

Redis快速入门


Redis的常见命令和客户端使用

1.初识Redis

Redis是一种键值型的NoSql数据库，这里有两个关键字：

- 键值型
- NoSql

其中**键值型**，是指Redis中存储的数据都是以key、value对的形式存储，而value的形式多种多样，可以是字符串、数值、甚至json：

	Key	Value
 键值数据库	id	1001
	name	张三
	age	21
		{
	1001	"id": 1001, "name": "张三", "age": 21 }

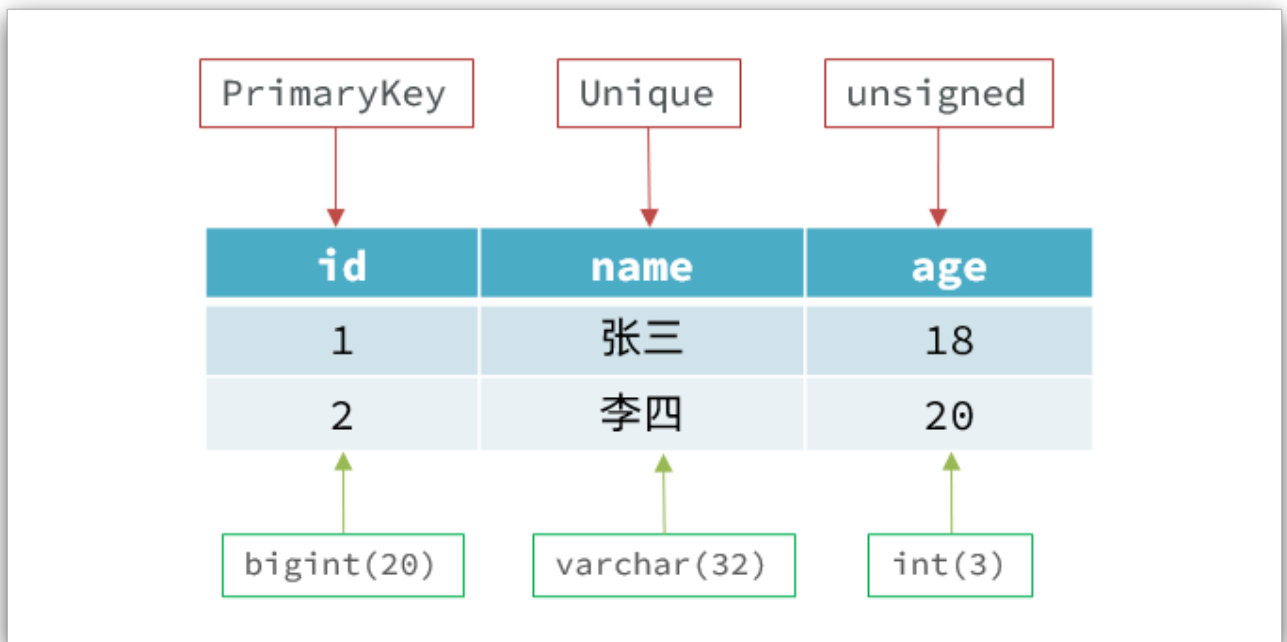
而NoSql则是相对于传统关系型数据库而言，有很大差异的一种数据库。

1.1.认识NoSQL

NoSql可以翻译做Not Only Sql（不仅仅是SQL），或者是No Sql（非Sql的）数据库。是相对于传统关系型数据库而言，有很大差异的一种特殊的数据库，因此也称之为**非关系型数据库**。

1.1.1.结构化与非结构化

传统关系型数据库是结构化数据，每一张表都有严格的约束信息：字段名、字段数据类型、字段约束等信息，插入的数据必须遵守这些约束：

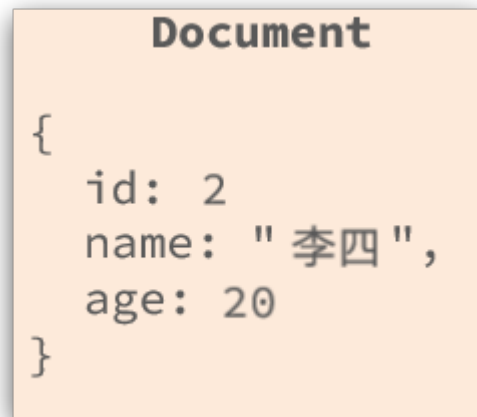


而NoSql则对数据库格式没有严格约束，往往形式松散，自由。

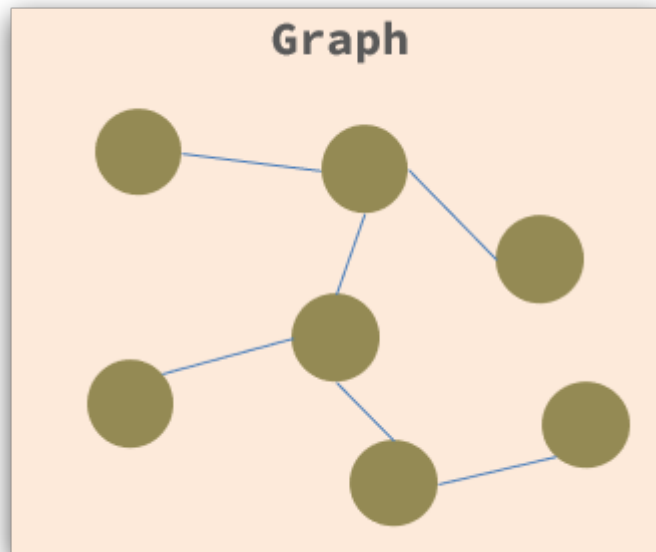
可以是键值型：

Key	Value
id	1
name	张三
age	18

也可以是文档型：

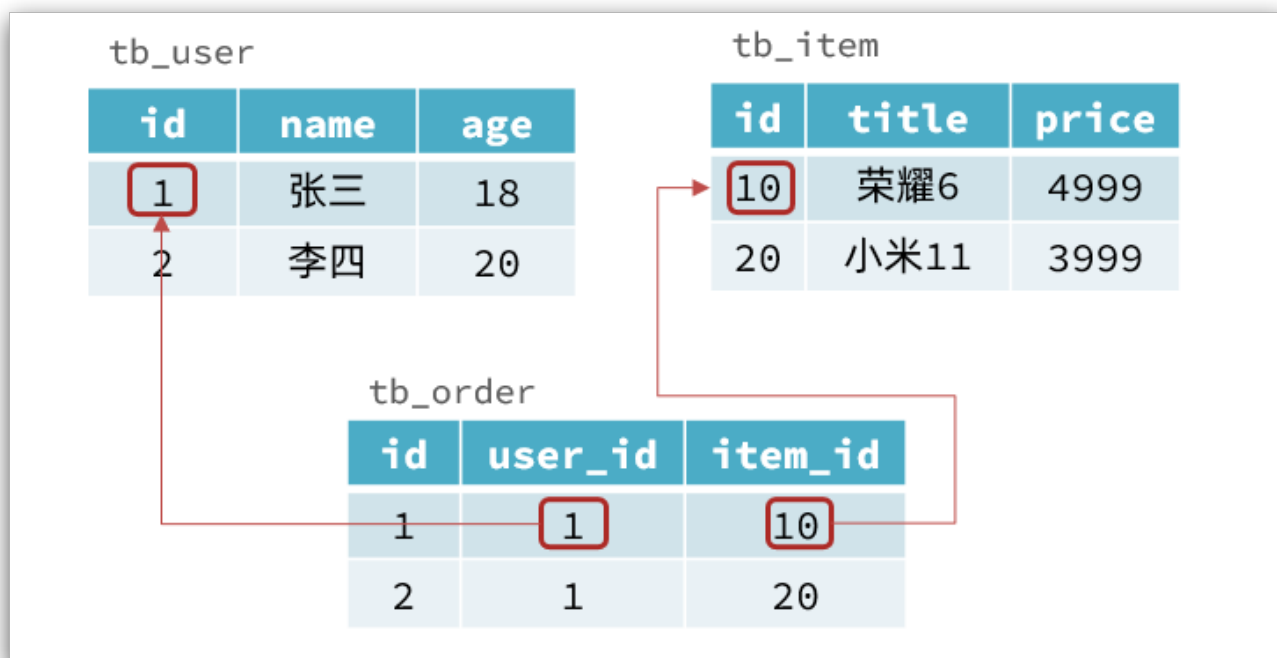


甚至可以是图格式：



1.1.2.关联和非关联

传统数据库的表与表之间往往存在关联，例如外键：



而非关系型数据库不存在关联关系，要维护关系要么靠代码中的业务逻辑，要么靠数据之间的耦合：

```
1 {
2   id: 1,
3   name: "张三",
4   orders: [
5     {
6       id: 1,
7       item: {
8         id: 10, title: "荣耀6", price: 4999
9       }
10    },
11    {
12      id: 2,
13      item: {
14        id: 20, title: "小米11", price: 3999
15      }
16    }
17  ]
18 }
```

此处要维护“张三”的订单与商品“荣耀”和“小米11”的关系，不得不冗余的将这两个商品保存在张三的订单文档中，不够优雅。还是建议用业务来维护关联关系。

1.1.3.查询方式

传统关系型数据库会基于Sql语句做查询，语法有统一标准；

而不同的非关系数据库查询语法差异极大，五花八门各种各样。

关系型: `SQL SELECT id, name age FROM tb_user WHERE id = 1`

非关系型: `Redis get user:1`

`MongoDB db.users.find({_id: 1})`

`elasticsearch GET http://localhost:9200/users/1`

1.1.4.事务

传统关系型数据库能满足事务ACID的原则。



而非关系型数据库往往不支持事务，或者不能严格保证ACID的特性，只能实现基本的一致性。

1.1.5.总结

除了上述四点以外，在存储方式、扩展性、查询性能上关系型与非关系型也都有着显著差异，总结如下：

	SQL	NoSQL
数据结构	结构化(Structured)	非结构化
数据关联	关联的(Relational)	无关联的
查询方式	SQL查询	非SQL
事务特性	ACID	BASE
存储方式	磁盘	内存
扩展性	垂直	水平
使用场景	1) 数据结构固定 2) 相关业务对数据安全性、一致性要求较高	1) 数据结构不固定 2) 对一致性、安全性要求不高 3) 对性能要求

#1 键值类型 (Redis)

#2 文档类型 (MongoDB)

#3 列类型 (HBase)

#4 Graph类型 (Neo4j)

- 存储方式
 - 关系型数据库基于磁盘进行存储，会有大量的磁盘IO，对性能有一定影响
 - 非关系型数据库，他们的操作更多的是依赖于内存来操作，内存的读写速度会非常快，性能自然会好一些
- 扩展性
 - 关系型数据库集群模式一般是主从，主从数据一致，起到数据备份的作用，称为垂直扩展。
 - 非关系型数据库可以将数据拆分，存储在不同机器上，可以保存海量数据，解决内存大小有限的问题。称为水平扩展。
 - 关系型数据库因为表之间存在关联关系，如果做水平扩展会给数据查询带来很多麻烦

1.2.认识Redis

Redis诞生于2009年全称是**Remote Dictionary Server** 远程词典服务器，是一个基于内存的键值型NoSQL数据库。

特征：

- 键值 (key-value) 型，value支持多种不同数据结构，功能丰富
- 单线程，每个命令具备原子性
- 低延迟，速度快（基于内存、IO多路复用、良好的编码）。
- 支持数据持久化
- 支持主从集群、分片集群
- 支持多语言客户端

作者：Antirez

Redis的官方网站地址：<https://redis.io/>

1.3.安装Redis

大多数企业都是基于Linux服务器来部署项目，而且Redis官方也没有提供Windows版本的安装包。因此课程中我们会基于Linux系统来安装Redis。

此处选择的Linux版本为CentOS 7。

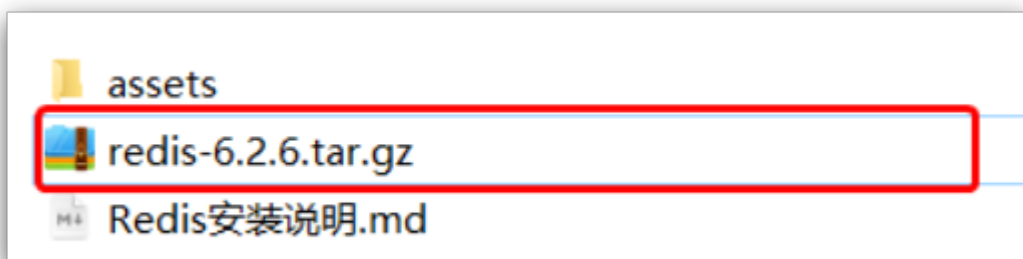
1.3.1.依赖库

Redis是基于C语言编写的，因此首先需要安装Redis所需要的gcc依赖：

```
1 | yum install -y gcc tc1
```

1.3.2.上传安装包并解压

然后将课前资料提供的Redis安装包上传到虚拟机的任意目录：



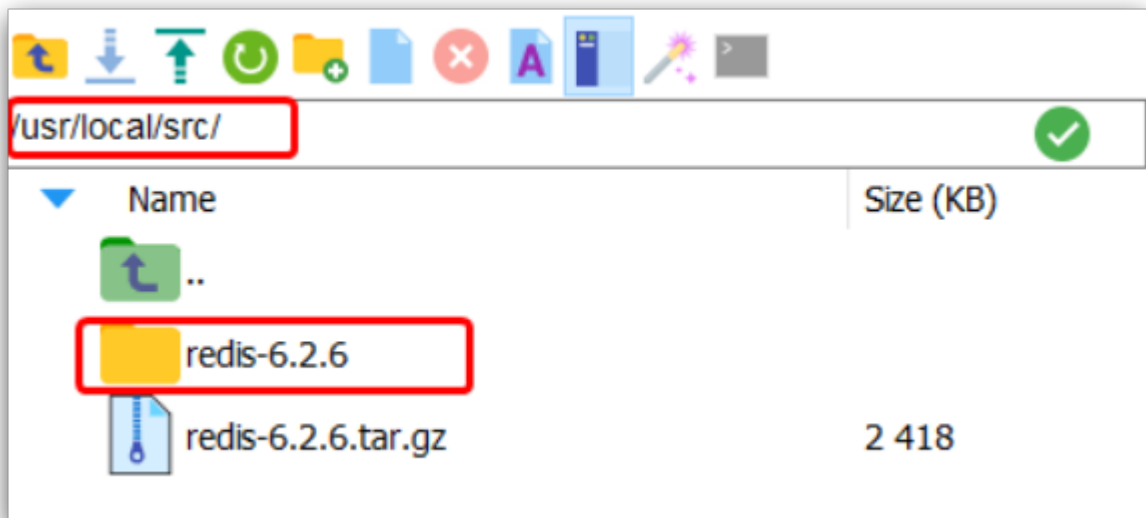
例如，我放到了/usr/local/src 目录：



解压缩：

```
1 | tar -xzf redis-6.2.6.tar.gz
```

解压后：



进入redis目录：

```
1 | cd redis-6.2.6
```

运行编译命令：

```
1 | make && make install
```

如果没有出错，应该就安装成功了。

默认的安装路径是在 /usr/local/bin 目录下：

该目录已经默认配置到环境变量，因此可以在任意目录下运行这些命令。其中：

- redis-cli：是redis提供的命令行客户端
- redis-server：是redis的服务端启动脚本
- redis-sentinel：是redis的哨兵启动脚本

1.3.3.启动

redis的启动方式有很多种，例如：

- 默认启动
- 指定配置启动
- 开机自启

1.3.4.默认启动

安装完成后，在任意目录输入redis-server命令即可启动Redis：

```
1 | redis-server
```

如图：

```
[root@heima redis-6.2.6]# redis-server
12572:C 11 Dec 2021 08:16:34.628 # o000o000o000o Redis is starting o000o000o000o
12572:C 11 Dec 2021 08:16:34.628 # Redis version=6.2.6, bits=64, commit=00000000, mod
fied=0, pid=12572, just started
12572:C 11 Dec 2021 08:16:34.628 # Warning: no config file specified, using the defau
t config. In order to specify a config file use redis-server /path/to/redis.conf
12572:M 11 Dec 2021 08:16:34.629 * Increased maximum number of open files to 10032 (
was originally set to 1024).
12572:M 11 Dec 2021 08:16:34.629 * monotonic clock: POSIX clock_gettime

Redis 6.2.6 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 12572

https://redis.io
```

这种启动属于 前台启动，会阻塞整个会话窗口，窗口关闭或者按下 CTRL + C 则Redis停止。不推荐使用。

1.3.5.指定配置启动

如果要让Redis以 后台 方式启动，则必须修改Redis配置文件，就在我们之前解压的redis安装包下 (/usr/local/src/redis-6.2.6)，名字叫redis.conf：

```
[root@heima redis-6.2.6]# pwd
/usr/local/src/redis-6.2.6
[root@heima redis-6.2.6]#
[root@heima redis-6.2.6]# ls
00-RELEASENOTES  CONTRIBUTING  dump.rdb  MANIFESTO  runtest
BUGS              COPYING      INSTALL   README.md  runtest-cluster
CONDUCT          deps         Makefile  redis.conf  runtest-moduleapi
```

我们先将这个配置文件备份一份：

```
1 | cp redis.conf redis.conf.bck
```

然后修改redis.conf文件中的一些配置：

```
1 # 允许访问的地址，默认是127.0.0.1，会导致只能在本地访问。修改为0.0.0.0则可以在任意IP访问，生产环境不要设置为0.0.0.0
2 bind 0.0.0.0
3 # 守护进程，修改为yes后即可后台运行
4 daemonize yes
5 # 密码，设置后访问Redis必须输入密码
6 requirepass 123321
```

Redis的其它常见配置：

```
1 # 监听的端口
2 port 6379
3 # 工作目录，默认是当前目录，也就是运行redis-server时的命令，日志、持久化等文件会保存在这个目录
4 dir .
5 # 数据库数量，设置为1，代表只使用1个库，默认有16个库，编号0~15
6 databases 1
7 # 设置redis能够使用的最大内存
8 maxmemory 512mb
9 # 日志文件，默认为空，不记录日志，可以指定日志文件名
10 logfile "redis.log"
```

启动Redis：

```
1 # 进入redis安装目录
2 cd /usr/local/src/redis-6.2.6
3 # 启动
4 redis-server redis.conf
```

停止服务：

```
1 # 利用redis-cli来执行 shutdown 命令，即可停止 Redis 服务，
2 # 因为之前配置了密码，因此需要通过 -u 来指定密码
3 redis-cli -u 123321 shutdown
```

1.3.6.开机自启

我们也可以通过配置来实现开机自启。

首先，新建一个系统服务文件：

```
1 vi /etc/systemd/system/redis.service
```

内容如下：

```
1 [Unit]
2 Description=redis-server
3 After=network.target
4
5 [Service]
6 Type=forking
7 ExecStart=/usr/local/bin/redis-server /usr/local/src/redis-6.2.6/redis.conf
8 PrivateTmp=true
9
10 [Install]
11 WantedBy=multi-user.target
```

然后重载系统服务：

```
1 | systemctl daemon-reload
```

现在，我们可以用下面这组命令来操作redis了：

```
1 # 启动
2 systemctl start redis
3 # 停止
4 systemctl stop redis
5 # 重启
6 systemctl restart redis
7 # 查看状态
8 systemctl status redis
```

执行下面的命令，可以让redis开机自启：

```
1 | systemctl enable redis
```

1.4.Redis桌面客户端

安装完成Redis，我们就可以操作Redis，实现数据的CRUD了。这需要用到Redis客户端，包括：

- 命令行客户端
- 图形化桌面客户端
- 编程客户端

1.4.1.Redis命令行客户端

Redis安装完成后就自带了命令行客户端：redis-cli，使用方式如下：

```
1 | redis-cli [options] [commands]
```

其中常见的options有：

- `-h 127.0.0.1`：指定要连接的redis节点的IP地址，默认是127.0.0.1
- `-p 6379`：指定要连接的redis节点的端口，默认是6379
- `-a 123321`：指定redis的访问密码

其中的commands就是Redis的操作命令，例如：

- `ping`：与redis服务端做心跳测试，服务端正常会返回 `pong`

不指定command时，会进入 `redis-cli` 的交互控制台：

```
[root@heima redis-6.2.6]# redis-cli -a 123321 通过-a参数指定密码并连接
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
127.0.0.1:6379>
127.0.0.1:6379>
[root@heima redis-6.2.6]# redis-cli
127.0.0.1:6379> auth 123321 进入控制台后，通过auth命令来指定密码
OK
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> █
```

1.4.2.图形化桌面客户端

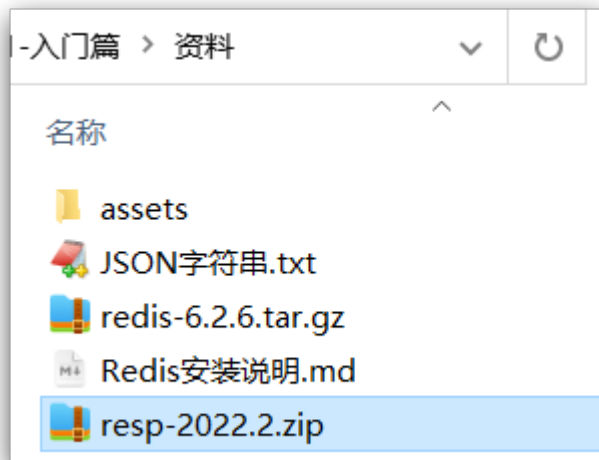
GitHub上的大神编写了Redis的图形化桌面客户端，地址：<https://github.com/uglide/RedisDesktopManager>

不过该仓库提供的是RedisDesktopManager的源码，并未提供windows安装包。

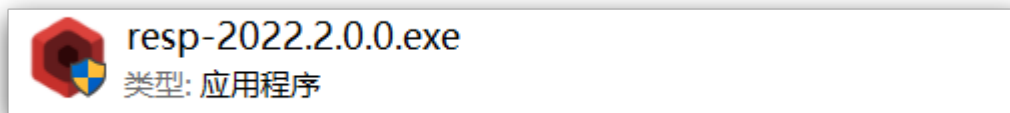
在下面这个仓库可以找到安装包：<https://github.com/lework/RedisDesktopManager-Windows/releases>

1.4.3.安装

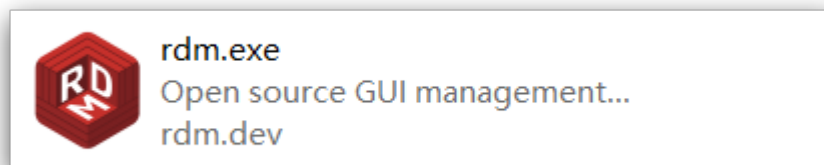
在课前资料中可以找到Redis的图形化桌面客户端：



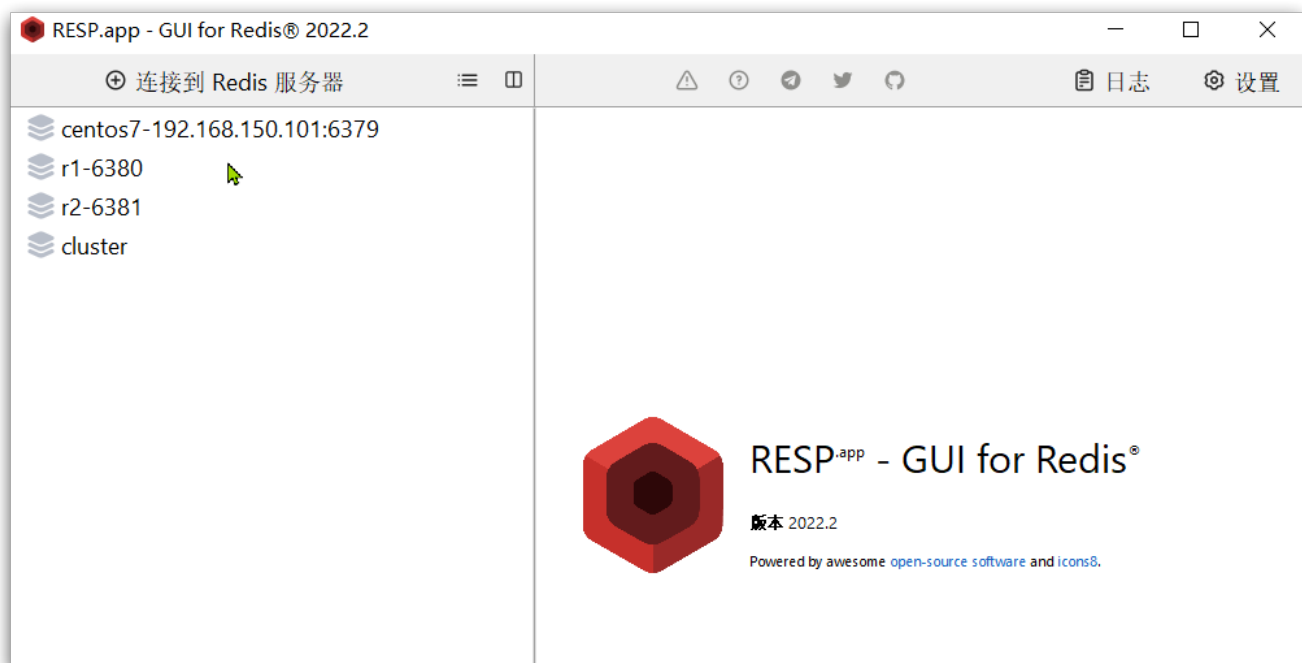
解压缩后，运行安装程序即可安装：



安装完成后，在安装目录下找到rdm.exe文件：



双击即可运行：



1.4.4.建立连接

点击左上角的 连接到Redis服务器 按钮：



在弹出的窗口中填写Redis服务信息：

新连接设置

×

怎么连接

连接设置

高级设置

名字：

centos7-192.168.150.101

地址

192.168.150.101

:

—

6379

+

密码：

●●●●●●

☐ 显示密码

用户名：

可选：服务端认证用户名 (Redis >6.0)

安全

☐ SSL / TLS

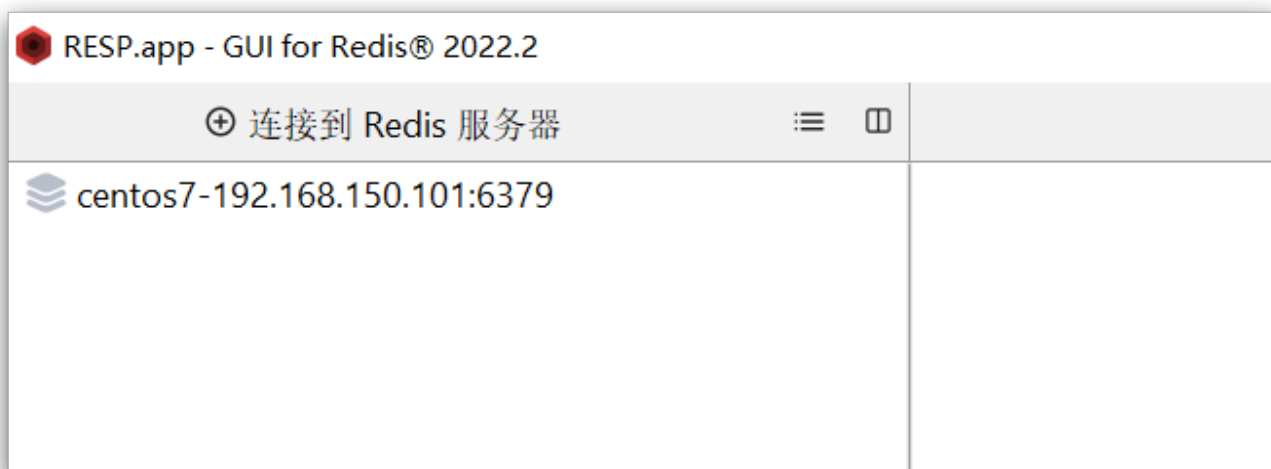
☐ SSH 通道

测试连接

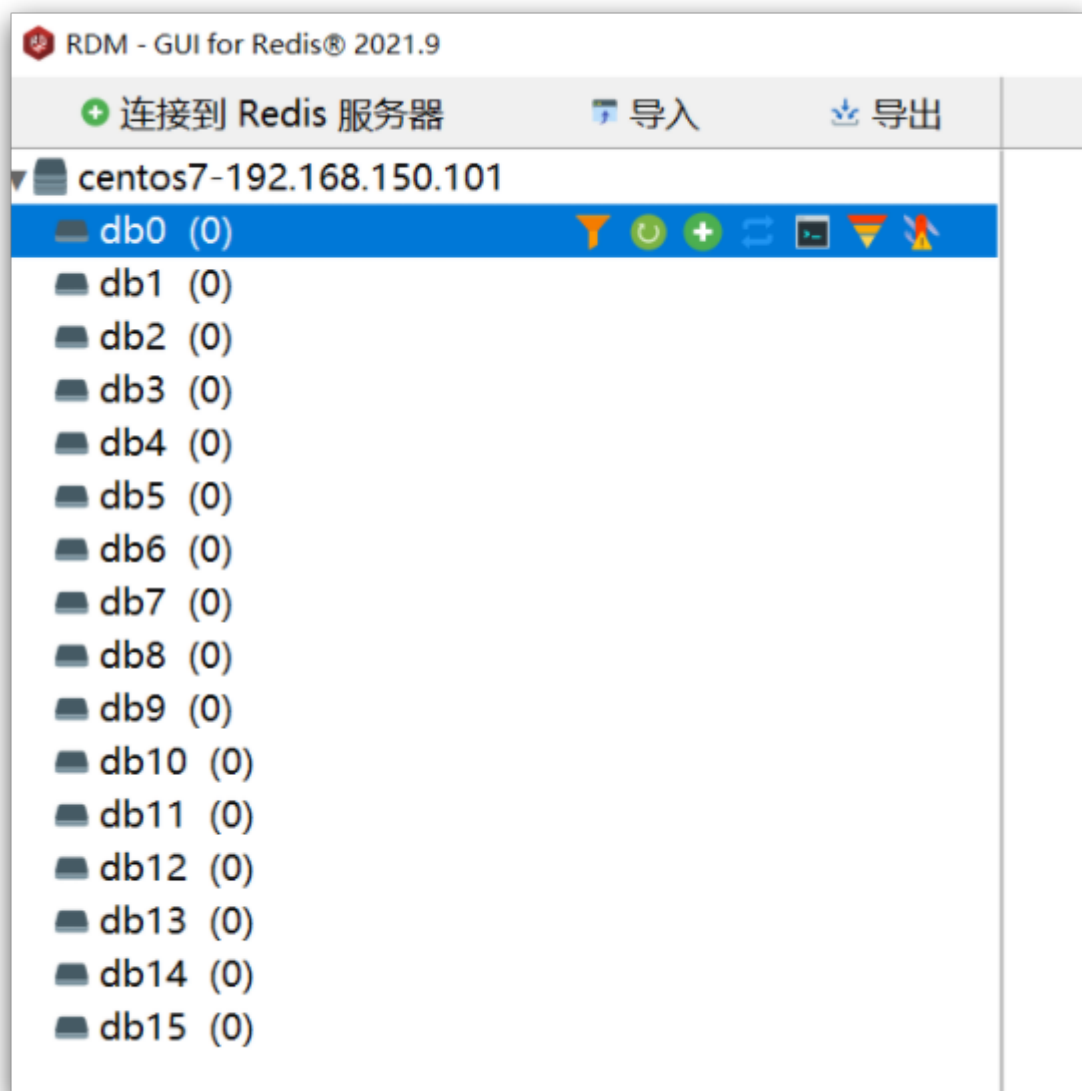
确定

取消

点击确定后，在左侧菜单会出现这个链接：



点击即可建立连接了。



Redis默认有16个仓库，编号从0至15. 通过配置文件可以设置仓库数量，但是不超过16，并且不能自定义仓库名称。

如果是基于redis-cli连接Redis服务，可以通过select命令来选择数据库：

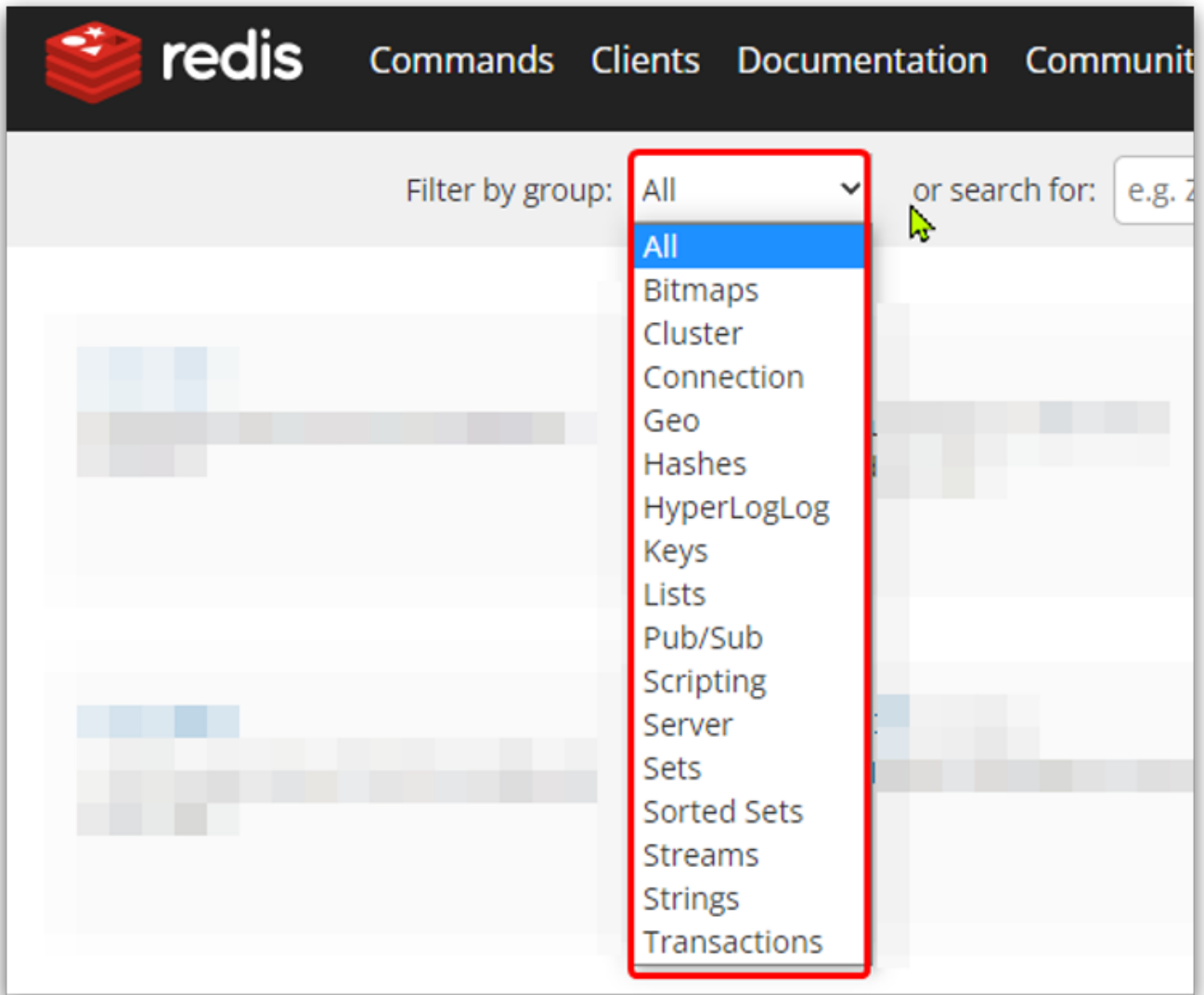
```
1 # 选择 0号库
2 select 0
```

2.Redis常见命令

Redis是典型的key-value数据库，key一般是字符串，而value包含很多不同的数据类型：

String	hello world	基本类型
Hash	{name: "Jack", age: 21}	
List	[A -> B -> C -> C]	
Set	{A, B, C}	
SortedSet	{A: 1, B: 2, C: 3}	
GEO	{A: (120.3, 30.5) }	特殊类型
BitMap	0110110101110101011	
HyperLog	0110110101110101011	

Redis为了方便我们学习，将操作不同数据类型的命令也做了分组，在官网（<https://redis.io/commands>）可以看到不同的命令：



不同类型的命令称为一个group，我们也可以通过help命令来查看各种不同group的命令：

```
127.0.0.1:6379> help
redis-cli 6.2.6
To get help about Redis commands type:
  "help @<group>" to get a list of commands in <group>
  "help <command>" for help on <command>
  "help <tab>" to get a list of possible help topics
  "quit" to exit

To set redis-cli preferences:
  ":set hints" enable online hints
  ":set nohints" disable online hints
Set your preferences in ~/.redisclirc
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> help @generic
```

查看通用的命令

接下来，我们就学习常见的五种基本数据类型的相关命令。

2.1.Redis通用命令

通用指令是部分数据类型的，都可以使用的指令，常见的有：

- KEYS：查看符合模板的所有key
- DEL：删除一个指定的key
- EXISTS：判断key是否存在
- EXPIRE：给一个key设置有效期，有效期到期时该key会被自动删除
- TTL：查看一个KEY的剩余有效期

通过help [command] 可以查看一个命令的具体用法，例如：

```
1 # 查看keys命令的帮助信息:
2 127.0.0.1:6379> help keys
3
4 KEYS pattern
5 summary: Find all keys matching the given pattern
6 since: 1.0.0
7 group: generic
```

2.2.String类型

String类型，也就是字符串类型，是Redis中最简单的存储类型。

其value是字符串，不过根据字符串的格式不同，又可以分为3类：

- string：普通字符串
- int：整数类型，可以做自增、自减操作
- float：浮点类型，可以做自增、自减操作

不管是哪种格式，底层都是字节数组形式存储，只不过是编码方式不同。字符串类型的最大空间不能超过512m.

KEY	VALUE
msg	hello world
num	10
score	92.5

2.2.1.String的常见命令

String的常见命令有：

- SET：添加或者修改已经存在的一个String类型的键值对

- GET：根据key获取String类型的value
- MSET：批量添加多个String类型的键值对
- MGET：根据多个key获取多个String类型的value
- INCR：让一个整型的key自增1
- INCRBY:让一个整型的key自增并指定步长，例如：incrby num 2 让num值自增2
- INCRBYFLOAT：让一个浮点类型的数字自增并指定步长
- SETNX：添加一个String类型的键值对，前提是这个key不存在，否则不执行
- SETEX：添加一个String类型的键值对，并且指定有效期

2.2.2.Key结构

Redis没有类似MySQL中的Table的概念，我们该如何区分不同类型的key呢？

例如，需要存储用户、商品信息到redis，有一个用户id是1，有一个商品id恰好也是1，此时如果使用id作为key，那就会冲突了，该怎么办？

我们可以通过给key添加前缀加以区分，不过这个前缀不是随便加的，有一定的规范：

Redis的key允许有多个单词形成层级结构，多个单词之间用':'隔开，格式如下：

```
1 | 项目名:业务名:类型:id
```

这个格式并非固定，也可以根据自己的需求来删除或添加词条。这样以来，我们就可以把不同类型的数据区分开了。从而避免了key的冲突问题。

例如我们的项目名称叫 heima，有user和product两种不同类型的数据，我们可以这样定义key：

- user相关的key：**heima:user:1**
- product相关的key：**heima:product:1**

如果Value是一个Java对象，例如一个User对象，则可以将对象序列化为JSON字符串后存储：

KEY	VALUE
heima:user:1	{"id":1, "name": "Jack", "age": 21}
heima:product:1	{"id":1, "name": "小米11", "price": 4999}

并且，在Redis的桌面客户端中，还会以相同前缀作为层级结构，让数据看起来层次分明，关系清晰：



2.3.Hash类型

Hash类型，也叫散列，其value是一个无序字典，类似于Java中的HashMap结构。

String结构是将对象序列化为JSON字符串后存储，当需要修改对象某个字段时很不方便：

KEY	VALUE
heima:user:1	{name:"Jack", age:21}
heima:user:2	{name:"Rose", age:18}

Hash结构可以将对象中的每个字段独立存储，可以针对单个字段做CRUD：

KEY	VALUE	
	field	value
heima:user:1	name	Jack
	age	21
heima:user:2	name	Rose
	age	18

Hash的常见命令有：

- HSET key field value：添加或者修改hash类型key的field的值
- HGET key field：获取一个hash类型key的field的值
- HMSET：批量添加多个hash类型key的field的值
- HMGET：批量获取多个hash类型key的field的值

- HGETALL: 获取一个hash类型的key中的所有的field和value
- HKEYS: 获取一个hash类型的key中的所有的field
- HINCRBY: 让一个hash类型key的字段值自增并指定步长
- HSETNX: 添加一个hash类型的key的field值, 前提是这个field不存在, 否则不执行

2.4.List类型

Redis中的List类型与Java中的LinkedList类似, 可以看做是一个双向链表结构。既可以支持正向检索和也可以支持反向检索。

特征也与LinkedList类似:

- 有序
- 元素可以重复
- 插入和删除快
- 查询速度一般

常用来存储一个有序数据, 例如: 朋友圈点赞列表, 评论列表等。

List的常见命令有:

- LPUSH key element ... : 向列表左侧插入一个或多个元素
- LPOP key: 移除并返回列表左侧的第一个元素, 没有则返回nil
- RPUSH key element ... : 向列表右侧插入一个或多个元素
- RPOP key: 移除并返回列表右侧的第一个元素
- LRANGE key start end: 返回一段角标范围内的所有元素
- BLPOP和BRPOP: 与LPOP和RPOP类似, 只不过在没有元素时等待指定时间, 而不是直接返回nil

2.5.Set类型

Redis的Set结构与Java中的HashSet类似, 可以看做是一个value为null的HashMap。因为也是一个hash表, 因此具备与HashSet类似的特征:

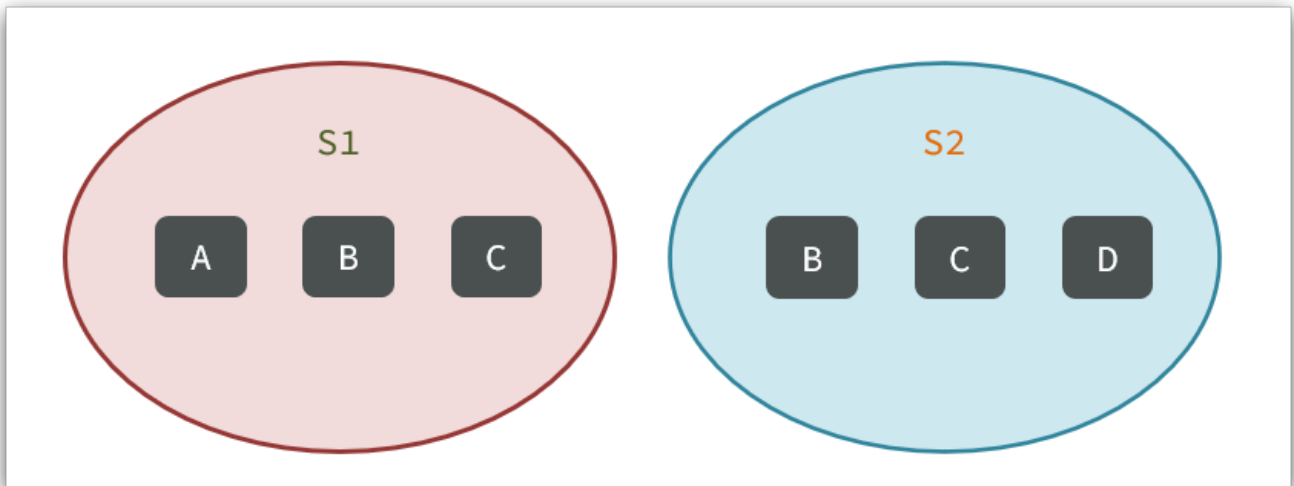
- 无序
- 元素不可重复
- 查找快
- 支持交集、并集、差集等功能

Set的常见命令有:

- SADD key member ... : 向set中添加一个或多个元素
- SREM key member ... : 移除set中的指定元素
- SCARD key: 返回set中元素的个数

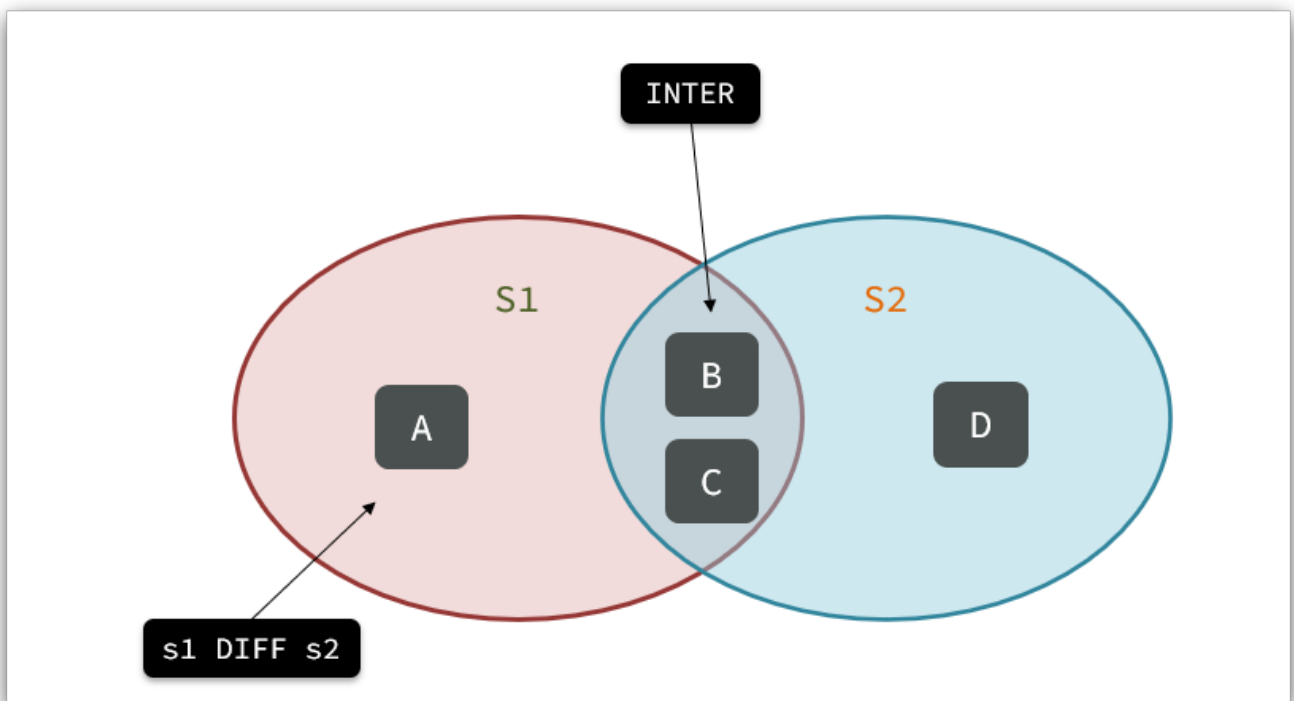
- SISMEMBER key member: 判断一个元素是否存在于set中
- SMEMBERS: 获取set中的所有元素
- SINTER key1 key2 ... : 求key1与key2的交集

例如两个集合: s1和s2:



求交集: SINTER s1 s2

求s1与s2的不同: SDIFF s1 s2



练习:

1. 将下列数据用Redis的Set集合来存储:
- 张三的好友有: 李四、王五、赵六

- 李四的好友有：王五、麻子、二狗

2. 利用Set的命令实现下列功能：

- 计算张三的好友有几人
- 计算张三和李四有哪些共同好友
- 查询哪些人是张三的好友却不是李四的好友
- 查询张三和李四的好友总共有哪些人
- 判断李四是否是张三的好友
- 判断张三是否是李四的好友
- 将李四从张三的好友列表中移除

2.6.SortedSet类型

Redis的SortedSet是一个可排序的set集合，与Java中的TreeSet有些类似，但底层数据结构却差别很大。SortedSet中的每一个元素都带有一个score属性，可以基于score属性对元素排序，底层的实现是一个跳表（SkipList）加 hash 表。

SortedSet具备下列特性：

- 可排序
- 元素不重复
- 查询速度快

因为SortedSet的可排序特性，经常被用来实现排行榜这样的功能。

SortedSet的常见命令有：

- ZADD key score member：添加一个或多个元素到sorted set，如果已经存在则更新其score值
- ZREM key member：删除sorted set中的一个指定元素
- ZSCORE key member：获取sorted set中的指定元素的score值
- ZRANK key member：获取sorted set 中的指定元素的排名
- ZCARD key：获取sorted set中的元素个数
- ZCOUNT key min max：统计score值在给定范围内的所有元素的个数
- ZINCRBY key increment member：让sorted set中的指定元素自增，步长为指定的increment值
- ZRANGE key min max：按照score排序后，获取指定排名范围内的元素
- ZRANGEBYSCORE key min max：按照score排序后，获取指定score范围内的元素
- ZDIFF、ZINTER、ZUNION：求差集、交集、并集

注意：所有的排名默认都是升序，如果要降序则在命令的Z后面添加REV即可，例如：

- **升序**获取sorted set 中的指定元素的排名：ZRANK key member
- **降序**获取sorted set 中的指定元素的排名：ZREVRANK key member

练习题：

将班级的下列学生得分存入Redis的SortedSet中：

Jack 85, Lucy 89, Rose 82, Tom 95, Jerry 78, Amy 92, Miles 76


并实现下列功能：

- 删除Tom同学
- 获取Amy同学的分数
- 获取Rose同学的排名
- 查询80分以下有几个学生
- 给Amy同学加2分
- 查出成绩前3名的同学
- 查出成绩80分以下的的所有同学

3.Redis的Java客户端

在Redis官网中提供了各种语言的客户端，地址：<https://redis.io/docs/clients/>

Browse by language:

ActionScript	ActiveX/COM+	Bash	Boomi	C	C#
C++	Clojure	Common Lisp	Crystal	D	Dart
Delphi	Elixir	emacs lisp	Erlang	Fancy	gawk
GNU Prolog	Go	Haskell	Haxe	Io	Java 
Julia	Lasso	Lua	Matlab	mruby	Nim
Node.js	Objective-C	OCaml	Pascal	Perl	PHP
PL/SQL	Prolog	Pure Data	Python	R	Racket
Rebol	Ruby	Rust	Scala	Scheme	Smalltalk
Swift	Tcl	VB	VCL	Xojo	Zig

其中Java客户端也包含很多：

Java

Redisson	Redisson - Redis Java client with features of In-Memory Data Grid. Over 50 Redis based Java objects and services: Set, Multimap, SortedSet, Map, List, Queue, Deque, Semaphore, Lock, AtomicLong, Map Reduce, Publish / Subscribe, Bloom filter, Spring Cache, Tomcat, Scheduler, JCache API, Hibernate, MyBatis, RPC, local cache ...	❤️	🔗 Apache-2.0	★ 18712	🏠
Jedis	Redis Java client designed for performance and ease of use.	❤️	🔗 MIT	★ 10362	
lettuce	Advanced Java Redis client for thread-safe sync, async, and reactive usage. Supports Cluster, Sentinel, Pipelining, and codecs.	❤️	🔗 Apache-2.0	★ 4496	🏠
vertx-redis-client	Redis client for Vert.x	🟢	🔗 Apache-2.0	★ 109	🏠
redis-protocol	Java client and server implementation of Redis		🔗 Other	★ 348	
JRedis	Java Client and Connectors for Redis		🔗 Apache-2.0	★ 309	🏠
java-redis-client	Low level Redis client (but you won't need more than this)		🔗 MIT	★ 42	

标记为*的就是推荐使用的java客户端，包括：

- Jedis和Lettuce：这两个主要是提供了Redis命令对应的API，方便我们操作Redis，而SpringDataRedis又对这两种做了抽象和封装，因此我们后期会直接以SpringDataRedis来学习。
- Redisson：是在Redis基础上实现了分布式的可伸缩的java数据结构，例如Map、Queue等，而且支持跨进程的同步机制：Lock、Semaphore等待，比较适合用来实现特殊的功能需求。

3.1.Jedis客户端

Jedis的官网地址：<https://github.com/redis/jedis>

3.1.1.快速入门

我们先来个快速入门：

1) 引入依赖：

```

1  <!-- jedis -->
2  <dependency>
3      <groupId>redis.clients</groupId>
4      <artifactId>jedis</artifactId>
5      <version>3.7.0</version>
6  </dependency>
7  <!-- 单元测试 -->
8  <dependency>
9      <groupId>org.junit.jupiter</groupId>
10     <artifactId>junit-jupiter</artifactId>
11     <version>5.7.0</version>
12     <scope>test</scope>
13 </dependency>

```

2) 建立连接

新建一个单元测试类，内容如下：

```

1  private Jedis jedis;
2
3  @BeforeEach
4  void setUp() {
5      // 1.建立连接
6      // jedis = new Jedis("192.168.150.101", 6379);
7      jedis = JedisConnectionFactory.getJedis();
8      // 2.设置密码
9      jedis.auth("123321");
10     // 3.选择库
11     jedis.select(0);
12 }

```

3) 测试：

```

1  @Test
2  void testString() {
3      // 存入数据
4      String result = jedis.set("name", "虎哥");
5      System.out.println("result = " + result);
6      // 获取数据
7      String name = jedis.get("name");
8      System.out.println("name = " + name);
9  }
10
11 @Test
12 void testHash() {
13     // 插入hash数据
14     jedis.hset("user:1", "name", "Jack");
15     jedis.hset("user:1", "age", "21");
16 }

```

```

17 // 获取
18 Map<String, String> map = jedis.hgetAll("user:1");
19 System.out.println(map);
20 }

```

4) 释放资源

```

1 @AfterEach
2 void tearDown() {
3     if (jedis != null) {
4         jedis.close();
5     }
6 }

```

3.1.2.连接池

Jedis本身是线程不安全的，并且频繁的创建和销毁连接会有性能损耗，因此我们推荐大家使用Jedis连接池代替Jedis的直连方式。

```

1 package com.heima.jedis.util;
2
3 import redis.clients.jedis.*;
4
5 public class JedisConnectionFactory {
6
7     private static JedisPool jedisPool;
8
9     static {
10         // 配置连接池
11         JedisPoolConfig poolConfig = new JedisPoolConfig();
12         poolConfig.setMaxTotal(8);
13         poolConfig.setMaxIdle(8);
14         poolConfig.setMinIdle(0);
15         poolConfig.setMaxWaitMillis(1000);
16         // 创建连接池对象，参数：连接池配置、服务端ip、服务端端口、超时时间、密码
17         jedisPool = new JedisPool(poolConfig, "192.168.150.101", 6379, 1000,
18             "123321");
19     }
20
21     public static Jedis getJedis(){
22         return jedisPool.getResource();
23     }
24 }

```

3.2.SpringDataRedis客户端

SpringData是Spring中数据操作的模块，包含对各种数据库的集成，其中对Redis的集成模块就叫做SpringDataRedis，官网地址：<https://spring.io/projects/spring-data-redis>

- 提供了对不同Redis客户端的整合（Lettuce和Jedis）
- 提供了RedisTemplate统一API来操作Redis
- 支持Redis的发布订阅模型
- 支持Redis哨兵和Redis集群
- 支持基于Lettuce的响应式编程
- 支持基于JDK、JSON、字符串、Spring对象的数据序列化及反序列化
- 支持基于Redis的JDKCollection实现

SpringDataRedis中提供了RedisTemplate工具类，其中封装了各种对Redis的操作。并且将不同数据类型的操作API封装到了不同的类型中：

API	返回值类型	说明
<code>redisTemplate.opsForValue()</code>	ValueOperations	操作String类型数据
<code>redisTemplate.opsForHash()</code>	HashOperations	操作Hash类型数据
<code>redisTemplate.opsForList()</code>	ListOperations	操作List类型数据
<code>redisTemplate.opsForSet()</code>	SetOperations	操作Set类型数据
<code>redisTemplate.opsForZSet()</code>	ZSetOperations	操作SortedSet类型数据
<code>redisTemplate</code>		通用的命令

3.2.1.快速入门

SpringBoot已经提供了对SpringDataRedis的支持，使用非常简单。

首先，新建一个maven项目，然后按照下面步骤执行：

1) 引入依赖

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>2.5.7</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.heima</groupId>
```

```
12 <artifactId>redis-demo</artifactId>
13 <version>0.0.1-SNAPSHOT</version>
14 <name>redis-demo</name>
15 <description>Demo project for Spring Boot</description>
16 <properties>
17     <java.version>1.8</java.version>
18 </properties>
19 <dependencies>
20     <!--redis依赖-->
21     <dependency>
22         <groupId>org.springframework.boot</groupId>
23         <artifactId>spring-boot-starter-data-redis</artifactId>
24     </dependency>
25     <!--common-pool-->
26     <dependency>
27         <groupId>org.apache.commons</groupId>
28         <artifactId>commons-pool2</artifactId>
29     </dependency>
30     <!--Jackson依赖-->
31     <dependency>
32         <groupId>com.fasterxml.jackson.core</groupId>
33         <artifactId>jackson-databind</artifactId>
34     </dependency>
35     <dependency>
36         <groupId>org.projectlombok</groupId>
37         <artifactId>lombok</artifactId>
38         <optional>true</optional>
39     </dependency>
40     <dependency>
41         <groupId>org.springframework.boot</groupId>
42         <artifactId>spring-boot-starter-test</artifactId>
43         <scope>test</scope>
44     </dependency>
45 </dependencies>
46
47 <build>
48     <plugins>
49         <plugin>
50             <groupId>org.springframework.boot</groupId>
51             <artifactId>spring-boot-maven-plugin</artifactId>
52             <configuration>
53                 <excludes>
54                     <exclude>
55                         <groupId>org.projectlombok</groupId>
56                         <artifactId>lombok</artifactId>
57                     </exclude>
58                 </excludes>
59             </configuration>
60         </plugin>
61     </plugins>
62 </build>
63
64 </project>
```

2) 配置Redis

```
1  spring:
2    redis:
3      host: 192.168.150.101
4      port: 6379
5      password: 123321
6      lettuce:
7        pool:
8          max-active: 8
9          max-idle: 8
10         min-idle: 0
11         max-wait: 100ms
```

3) 注入RedisTemplate

因为有了SpringBoot的自动装配，我们可以拿来就用：

```
1  @SpringBootTest
2  class RedisStringTests {
3
4      @Autowired
5      private RedisTemplate redisTemplate;
6  }
```

4) 编写测试

```
1  @SpringBootTest
2  class RedisStringTests {
3
4      @Autowired
5      private RedisTemplate redisTemplate;
6
7      @Test
8      void testString() {
9          // 写入一条String数据
10         redisTemplate.opsForValue().set("name", "虎哥");
11         // 获取string数据
12         Object name = redisTemplate.opsForValue().get("name");
13         System.out.println("name = " + name);
14     }
15 }
```

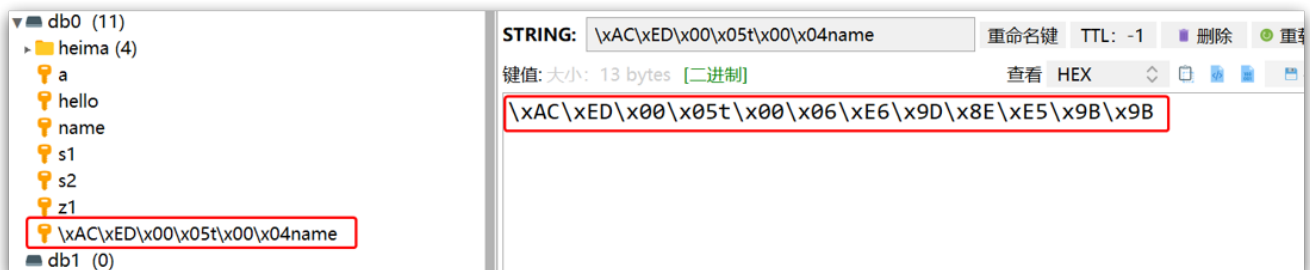
3.2.2.自定义序列化

RedisTemplate可以接收任意Object作为值写入Redis:

```
@Test
void testString() {

    redisTemplate.opsForValue().
        get(Object key)
        set(Object key, Object value)
```

只不过写入前会把Object序列化为字节形式，默认是采用JDK序列化，得到的结果是这样的：



缺点:

- 可读性差
- 内存占用较大

我们可以自定义RedisTemplate的序列化方式，代码如下：

```
1  @Configuration
2  public class RedisConfig {
3
4      @Bean
5      public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
connectionFactory){
6          // 创建RedisTemplate对象
7          RedisTemplate<String, Object> template = new RedisTemplate<>();
8          // 设置连接工厂
9          template.setConnectionFactory(connectionFactory);
10         // 创建JSON序列化工具
11         GenericJackson2JsonRedisSerializer jsonRedisSerializer =
12             new GenericJackson2JsonRedisSerializer();
13         // 设置Key的序列化
```

```

14     template.setKeySerializer(RedisSerializer.string());
15     template.setHashKeySerializer(RedisSerializer.string());
16     // 设置Value的序列化
17     template.setValueSerializer(jsonRedisSerializer);
18     template.setHashValueSerializer(jsonRedisSerializer);
19     // 返回
20     return template;
21 }
22 }

```

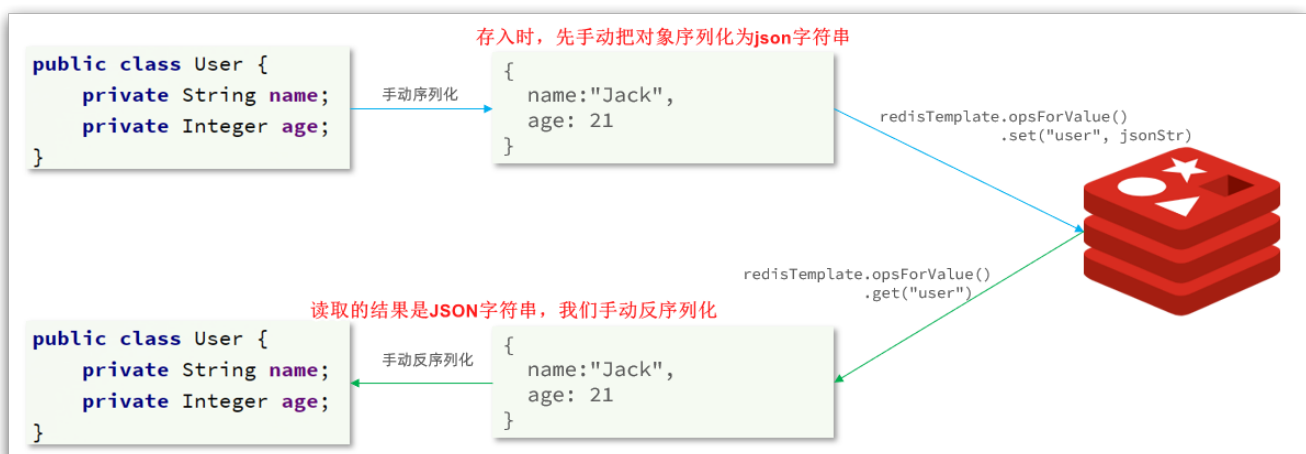
这里采用了JSON序列化来代替默认的JDK序列化方式。最终结果如图：



整体可读性有了很大提升，并且能将Java对象自动的序列化为JSON字符串，并且查询时能自动把JSON反序列化为Java对象。不过，其中记录了序列化时对应的class名称，目的是为了查询时实现自动反序列化。这会带来额外的内存开销。

3.2.3.StringRedisTemplate

为了节省内存空间，我们可以不使用JSON序列化器来处理value，而是统一使用String序列化器，要求只能存储String类型的key和value。当需要存储Java对象时，手动完成对象的序列化和反序列化。



因为存入和读取时的序列化及反序列化都是我们自己实现的，SpringDataRedis就不会将class信息写入Redis了。

这种用法比较普遍，因此SpringDataRedis就提供了RedisTemplate的子类：StringRedisTemplate，它的key和value的序列化方式默认就是String方式。

```
Author: Costin Leau, Mark Paluch  
public class StringRedisTemplate extends RedisTemplate<String, String> {  
  
    Constructs a new StringRedisTemplate instance.  
    setConnectionFactory(RedisConnectionFactory)  
    and afterPropertiesSet() still need to be called.  
  
    public StringRedisTemplate() {  
        setKeySerializer(RedisSerializer.string());  
        setValueSerializer(RedisSerializer.string());  
        setHashKeySerializer(RedisSerializer.string());  
        setHashValueSerializer(RedisSerializer.string());  
    }  
}
```

省去了我们自定义RedisTemplate的序列化方式的步骤，而是直接使用：

```
1  @Autowired  
2  private StringRedisTemplate stringRedisTemplate;  
3  // JSON序列化工具  
4  private static final ObjectMapper mapper = new ObjectMapper();  
5  
6  @Test  
7  void testSaveUser() throws JsonProcessingException {  
8      // 创建对象  
9      User user = new User("虎哥", 21);  
10     // 手动序列化  
11     String json = mapper.writeValueAsString(user);  
12     // 写入数据  
13     stringRedisTemplate.opsForValue().set("user:200", json);  
14  
15     // 获取数据  
16     String jsonUser = stringRedisTemplate.opsForValue().get("user:200");  
17     // 手动反序列化  
18     User user1 = mapper.readValue(jsonUser, User.class);  
19     System.out.println("user1 = " + user1);  
20 }  
21
```