# 课程安排

1. 介绍
2. 安装启动及关闭
3. Jedis使用
4. Redis各种数据类型的操作
5. Redis持久化方案
6. 主从复制(了解)

# 课程目标

掌握Redis的安装

掌握redis常用数据结构(string/hash/list/sortedset)的存取数据方法

掌握Jedis的用法

# redis介绍

## 什么是NoSQL

NoSQL，泛指非关系型的数据库，NoSQL即Not-Only SQL，它可以作为关系型数据库的良好补充。随着互联网web2.0网站的兴起，非关系型的数据库现在成了一个极其热门的新领域，非关系数据库产品的发展非常迅速。而传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，例如：

1、High performance - 对数据库高并发读写的需求

web2.0网站要根据用户个性化信息来实时生成动态页面和提供动态信息，所以基本上无法使用动态页面静态化技术，因此数据库并发负载非常高，往往要达到每秒上万次读写请求。关系数据库应付上万次SQL查询还勉强顶得住，但是应付上万次SQL写数据请求，硬盘IO就已经无法承受了。其实对于普通的BBS网站，往往也存在对高并发写请求的需求，例如网站的实时统计在线用户状态，记录热门帖子的点击次数，投票计数等，因此这是一个相当普遍的需求。

2、Huge Storage - 对海量数据的高效率存储和访问的需求

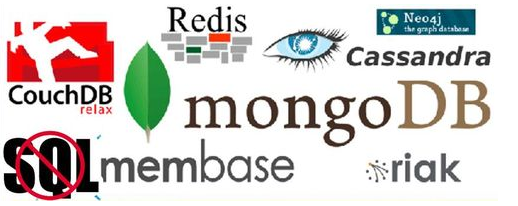
类似Facebook，twitter，Friendfeed这样的SNS网站，每天用户产生海量的用户动态，以Friendfeed为例，一个月就达到了2.5亿条用户动态，对于关系数据库来说，在一张2.5亿条记录的表里面进行SQL查询，效率是极其低下乃至不可忍受的。再例如大型web网站的用户登录系统，例如腾讯，盛大，动辄数以亿计的帐号，关系数据库也很难应付。

3、High Scalability && High Availability- 对数据库的高可扩展性和高可用性的需求

在基于web的架构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，你的数据库却没有办法像web server和app server那样简单的通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供24小时不间断服务的网站来说，对数据库系统进行升级和扩展是非常痛苦的事情，往往需要停机维护和数据迁移，为什么数据库不能通过不断的添加服务器节点来实现扩展呢？

NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。

一些主流的NOSQL产品：



NoSQL数据库的四大分类如下：

* 键值(Key-Value)存储数据库

相关产品： Tokyo Cabinet/Tyrant、Redis、Voldemort、Berkeley DB

典型应用： 内容缓存，主要用于处理大量数据的高访问负载。

数据模型： 一系列键值对

优势： 快速查询

劣势： 存储的数据缺少结构化

* 列存储数据库

相关产品：Cassandra, HBase, Riak

典型应用：分布式的文件系统

数据模型：以列簇式存储，将同一列数据存在一起

优势：查找速度快，可扩展性强，更容易进行分布式扩展

劣势：功能相对局限

* 文档型数据库

相关产品：CouchDB、MongoDB

典型应用：Web应用（与Key-Value类似，Value是结构化的）

数据模型： 一系列键值对

优势：数据结构要求不严格

劣势： 查询性能不高，而且缺乏统一的查询语法

* 图形(Graph)数据库

相关数据库：Neo4J、InfoGrid、Infinite Graph

典型应用：社交网络

数据模型：图结构

优势：利用图结构相关算法。

劣势：需要对整个图做计算才能得出结果，不容易做分布式的集群方案。

## redis历史发展

2008年，意大利的一家创业公司Merzia推出了一款基于MySQL的网站实时统计系统LLOOGG，然而没过多久该公司的创始人 Salvatore Sanfilippo便 对MySQL的性能感到失望，于是他决定亲自为LLOOGG量身定做一个数据库，并于2009年开发完成，这个数据库就是Redis。 不过Salvatore Sanfilippo并不满足只将Redis用于LLOOGG这一款产品，而是希望更多的人使用它，于是在同一年Salvatore Sanfilippo将Redis开源发布，并开始和Redis的另一名主要的代码贡献者Pieter Noordhuis一起继续着Redis的开发，直到今天。

Salvatore Sanfilippo自己也没有想到，短短的几年时间，Redis就拥有了庞大的用户群体。Hacker News在2012年发布了一份数据库的使用情况调查，结果显示有近12%的公司在使用Redis。国内如新浪微博、街旁网、知乎网，国外如GitHub、Stack Overflow、Flickr等都是Redis的用户。

VMware公司从2010年开始赞助Redis的开发， Salvatore Sanfilippo和Pieter Noordhuis也分别在3月和5月加入VMware，全职开发Redis。

## 什么是redis

Redis是用C语言开发的一个开源的高性能键值对（key-value）数据库。它通过提供多种键值数据类型来适应不同场景下的存储需求，目前为止Redis支持的键值数据类型如

下：

字符串类型

散列类型

列表类型

集合类型

有序集合类型。

## redis的应用场景

缓存（数据查询、短连接、新闻内容、商品内容等等）。（最多使用）

分布式集群架构中的session分离。

聊天室的在线好友列表。

任务队列。（秒杀、抢购、12306等等）

应用排行榜。

网站访问统计。

数据过期处理（可以精确到毫秒）

# 测试环境

redis需要安装在linux服务器上运行测试，本教程使用 linux虚拟机及ssh客户端进行功能测试。

## 虚拟机

创建Centos6.4虚拟机作为linux服务器。

虚拟机ip：192.168.93.88

## ssh客户端

在开发环境(windows)安装ssh客户端，本教程使用SecureCRT作为ssh客户端连接虚拟机。

# 安装运行redis

## redis安装环境

redis是C语言开发，建议在linux上运行，本教程使用Centos6.4作为安装环境。

安装redis需要先将官网下载的源码进行编译，编译依赖gcc环境，如果没有gcc环境，需要安装gcc：yum install gcc-c++

## redis安装

* 版本说明

本教程使用redis3.0版本。3.0版本主要增加了redis集群功能。

* 源码下载

从官网下载

http://download.redis.io/releases/redis-3.0.0.tar.gz

将redis-3.0.0.tar.gz拷贝到/usr/local下

* 解压源码

tar -zxvf redis-3.0.0.tar.gz

* 进入解压后的目录进行编译

cd /usr/local/redis-3.0.0

make

* 安装到指定目录,如 /usr/local/redis

cd /usr/local/redis-3.0.0

make PREFIX=/usr/local/redis install

* redis.conf

redis.conf是redis的配置文件，redis.conf在redis源码目录。

注意修改port作为redis进程的端口,port默认6379。

* 拷贝配置文件到安装目录下

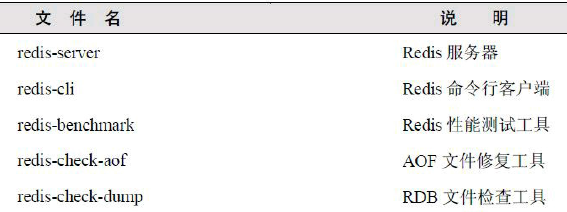
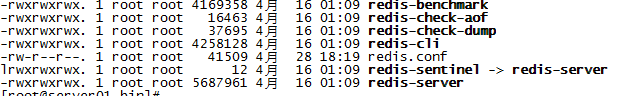
进入源码目录，里面有一份配置文件 redis.conf，然后将其拷贝到安装路径下

cd /usr/local/redis

mkdir conf

cp /usr/local/redis-3.0.0/redis.conf /usr/local/redis/bin

* 安装目录bin下的文件列表



redis3.0新增的redis-sentinel是redis集群管理工具可实现高可用。

配置文件目录：



## redis启动

### 前端模式启动

直接运行bin/redis-server将以前端模式启动，前端模式启动的缺点是ssh命令窗口关闭则redis-server程序结束，不推荐使用此方法。如下图：



### 后端模式启动

修改redis.conf配置文件， daemonize yes 以后端模式启动。

执行如下命令启动redis：

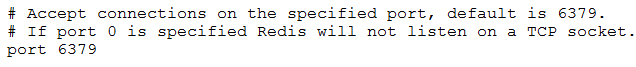
cd /usr/local/redis

./bin/redis-server ./redis.conf

redis默认使用6379端口。



也可更改redis.conf文件，修改端口号：



### 启动多个redis进程

方法1：

启动时指定端口可在一台服务器启动多个redis进程。

cd /usr/local/redis/bin

./redis-server ./redis.conf --port 6380

方法2(推荐此方法)：

创建多个redis目录，以端口号命名，比如：创建6379、6380两个目录，将redis的安装文件bin和conf拷贝至这两个目录。

修改6379目录下的redis.conf设置端口号为6379

修改6380目录下的redis.conf设置端口号为6380

启动6379和6380目录下的redis-server程序：

cd 6379

./redis-server . /redis.conf

cd 6380

./redis-server . /redis.conf

查询当前redis的进程：



## redis停止

强行终止Redis进程可能会导致redis持久化数据丢失。正确停止Redis的方式应该是向Redis发送SHUTDOWN命令，方法为：

cd /usr/local/redis

./bin/redis-cli shutdown

## redis客户端

在redis的安装目录中有redis的客户端，即redis-cli（Redis Command Line Interface），它是Redis自带的基于命令行的Redis客户端。

### 连接redis服务端

执行bin/redis-cli连接redis服务端：



从上图得知redis-cli默认连接本机的redis，本机的redis没有启动则报上图中的错误。

指定连接redis服务的ip和端口：



### 向redis服务端发送命令

redis-cli连上redis服务后，可以在命令行发送命令。

* ping

Redis提供了PING命令来测试客户端与Redis的连接是否正常，如果连接正常会收到回复PONG



* set/get

使用set和get可以向redis设置数据、获取数据。



* del

删除指定key的内容。

例如：del name

* Keys \*

查看当前库中所有的key值

## redis多数据库

### redis实例

一个redis进程就是一个redis实例，一台服务器可以同时有多个redis实例，不同的redis实例提供不同的服务端口对外提供服务，每个redis实例之间互相影响。每个redis实例都包括自己的数据库，数据库中可以存储自己的数据。

### 多数据库测试

一个Redis实例可以包括多个数据库，客户端可以指定连接某个redis实例的哪个数据库，就好比一个mysql中创建多个数据库，客户端连接时指定连接哪个数据库。

一个redis实例最多可提供16个数据库，下标从0到15，客户端默认连接第0号数据库，也可以通过select选择连接哪个数据库，如下连接1号库：



在1号库中查询上节设置的数据，结果查询不到：



重新选择第0号数据库，查询数据：



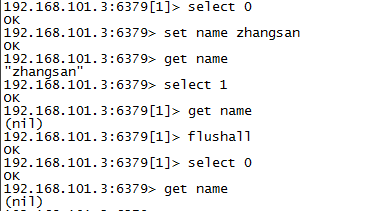
如果选择一个不存在数据库则会报错：



注意：redis不支持修改数据库的名称，只能通过select 0、select 1...选择数据库。

### 注意问题

在0号数据库存储数据，在1号数据库执行清空数据命令却把0号数据库的数据给清空了：



**建议：不同的应用系统要使用不同的redis实例而不是使用同一个redis实例下的不同数据库。**

# jedis

## jedis介绍

Redis不仅是使用命令来操作，现在基本上主流的语言都有客户端支持，比如java、C、C#、C++、php、Node.js、Go等。

在官方网站里列一些Java的客户端，有Jedis、Redisson、Jredis、JDBC-Redis、等其中官方推荐使用Jedis和Redisson。 在企业中用的最多的就是Jedis，下面我们就重点学习下Jedis。

Jedis同样也是托管在github上，地址：<https://github.com/xetorthio/jedis>

## 通过jedis连接redis单机

### jar包

pom坐标：

<dependency>

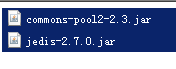
<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.7.0</version>

</dependency>

jar包如下：



### 单实例连接

通过创建单实例jedis对象连接redis服务，如下代码：

// 单实例连接redis

@Test

**public** **void** testJedisSingle() {

Jedis jedis = **new** Jedis("192.168.101.3", 6379);

jedis.set("name", "bar");

String name = jedis.get("name");

System.*out*.println(name);

jedis.close();

}

#### 连接超时解决

由于linux防火墙默认开启，redis的服务端口6379并不在开放规则之内，所有需要将此端口开放访问或者关闭防火墙。

关闭防火墙命令：sevice iptables stop

如果是修改防火墙规则，可以修改：/etc/sysconfig/iptables文件

### 使用连接池连接

通过单实例连接redis不能对redis连接进行共享，可以使用连接池对redis连接进行共享，提高资源利用率，使用jedisPool连接redis服务，如下代码：

@Test

**public** **void** pool() {

JedisPoolConfig config = **new** JedisPoolConfig();

//最大连接数

config.setMaxTotal(30);

//最大连接空闲数

config.setMaxIdle(2);

JedisPool pool = **new** JedisPool(config, "192.168.101.3", 6379);

Jedis jedis = **null**;

**try** {

jedis = pool.getResource();

jedis.set("name", "lisi");

String name = jedis.get("name");

System.*out*.println(name);

}**catch**(Exception ex){

ex.printStackTrace();

}**finally**{

**if**(jedis != **null**){

//关闭连接

jedis.close();

}

}

}

详细的连接池配置参数参考下节jedis和spring整合中applicationContext.xml的配置内容。

### jedis与spring整合

配置spring配置文件applicationContext.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.2.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-3.2.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-3.2.xsd ">

<!-- 连接池配置 -->

<bean id=*"jedisPoolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>

<!-- 最大连接数 -->

<property name=*"maxTotal"* value=*"30"* />

<!-- 最大空闲连接数 -->

<property name=*"maxIdle"* value=*"10"* />

<!-- 每次释放连接的最大数目 -->

<property name=*"numTestsPerEvictionRun"* value=*"1024"* />

<!-- 释放连接的扫描间隔（毫秒） -->

<property name=*"timeBetweenEvictionRunsMillis"* value=*"30000"* />

<!-- 连接最小空闲时间 -->

<property name=*"minEvictableIdleTimeMillis"* value=*"1800000"* />

<!-- 连接空闲多久后释放, 当空闲时间>该值 且 空闲连接>最大空闲连接数 时直接释放 -->

<property name=*"softMinEvictableIdleTimeMillis"* value=*"10000"* />

<!-- 获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1 -->

<property name=*"maxWaitMillis"* value=*"1500"* />

<!-- 在获取连接的时候检查有效性, 默认false -->

<property name=*"testOnBorrow"* value=*"true"* />

<!-- 在空闲时检查有效性, 默认false -->

<property name=*"testWhileIdle"* value=*"true"* />

<!-- 连接耗尽时是否阻塞, false报异常,ture阻塞直到超时, 默认true -->

<property name=*"blockWhenExhausted"* value=*"false"* />

</bean>

<!-- redis单机 通过连接池 -->

<bean id=*"jedisPool"* class=*"redis.clients.jedis.JedisPool"* destroy-method*="close"*>

<constructor-arg name="poolConfig" ref="jedisPoolConfig"/>

<constructor-arg name="host" value="192.168.25.145"/>

<constructor-arg name="port" value="6379"/>

</bean>

</beans>

测试代码：

**private** ApplicationContext applicationContext;

@Before

**public** **void** init() {

applicationContext = **new** ClassPathXmlApplicationContext(

"classpath:applicationContext.xml");

}

@Test

**public** **void** testJedisPool() {

JedisPool pool = (JedisPool) applicationContext.getBean("jedisPool");

**try** {

Jedis jedis = pool.getResource();

jedis.set("name", "lisi");

String name = jedis.get("name");

System.*out*.println(name);

}**catch**(Exception ex){

ex.printStackTrace();

}**finally**{

**if**(jedis != **null**){

//关闭连接

jedis.close();

}

}

}

# 数据类型--string

## redis string介绍

redis中没有使用C语言的字符串表示，而是自定义一个数据结构叫**SDS（simple dynamic string）即简单动态字符串**。

打开下载的redis源码包，找到src下的sds.h文件查看sds源码：

struct sdshdr {

//字符串长度

unsigned int len;

//buf数组中未使用的字节数量

unsigned int free;

//用于保存字符串

char buf[];

};

c语言对字符串的存储是使用字符数组，遇到'\0'字符则认为字符串结束，redis的字符串可以存储任何类型的数据，因为任何类型数据都可以表示成二进制，sds结构中的char buf[]就是存储了二进制数据。

redis的字符串是二进制安全的，什么是二进制安全？简单理解就是存入什么数据取出的还是什么数据。redis中的sds不像c语言处理字符串那样遇到'\0'字符则认证字符串结束，它不会对存储进去的二进制数据进行处理，存入什么数据取出还是什么数据。

## 命令

### 赋值

*SET key value*

*127.0.0.1:6379> set test 123*

*OK*

### 取值

赋值与取值：

*GET key*

*127.0.0.1:6379> get test*

*"123“*

当键不存在时返回空结果。

-------------------

GETSET key value

取值时同时对key进行赋值操作。

### 删除

Del key

*127.0.0.1:6379> del test*

(integer) 1

### 数值增减

* 递增数字

*INCR key*

当存储的字符串是整数时，Redis提供了一个实用的命令INCR，其作用是让当前键值递增，并返回递增后的值。

127.0.0.1:6379> incr num

(integer) 1

127.0.0.1:6379> incr num

(integer) 2

127.0.0.1:6379> incr num

(integer) 3

* 增加指定的整数

*INCRBY key increment*

示例：

127.0.0.1:6379> incrby num 2

(integer) 5

127.0.0.1:6379> incrby num 2

(integer) 7

127.0.0.1:6379> incrby num 2

(integer) 9

* 递减数值

*DECR key*

### 其它命令(自学)

* 减少指定的整数

*DECRBY key decrement*

示例：

127.0.0.1:6379> decr num

(integer) 6

127.0.0.1:6379> decr num

(integer) 5

127.0.0.1:6379> decrby num 3

(integer) 2

127.0.0.1:6379> decrby num 3

(integer) -1

* 向尾部追加值

*APPEND key value*

APPEND的作用是向键值的末尾追加value。如果键不存在则将该键的值设置为value，即相当于 SET key value。返回值是追加后字符串的总长度。

127.0.0.1:6379> set str hello

OK

127.0.0.1:6379> append str " world!"

(integer) 12

127.0.0.1:6379> get str

"hello world!"

* 获取字符串长度

*STRLEN key*

STRLEN命令返回键值的长度，如果键不存在则返回0。

*127.0.0.1:6379> strlen str*

*(integer) 0*

*127.0.0.1:6379> set str hello*

*OK*

*127.0.0.1:6379> strlen str*

*(integer) 5*

* 同时设置/获取多个键值

*MSET key value [key value …]*

*MGET key [key …]*

127.0.0.1:6379> mset k1 v1 k2 v2 k3 v3

OK

127.0.0.1:6379> get k1

"v1"

127.0.0.1:6379> mget k1 k3

1) "v1"

2) "v3"

## 应用

### 自增主键

商品编号、订单号采用string的递增数字特性生成。

定义商品编号key：items:id

192.168.101.3:7003> INCR items:id

(integer) 2

192.168.101.3:7003> INCR items:id

(integer) 3

java代码参考测试工程。

# 数据类型--hash

## 使用string的问题

假设有User对象以JSON序列化的形式存储到Redis中，User对象有id，username、password、age、name等属性，存储的过程如下：

保存、更新：

User对象 🡪 json(string) 🡪 redis

如果在业务上只是更新age属性，其他的属性并不做更新我应该怎么做呢？ 如果仍然采用上边的方法在传输、处理时会造成资源浪费，下边讲的hash可以很好的解决这个问题。

## redis hash介绍

hash叫散列类型，它提供了字段和字段值的映射。字段值只能是字符串类型，不支持散列类型、集合类型等其它类型。如下：



思考：redis hash存储比关系数据库的好处？

## 命令

### 赋值

*HSET key field value 一次只能设置一个字段值*

*127.0.0.1:6379> hset user username zhangsan*

*(integer) 1*

*-----------------------------*

*HMSET key field value [field value ...] 一次可以设置多个字段值*

*127.0.0.1:6379> hmset user age 20 username lisi*

*OK*

### 取值

*HGET key field 一次只能获取一个字段值*

*127.0.0.1:6379> hget user username*

*"zhangsan“*

*----------------------------*

*HMGET key field [field ...] 一次可以获取多个字段值*

*127.0.0.1:6379> hmget user age username*

*1) "20"*

*2) "lisi"*

*----------------------------*

*HGETALL key*

*127.0.0.1:6379> hgetall user*

*1) "age"*

*2) "20"*

*3) "username"*

*4) "lisi"*

HSET命令不区分插入和更新操作，当执行插入操作时HSET命令返回1，当执行更新操作时返回0.

### 删除字段

**可以删除一个或多个字段，返回值是被删除的字段个数**

*HDEL key field [field ...]*

*127.0.0.1:6379> hdel user age*

*(integer) 1*

*127.0.0.1:6379> hdel user age name*

*(integer) 0*

*127.0.0.1:6379> hdel user age username*

*(integer) 1*

### 增加数字

*HINCRBY key field increment*

*127.0.0.1:6379> hincrby user age 2 将用户的年龄加2*

*(integer) 22*

*127.0.0.1:6379> hget user age 获取用户的年龄*

*"22“*

### 其它命令(自学)

* 判断字段是否存在

*HEXISTS key field*

*127.0.0.1:6379> hexists user age 查看user中是否有age字段*

*(integer) 1*

*127.0.0.1:6379> hexists user name 查看user中是否有name字段*

*(integer) 0*

*HSETNX key field value*

当**字段**不存在时赋值，类似HSET，区别在于如果字段已经存在，该命令不执行任何操作。

*127.0.0.1:6379> hsetnx user age 30 如果user中没有age字段则设置age值为30，否则不做任何操作*

*(integer) 0*

* 只获取字段名或字段值

*HKEYS key*

*HVALS key*

*127.0.0.1:6379> hmset user age 20 name lisi*

*OK*

*127.0.0.1:6379> hkeys user*

*1) "age"*

*2) "name"*

*127.0.0.1:6379> hvals user*

*1) "20"*

*2) "lisi"*

* 获取字段数量

*HLEN key*

*127.0.0.1:6379> hlen user*

*(integer) 2*

## 应用

### 商品信息

商品id、商品名称、商品描述、商品库存、商品好评

定义商品信息的key：

商品1001的信息在 redis中的key为：items:1001

存储商品信息

192.168.101.3:7003> HMSET items:1001 id 3 name apple price 999.9

OK

获取商品信息

192.168.101.3:7003> HGET items:1001 id

"3"

192.168.101.3:7003> HGETALL items:1001

1) "id"

2) "3"

3) "name"

4) "apple"

5) "price"

6) "999.9"

java代码参考测试工程。

# 数据类型--list

## ArrayList与LinkedList的区别

ArrayList使用数组方式存储数据，所以根据索引查询数据速度快，而新增或者删除元素时需要设计到位移操作，所以比较慢。

LinkedList使用双向链接方式存储数据，每个元素都记录前后元素的指针，所以插入、删除数据时只是更改前后元素的指针指向即可，速度非常快，然后通过下标查询元素时需要从头开始索引，所以比较慢，但是如果查询前几个元素或后几个元素速度比较快。



## redis list介绍

列表类型（list）可以存储一个有序的字符串列表，常用的操作是向列表两端添加元素，或者获得列表的某一个片段。

列表类型内部是使用双向链表（double linked list）实现的，所以向列表两端添加元素的时间复杂度为0(1)，获取越接近两端的元素速度就越快。这意味着即使是一个有几千万个元素的列表，获取头部或尾部的10条记录也是极快的。

## 命令

### 向列表两端增加元素。

*LPUSH key value [value ...]*

*RPUSH key value [value ...]*

*向列表左边增加元素*

*127.0.0.1:6379> lpush list:1 1 2 3*

*(integer) 3*

*向列表右边增加元素*

*127.0.0.1:6379> rpush list:1 4 5 6*

*(integer) 3*

### 查看列表

*LRANGE key start stop*

LRANGE命令是列表类型最常用的命令之一，获取列表中的某一片段，将返回start、stop之间的所有元素（包含两端的元素），索引从0开始。索引可以是负数，如：“-1”代表最后边的一个元素。

*127.0.0.1:6379> lrange list:1 0 2*

*1) "2" -> 3*

*2) "1" ->2*

*3) "4" ->1*

### 从列表两端弹出元素

*LPOP key*

*RPOP key*

LPOP命令从列表左边弹出一个元素，会分两步完成，第一步是将列表左边的元素从列表中移除，第二步是返回被移除的元素值。

127.0.0.1:6379> lpop list:1

"3“

127.0.0.1:6379> rpop list:1

"6“

### 获取列表中元素的个数

*LLEN key*

*127.0.0.1:6379> llen list:1*

*(integer) 2*

### 其它命令(自学)

* 删除列表中指定的值

*LREM key count value*

LREM命令会删除列表中前count个值为value的元素，返回实际删除的元素个数。根据count值的不同，该命令的执行方式会有所不同：

* 当count>0时， LREM会从列表左边开始删除。
* 当count<0时， LREM会从列表后边开始删除。
* 当count=0时， LREM删除所有值为value的元素。
* 获得/设置指定索引的元素值

LINDEX key index

LSET key index value

127.0.0.1:6379> lindex l:list 2

"1"

127.0.0.1:6379> lset l:list 2 2

OK

127.0.0.1:6379> lrange l:list 0 -1

1) "6"

2) "5"

3) "2"

4) "2"

* 只保留列表指定片段，指定范围和LRANGE一致

*LTRIM key start stop*

*127.0.0.1:6379> lrange l:list 0 -1*

*1) "6"*

*2) "5"*

*3) "0"*

*4) "2"*

*127.0.0.1:6379> ltrim l:list 0 2*

*OK*

*127.0.0.1:6379> lrange l:list 0 -1*

*1) "6"*

*2) "5"*

*3) "0"*

* 向列表中插入元素

*LINSERT key BEFORE|AFTER pivot value*

该命令首先会在列表中从左到右查找值为pivot的元素，然后根据第二个参数是BEFORE还是AFTER来决定将value插入到该元素的前面还是后面。

127.0.0.1:6379> lrange list 0 -1

1) "3"

2) "2"

3) "1"

127.0.0.1:6379> linsert list after 3 4

(integer) 4

127.0.0.1:6379> lrange list 0 -1

1) "3"

2) "4"

3) "2"

4) "1"

* 将元素从一个列表转移到另一个列表中

*RPOPLPUSH source destination*

*127.0.0.1:6379> rpoplpush list newlist*

*"1"*

*127.0.0.1:6379> lrange newlist 0 -1*

*1) "1"*

*127.0.0.1:6379> lrange list 0 -1*

*1) "3"*

*2) "4"*

*3) "2"*

## 应用

### 商品评论列表

思路：

在redis中创建商品评论列表

用户发布商品评论，将评论信息转成json存储到list中。

用户在页面查询评论列表，从redis中取出json数据展示到页面。

定义商品评论列表key：

商品编号为1001的商品评论key：items: comment:1001

192.168.101.3:7001> LPUSH items:comment:1001 '{"id":1,"name":"商品不错，很好！！","date":1430295077289}'

java代码参考测试工程。

# 数据类型--set

## redis set介绍

在集合中的每个元素都是不同的，且没有顺序。

集合类型和列表类型的对比：



集合类型的常用操作是向集合中加入或删除元素、判断某个元素是否存在等，由于集合类型的Redis内部是使用值为空的散列表实现，所有这些操作的时间复杂度都为0(1)。

Redis还提供了多个集合之间的交集、并集、差集的运算。

## 命令

### 增加/删除元素

*SADD key member [member ...]*

*SREM key member [member ...]*

*127.0.0.1:6379> sadd set a b c*

*(integer) 3*

*127.0.0.1:6379> sadd set a*

*(integer) 0*

*127.0.0.1:6379> srem set c d*

*(integer) 1*

### 获得集合中的所有元素

*SMEMBERS key*

127.0.0.1:6379> smembers set

1) "b"

2) "a”

判断元素是否在集合中，无论集合中有多少元素都可以极速的返回结果。

*SISMEMBER key member*

127.0.0.1:6379> sismember set a

(integer) 1

127.0.0.1:6379> sismember set h

(integer) 0

## 其它命令

### 集合的差集运算 A-B

属于A并且不属于B的元素构成的集合。



*SDIFF key [key ...]*

*127.0.0.1:6379> sadd setA 1 2 3*

*(integer) 3*

*127.0.0.1:6379> sadd setB 2 3 4*

*(integer) 3*

*127.0.0.1:6379> sdiff setA setB*

*1) "1"*

*127.0.0.1:6379> sdiff setB setA*

*1) "4"*

### 集合的交集运算 A ∩ B

属于A且属于B的元素构成的集合。



SINTER key [key ...]

127.0.0.1:6379> sinter setA setB

1) "2"

2) "3"

### 集合的并集运算 A ∪ B

属于A或者属于B的元素构成的集合



SUNION key [key ...]

*127.0.0.1:6379> sunion setA setB*

*1) "1"*

*2) "2"*

*3) "3"*

*4) "4"*

### 其它命令(自学)

* 获得集合中元素的个数

*SCARD key*

127.0.0.1:6379> smembers setA

1) "1"

2) "2"

3) "3"

127.0.0.1:6379> scard setA

(integer) 3

从集合中弹出一个元素

*SPOP key*

*127.0.0.1:6379> spop setA*

*"1“*

注意：由于集合是无序的，所有SPOP命令会从集合中随机选择一个元素弹出

# 数据类型--sorted set

## redis sorted set介绍

在集合类型的基础上有序集合类型为集合中的每个元素都关联一个分数，这使得我们不仅可以完成插入、删除和判断元素是否存在在集合中，还能够获得分数最高或最低的前N个元素、获取指定分数范围内的元素等与分数有关的操作。

在某些方面有序集合和列表类型有些相似。

1、二者都是有序的。

2、二者都可以获得某一范围的元素。

但是，二者有着很大区别：

1、列表类型是通过链表实现的，获取靠近两端的数据速度极快，而当元素增多后，访问中间数据的速度会变慢。

2、有序集合类型使用散列表实现，所有即使读取位于中间部分的数据也很快。

3、列表中不能简单的调整某个元素的位置，但是有序集合可以（通过更改分数实现）

4、有序集合要比列表类型更耗内存。

## 命令

### 增加元素

向有序集合中加入一个元素和该元素的分数，如果该元素已经存在则会用新的分数替换原有的分数。返回值是**新**加入到集合中的元素个数，不包含之前已经存在的元素。

*ZADD key score member [score member ...]*

*127.0.0.1:6379> zadd scoreboard 80 zhangsan 89 lisi 94 wangwu*

*(integer) 3*

*127.0.0.1:6379> zadd scoreboard 97 lisi*

*(integer) 0*

获取元素的分数

ZSCORE key member

127.0.0.1:6379> zscore scoreboard lisi

"97"

### 删除元素

ZREM key member [member ...]

移除有序集key中的一个或多个成员，不存在的成员将被忽略。

当key存在但不是有序集类型时，返回一个错误。

127.0.0.1:6379> zrem scoreboard lisi

(integer) 1

### 获得排名在某个范围的元素列表

获得排名在某个范围的元素列表

*ZRANGE key start stop [WITHSCORES]* 照元素分数从小到大的顺序返回索引从start到stop之间的所有元素（包含两端的元素）

127.0.0.1:6379> zrange scoreboard 0 2

1) "zhangsan"

2) "wangwu"

3) "lisi“

*ZREVRANGE key start stop [WITHSCORES]* 照元素分数从大到小的顺序返回索引从start到stop之间的所有元素（包含两端的元素）

127.0.0.1:6379> zrevrange scoreboard 0 2

1) " lisi "

2) "wangwu"

3) " zhangsan “

如果需要获得元素的分数的可以在命令尾部加上*WITHSCORES参数*

127.0.0.1:6379> zrange scoreboard 0 1 WITHSCORES

1) "zhangsan"

2) "80"

3) "wangwu"

4) "94"

### 其它命令(自学)

* 获得指定分数范围的元素

*ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]*

*127.0.0.1:6379> ZRANGEBYSCORE scoreboard 90 97 WITHSCORES*

*1) "wangwu"*

*2) "94"*

*3) "lisi"*

*4) "97"*

*127.0.0.1:6379> ZRANGEBYSCORE scoreboard 70 100 limit 1 2*

*1) "wangwu"*

*2) "lisi"*

* 增加某个元素的分数，返回值是更改后的分数。

*ZINCRBY key increment member*

给lisi加4分

127.0.0.1:6379> ZINCRBY scoreboard 4 lisi

"101“

* 获得集合中元素的数量

*ZCARD key*

127.0.0.1:6379> ZCARD scoreboard

(integer) 3

* 获得指定分数范围内的元素个数

*ZCOUNT key min max*

*127.0.0.1:6379> ZCOUNT scoreboard 80 90*

*(integer) 1*

* 按照排名范围删除元素

*ZREMRANGEBYRANK key start stop*

*127.0.0.1:6379> ZREMRANGEBYRANK scoreboard 0 1*

*(integer) 2*

*127.0.0.1:6379> ZRANGE scoreboard 0 -1*

*1) "lisi"*

*ZREMRANGEBYSCORE key min max*

按照分数范围删除元素

127.0.0.1:6379> zadd scoreboard 84 zhangsan

(integer) 1

127.0.0.1:6379> ZREMRANGEBYSCORE scoreboard 80 100

(integer) 1

* 获取元素的排名

*ZRANK key member*

*ZREVRANK key member*

从小到大

127.0.0.1:6379> ZRANK scoreboard lisi

(integer) 0

从大到小

127.0.0.1:6379> ZREVRANK scoreboard zhangsan

(integer) 1

## 应用

### 商品销售排行榜

根据商品销售量对商品进行排行显示，定义sorted set集合，商品销售量为元素的分数。

定义商品销售排行榜key：items:sellsort

写入商品销售量：

商品编号1001的销量是9，商品编号1002的销量是10

192.168.101.3:7007> ZADD items:sellsort 9 1001 10 1002

商品编号1001的销量加1

192.168.101.3:7001> ZINCRBY items:sellsort 1 1001

商品销量前10名：

192.168.101.3:7001> ZRANGE items:sellsort 0 9 withscores

java代码参考测试工程。

# keys命令

## 设置key的生存时间

Redis在实际使用过程中更多的用作缓存，然而缓存的数据一般都是需要设置生存时间的，即：到期后数据销毁。

*EXPIRE key seconds 设置key的生存时间（单位：秒）key在多少秒后会自动删除*

*TTL key 查看key生于的生存时间*

*PERSIST key 清除生存时间*

*PEXPIRE key milliseconds 生存时间设置单位为：毫秒*

*例子：*

*192.168.101.3:7002> set test 1 设置test的值为1*

*OK*

*192.168.101.3:7002> get test 获取test的值*

*"1"*

*192.168.101.3:7002> EXPIRE test 5 设置test的生存时间为5秒*

*(integer) 1*

*192.168.101.3:7002> TTL test 查看test的生于生成时间还有1秒删除*

*(integer) 1*

*192.168.101.3:7002> TTL test*

*(integer) -2*

*192.168.101.3:7002> get test 获取test的值，已经删除*

*(nil)*

## 其它命令（自学）

* keys

返回满足给定pattern 的所有key

redis 127.0.0.1:6379> keys mylist\*

1) "mylist"

2) "mylist5"

3) "mylist6"

4) "mylist7"

5) "mylist8"

* exists

确认一个key 是否存在

redis 127.0.0.1:6379> exists HongWan

(integer) 0

redis 127.0.0.1:6379> exists age

(integer) 1

redis 127.0.0.1:6379>

从结果来数据库中不存在HongWan 这个key，但是age 这个key 是存在的

* del

删除一个key

redis 127.0.0.1:6379> del age

(integer) 1

redis 127.0.0.1:6379> exists age

(integer) 0

redis 127.0.0.1:6379>

从结果来数据库中不存在HongWan 这个key，但是age 这个key 是存在的

* rename

重命名key

redis 127.0.0.1:6379[1]> keys \*

1) "age"

redis 127.0.0.1:6379[1]> rename age age\_new

OK

redis 127.0.0.1:6379[1]> keys \*

1) "age\_new"

redis 127.0.0.1:6379[1]>

age 成功的被我们改名为age\_new 了

* type

返回值的类型

redis 127.0.0.1:6379> type addr

string

redis 127.0.0.1:6379> type myzset2

zset

redis 127.0.0.1:6379> type mylist

list

redis 127.0.0.1:6379>

这个方法可以非常简单的判断出值的类型

# 服务器命令(自学)

* ping

测试连接是否存活

redis 127.0.0.1:6379> ping

PONG

//执行下面命令之前，我们停止redis 服务器

redis 127.0.0.1:6379> ping

Could not connect to Redis at 127.0.0.1:6379: Connection refused

//执行下面命令之前，我们启动redis 服务器

not connected> ping

PONG

redis 127.0.0.1:6379>

第一个ping 时，说明此连接正常

第二个ping 之前，我们将redis 服务器停止，那么ping 是失败的

第三个ping 之前，我们将redis 服务器启动，那么ping 是成功的

* echo

在命令行打印一些内容

redis 127.0.0.1:6379> echo HongWan

"HongWan"

redis 127.0.0.1:6379>

* select

选择数据库。Redis 数据库编号从0~15，我们可以选择任意一个数据库来进行数据的存取。

redis 127.0.0.1:6379> select 1

OK

redis 127.0.0.1:6379[1]> select 16

(error) ERR invalid DB index

redis 127.0.0.1:6379[16]>

当选择16 时，报错，说明没有编号为16 的这个数据库

* quit

退出连接。

redis 127.0.0.1:6379> quit

* dbsize

返回当前数据库中key 的数目。

redis 127.0.0.1:6379> dbsize

(integer) 18

redis 127.0.0.1:6379>

结果说明此库中有18 个key

* info

获取服务器的信息和统计。

redis 127.0.0.1:6379> info

redis\_version:2.2.12

redis\_git\_sha1:00000000

redis\_git\_dirty:0

arch\_bits:32

multiplexing\_api:epoll

process\_id:28480

uptime\_in\_seconds:2515

uptime\_in\_days:0

。。。。

。。。。

* flushdb

删除当前选择数据库中的所有key。

redis 127.0.0.1:6379> dbsize

(integer) 18

redis 127.0.0.1:6379> flushdb

OK

redis 127.0.0.1:6379> dbsize

(integer) 0

redis 127.0.0.1:6379>

在本例中我们将0 号数据库中的key 都清除了。

* flushall

删除所有数据库中的所有key。

redis 127.0.0.1:6379[1]> dbsize

(integer) 1

redis 127.0.0.1:6379[1]> select 0

OK

redis 127.0.0.1:6379> flushall

OK

redis 127.0.0.1:6379> select 1

OK

redis 127.0.0.1:6379[1]> dbsize

(integer) 0

redis 127.0.0.1:6379[1]>

在本例中我们先查看了一个1 号数据库中有一个key，然后我切换到0 号库执行flushall 命令，结果1 号库中的key 也被清除了，说是此命令工作正常。

# 持久化(了解)

Redis的高性能是由于其将所有数据都存储在了内存中，为了使Redis在重启之后仍能保证数据不丢失，需要将数据从内存中同步到硬盘中，这一过程就是持久化。

Redis支持两种方式的持久化，一种是RDB方式，一种是AOF方式。可以单独使用其中一种或将二者结合使用。

## RDB持久化

RDB方式的持久化是通过快照（snapshotting）完成的，当符合一定条件时Redis会自动将内存中的数据进行快照并持久化到硬盘。

RDB是Redis默认采用的持久化方式，在redis.conf配置文件中默认有此下配置：

save 900 1

save 300 10

save 60 10000

save 开头的一行就是持久化配置，可以配置多个条件（每行配置一个条件），每个条件之间是“或”的关系，“save 900 1”表示15分钟（900秒钟）内至少1个键被更改则进行快照，“save 300 10”表示5分钟（300秒）内至少10个键被更改则进行快照。

在redis.conf中：

配置dir指定rdb快照文件的位置

配置dbfilenam指定rdb快照文件的名称

Redis启动后会读取RDB快照文件，将数据从硬盘载入到内存。根据数据量大小与结构和服务器性能不同，这个时间也不同。通常将记录一千万个字符串类型键、大小为1GB的快照文件载入到内存中需要花费20～30秒钟。

问题总结：

通过RDB方式实现持久化，一旦Redis异常退出，就会丢失最后一次快照以后更改的所有数据。这就需要开发者根据具体的应用场合，通过组合设置自动快照条件的方式来将可能发生的数据损失控制在能够接受的范围。如果数据很重要以至于无法承受任何损失，则可以考虑使用AOF方式进行持久化。

## AOF持久化

默认情况下Redis没有开启AOF（append only file）方式的持久化，可以通过appendonly参数开启：

appendonly yes

开启AOF持久化后每执行一条会更改Redis中的数据的命令，Redis就会将该命令写入硬

盘中的AOF文件。AOF文件的保存位置和RDB文件的位置相同，都是通过dir参数设置的，默认的文件名是appendonly.aof，可以通过appendfilename参数修改：appendfilename appendonly.aof

# 主从复制（了解）

## 什么是主从复制

持久化保证了即使redis服务重启也会丢失数据，因为redis服务重启后会将硬盘上持久化的数据恢复到内存中，但是当redis服务器的硬盘损坏了可能会导致数据丢失，如果通过redis的主从复制机制就可以避免这种单点故障，如下图：

主Redis

(master)

从Redis1

(slave)

从Redis2

(slave)

说明：

* 主redis中的数据有两个副本（replication）即从redis1和从redis2，即使一台redis服务器宕机其它两台redis服务也可以继续提供服务。
* 主redis中的数据和从redis上的数据保持实时同步，当主redis写入数据时通过主从复制机制会复制到两个从redis服务上。
* 只有一个主redis，可以有多个从redis。
* 主从复制不会阻塞master，在同步数据时，master 可以继续处理client 请求
* 一个redis可以即是主又是从，如下图：

主Redis

(master)

从Redis1

(slave)

从Redis2

(slave)

从Redis3

(slave)

从Redis4

(slave)

## 主从配置

### 主redis配置

无需特殊配置。

### 从redis配置

修改从redis服务器上的redis.conf文件，添加**slaveof 主redisip 主redis端口**



上边的配置说明当前该从redis服务器所对应的主redis是192.168.101.3，端口是6379

## 主从复制过程

### 完整复制

在redis2.8版本之前主从复制过程如下图：

主Redis

从Redis

发送sync命令

发送RDB文件

发送缓冲获得所有写命令

复制过程说明：

1. slave 服务启动，slave 会建立和master 的连接，发送sync 命令。

2、master启动一个后台进程将数据库快照保存到RDB文件中

注意：此时如果生成RDB文件过程中存在写数据操作会导致RDB文件和当前主redis数据不一致，所以此时master 主进程会开始收集写命令并缓存起来。

3、master 就发送RDB文件给slave

4、slave 将文件保存到磁盘上，然后加载到内存恢复

5、master把缓存的命令转发给slave

注意：后续master 收到的写命令都会通过开始建立的连接发送给slave。

当master 和slave 的连接断开时slave 可以自动重新建立连接。如果master 同时收到多个slave 发来的同步连接命令，只会启动一个进程来写数据库镜像，然后发送给所有slave。

完整复制的问题：

在redis2.8之前从redis每次同步都会从主redis中复制全部的数据，如果从redis是新创建的从主redis中复制全部的数据这是没有问题的，但是，如果当从redis停止运行，再启动时可能只有少部分数据和主redis不同步，此时启动redis仍然会从主redis复制全部数据，这样的性能肯定没有只复制那一小部分不同步的数据高。

### 部分复制

主Redis

从Redis

发送psync命令(首次同步)

如果是首次同步则发送RDB文件

发送缓冲获得所有写命令

如果是非首次同步则发送部分非同步数据

发送psync命令(非首次同步骤)

部分复制说明：

从机连接主机后，会主动发起 PSYNC 命令，从机会提供 master的runid(机器标识，随机生成的一个串) 和 offset（数据偏移量，如果offset主从不一致则说明数据不同步），主机验证 runid 和 offset 是否有效， runid 相当于主机身份验证码，用来验证从机上一次连接的主机，如果runid验证未通过则，则进行全同步，如果验证通过则说明曾经同步过，根据offset同步部分数据。