**Redis**

**1. 概述**

**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**。

Redis 是一款高性能的基于内存的NoSQL 系列的非关系型数据库

**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。键值型、文档型、图格式......

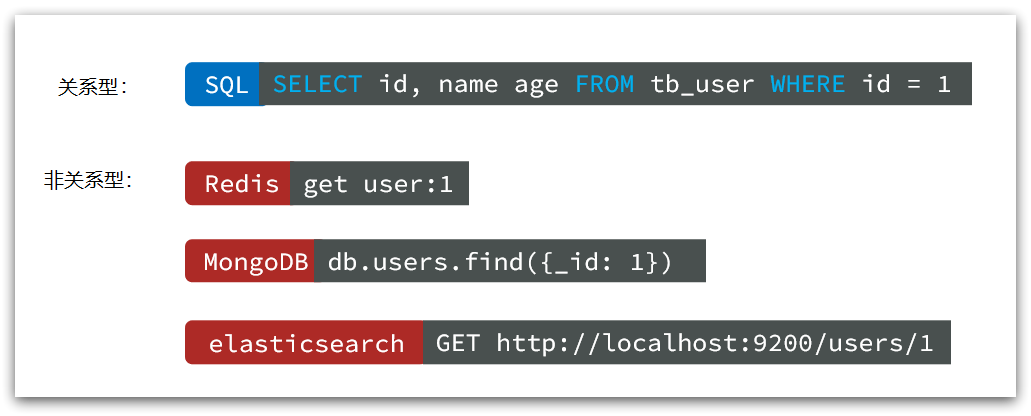
**1.1.2 关联和非关联**

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

**1.1.3.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。

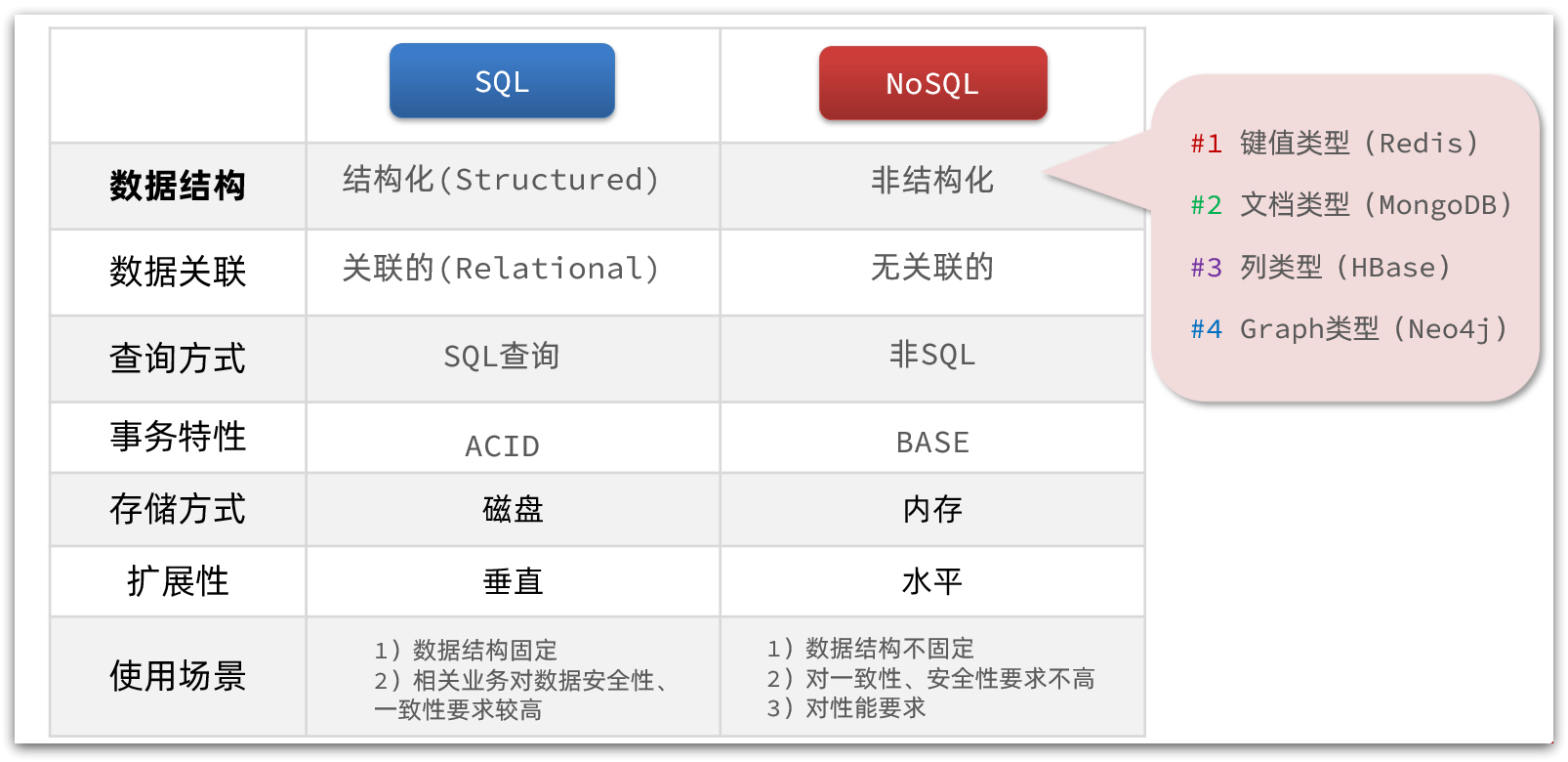


**1.1.4.事务**

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。

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：



* 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

【非关系型数据库的优势】

* 性能 NoSQL 是基于键值对的，可以想象成表中的主键和值的对应关系，而且不需要经过 SQL 层的解析，所以性能非常高
* 可扩展性同样也是因为基于键值对，数据之间没有耦合性，所以非常容易水平扩展

【关系型数据库的优势】

* 复杂查询可以用 SQL 语句方便的在一个表以及多个表之间做非常复杂的数据查询
* 事务支持使得对于安全性能很高的数据访问要求得以实现

【总结】

关系型数据库与 NoSQL 数据库并非对立而是互补的关系，即通常情况下使用关系型数据库，在适合使用 NoSQL 的时候使用 NoSQL 数据库，让 NoSQL 数据库对关系型数据库的不足进行弥补，一般会将数据存储在关系型数据库中，在 NoSQL 数据库中备份存储关系型数据库的数据。

主流的 NoSQL 产品：

【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:

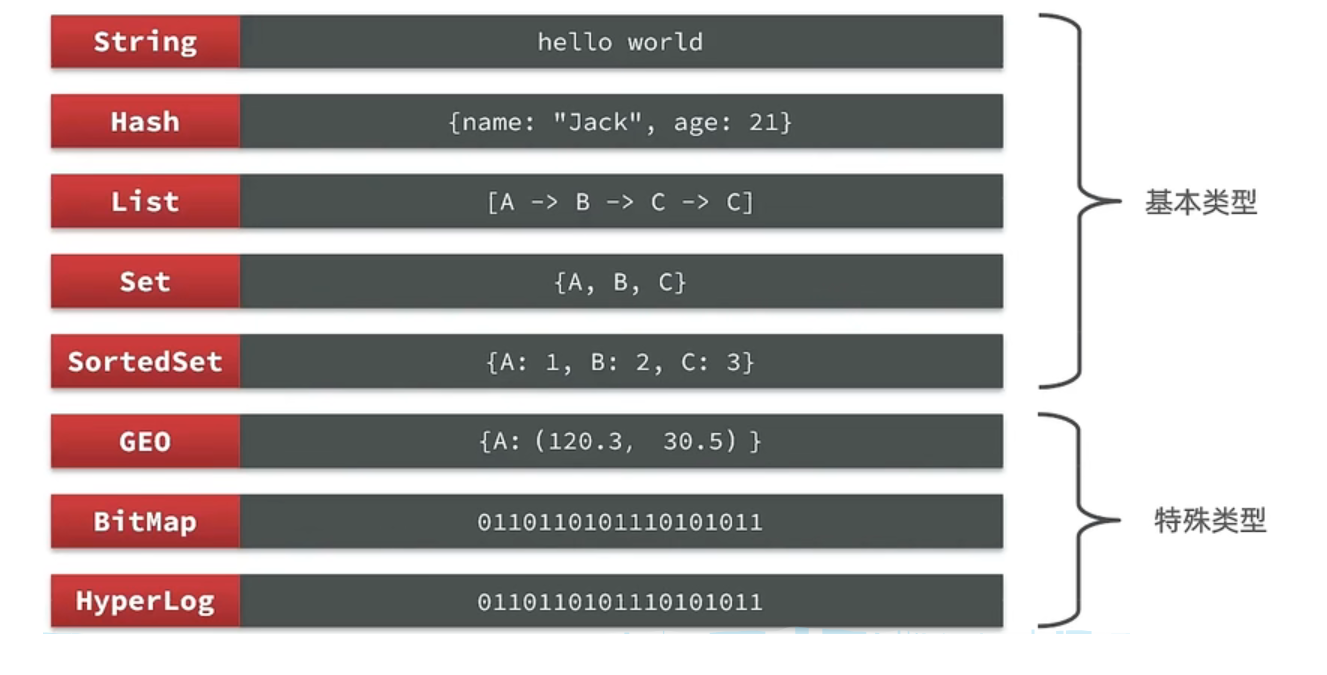
* 键值( key-value)型,value支持多种不同数据结构，功能丰富
* 单线程，每个命令具备原子性
* 低延迟，速度快（基于内存、IO多路复用、良好的编码)。
* 支持数据持久化
* 支持主从集群、分片集群
* 支持多语言客户端

**2. 数据结构**

Redis存储的是 key-value 格式的数据，其中 key 一般是字符串，value 的形式多样

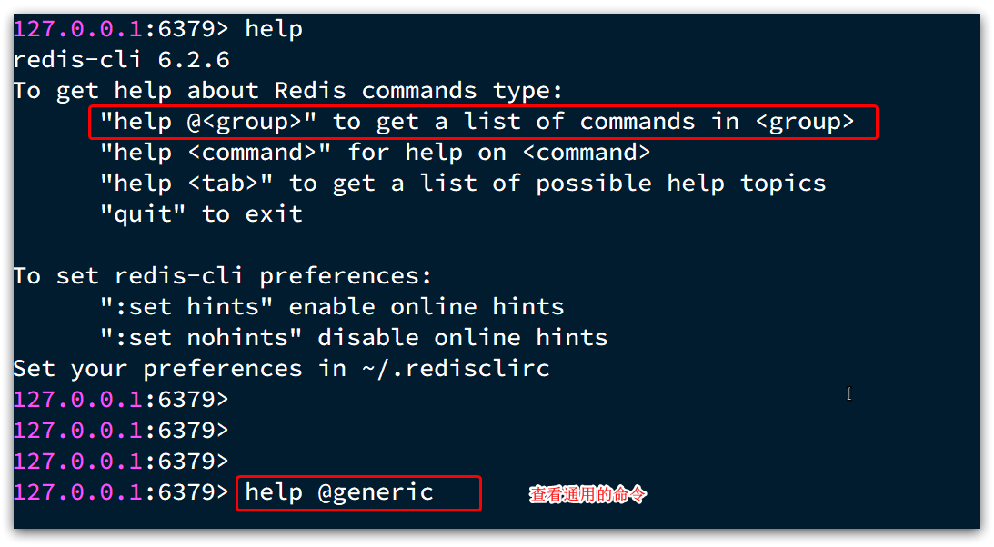
五种基本类型：

* 字符串类型 String
* 哈希类型 Hash：map 格式
* 列表类型 List：LinkedList 格式（支持重复元素）
* 集合类型 Set：HashSet 格式（不允许重复元素）
* 有序集合类型 SortedSet：不允许重复元素，且元素有顺序



在官网（ <https://redis.io/commands> ）可以查看到不同的命令

也可以通过Help命令来帮助我们去查看命令



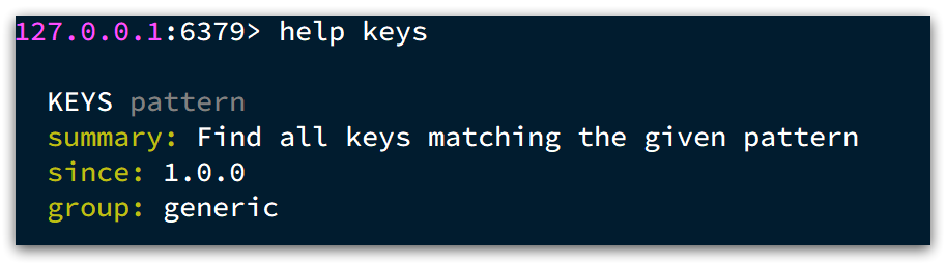
**3.常见命令**

**3.1.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：



* KEYS

KEYS \*：查询所有键  
KEYS key：查询指定名的键  
KEYS 正则表达式：通过正则表达式查询键   # 查询以a开头的key keys a\*

**在生产环境下，不推荐使用keys 命令，因为这个命令在key过多的情况下，效率不高**

* DEL

127.0.0.1:6379> help del  
​  
DEL key [key ...]  
summary: Delete a key  
since: 1.0.0  
group: generic  
​  
127.0.0.1:6379> del name #删除单个  
(integer) 1 #成功删除1个  
​  
127.0.0.1:6379> keys \*  
1) "k3"  
2) "k2"  
3) "k1"  
4) "age"  
​  
127.0.0.1:6379> del k1 k2 k3 k4  
(integer) 3   #此处返回的是成功删除的key，由于redis中只有k1,k2,k3 所以只成功删除3个，最终返回  
127.0.0.1:6379>

* EXISTS

127.0.0.1:6379> exists age  
(integer) 1

* EXPIRE

内存非常宝贵，对于一些数据，我们应当给他一些过期时间，当过期时间到了之后，他就会自动被删除

127.0.0.1:6379> expire age 10  
(integer) 1  
​  
127.0.0.1:6379> ttl age  
(integer) 8  
​  
127.0.0.1:6379> ttl age  
(integer) 6  
​  
127.0.0.1:6379> ttl age  
(integer) -2  
​  
127.0.0.1:6379> ttl age  
(integer) -2 #当这个key过期了，那么此时查询出来就是-2   
​  
127.0.0.1:6379> keys \*  
(empty list or set)  
​  
127.0.0.1:6379> set age 10 #如果没有设置过期时间  
OK  
​  
127.0.0.1:6379> ttl age  
(integer) -1 # ttl的返回值就是-1

**3.2.String命令**

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

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

以上命令除了INCRBYFLOAT 都是常用命令

* SET 和GET: 如果key不存在则是新增，如果存在则是修改

127.0.0.1:6379> set name Rose //原来不存在  
OK  
​  
127.0.0.1:6379> get name   
"Rose"  
​  
127.0.0.1:6379> set name Jack //原来存在，就是修改  
OK  
​  
127.0.0.1:6379> get name  
"Jack"

* MSET和MGET

127.0.0.1:6379> MSET k1 v1 k2 v2 k3 v3  
OK  
​  
127.0.0.1:6379> MGET name age k1 k2 k3  
1) "Jack" //之前存在的name  
2) "10"   //之前存在的age  
3) "v1"  
4) "v2"  
5) "v3"

* INCR和INCRBY和DECY

127.0.0.1:6379> get age   
"10"  
​  
127.0.0.1:6379> incr age //增加1  
(integer) 11  
     
127.0.0.1:6379> get age //获得age  
"11"  
​  
127.0.0.1:6379> incrby age 2 //一次增加2  
(integer) 13 //返回目前的age的值  
     
127.0.0.1:6379> incrby age 2  
(integer) 15  
     
127.0.0.1:6379> incrby age -1 //也可以增加负数，相当于减  
(integer) 14  
     
127.0.0.1:6379> incrby age -2 //一次减少2个  
(integer) 12  
     
127.0.0.1:6379> DECR age //相当于 incr 负数，减少正常用法  
(integer) 11  
     
127.0.0.1:6379> get age   
"11"  
​

* 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

127.0.0.1:6379> help setnx  
​  
SETNX key value  
summary: Set the value of a key, only if the key does not exist  
since: 1.0.0  
group: string  
​  
127.0.0.1:6379> set name Jack //设置名称  
OK  
127.0.0.1:6379> setnx name lisi //如果key不存在，则添加成功  
(integer) 0  
127.0.0.1:6379> get name //由于name已经存在，所以lisi的操作失败  
"Jack"  
127.0.0.1:6379> setnx name2 lisi //name2 不存在，所以操作成功  
(integer) 1  
127.0.0.1:6379> get name2   
"lisi"

* SETEX:添加一个String类型的键值对，并且指定有效期

127.0.0.1:6379> setex name 10  
OK  
​  
127.0.0.1:6379> ttl name  
(integer) 8  
​  
127.0.0.1:6379> ttl name  
(integer) 7  
​  
127.0.0.1:6379> ttl name  
(integer) 5

**3.3.Key的层级结构**

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：

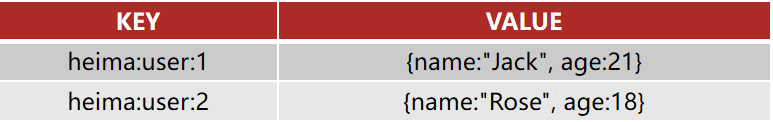


这个格式并非固定，也可以根据自己的需求来删除或添加词条。

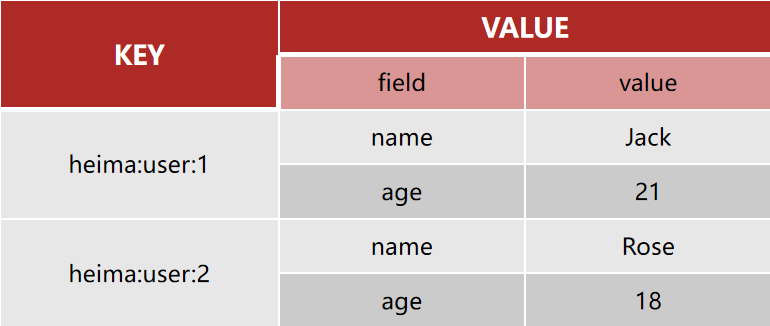
**3.4.Hash命令**

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



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



**Hash类型的常见命令**

* 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
* 删除：HDEL key field
* HSET和HGET

127.0.0.1:6379> HSET heima:user:3 name Lucy//大key是 heima:user:3 小key是name，小value是Lucy  
(integer) 1  
127.0.0.1:6379> HSET heima:user:3 age 21// 如果操作不存在的数据，则是新增  
(integer) 1  
127.0.0.1:6379> HSET heima:user:3 age 17 //如果操作存在的数据，则是修改  
(integer) 0  
127.0.0.1:6379> HGET heima:user:3 name   
"Lucy"  
127.0.0.1:6379> HGET heima:user:3 age  
"17"

* HMSET和HMGET

127.0.0.1:6379> HMSET heima:user:4 name HanMeiMei  
OK  
127.0.0.1:6379> HMSET heima:user:4 name LiLei age 20 sex man  
OK  
127.0.0.1:6379> HMGET heima:user:4 name age sex  
1) "LiLei"  
2) "20"  
3) "man"

* HGETALL：获取一个hash类型的key中的所有的field和value

127.0.0.1:6379> HGETALL heima:user:4  
1) "name"  
2) "LiLei"  
3) "age"  
4) "20"  
5) "sex"  
6) "man"

* HKEYS和HVALS

127.0.0.1:6379> HKEYS heima:user:4  
1) "name"  
2) "age"  
3) "sex"  
127.0.0.1:6379> HVALS heima:user:4  
1) "LiLei"  
2) "20"  
3) "man"

* HINCRBY：让一个hash类型key的字段值自增并指定步长

127.0.0.1:6379> HINCRBY heima:user:4 age 2  
(integer) 22  
127.0.0.1:6379> HVALS heima:user:4  
1) "LiLei"  
2) "22"  
3) "man"  
127.0.0.1:6379> HINCRBY heima:user:4 age -2  
(integer) 20

* 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

127.0.0.1:6379> HSETNX heima:user:4 sex woman  
(integer) 1  
127.0.0.1:6379> HGETALL heima:user:3  
1) "name"  
2) "Lucy"  
3) "age"  
4) "17"  
127.0.0.1:6379> HSETNX heima:user:3 sex woman  
(integer) 1  
127.0.0.1:6379> HGETALL heima:user:3  
1) "name"  
2) "Lucy"  
3) "age"  
4) "17"  
5) "sex"  
6) "woman"

* 删除：HDEL key field

127.0.0.1:6379> HDEL myhash username  
(integer) 1

**3.5.List命令**

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。

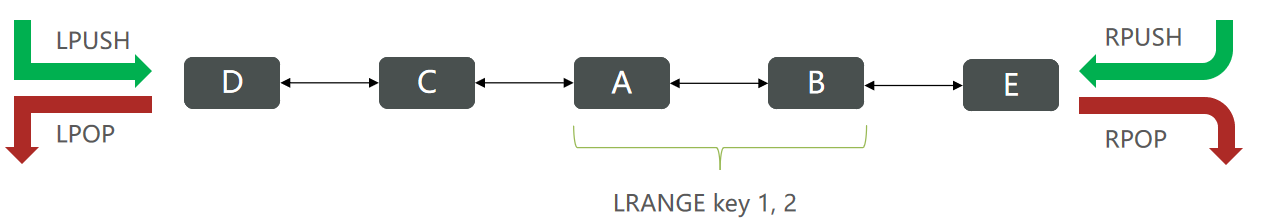
特征也与LinkedList类似：

* 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 star 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



* LPUSH和RPUSH

127.0.0.1:6379> LPUSH users 1 2 3  
(integer) 3  
127.0.0.1:6379> RPUSH users 4 5 6  
(integer) 6

* LPOP和RPOP

127.0.0.1:6379> LPOP users  
"3"  
127.0.0.1:6379> RPOP users  
"6"

* LRANGE

127.0.0.1:6379> LRANGE users 1 2  
1) "1"  
2) "2"

**3.6.Set命令**

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类型的常见命令**

* 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
* SDIFF key1 key2 ... ：求key1与key2的差集
* SUNION key1 key2 ..：求key1和key2的并集
* 127.0.0.1:6379> sadd s1 a b c  
  (integer) 3  
  127.0.0.1:6379> smembers s1  
  1) "c"  
  2) "b"  
  3) "a"  
  127.0.0.1:6379> srem s1 a  
  (integer) 1  
       
  127.0.0.1:6379> SISMEMBER s1 a  
  (integer) 0  
       
  127.0.0.1:6379> SISMEMBER s1 b  
  (integer) 1  
       
  127.0.0.1:6379> SCARD s1  
  (integer) 2

**案例**

* 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
* 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

127.0.0.1:6379> SADD zs lisi wangwu zhaoliu  
(integer) 3  
     
127.0.0.1:6379> SADD ls wangwu mazi ergou  
(integer) 3  
     
127.0.0.1:6379> SCARD zs  
(integer) 3  
     
127.0.0.1:6379> SINTER zs ls  
1) "wangwu"  
     
127.0.0.1:6379> SDIFF zs ls  
1) "zhaoliu"  
2) "lisi"  
     
127.0.0.1:6379> SUNION zs ls  
1) "wangwu"  
2) "zhaoliu"  
3) "lisi"  
4) "mazi"  
5) "ergou"  
     
127.0.0.1:6379> SISMEMBER zs lisi  
(integer) 1  
     
127.0.0.1:6379> SISMEMBER ls zhangsan  
(integer) 0  
     
127.0.0.1:6379> SREM zs lisi  
(integer) 1  
     
127.0.0.1:6379> SMEMBERS zs  
1) "zhaoliu"  
2) "wangwu"

**3.7.SortedSet命令**

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 memeber

**4.Redis的Java客户端-Jedis**

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

* 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。
* 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。

Jedis 直连，本质是定义一个 tcp 连接，然后使用 socket 技术进行通信。

**4.1.使用**

1）引入依赖：

<!--jedis-->  
<dependency>  
   <groupId>redis.clients</groupId>  
   <artifactId>jedis</artifactId>  
   <version>3.7.0</version>  
</dependency>

2）建立连接

public class redis {  
   public static void main(String[] args) {  
private Jedis jedis;  
   // 1.建立连接  
   // jedis = new Jedis("localhost", 6379);   默认 6379  
   // Jedis jedis = new Jedis(); 默认 localhost:6379  
   jedis = JedisConnectionFactory.getJedis();  
   // 2.设置密码（如果密码为空，则不需要）  
   jedis.auth("123456");  
   // 3.选择库  
   jedis.select(0);  
   // 4.操作  
   jedis.set("username", "zhangsan");  
   // 插入hash数据  
   jedis.hset("user:1", "name", "Jack");  
   jedis.hset("user:1", "age", "21");  
   // 获取  
   Map<String, String> map = jedis.hgetAll("user:1");  
   System.out.println(map);  
   // 5.关闭连接  
   if (jedis != null) {  
       jedis.close();  
  }  
  }  
}

**4.2.Jedis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

Jedis 连接池：JedisPool（Jedis 自带）

<!--common-pool-->  
       <dependency>  
           <groupId>org.apache.commons</groupId>  
           <artifactId>commons-pool2</artifactId>  
       </dependency>

public class JedisConnectionFacotry {  
​  
    private static final JedisPool jedisPool;  
​  
    static {  
        //配置连接池  
        JedisPoolConfig poolConfig = new JedisPoolConfig();  
        poolConfig.setMaxTotal(8);  
        poolConfig.setMaxIdle(8);  
        poolConfig.setMinIdle(0);  
        poolConfig.setMaxWaitMillis(1000);  
        //创建连接池对象  
        jedisPool = new JedisPool(poolConfig,  
                "192.168.150.101",6379,1000,"123321");  
    }  
​  
    public static Jedis getJedis(){  
        return jedisPool.getResource();  
    }  
}

**代码说明：**

* 1） JedisConnectionFacotry：工厂设计模式是实际开发中非常常用的一种设计模式，我们可以使用工厂，去降低代的耦合，比如Spring中的Bean的创建，就用到了工厂设计模式
* 2）静态代码块：随着类的加载而加载，确保只能执行一次，我们在加载当前工厂类的时候，就可以执行static的操作完成对 连接池的初始化
* 3）最后提供返回连接池中连接的方法.

当我们使用了连接池后，当我们关闭连接其实并不是关闭，而是将Jedis还回连接池

【Jedis 详细配置】

#最大活动对象数       
redis.pool.maxTotal=1000      
#最大能够保持idel状态的对象数        
redis.pool.maxIdle=100    
#最小能够保持idel状态的对象数     
redis.pool.minIdle=50      
#当池内没有返回对象时，最大等待时间      
redis.pool.maxWaitMillis=10000      
#当调用borrow Object方法时，是否进行有效性检查      
redis.pool.testOnBorrow=true      
#当调用return Object方法时，是否进行有效性检查      
redis.pool.testOnReturn=true    
#“空闲链接”检测线程，检测的周期，毫秒数。如果为负值，表示不运行“检测线程”。默认为-1.    
redis.pool.timeBetweenEvictionRunsMillis=30000    
#向调用者输出“链接”对象时，是否检测它的空闲超时；    
redis.pool.testWhileIdle=true    
# 对于“空闲链接”检测线程而言，每次检测的链接资源的个数。默认为3.    
redis.pool.numTestsPerEvictionRun=50    
#redis服务器的IP      
redis.ip=xxxxxx    
#redis服务器的Port      
redis1.port=6379

**5.Redis的Java客户端-SpringDataRedis**

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：



**5.1.导入pom坐标**

<!--redis依赖-->  
       <dependency>  
           <groupId>org.springframework.boot</groupId>  
           <artifactId>spring-boot-starter-data-redis</artifactId>  
       </dependency>  
       <!--common-pool-->  
       <dependency>  
           <groupId>org.apache.commons</groupId>  
           <artifactId>commons-pool2</artifactId>  
       </dependency>  
       <!--Jackson依赖-->  
       <dependency>  
           <groupId>com.fasterxml.jackson.core</groupId>  
           <artifactId>jackson-databind</artifactId>  
       </dependency>

**5.2.配置文件**

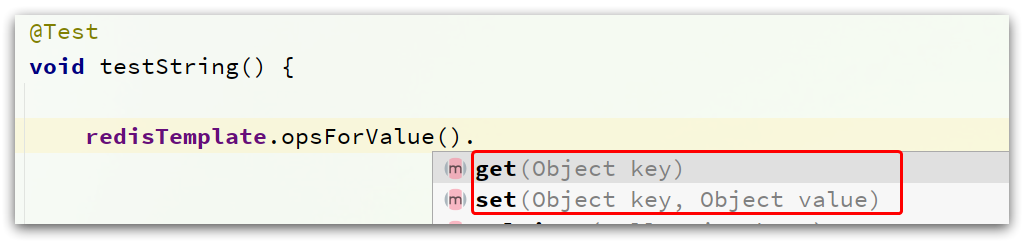
spring:  
redis:  
  host: 192.168.150.101  
  port: 6379  
  password: 123456  
  lettuce:  
    pool:  
      max-active: 8  #最大连接  
      max-idle: 8   #最大空闲连接  
      min-idle: 0   #最小空闲连接  
      max-wait: 100ms #连接等待时间

**5.3.测试**

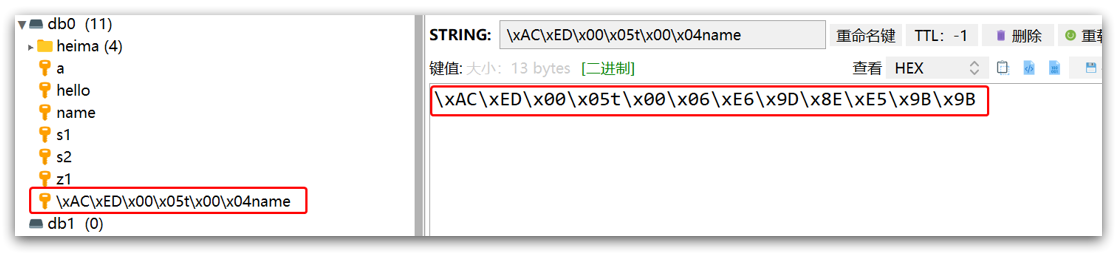
@SpringBootTest  
class RedisDemoApplicationTests {  
​  
   @Autowired  
   private RedisTemplate<String, Object> redisTemplate;  
​  
   @Test  
   void testString() {  
       // 写入一条String数据  
       redisTemplate.opsForValue().set("name", "虎哥");  
       // 获取string数据  
       Object name = redisTemplate.opsForValue().get("name");  
       System.out.println("name = " + name);  
  }  
}

**5.4.数据序列化器**

RedisTemplate可以接收任意Object作为值写入Redis：



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：



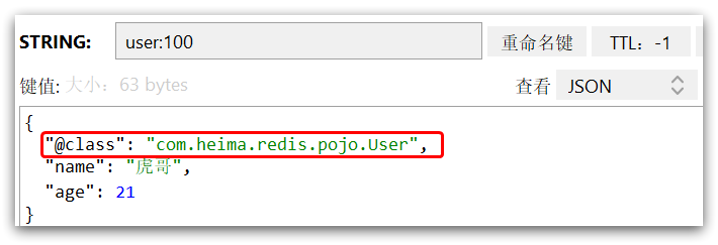
缺点：

* 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：

@Configuration  
public class RedisConfig {  
​  
   @Bean  
   public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory connectionFactory){  
       // 创建RedisTemplate对象  
       RedisTemplate<String, Object> template = new RedisTemplate<>();  
       // 设置连接工厂  
       template.setConnectionFactory(connectionFactory);  
       // 创建JSON序列化工具  
       GenericJackson2JsonRedisSerializer jsonRedisSerializer =   
          new GenericJackson2JsonRedisSerializer();  
       // 设置Key的序列化  
       template.setKeySerializer(RedisSerializer.string());  
       template.setHashKeySerializer(RedisSerializer.string());  
       // 设置Value的序列化  
       template.setValueSerializer(jsonRedisSerializer);  
       template.setHashValueSerializer(jsonRedisSerializer);  
       // 返回  
       return template;  
  }  
}

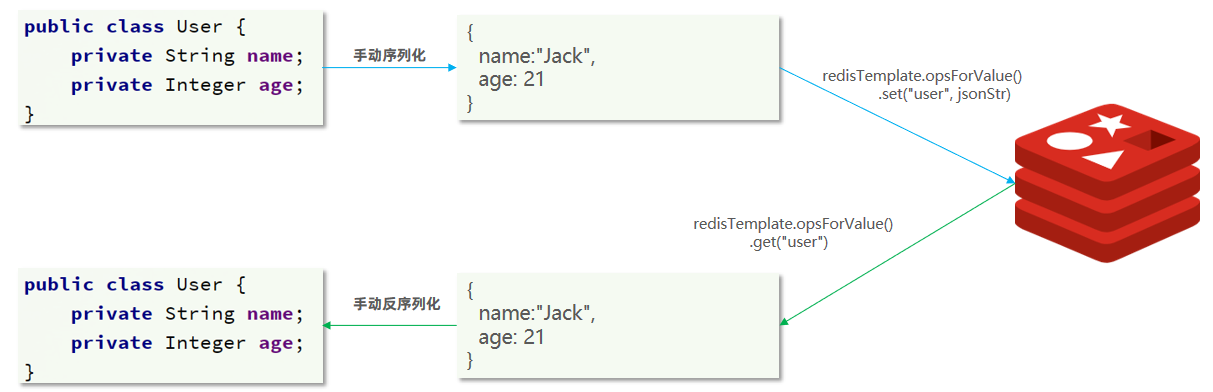
这里采用了JSON序列化来代替默认的JDK序列化方式。最终结果如图：



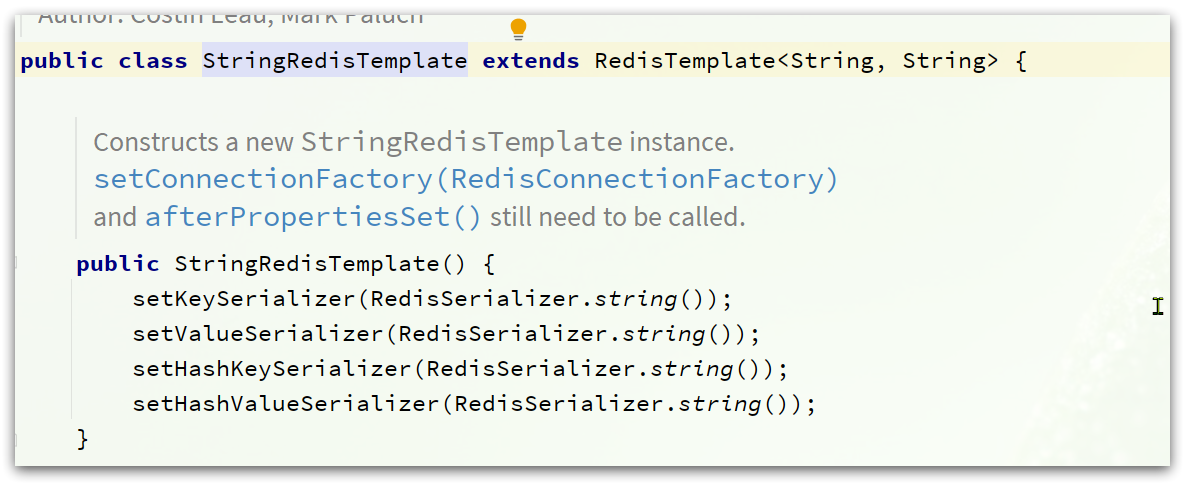
整体可读性有了很大提升，并且能将Java对象自动的序列化为JSON字符串，并且查询时能自动把JSON反序列化为Java对象。不过，其中记录了序列化时对应的class名称，目的是为了查询时实现自动反序列化。这会带来额外的内存开销。

**5.5.StringRedisTemplate**

为了减少内存的消耗，我们可以采用手动序列化的方式，换句话说，就是不借助默认的序列化器，而是我们自己来控制序列化的动作，同时，我们只采用String的序列化器，这样，在存储value时，我们就不需要在内存中就不用多存储数据，从而节约我们的内存空间



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。



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

@SpringBootTest  
class RedisStringTests {  
​  
   @Autowired  
   private StringRedisTemplate stringRedisTemplate;  
​  
   @Test  
   void testString() {  
       // 写入一条String数据  
       stringRedisTemplate.opsForValue().set("verify:phone:13600527634", "124143");  
       // 获取string数据  
       Object name = stringRedisTemplate.opsForValue().get("name");  
       System.out.println("name = " + name);  
  }  
​  
   private static final ObjectMapper mapper = new ObjectMapper();  
​  
   @Test  
   void testSaveUser() throws JsonProcessingException {  
       // 创建对象  
       User user = new User("虎哥", 21);  
       // 手动序列化  
       String json = mapper.writeValueAsString(user);  
       // 写入数据  
       stringRedisTemplate.opsForValue().set("user:200", json);  
​  
       // 获取数据  
       String jsonUser = stringRedisTemplate.opsForValue().get("user:200");  
       // 手动反序列化  
       User user1 = mapper.readValue(jsonUser, User.class);  
       System.out.println("user1 = " + user1);  
  }  
​  
}

RedisTemplate的两种序列化实践方案：

* 方案一：
  + 自定义RedisTemplate
  + 修改RedisTemplate的序列化器为GenericJackson2JsonRedisSerializer
* 方案二：
  + 使用StringRedisTemplate
  + 写入Redis时，手动把对象序列化为JSON
  + 读取Redis时，手动把读取到的JSON反序列化为对象

**5.5.6Hash结构操作**

@SpringBootTest  
class RedisStringTests {  
​  
   @Autowired  
   private StringRedisTemplate stringRedisTemplate;  
​  
​  
   @Test  
   void testHash() {  
       stringRedisTemplate.opsForHash().put("user:400", "name", "虎哥");  
       stringRedisTemplate.opsForHash().put("user:400", "age", "21");  
​  
       Map<Object, Object> entries = stringRedisTemplate.opsForHash().entries("user:400");  
       System.out.println("entries = " + entries);  
  }  
}

**6.持久化**

Redis 是一个内存数据库，当 Redis 服务器重启，或者计算机重启，数据会丢失，我们可以将 Redis 内存中的数据持久化保存到硬盘的文件中。

Redis 持久化机制：

* 【RDB】：默认方式，不需要进行配置，默认就使用这种机制

在一定的间隔时间中，检测 key 的变化情况，然后持久化数据，同时默认在 Redis 根目录生成一个 dump.rdb 持久化文件。

1. 编辑 redis.windwos.conf 文件

#   after 900 sec (15 min) if at least 1 key changed  
# 在 15 分钟之后，只要有 1 个 key 发生改变，就持久化一次  
save 900 1  
#   after 300 sec (5 min) if at least 10 keys changed  
# 在 5 分钟之后，只要有 10 个 key 发生改变，就持久化一次  
save 300 10  
#   after 60 sec if at least 10000 keys changed  
# 在 60 秒之后，只要有 10000 个 key 发生改变，就持久化一次  
save 60 10000

1. 重新启动 Redis 服务，并指定配置文件名称

redis-server redis.windows.conf

* 【AOF】：日志记录的方式，可以记录每一条命令的操作，可以每一次命令操作后，持久化数据（对性能消耗较大，一般不推荐！），同时默认在 Redis 根目录生成一个 appendonly.aof 持久化文件。

1. 编辑 redis.windwos.conf 文件

appendonly no（关闭） --> appendonly yes （开启）  
  
# appendfsync always ： 每一次操作都进行持久化  
appendfsync everysec ： 每隔一秒进行一次持久化  
# appendfsync no ： 不进行持久化

1. 重新启动 Redis 服务，并指定配置文件名称

redis-server redis.windows.conf

注意：虽然 Redis 有持久化机制，但是其持久化是存在一定风险的，还要根据具体业务具体分析！