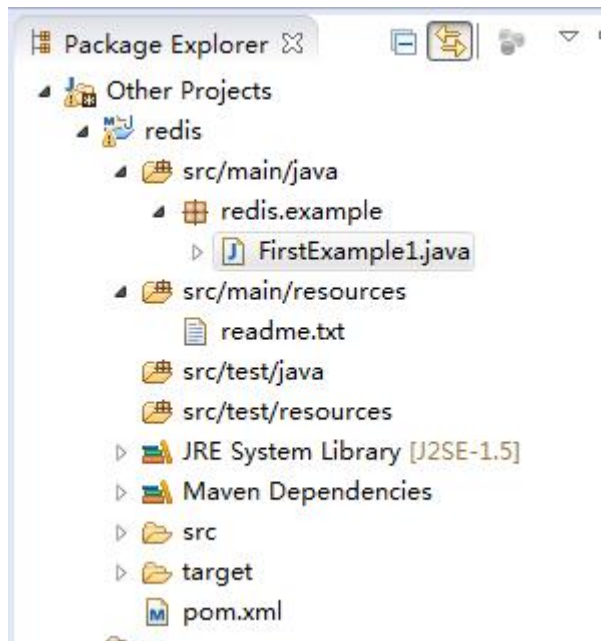
2) 从机 conf 配置

## 六、Redis 在 Java 下使用

### 6.1 Redis 开发准备工作

1) 打开 Eclipse 新建一个 maven 工程.  
这个步骤就不截图了





- 2) 在工程加入 redis 依赖的 jar  
目前 maven repository 的最高版本是 2.8.x

Version	
2.8.x	2.8.0
	2.7.3
2.7.x	2.7.2
	2.7.1
	2.7.0
	2.6.3

```
<dependency>  
  <groupId>redis.clients</groupId>  
  <artifactId>jedis</artifactId>  
  <version>2.8.0</version>  
</dependency>
```

- 3) 在工程建一个 java 类测试，在测试工程中必须要保证这个客户端是启动的

```
C:\windows\system32\cmd.exe
D:\Server\redis\redisbin_x32>redis-server redis.conf
[1440] 16 Dec 11:27:39.192 # Warning: 32 bit instance detected but no memory limit set. Setting 3 GB maxmemory limit with 'noeviction' policy now.

Redis 2.6.12 (00000000/0) 32 bit

Running in stand alone mode
Port: 6379
PID: 1440

http://redis.io

[1440] 16 Dec 11:27:39.234 # Server started, Redis version 2.6.12
[1440] 16 Dec 11:27:39.234 * The server is now ready to accept connections on port 6379
```

## 6.2 Redis 字符串使用.

```
10  *@Copyright(c)liangjilong
11  *@Description:
12  */
13  @SuppressWarnings("all")
14  public class FirstExample1 {
15
16      public static void main(String[] args) {
17          //连接本地的 Redis 服务
18          Jedis jedis = new Jedis("localhost");//加上主机/或者可以用127.0.0.1
19          System.out.println("Connection to server sucessfully");
20          //查看服务是否运行
21          System.out.println("Server is running: "+jedis.ping());
22      }
23  }
24  }
25  }
```

<terminated> FirstExample1 [Java Application] C:\Program Files\Java\jdk1.7.0\_80\bin\javaw.exe (2015年12月16日 上午11:32:11)

```
Connection to server sucessfully
Server is running: PONG
```

## 6.3 Redis 字符串使用.

```
package redis.example;

import redis.clients.jedis.Jedis;
```

```

public class RedisStringJava {
    public static void main(String[] args) {
        //连接本地的 Redis 服务
        Jedis jedis = new Jedis("localhost");
        System.out.println("Connection to server sucessfully");
        //设置 redis 字符串数据
        jedis.set("w3ckey", "Redis tutorial");
        // 获取存储的数据并输出
        System.out.println("Stored string in redis:: "+
jedis.get("w3ckey"));
    }
}

```

## 6.4Redis 数组使用.

```

package redis.example;

import java.util.List;

import redis.clients.jedis.Jedis;
public class RedisListJava {
    public static void main(String[] args) {
        //连接本地的 Redis 服务
        Jedis jedis = new Jedis("localhost");
        System.out.println("Connection to server sucessfully");
        //存储数据到列表中
        jedis.lpush("tutorial-list", "Redis");
        jedis.lpush("tutorial-list", "Mongodb");
        jedis.lpush("tutorial-list", "Mysql");
        // 获取存储的数据并输出
        List<String> list = jedis.lrange("tutorial-list", 0 ,5);
        for(int i=0; i<list.size(); i++) {
            System.out.println("Stored string in redis::
"+list.get(i));
        }
    }
}

```

## 6.5 Redis-Keys 使用.

```
package redis.example;

import java.util.List;

import redis.clients.jedis.Jedis;

public class RedisKeyJava {
    public static void main(String[] args) {
        //连接本地的 Redis 服务
        Jedis jedis = new Jedis("localhost");
        System.out.println("Connection to server sucessfully");

        // 获取数据并输出
        List<String> list = (List<String>) jedis.keys("*");
        for(int i=0; i<list.size(); i++) {
            System.out.println("List of stored keys:: "+list.get(i));
        }
    }
}
```

## 6.6 学习网

1、此工程我是用了 redis2.6.x 版本，因为我的服务是用了 2.6.x 版本，我不知道用高版本会不会有问题，为了学习  
就不纠结这个版本问题，自己可以通过多试一试验证一下。

此工程是用了 mavne 搭建，所有有些 jar 我就不提供，mavne 会自动下载，maven 库可以自己找

<http://www.mvnrepository.com/artifact/redis.clients/jedis/2.6.1>

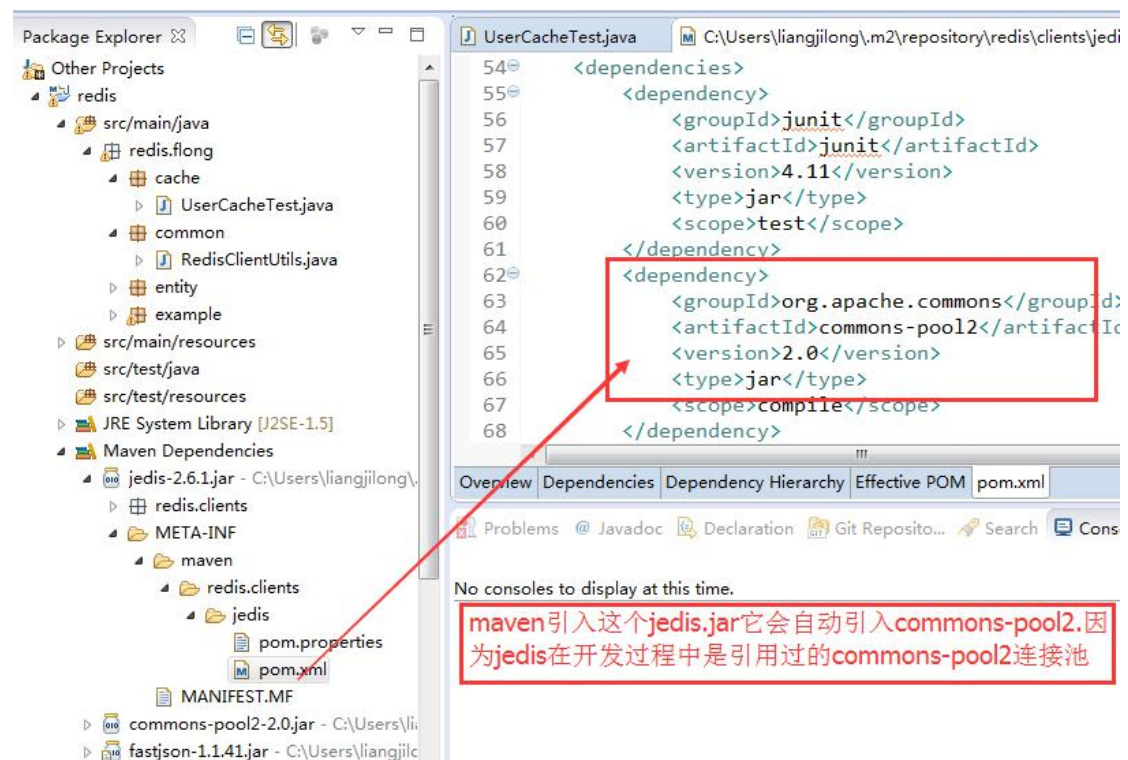
### 6.1

代码我就不上传到 git 代码库了，可以介绍个网站给大家伙学习。

<http://www.runoob.com/redis/redis-java.html>

## 七、Redis 的 Cache 缓存

## 7.1 Redis 准备工作



引入一个阿里巴巴的 fastjson 包

```
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.1.41</version>
</dependency>
```

## 7.2 简单的 redis 缓存例子

1)redis.properties 配置文件的参数

```
rediscli_history x redis.properties x
1 #最大分配的对象数
2 redis.pool.maxActive= 100
3 #最大能够保持idle状态的对象数
4 redis.pool.maxIdle= 20
5 #当池内没有返回对象时，最大等待时间
6 redis.pool.maxWait= 3000
7 #当调用borrow Object方法时，是否进行有效性检查
8 redis.pool.testOnBorrow=true
9 #redis链接的IP
10 redis.ip= localhost
11 #redis链接的端口
12 redis.port= 6379
```

2)Redis 读取配置帮助类的封装

```
package redis.flong.common;
import java.util.ResourceBundle;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;
import com.alibaba.fastjson.JSON;
/**
 * @Author:liangjilong
 * @Date:2015 年 12 月 18 日上午 8:52:29
 * @Email:jilongliang@sina.com
 * @Version:1.0
 * @Copyright(c)liangjilong
 * @Description:
 */
public class RedisClientUtils {

    public static JedisPool jedisPool; // 池化管理 jedis 链接池

    /**
     * 定义代码块=====
     */
    static {
        //读取 redis.properties 配置，ResourceBundle 类不需要加上后缀
        文件,直接放个文件名即可。
        ResourceBundle resourceBundle =
        ResourceBundle.getBundle("redis");
```



```

        int maxActive =
Integer.parseInt(resourceBundle.getString("redis.pool.maxActive"));
//最大激活
        int maxIdle =
Integer.parseInt(resourceBundle.getString("redis.pool.maxIdle"));
//最大 Id
        int maxWait =
Integer.parseInt(resourceBundle.getString("redis.pool.maxWait"));
//最大等待链接

        String ip = resourceBundle.getString("redis.ip");
//读取 ip
        int port =
Integer.parseInt(resourceBundle.getString("redis.port"));
//读取
端口

        //创建 JedisPoolConfig 配置对象
        JedisPoolConfig config = new JedisPoolConfig();
        //设置最大连接数
        config.setMaxTotal(maxActive);
        //设置最大空闲数
        config.setMaxIdle(maxIdle);
        //设置超时时间
        config.setMaxWaitMillis(maxWait);

        //初始化连接池
        jedisPool = new JedisPool(config, ip, port);

    }

    /**
     * 向缓存中设置字符串内容
     * @param key key
     * @param value value
     * @return
     * @throws Exception
     */
    public static boolean setValue(String key,String value)
throws Exception{
        Jedis jedis = null;
        try {
            jedis = jedisPool.getResource();
            jedis.set(key, value);
            return true;
        } catch (Exception e) {

```

```
        e.printStackTrace();
        return false;
    }finally{
        jedisPool.returnResource(jedis);
    }
}

/**
 * 向缓存中设置对象
 * @param key
 * @param value
 * @return
 */
public static boolean setValue(String key, Object value){
    Jedis jedis = null;
    try {
        String objectJson = JSON.toJSONString(value);
        jedis = jedisPool.getResource();
        jedis.set(key, objectJson);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }finally{
        jedisPool.returnResource(jedis);
    }
}

/**
 * 删除缓存中得对象，根据 key
 * @param key
 * @return
 */
public static boolean deleteByKey(String key){
    Jedis jedis = null;
    try {
        jedis = jedisPool.getResource();
        jedis.del(key);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }finally{
        jedisPool.returnResource(jedis);
    }
}
```

```

    }
}

/**
 * 根据 key 获取内容
 * @param key
 * @return
 */
public static Object getValueByKey(String key){
    Jedis jedis = null;
    try {
        jedis = jedisPool.getResource();
        Object value = jedis.get(key);
        return value;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }finally{
        jedisPool.returnResource(jedis);
    }
}

/**
 * 根据 key 获取对象
 * @param key
 * @return
 */
public static <T> T getValueByObject(String key,Class<T>
clazz){
    Jedis jedis = null;
    try {
        jedis = jedisPool.getResource();
        String value = jedis.get(key);
        return JSON.parseObject(value, clazz);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }finally{
        jedisPool.returnResource(jedis);
    }
}
}

```

### 7.3 实体

```
package redis.flong.entity;

import java.io.Serializable;

/**
 * @Author:liangjilong
 * @Date:2015 年 12 月 18 日上午 8:43:14
 * @Email:jilongliang@sina.com
 * @Version:1.0
 * @Copyright(c)liangjilong
 * @Description:
 */
@SuppressWarnings("all")
public class User implements Serializable{
    private int userId;//id

    private int age;//年龄
    private String userName;//名字
    private String sex;//性别
    private String address;//地址
    /**
     * 构造方法
     * @param userId
     * @param age
     * @param userName
     * @param sex
     * @param address
     */
    public User(int userId, int age, String userName, String sex,
String address) {
        super();
        this.userId = userId;
        this.age = age;
        this.userName = userName;
        this.sex = sex;
        this.address = address;
    }

    public User() {
        super();
    }
}
```

```
}

public int getUserId() {
    return userId;
}
public void setUserId(int userId) {
    this.userId = userId;
}
public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}
public String getUserName() {
    return userName;
}
public void setUserName(String userName) {
    this.userName = userName;
}
public String getSex() {
    return sex;
}
public void setSex(String sex) {
    this.sex = sex;
}
public String getAddress() {
    return address;
}
public void setAddress(String address) {
    this.address = address;
}
}
```

## 7.4 Redis 测试类

```
package redis.flong.cache;
import redis.flong.common.RedisClientUtils;
import redis.flong.entity.User;
/**
 * @Author:Liangjilong
 * @Date:2015 年12 月18 日上午8:57:25
 * @Email:jilongliang@sina.com
 * @Version:1.0
 * @Copyright(c)Liangjilong
 * @Description:
 */
public class UserCacheTest {
    /**
     * 设置缓存值=====
     */
    public static void setUserCacheValue(){
        //向构造方法保存值
        User userCache = new User(1001,100,"周伯通","男","神仙鬼谷情居");
        //调用方法处理
        Boolean resultCache = RedisClientUtils.setValue("userCache", userCache);
        if(resultCache) {
            System.out.println("向缓存中保存对象成功。");
        }else{
            System.out.println("向缓存中保存对象失败。");
        }
    }
    /**
     * 获取缓存数据
     */
    public static void getUserCacheValue(){
        //如果使用了一个构造方法是传值的话，必须要提供一个默认无参数的构造方法，否则的话阿里巴巴的fastjson
        //无法解析这个com.alibaba.fastjson.JSONException: default constructor not found. class
        //自己可以去掉User这个实体类的,public User() {super();},而且
        //在开发中，一般提供了有参数和无参数的构造方法.这样就
        //减少错误发生.不用构造方法也可以，可以使用,New一个对象一个
        //一个属性的设值。
        User userCache = RedisClientUtils.getValueByObject("userCache",User.class);
        if(userCache != null){
            System.out.println("从缓存中获取的对象，" + userCache.getUserName() +
            "*****" + userCache.getAddress());
        }
    }
}
```



```
    }  
}  
public static void main(String[] args) {  
    setUserCacheValue();  
    getUserCacheValue();  
}  
}
```

## 八、Redis 集群

### 8.1 集群简介

实现集群的技术好多，比如：Oracle，WebLogic，[SolrCloud](#)，[Dubbo](#)，[Redis](#) 等等

**集群（cluster）**技术是一种较新的技术，通过集群技术，可以在付出较低成本的情况下获得在性能、可靠性、灵活性方面的相对较高的收益，其任务调度则是集群系统中的**核心技术**。

集群是一组相互独立的、通过高速**网络互联**的计算机，它们构成了一个组，并以单一系统的模式加以管理。一个客户与集群相互作用时，集群像是一个独立的**服务器**。集群配置是用于提高可用性和可缩放性。

#### 1 提高性能

一些计算密集型应用，如：天气预报、核试验模拟等，需要计算机要有很强的运算处理能力，现有的技术，即使普通的大型机器计算也很难胜任。这时，一般都使用**计算机集群**技术，集中几十台甚至上百台计算机的运算能力来满足要求。提高处理性能一直是集群技术研究的一个重要目标之一。

#### 2 降低成本

通常一套较好的集群配置，其软硬件开销要超过 100000 美元。但与价值上百万美元的专用**超级计算机**相比已属相当便宜。在达到同样性能的条件下，采用计算机集群比采用同等运算能力的**大型计算机**具有更高的性价比。

#### 3 提高可扩展性

用户若想扩展系统能力，不得不购买更高性能的**服务器**，才能获得额外所需的 CPU 和**存储器**。如果采用集群技术，则只需要将新的服务器加入集群中即可，对于客户来看，服务

无论从连续性还是性能上都几乎没有变化，好像系统在不知不觉中完成了升级。

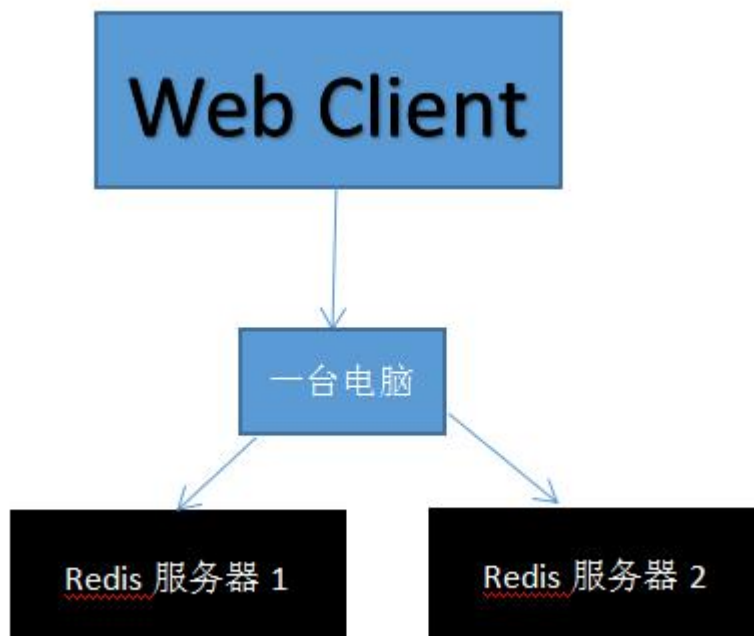
#### 4 增强可靠性

集群技术使系统在故障发生时仍可以继续工作，将系统停运时间减到最小。集群系统提高系统的可靠性的同时，也大大减小了故障损失。

详细说明请看：[百度百科](#)

## 8.2 单机集群

单机集群就只有一台机器运行，但不一定是一个 redis 服务器进行部署，也叫单机版集群。



```
31
32 public static void main(String[] args) throws Exception {
33     //BinaryJedisCluster jc;
34
35     Set<HostAndPort> jedisClusterNodes = new HashSet<HostAndPort>();
36     //jedisClusterNodes.add(new HostAndPort("127.0.0.1", 6379)); //服务1
37     jedisClusterNodes.add(new HostAndPort("112.74.106.211", 7379)); //服务2
38     //jc = new BinaryJedisCluster(jedisClusterNodes);
39     JedisCluster jc = new JedisCluster(jedisClusterNodes);
40     System.out.println(jc);
41
42     //jc.set("M:Key", "M:Value"); //不设置超时 这个key=M:Key, value=M:Value

```

Problems Git Repositories @ Javadoc Declaration Console Progress Servers

<terminated> ClusterHost [Java Application] C:\Program Files\Java\jdk1.7.0\_80\bin\javaw.exe (2015年12月23日 下午4:1  
redis.clients.jedis.JedisCluster@12524b0

在 Window 搭建 Redis 集群没在 linux 搭建集群方便，下面介绍一下 keepalived 介绍，由于本机没 linux 环境，就借助网友的好文章描述整理一下，所谓“授人以鱼，不如授人以渔”

### 8.3 Keepalived 介绍

来源：<http://my.oschina.net/pwd/blog/381212>

参考官方文档：<http://keepalived.org/pdf/sery-lvs-cluster.pdf>

Keepalived 是一个用 c 写的路由选择软件，配合 IPVS 负载均衡实用，通过 VRRP 协议提供高可用。目前最新版本 1.2.7.Keepalived 机器之间实用 VRRP 路由协议切换 VIP，切换速度秒级，且不存在脑裂问题。可以实现

可以实现一主多备，主挂后备自动选举，漂移 VIP，切换速度秒级；切换时可通过运行指定脚本更改业务服务状态。

如两台主机 A、B，可以实现如下切换：

1. A 、B 依次启动，A 作为主、B 为从
- 2.主 A 挂掉，B 接管业务，作为主
- 3.A 起来，作为从 SLAVEOF B
- 4.B 挂掉，A 切回主

将一台全部作为主，即可实现主从，可做读写分离；也可以通过多个 VIP，在一台机器上多个实例中一半主、一半从，实现互备份，两机同时负责部分业务，一台宕机后业务都集中在一台上

安装配置都比较简单：

需要依赖包：openssl-devel（ubuntu 中为 libssl-dev),popt-devel（ubuntu 中为 libpopt-dev）。

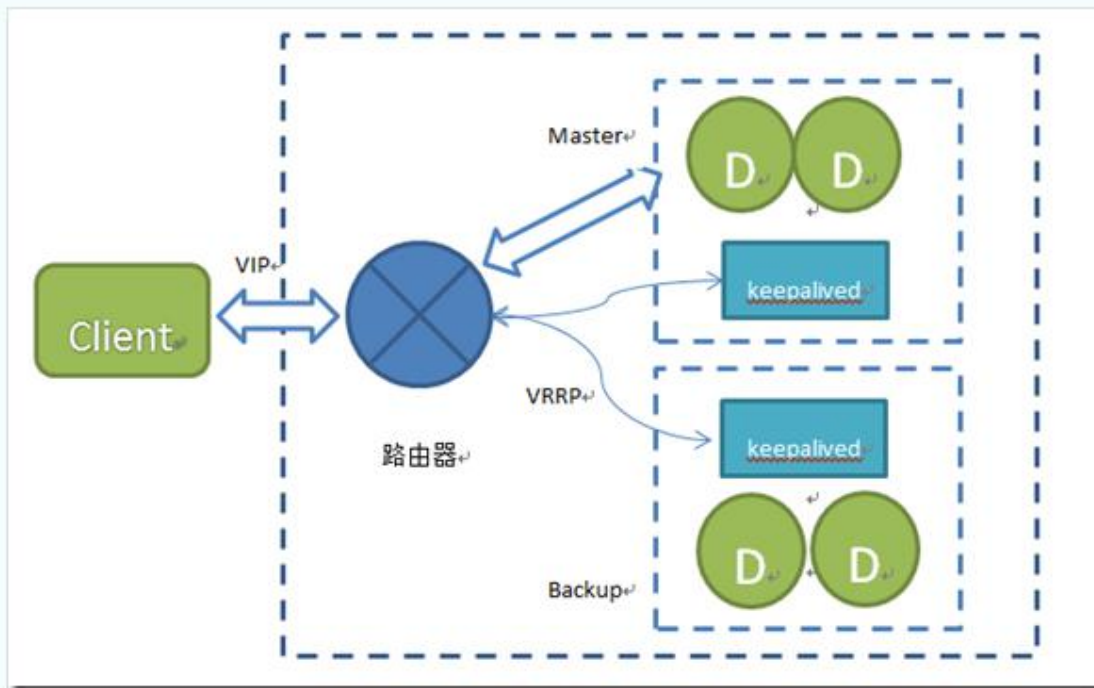
配置文件默认路径：/etc/keepalived/keepalived.conf 也可以手动指定路径，不过要注意的是手动指定需要使用绝对路径。主要要确保配置文件的正确性，keepalived 不会检查配置是否符合规则。

使用 keepalived -D 运行，即可启动 3 个守护进程：一个父进程，一个 check 健康检查，一个 Vrrp，-D 将日志写入/var/log/message，可以通过日志查看切换状况。

注意问题：

1. VRRP 协议是组播协议，需要保证主、备、VIP 都在同一个 VLAN 下
2. 不同的 VIP 需要与不同的 VRID 对应，一个 VLAN 中 VRID 不能和其他组冲突
3. 在 keepalived 有两个角色：Master(一个)、Backup（多个），如果设置一个为 Master，但 Master 挂了后再起来，必然再次业务又一次切换，这对于有 状态服务是不可接受的。解决方案就是两台机器都设置为 Backup，而且优先级高的 Backup 设置为 nopreemt 不抢占。

## 8.4 通过 keepalived 实现的高可用方案



切换流程：

1. 当 Master 挂了后，VIP 漂移到 Slave；Slave 上 keepalived 通知 redis 执行：slaveof no one，开始提供业务
2. 当 Master 起来后，VIP 地址不变，Master 的 keepalived 通知 redis 执行 slaveof slave IP host，开始作为从同步数据
3. 依次类推

主从同时 Down 机情况：

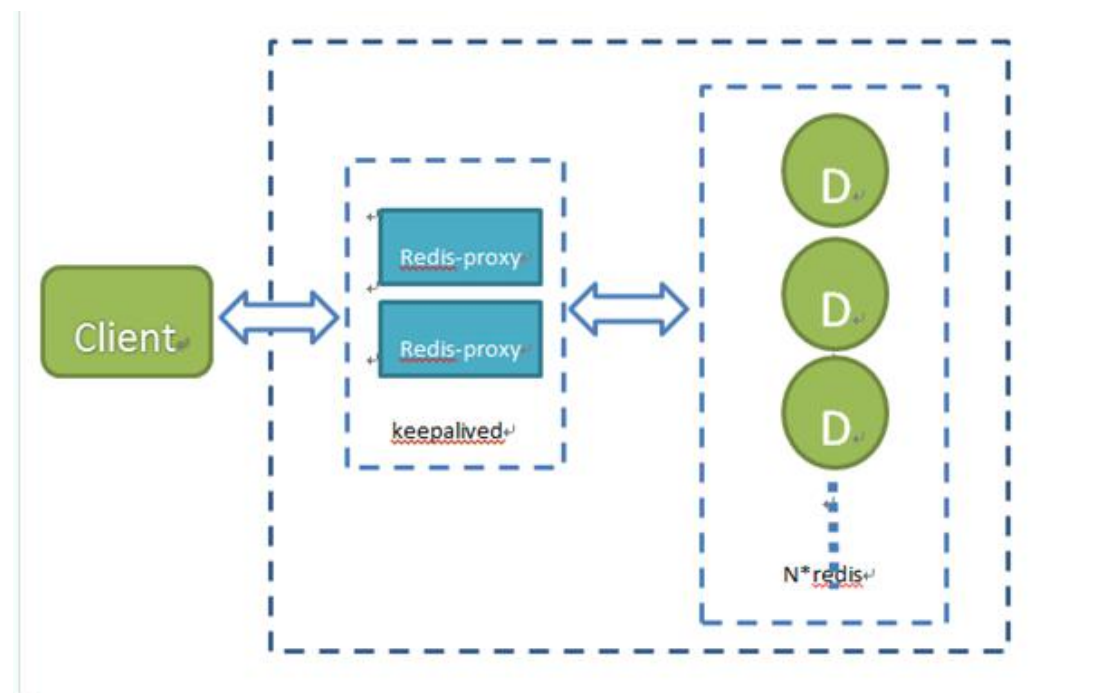
1. 非计划性，不做考虑，一般也不会存在这种问题
2. 计划性重启，重启之前通过运维手段 SAVE DUMP 主库数据；需要注意顺序：
  1. 关闭其中一台机器上所有 redis，使得 master 全部切到另外一台机器（多实例部署，单机上既有主又有从的情况）；并关闭机器
  2. 依次 dump 主上 redis 服务
  3. 关闭主
  4. 启动主，并等待数据 load 完毕
  5. 启动从删除 DUMP 文件（避免重启加载慢）

## 8.5 使用 Twemproxy 实现集群方案

Twemproxy 一个由 twitter 开源的 c 版本 proxy，同时支持 memcached 和 redis，目前最新版本为：0.2.4，持续开发中;<https://github.com/twitter/twemproxy> .twitter 用它主要减少前端与缓存服务间网络连接数。

特点：快、轻量级、减少后端 Cache Server 连接数、易配置、支持 ketama、modula、random、常用 hash 分片算法。

这里使用 keepalived 实现高可用主备方案，解决 proxy 单点问题；



优点：

1. 对于客户端而言，redis 集群是透明的，客户端简单，便于动态扩容
2. Proxy 为单点、处理一致性 hash 时，集群节点可用性检测不存在脑裂问题
3. 高性能，CPU 密集型，而 redis 节点集群多 CPU 资源冗余，可部署在 redis 节点集群上，不需要额外设备

### redis读写瓶颈

匿名 | 浏览 2588 次

举报 | 分享 | 2014-03-21 13:19

硬件

redis做写放测试，服务器是16G内存8核CPU的IBM，为什么在循环写入数据的时候，差不多写50000条左右redis服务就挂了，redis.conf用的是默认的参数。

redis配置文件要怎么调整，数据结构用的是hash，循环写入50000条记录服务就挂了，而且写入的速度非常慢，一秒只写50条左右，大家使用redis的时候，性能情况是怎样的

## 8.7 监控工具

历史 redis 运行查询：CPU、内存、命中率、请求量、主从切换等

实时监控曲线

短信报警

使用基于开源 Redis Live 修改工具，便于批量实例监控，基础功能都已实现，细节也将逐步完善。

源码地址如下：

<https://github.com/LittlePeng/redis-monitor>

## 8.6 多台机集群

Redis Node 由一组 Redis Instance 组成，一组 Redis Instance 可以有一个 Master Instance，多个 Slave Instance

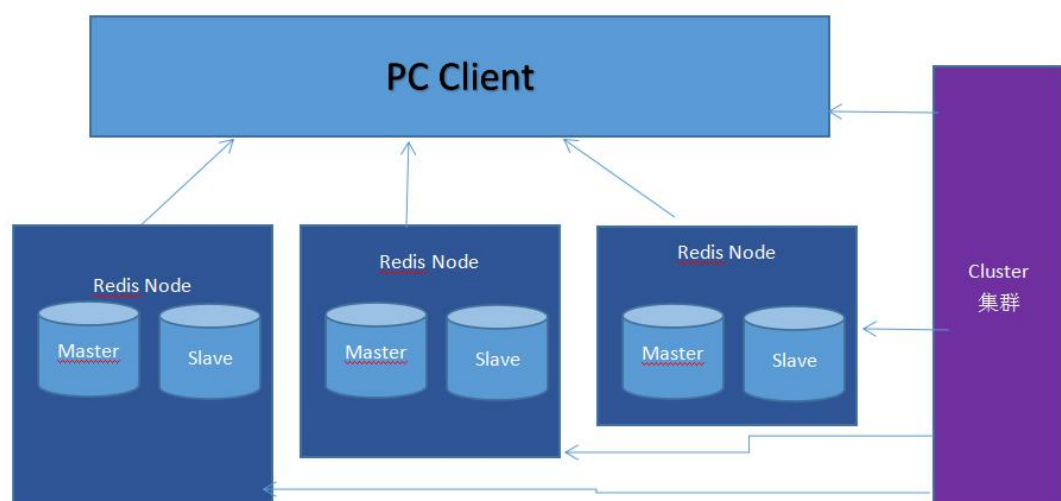
Redis 官方的 cluster 还在 beta 版本，参看 [Redis cluster tutorial](#)

在做调研的时候，曾经特别关注过 KeepAlived+VIP 和 Twemproxy

不过最后还是决定基于 Redis Sentinel 实现一套

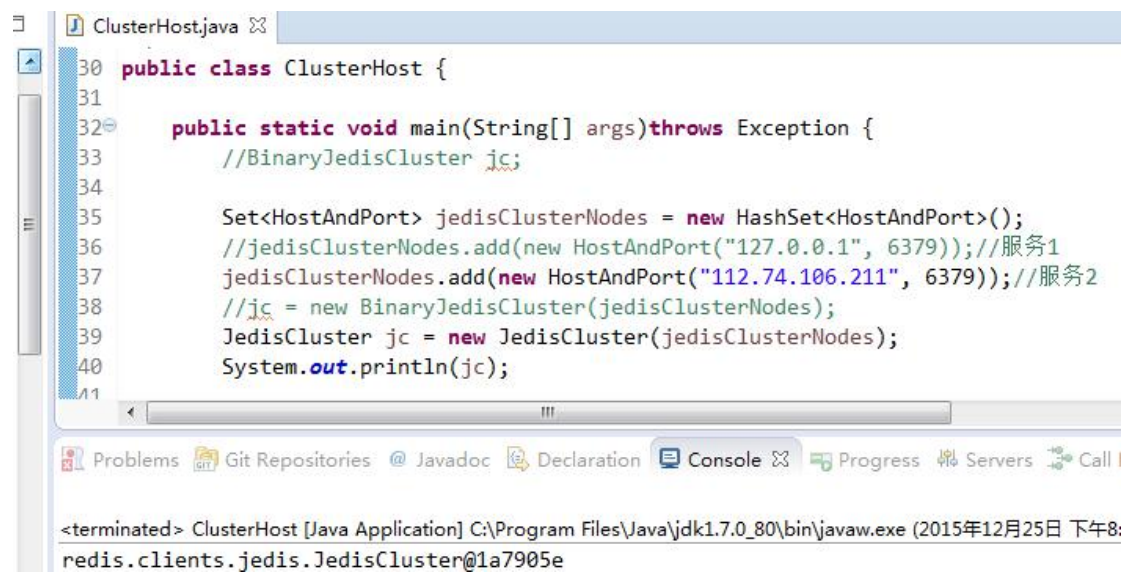
整体设计

1. 数据 Hash 分布在不同的 Redis Instance 上
2. M/S 的切换采用 Sentinel
3. 写：只会写 master Instance，从 sentinel 获取当前的 master Instance
4. 读：从 Redis Node 中基于权重选取一个 Redis Instance 读取，失败/超时则轮询其他 Instance
5. 通过 RPC 服务访问，RPC server 端封装了 Redis 客户端，客户端基于 jedis 开发
6. 批量写/删除：不保证事务





参考博客: <http://hot66hot.iteye.com/blog/2050676>



The screenshot shows an IDE window with a file named `ClusterHost.java`. The code is as follows:

```
30 public class ClusterHost {
31
32     public static void main(String[] args) throws Exception {
33         //BinaryJedisCluster jc;
34
35         Set<HostAndPort> jedisClusterNodes = new HashSet<HostAndPort>();
36         //jedisClusterNodes.add(new HostAndPort("127.0.0.1", 6379)); //服务1
37         jedisClusterNodes.add(new HostAndPort("112.74.106.211", 6379)); //服务2
38         //jc = new BinaryJedisCluster(jedisClusterNodes);
39         JedisCluster jc = new JedisCluster(jedisClusterNodes);
40         System.out.println(jc);
41     }
42 }
```

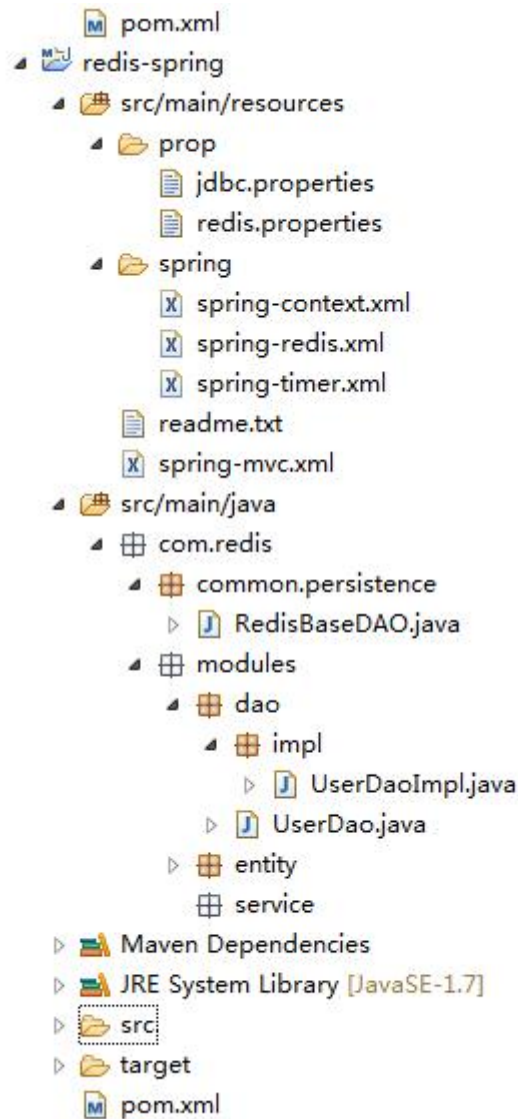
Below the code editor, the console output is visible:

```
<terminated> ClusterHost [Java Application] C:\Program Files\Java\jdk1.7.0_80\bin\javaw.exe (2015年12月25日 下午8:
redis.clients.jedis.JedisCluster@1a7905e
```

## 8.7 Spring+redis 整合

参考博客: <http://snowolf.iteye.com/blog/1666908>

内事问度娘, 外事问谷歌



## Redis.xml

```
<!-- 加载配置属性文件 -->
<context:property-placeholder ignore-unresolvable="true"
location="classpath:/prop/redis.properties" />

<!-- 使用 Annotation 自动注册 Bean，解决事物失效问题：在主容器中不扫描
@Controller 注解，在 SpringMvc 中只扫描@Controller 注解。 -->
<context:component-scan base-package="com.redis">
<!-- base-package 如果多个，用“,”分隔 -->
<context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller" />
</context:component-scan>

<!-- 配置 Annotation 驱动，定义事务 -->
```

```

<tx:annotation-driven transaction-manager="transactionManager"
proxy-target-class="true" />

<bean id="jedisConnectionFactory"
class="org.springframework.data.redis.connection.jedis.JedisConnecti
onFactory">
    <property name="hostName" value="${redis.ip}" />
    <property name="port" value="${redis.port}" />
    <property name="poolConfig" ref="jedisPoolConfig" />
</bean>
<bean class="org.springframework.data.redis.core.RedisTemplate"
p:connection-factory-ref="jedisConnectionFactory" />
<bean id="jedisPoolConfig"
class="redis.clients.jedis.JedisPoolConfig">
    <property name="maxActive" value="${redis.pool.maxActive}" />
    <property name="maxIdle" value="${redis.pool.maxIdle}" />
    <property name="maxWait" value="${redis.pool.maxWait}" />
    <property name="testOnBorrow"
value="${redis.pool.testOnBorrow}" />
</bean>

```

## Redis.properties 配置

```

#最大分配的对象数
redis.pool.maxActive=1024
#最大能够保持 idle 状态的对象数
redis.pool.maxIdle=200
#当池内没有返回对象时，最大等待时间
redis.pool.maxWait=1000
#当调用 borrow Object 方法时，是否进行有效性检查
redis.pool.testOnBorrow=true
#IP
redis.ip=127.0.0.1
#Port
redis.port=6379

```

## Pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>redis-spring</groupId>

```

```
<artifactId>redis-spring</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>redis-spring Maven Webapp</name>
<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
>
<java-version>1.7</java-version>
<spring.version>3.1.1.RELEASE</spring.version>
<junit.version>4.11</junit.version>
<log4j.version>1.2.9</log4j.version>
<mysql.version>5.1.22</mysql.version>
<jdk.version>1.7</jdk.version>
<jetty.version>7.6.9.v20130131</jetty.version>
</properties>

<dependencies>

<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>${junit.version}</version>
<scope>test</scope>
</dependency>

<dependency>
<groupId>org.springframework</groupId>
<artifactId>org.springframework.core</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>org.springframework.web</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>org.springframework.transaction</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
```

```
<groupId>org.springframework</groupId>
<artifactId>org.springframework.orm</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>org.springframework.jdbc</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>org.springframework.web.servlet</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>org.springframework.context</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>org.springframework.aop</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>org.springframework.expression</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>org.springframework.test</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-asm</artifactId>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>${mysql.version}</version>
</dependency>
```

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>${log4j.version}</version>
</dependency>

<dependency>
  <groupId>commons-pool</groupId>
  <artifactId>commons-pool</artifactId>
  <version>1.4</version>
</dependency>
<dependency>
  <groupId>c3p0</groupId>
  <artifactId>c3p0</artifactId>
  <version>0.9.1.2</version>
</dependency>
<dependency>
  <groupId>commons-dbc</groupId>
  <artifactId>commons-dbc</artifactId>
  <version>1.4</version>
</dependency>
<dependency>
  <groupId>commons-collections</groupId>
  <artifactId>commons-collections</artifactId>
  <version>3.2.1</version>
</dependency>
<dependency>
  <groupId>commons-logging</groupId>
  <artifactId>commons-logging</artifactId>
  <version>1.2</version>
</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.7</version>
</dependency>
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.4</version>
</dependency>
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
```



```
        <version>3.1</version>
    </dependency>
    <dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.2.2</version>
    </dependency>

    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.codehaus.jackson</groupId>
        <artifactId>jackson-core-asl</artifactId>
        <version>1.9.2</version>
    </dependency>
    <dependency>
        <groupId>org.codehaus.jackson</groupId>
        <artifactId>jackson-mapper-asl</artifactId>
        <version>1.9.2</version>
    </dependency>
    <dependency>
        <groupId>commons-beanutils</groupId>
        <artifactId>commons-beanutils</artifactId>
        <version>1.8.3</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

    <!-- JSTL -->
    <dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>1.1.2</version>
        <type>jar</type>
    </dependency>
    <dependency>
```

```
<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
<version>1.2</version>
<type>jar</type>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>org.jsoup</groupId>
    <artifactId>jsoup</artifactId>
    <version>1.8.1</version>
</dependency>

<!-- *****Spring-redis***** -->
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-redis</artifactId>
    <version>1.0.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.6.1</version>
</dependency>

<dependency>
    <groupId>org.springframework.integration</groupId>
    <artifactId>spring-integration-redis</artifactId>
    <version>2.1.3.RELEASE</version>
</dependency>

</dependencies>
<build>
    <finalName>redis-spring</finalName>
</build>
</project>
```

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
  <display-name>flong</display-name>

  <!-- 参数初始化类配置 -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      <u>classpath*:spring/spring-context.xml,</u>
      <u>classpath*:spring/spring-redis.xml,</u>
      <u>classpath*:spring/spring-timer.xml</u>
    </param-value>
  </context-param>
  <context-param>
    <param-name>spring.profiles.default</param-name>
    <param-value>production</param-value>
  </context-param>
  <!--
=====encoding=====
  <listener>

  <listener-class>org.springframework.web.context.ContextLoaderListene
r</listener-class>
  </listener>
  <filter>
    <filter-name>encodingFilter</filter-name>

  <filter-class>org.springframework.web.filter.CharacterEncodingFilter
</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
      <param-name>forceEncoding</param-name>
      <param-value>true</param-value>
    </init-param>
  </filter>
  <filter-mapping>

```

```

    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- =====SpringMVC=====
-->
<servlet>
    <servlet-name>SpringMVC</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</se
rvlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>SpringMVC</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>

```

## 九、总结

- 1) 一分耕耘一分收获
- 2) 站在巨人的肩膀上成长
- 3) 少说多动手，
- 4) 互相学习，共同进步