# [Redis支持的数据类型](https://www.cnblogs.com/jasontec/p/9699242.html)和[持久化方式]( https://blog.csdn.net/yhl_jxy/article/details/91879874 )

## 1.Redis支持的数据类型？

### 1.1 String的实际应用场景比较广泛的有：

* **缓存功能：String**字符串是最常用的数据类型，不仅仅是**Redis**，各个语言都是最基本类型，因此，利用**Redis**作为缓存，配合其它数据库作为存储层，利用**Redis**支持高并发的特点，可以大大加快系统的读写速度、以及降低后端数据库的压力。
* **计数器：**许多系统都会使用**Redis**作为系统的实时计数器，可以快速实现计数和查询的功能。而且最终的数据结果可以按照特定的时间落地到数据库或者其它存储介质当中进行永久保存。
  + **incr命令**是增加键的整数值一次，如果键没有被设置，则设置为1，返回自增后的数值。这里利用了redis的原子性，即同一个键只允许一个客户端操作，确保自增值准确。
* **共享用户Session：**用户重新刷新一次界面，可能需要访问一下数据进行重新登录，或者访问页面缓存**Cookie**，但是可以利用**Redis**将用户的**Session**集中管理，在这种模式只需要保证**Redis**的高可用，每次用户**Session**的更新和获取都可以快速完成。大大提高效率。

### 1.2 Hash：

这个是类似 **Map** 的一种结构，这个一般就是可以将结构化的数据，比如一个对象（前提是**这个对象没嵌套其他的对象**）给缓存在 **Redis** 里，然后每次读写缓存的时候，可以就操作 **Hash** 里的**某个字段**。

但是这个的场景其实还是多少单一了一些，因为现在很多对象都是比较复杂的，比如你的商品对象可能里面就包含了很多属性，其中也有对象。我自己使用的场景用得不是那么多。

[详细使用参考Redis的Hash用法](https://blog.csdn.net/qq_41307443/article/details/79766930)

1.hset key value(key value) :向Hash中存入值。

2.hget key value(key）:取出Hash中key的值。

3.hmset ：向Hash表中存入该对象的多个属性值。

注意：当向同一个对象的同一个属性赋多个值时，会覆盖。不同属性时，会拼接。

4.hsetnx key：判断新增的属性值是否存在，存在新增失败返回0，不存在新增成功，返回1.

### 1.3 List：

**List** 是有序列表，这个还是可以玩儿出很多花样的。

* 比如可以通过 **List** 存储一些列表型的数据结构，类似粉丝列表、文章的评论列表之类的东西。
* 比如可以通过 **lrange** 命令，读取某个闭区间内的元素，可以基于 **List** 实现分页查询，这个是很棒的一个功能，基于 **Redis** 实现简单的高性能分页，可以做类似微博那种下拉不断分页的东西，性能高，就一页一页走。
  + Redis Lrange 返回列表中指定区间内的元素，区间以偏移量 START 和 END 指定。 其中 0 表示列表的第一个元素， 1 表示列表的第二个元素，以此类推。 你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。
  + redis Lrange 命令基本语法如下：
  + redis 127.0.0.1:6379> LRANGE KEY\_NAME START END
* 比如可以搞个简单的消息队列，从 **List** 头怼进去，从 **List** 屁股那里弄出来。

**List**本身就是我们在开发过程中比较常用的数据结构了，热点数据更不用说了。

* **消息队列：Redis**的链表结构，可以轻松实现阻塞队列，可以使用左进右出的命令组成来完成队列的设计。比如：数据的生产者可以通过**Lpush**命令从左边插入数据，多个数据消费者，可以使用**BRpop**命令阻塞的“抢”列表尾部的数据。
* 文章列表或者数据分页展示的应用。

比如，我们常用的博客网站的文章列表，当用户量越来越多时，而且每一个用户都有自己的文章列表，而且当文章多时，都需要分页展示，这时可以考虑使用**Redis**的列表，列表不但有序同时还支持按照范围内获取元素，可以完美解决分页查询功能。大大提高查询效率。

### 1.4 Set：

**Set** 是无序集合，会自动去重的那种。

直接基于 **Set** 将系统里需要去重的数据扔进去，自动就给去重了，如果你需要对一些数据进行快速的全局去重，你当然也可以基于 **JVM** 内存里的 **HashSet** 进行去重，但是如果你的某个系统部署在多台机器上呢？得基于**Redis**进行全局的 **Set** 去重。

可以基于 **Set** 玩儿交集、并集、差集的操作，比如交集吧，我们可以把两个人的好友列表整一个交集，看看俩人的共同好友是谁？对吧。

反正这些场景比较多，因为对比很快，操作也简单，两个查询一个**Set**搞定。

### 1.5 Sorted Set：

**Sorted set** 是排序的 **Set**，去重但可以排序，写进去的时候给一个分数，自动根据分数排序。

有序集合的使用场景与集合类似，但是set集合不是自动有序的，而**Sorted set**可以利用分数进行成员间的排序，而且是插入时就排序好。所以当你需要一个有序且不重复的集合列表时，就可以选择**Sorted set**数据结构作为选择方案。

* 排行榜：有序集合经典使用场景。例如视频网站需要对用户上传的视频做排行榜，榜单维护可能是多方面：按照时间、按照播放量、按照获得的赞数等。
* 用**Sorted Sets**来做带权重的队列，比如普通消息的score为1，重要消息的score为2，然后工作线程可以选择按score的倒序来获取工作任务。让重要的任务优先执行。

微博热搜榜，就是有个后面的热度值，前面就是名称

redis 127.0.0.1:6379> ZADD runoobkey 1 redis
  
(integer) 1
  
redis 127.0.0.1:6379> ZADD runoobkey 2 mongodb
  
(integer) 1
  
redis 127.0.0.1:6379> ZADD runoobkey 3 mysql
  
(integer) 1
  
redis 127.0.0.1:6379> ZADD runoobkey 3 mysql
  
(integer) 0
  
redis 127.0.0.1:6379> ZADD runoobkey 4 mysql
  
(integer) 0
  
redis 127.0.0.1:6379> ZRANGE runoobkey 0 10 WITHSCORES
  
   
1) "redis"
  
2) "1"
  
3) "mongodb"
  
4) "2"
  
5) "mysql"
  
6) "4"

### 高级用法：

### **Bitmap** :

位图是支持按 bit 位来存储信息，可以用来实现 **布隆过滤器（BloomFilter）**；

## 2.什么是Redis持久化？Redis有哪几种持久化方式？优缺点是什么？

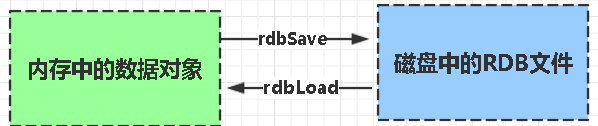
持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失。

Redis 提供了两种持久化方式:RDB（默认） 和AOF

### 2.1RDB：

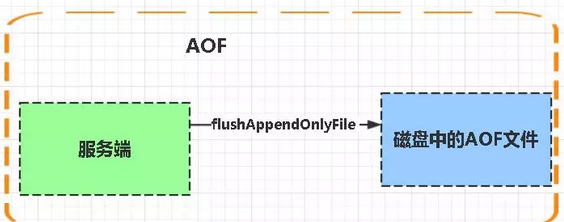
rdb是Redis DataBase缩写

功能核心函数rdbSave(生成RDB文件)和rdbLoad（从文件加载内存）两个函数



### 2.2AOF:

Aof是Append-only file缩写



每当执行服务器(定时)任务或者函数时flushAppendOnlyFile 函数都会被调用， 这个函数执行以下两个工作

aof写入保存：

WRITE：根据条件，将 aof\_buf 中的缓存写入到 AOF 文件

SAVE：根据条件，调用 fsync 或 fdatasync 函数，将 AOF 文件保存到磁盘中。

**存储结构:**

内容是redis通讯协议(RESP )格式的命令文本存储。

**比较**：

1、aof文件比rdb更新频率高，优先使用aof还原数据。

2、aof比rdb更安全也更大

3、rdb性能比aof好

4、如果两个都配了优先加载AOF

### [2.3混合持久化]( https://blog.csdn.net/yhl_jxy/article/details/91879874 )

重启Redis时，我们很少使用rdb来恢复内存状态，因为会丢失大量数据。我们通常使用AOF日志重写，

但是AOF重写性能相对rdb来说要慢很多，这样在Redis实例很大的情况下，启动需要花费很长的时间。

Redis 4.0 为了解决这个问题，带来了一个新的持久化选项——混合持久化。

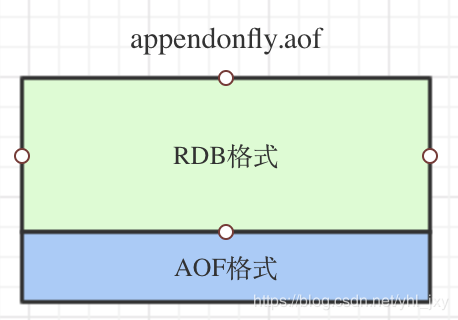
1、混合持久化  
 AOF在进行文件重写(aof文件里可能有太多没用指令，所以aof会定期根据内存的最新数据生成aof文件)

时将重写这一刻之前的内存rdb快照文件的内容和增量的AOF修改内存数据的命令日志文件存在一起，

都写入新的aof文件，新的文件一开始不叫appendonly.aof，等到重写完新的AOF文件才会进行改名，

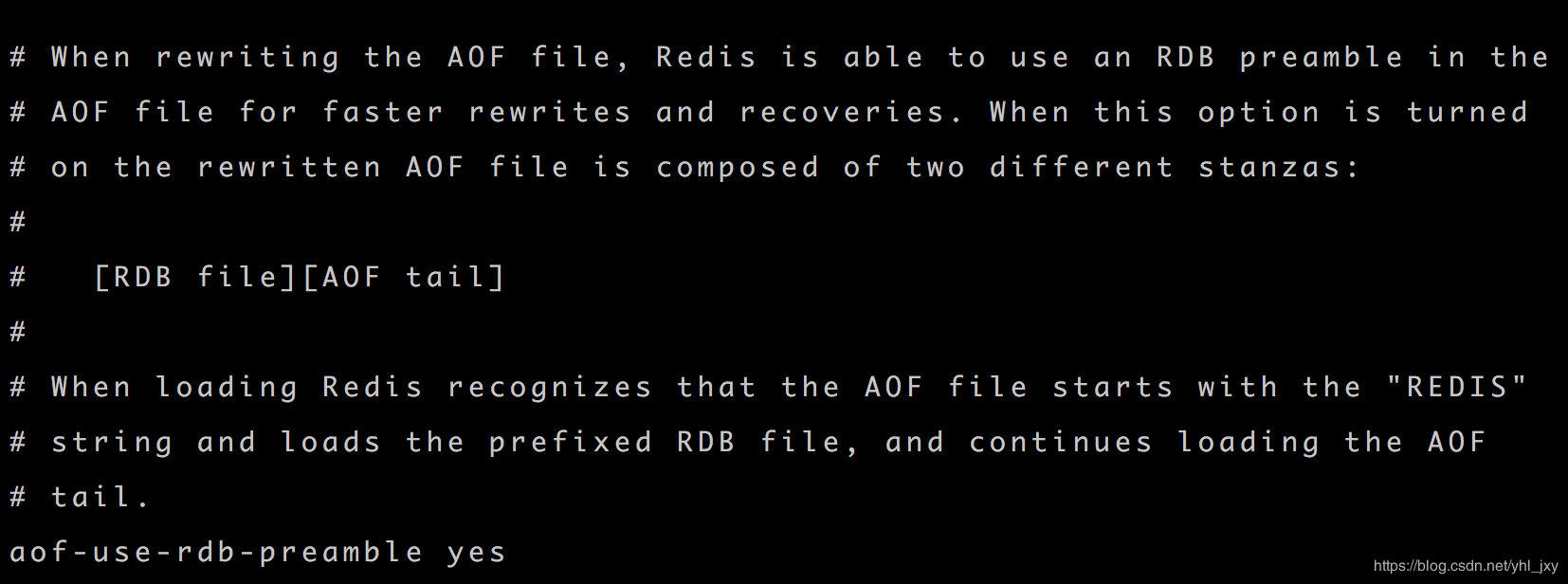
原子的覆盖原有的AOF文件，完成新旧两个AOF文件的替换。

混合持久化文件结构：



2、混合持久化配置  
开启混合持久化：

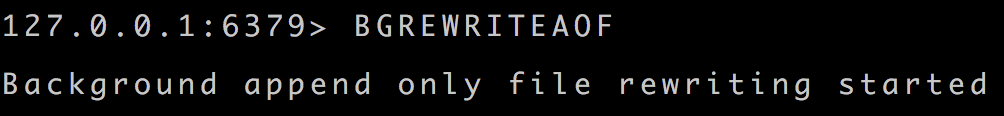
aof-use-rdb-preamble yes



重启Redis服务。

3、bgrewriteaof重写AOF  
AOF根据配置规则在后台自动重写aof文件，也可以人为执行命令bgrewriteaof重写AOF。 于是在Redis重启的时候，

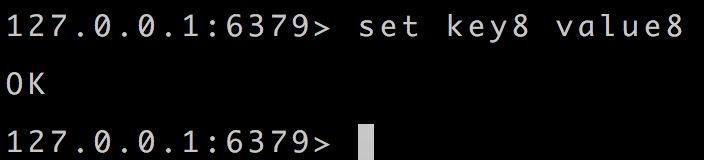
4、实例验证  
通过bgrewriteaof手动重写aof。



查看appendonly.aof文件内容，看到rdb格式的二进制文件。



再执行一条set命令。



然后再看appendonly.aof文件内容。



从文件格式可以看到，混合持久化appendonly.aof文件由rdb格式和aof指令格式两大部分组成。

Redis在重启时，先重写rdb到内存，然后在重写aof到内存，重启效率高，还能减少数据的丢失。