ALIBABA CLOUD

# 阿里云

## 专有云企业版

## 云数据库 Redis 版

## 用户指南

产品版本：v3.16.2

文档版本：20220915

阿里云

# 法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

# 通用约定

| 格式 | 说明 | 样例 |
| --- | --- | --- |
| 危险 | 该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 危险<br>重置操作将丢失用户配置数据。 |
| 警告 | 该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 警告<br>重启操作将导致业务中断，恢复业务时间约十分钟。 |
| 注意 | 用于警示信息、补充说明等，是用户必须了解的内容。 | 注意<br>权重设置为0，该服务器不会再接受新请求。 |
| 说明 | 用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。 | 说明<br>您也可以通过按Ctrl+A选中全部文件。 |
| > | 多级菜单递进。 | 单击**设置**> **网络**> **设置网络类型**。 |
| **粗体** | 表示按键、菜单、页面名称等UI元素。 | 在**结果确认**页面，单击**确定**。 |
| Courier字体 | 命令或代码。 | 执行 `cd /d C:/window` 命令，进入Windows系统文件夹。 |
| *斜体* | 表示参数、变量。 | `bae log list --instanceid`<br>*Instance_ID* |
| [] 或者 [a\|b] | 表示可选项，至多选择一个。 | `ipconfig [-all\|-t]` |
| {} 或者 {a\|b} | 表示必选项，至多选择一个。 | `switch {active\|stand}` |

# 目录

# 1.什么是云数据库Redis

云数据库Redis（KVStore for Redis）是兼容开源Redis协议标准的数据库服务，基于双机热备架构及集群架构，可满足高吞吐、低延迟及弹性变配等业务需求。

## 功能特性

- 支持String（字符串）、List（链表）、Set（集合）、Sorted Set（有序集合）、Hash（哈希表）、Stream（流数据）等多种数据结构，同时支持Transaction（事务）、Pub/Sub（消息订阅与发布）等高级功能。
- 在社区版的基础上推出企业级缓存服务产品，提供性能增强型供您选择。

## 支持的实例类型

| 实例类型 | 简介 |
| --- | --- |
| 社区版 | 兼容开源Redis的高性能数据缓存服务，支持主从双副本、集群和读写分离架构。 |
| 企业版（性能增强型） | 在云数据库Redis社区版的基础上开发的强化版Redis服务，推出性能增强型产品，采用多线程模型，集成阿里巴巴Tair的部分特性，支持多种Tair数据结构，对于部分特殊业务有很高的适用性。 |

# 2.使用限制

| 项目 | 说明 |
| --- | --- |
| List数据类型 | 没有List个数限制，单个元素最大值为512 MB，推荐List的元素个数小于8192，value最大长度不超过1 MB。 |
| Set数据类型 | 没有Set个数限制，单个元素最大值为512 MB，推荐Set的元素个数小于8192，value最大长度不超过1 MB。 |
| Sorted Set数据类型 | 没有Sorted Set个数限制，单个元素最大值为512 MB，推荐Sorted Set的元素个数小于8192， value最大长度不超过1 MB。 |
| Hash数据类型 | 没有field个数限制，单个元素最大值为512 MB，推荐元素个数小于8192，value最大长度不超过1 MB。 |
| DB数限制 | 每个实例支持256个DB，建议通过拆分Redis的方式来使用多DB。 |
| 数据过期删除策略 | • 主动过期：系统后台会周期性的检测，若发现已过期的key时，会将其删除，但不保证时效性。<br>• 被动过期：将在访问key时同步生效。当用户访问某个key时，若该key已经过期，则将其删除。<br>**说明** 在Redis 4.0版本之前，删除大key会消耗较大资源，可能会导致业务抖动。 |
| 空闲连接回收机制 | 服务端不主动回收Redis空闲连接，由用户管理。 |
| 数据持久化策略 | 采用AOF_FSYNC_EVERYSEC方式，每秒fysnc。 |

# 3.企业版介绍与命令支持

## 3.1. 企业版性能增强型介绍

Redis企业版性能增强型（简称性能增强型）适合并发量大、读写热点多，对性能要求较高的场景。相比社区版，性能增强型主要在多线程和多模块集成方面进行了优化。

### 主要优势

| 类别 | 说明 |
| --- | --- |
| 性能 | • 读写性能达到同规格原生Redis或者云Redis社区版实例的3倍，能够更好地应对热点数据的高频读写。<br>• 相比原生Redis，高QPS场景下响应时间更短，性能表现更佳。<br>• 在大并发场景下运行稳定，可以极大地缓解突发大量请求导致的连接问题，从容应对业务高峰。<br>• 全量同步和增量同步在I/O线程中进行，提高同步速度。 |
| 自研模块 | 集成多个自研的Redis模块，包括CAS和CAD命令、TairString命令、TairHash命令、TairGIS命令、TairBloom命令、TairDoc命令，扩展了Redis的适用性，同时降低了复杂场景下业务的开发难度，让您专注于业务创新。 |
| 兼容性 | 100%兼容原生Redis，无需修改业务代码。 |
| 扩展性 | 支持主从和集群架构，可根据业务需求执行垂直扩容（升级规格）或水平扩容（升级为集群版）。 |

### 适用场景

适用于视频直播、电商秒杀和在线教育等场景，例如：

| 业务上遇到的问题 | 说明 |
| --- | --- |
| 社区版主从架构实例无法满足高QPS需求。 | 使用Redis社区版主从实例在秒杀场景中构建缓存，部分热点Key的QPS要求高达20万以上，社区版实例无法满足业务高峰期的需求。<br>采用性能增强型（主从版）实例后，热门商品秒杀过程流畅，未发生性能问题。 |
| 需要沿用现有主从架构并提升实例性能。 | 由于集群版在存在一定的使用限制，需要保持现有的主从架构。<br>采用性能增强型（主从版）实例，在满足性能需求的同时保持了现有架构，避免升级至集群带来的限制，无需调整现有业务实现方式。 |
| 自建Redis集群分片多、成本高、性能损耗大。 | 随着业务的高速发展，自建Redis集群的分片数不断增加，导致管理维护成本上升，且带来了一定的性能损耗。<br>采用性能增强型（集群版）实例后，得益于其出色的性能，分片数缩减至原有的三分之一，降低了性能损耗，同时，云数据库提供丰富的管理功能帮助您便捷地管理集群。 |

## 性能对比

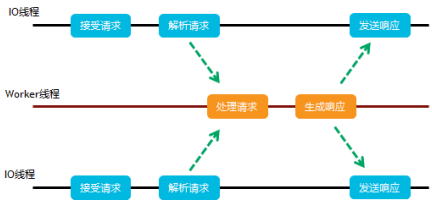
- Redis社区版实例采用与原生Redis相同的单线程模型，每个数据节点支持8万到10万的QPS。
- Redis性能增强型采用多线程模型，由I/O线程、Worker线程和辅助线程共同完成数据处理，单节点性能为社区版实例的3倍左右。

下表展示了不同架构下，社区版和企业版（性能增强型）实例的适用场景对比。

| 架构 | 实例类型 | 说明 |
| --- | --- | --- |
| 标准架构（主从版） | 社区版 | 不适用于单节点QPS要求超过10万的场景。 |
| | 企业版（性能增强型） | 可应用于QPS高于10万的场景。 |
| 集群架构（集群版） | 社区版 | 包含多个数据节点，每个节点的性能与主从版实例相似。当某个节点储存了热度较高的数据并面临大并发量的请求时，该节点中其它数据的读写可能受到影响，形成性能瓶颈。 |
| | 企业版（性能增强型） | 能更好地应对热读写，降低维护成本。 |

## 线程模型对比

| 线程架构 | 说明 |
| --- | --- |
| Redis单线程模型<br>  | Redis社区版和原生Redis采用单线程模型，数据处理流程为：读取请求，解析请求，处理数据，发送响应。其中网络IO和请求解析占用了大部分的资源。 |

| 线程架构 | 说明 |
| --- | --- |
| Redis多线程模型<br>  <br> | 性能增强型将Redis服务各阶段的任务进行分离，通过分工明确的多个线程并行处理各阶段任务，达到提高性能的目的。<br>• IO线程：负责请求读取、响应发送、命令解析等。<br>• Worker线程：负责命令处理、定时器事件等。<br>• 辅助线程：负责高可用探测、保活等。<br>IO线程读取用户的请求并进行解析，之后将解析结果以命令的形式放在队列中发送给Worker线程处理。Worker线程将命令处理完成后生成响应，通过另一条队列发送给IO线程。<br>Redis性能增强型最多支持4个IO线程并发运行。为了提高线程的并行度，IO线程和Worker线程之间采用无锁队列和管道进行数据交换。<br>**说明**<br>• 对于通用的类型，例如：String、List、Hash、Set、Zset、 Hyperloglog、Geo，以及扩展结构都有很好的加速效果。<br>• 对于pub、sub、blocking系列API，由于其复制是在worker中完成的，可被加速提升吞吐，性能提升约50%。<br>• 由于transaction、lua script要求串行执行，无加速效果。 |

**说明** 区别于Redis社区版6.0的多线程（性能至多提升2倍，且CPU资源消耗高），性能增强型的Real Multi-IO能够将IO加速地更彻底，具备更高的抗连接冲击性，且可以线性地提升吞吐能力。

# 3.2. CAS和CAD命令

本文介绍云数据库Redis企业版性能增强型实例中新增的String增强类命令，包括CAS和CAD。

## 使用前提

请注意，本文介绍的命令只有在满足以下条件时才能生效。

- Redis实例为企业版性能增强型。
- 操作对象为性能增强型实例中的Redis String数据。

**说明** 性能增强型实例中可同时设置Redis String（即Redis原生String）和TairString，CAS和CAD只能对Redis String使用。

## 命令列表

## String增强命令

| 命令 | 语法 | 说明 |
| --- | --- | --- |
| CAS | CAS <key> <oldvalue> <newvalue> | 当oldvalue和key的value相等时，修改value的值为newvalue；不相等则不修改。<br>**说明** CAS仅适用于操作Redis String类型的数据，如需对TairString做相同的操作，请使用EXCAS。 |
| CAD | CAD <key> <value> | 当oldvalue和key的value相等时，删除该key；不相等则不删除。<br>**说明** CAD仅适用于操作Redis String类型的数据，如需对TairString做相同的操作，请使用EXCAD。 |

## CAS

- 语法

  CAS <key> <oldvalue> <newvalue>

- 时间复杂度

  O(1)

- 命令描述

  CAS（Compare And Set），查看某个key的value是否等于一个指定的值，如果相等，则将value修改为一个新的值；不相等则不修改。

- 参数及选项说明

| 参数/选项 | 说明 |
| --- | --- |
| key | String的key，用于指定作为命令调用对象的String。 |
| oldvalue | 用于跟key的现有value比较的值。 |
| newvalue | 当oldvalue和key的现有value相等时，将value修改为newvalue。 |

- 返回值
  - 成功：1。
  - key不存在：-1。
  - 失败：0。
  - 其它情况返回异常。
- 使用示例

```
127.0.0.1:6379> SET foo bar
OK
127.0.0.1:6379> CAS foo baa bzz
(integer) 0
127.0.0.1:6379> GET foo
"bar"
127.0.0.1:6379> CAS foo bar bzz
(integer) 1
127.0.0.1:6379> GET foo
"bzz"
```

## CAD

- 语法

  CAD <key> <value>

- 时间复杂度

  O(1)

- 命令描述

  CAD（Compare And Delete），查看某个key的value是否等于一个指定的值，如果相等，删除该key；不相等则不删除。

- 参数及选项说明

| 参数/选项 | 说明 |
| --- | --- |
| key | String的key，用于指定作为命令调用对象的String。 |
| value | 用于跟key的现有value比较的值。 |

- 返回值
  - 成功：1。
  - key不存在：-1。
  - 失败：0。
  - 其它情况返回异常。

- 使用示例

```
127.0.0.1:6379> SET foo bar
OK
127.0.0.1:6379> CAD foo bzz
(integer) 0
127.0.0.1:6379> CAD not-exists xxx
(integer) -1
127.0.0.1:6379> CAD foo bar
(integer) 1
127.0.0.1:6379> GET foo
(nil)
```

# 3.3. TairString命令

本文介绍TairString数据支持的命令。

## TairString简介

TairString是一种带版本号的string类型数据结构。Redis的String仅由key和value组成，而TairString不仅包含key和value，还携带了版本（version），可用于乐观锁等场景。除此之外，TairString在Redis String加减功能的基础上支持了边界设置，可以将**INCRBY**、**INCRBYFLOAT**的结果限制在一定的范围内，超出范围则提示错误。

主要特性：

- value携带版本号。
- 使用**INCRBY**、**INCRBYFLOAT**递增数据时可设置变更范围。

> **警告** TairString与Redis原生String是两种不同的数据结构，相关命令不可混用。

## 使用前提

请注意，本文介绍的命令只有在满足以下条件时才能生效。

- Redis实例为企业版性能增强型。
- 操作对象为性能增强型实例中的TairString数据。

> **说明** 性能增强型实例中可同时设置Redis String（即Redis原生String）和TairString，本文的命令无法对Redis String使用。

## 命令列表

## TairString命令

| 命令 | 语法 | 简介 |
| --- | --- | --- |
| EXSET | EXSET <key> <value> [EX time] [PX time] [EXAT time] [PXAT time] [NX \| XX] [VER version \| ABS version] | 将value保存到key中。 |
| EXGET | EXGET <key> | 返回TairString的value和version。 |
| EXSETVER | EXSETVER <key> <version> | 直接对一个key设置version。 |
| EXINCRBY | EXINCRBY <key> <num> [EX time] [PX time] [EXAT time] [EXAT time] [PXAT time] [NX \| XX] [VER version \| ABS version] [MIN minval] [MAX maxval] | 对Key做自增自减操作，num的范围为long。 |
| EXINCRBYFLOAT | EXINCRBYFLOAT <key> <num> [EX time] [PX time] [EXAT time] [EXAT time] [PXAT time] [NX \| XX] [VER version \| ABS version] [MIN minval] [MAX maxval] | 对Key做自增自减操作，num的范围为double。 |

| 命令 | 语法 | 简介 |
| --- | --- | --- |
| EXCAS | EXCAS <key> <newvalue> <version> | 指定version将value更新，当引擎中的version和指定的相同时才更新成功，不成功会返回旧的value和version。 |
| EXCAD | EXCAD <key> <version> | 当指定version和引擎中version相等时候删除Key，否则失败。 |
| DEL | DEL <key> [key ...] | 使用原生Redis的DEL命令可以删除一条或多条TairString数据。 |

## EXSET

- 语法

  EXSET <key> <value> [EX time] [PX time] [EXAT time] [EXAT time] [PXAT time] [NX | XX] [VER version | ABS version]

- 时间复杂度

  O(1)

- 命令描述

  将value保存到key中。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairString的key，用于指定作为命令调用对象的TairString。 |
| value | 需要为key设置的value。 |
| EX | 指定key的相对超时时间，单位为秒，为0表示立即过期。 |
| EXAT | 指定key的绝对超时时间，单位为秒，为0表示立即过期。 |
| PX | 指定key的相对超时时间，单位为毫秒，为0表示立即过期。 |
| PXAT | 指定key的绝对超时时间，单位为毫秒 ，为0表示立即过期。 |
| NX | 只在key不存在时写入。 |
| XX | 只在key存在时写入。 |
| VER | 版本号。<br>○ 如果key存在，和当前版本号做比较：<br>■ 如果相等，写入，且版本号加1。<br>■ 如果不相等，返回异常。<br>○ 如果key不存在或者key当前版本为0，忽略传入的版本号直接设置value，成功后版本号变为1。 |

| 参数或选项 | 说明 |
| --- | --- |
| ABS | 绝对版本号。设置后，无论key当前的版本号是多少，完成写入并将key的版本号覆盖为该选项中设置的值。 |

- 返回值
  - 成功：OK。
  - 其它情况返回异常。
- 使用示例

```
127.0.0.1:6379> EXSET foo bar XX
(nil)
127.0.0.1:6379> EXSET foo bar NX
OK
127.0.0.1:6379> EXSET foo bar NX
(nil)
127.0.0.1:6379> EXGET foo
1) "bar"
2) (integer) 1
127.0.0.1:6379> EXSET foo bar1 VER 10
(error) ERR update version is stale
127.0.0.1:6379> EXSET foo bar1 VER 1
OK
127.0.0.1:6379> EXGET foo
1) "bar1"
2) (integer) 2
127.0.0.1:6379> EXSET foo bar2 ABS 100
OK
127.0.0.1:6379> EXGET foo
1) "bar2"
2) (integer) 100
```

## EXGET

- 语法

  EXGET <key>
- 时间复杂度

  O(1)
- 命令描述

  获取TairString的value和version。
- 参数及选项说明

  key：TairString的key，用于指定作为命令调用对象的TairString。
- 返回值
  - 成功：value+version。
  - 其它情况返回异常。
- 使用示例

```
127.0.0.1:6379> EXSET foo bar ABS 100
OK
127.0.0.1:6379> EXGET foo
1) "bar"
2) (integer) 100
127.0.0.1:6379> DEL foo
(integer) 1
127.0.0.1:6379> EXGET foo
(nil)
```

## EXSETVER

- 语法

  EXSETVER <key> <version>

- 时间复杂度

  O(1)

- 命令描述

  直接对一个key设置version。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairString的key，用于指定作为命令调用对象的TairString。 |
| version | 需要设置的版本号。 |

- 返回值
  - 成功：1。
  - key不存在：0。
  - 其它情况返回异常。
- 使用示例

```
127.0.0.1:6379> EXSET foo bar
OK
127.0.0.1:6379> EXGET foo
1) "bar"
2) (integer) 1
127.0.0.1:6379> EXSETVER foo 2
(integer) 1
127.0.0.1:6379> EXGET foo
1) "bar"
2) (integer) 2
127.0.0.1:6379> EXSETVER not-exists 0
(integer) 0
```

## EXINCRBY

- 语法

EXINCRBY |EXINCRBY <key> <num> [EX time] [PX time] [EXAT time] [EXAT time] [PXAT time] [NX | XX] [VER version | ABS version] [MIN minval] [MAX maxval]

- 时间复杂度

  O(1)

- 命令描述

  对TairString的value做自增自减操作，num的范围为long。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairString的key，用于指定作为命令调用对象的TairString。 |
| num | TairString自增的数值，必须为整数。 |
| EX | 指定key的相对超时时间，单位为秒，为0表示立即过期。 |
| EXAT | 指定key的绝对超时时间，单位为秒，为0表示立即过期。 |
| PX | 指定key的相对超时时间，单位为毫秒，为0表示立即过期。 |
| PXAT | 指定key的绝对超时时间，单位为毫秒 ，为0表示立即过期。 |
| NX | 只在key不存在时写入。 |
| XX | 只在key存在时写入。 |
| VER | 版本号。<br>○ 如果key存在，和当前版本号做比较：<br>■ 如果相等，进行自增，且版本号加1。<br>■ 如果不相等，返回异常。<br>○ 如果key不存在或者key当前版本为0，忽略传入的版本号并进行自增操作，成功后版本号变为1。 |
| ABS | 绝对版本号。设置后，无论key当前的版本号是多少，完成自增并将key的版本号覆盖为该选项中设置的值。 |
| MIN | TairString value的最小值。 |
| MAX | TairString value的最大值。 |

- 返回值
  - 成功：value的当前值。
  - 其它情况返回异常。
- 使用示例

```
127.0.0.1:6379> EXINCRBY foo 100
(integer) 100
127.0.0.1:6379> EXINCRBY foo 100 MAX 150
(error) ERR increment or decrement would overflow
127.0.0.1:6379> FLUSHALL
OK
127.0.0.1:6379> EXINCRBY foo 100
(integer) 100
127.0.0.1:6379> EXINCRBY foo 100 MAX 150
(error) ERR increment or decrement would overflow
127.0.0.1:6379> EXINCRBY foo 100 MAX 300
(integer) 200
127.0.0.1:6379> EXINCRBY foo 100 MIN 500
(error) ERR increment or decrement would overflow
127.0.0.1:6379> EXINCRBY foo 100 MIN 500 MAX 100
(error) ERR min or max is specified, but not valid
127.0.0.1:6379> EXINCRBY foo 100 MIN 50
(integer) 300
```

## EXINCRBYFLOAT

- 语法

  EXINCRBYFLOAT |EXINCRBYFLOAT <key> <num> [EX time] [PX time] [EXAT time] [EXAT time] [PXAT time] [NX | XX] [VER version | ABS version] [MIN minval] [MAX maxval]

- 时间复杂度

  O(1)

- 命令描述

  对TairString的value做自增自减操作，num的范围为double。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairString的key，用于指定作为命令调用对象的TairString。 |
| num | TairString自增的数值，类型为浮点数。 |
| EX | 指定key的相对超时时间，单位为秒，为0表示立即过期。 |
| EXAT | 指定key的绝对超时时间，单位为秒，为0表示立即过期。 |
| PX | 指定key的相对超时时间，单位为毫秒，为0表示立即过期。 |
| PXAT | 指定key的绝对超时时间，单位为毫秒 ，为0表示立即过期。 |
| NX | 当数据不存在时写入。 |
| XX | 当数据存在时写入。 |

| 参数或选项 | 说明 |
| --- | --- |
| VER | 版本号。<br>○ 如果key存在，和当前版本号做比较：<br>■ 如果相等，进行自增，且版本号加1。<br>■ 如果不相等，返回异常。<br>○ 如果key不存在或者key当前版本为0，忽略传入的版本号并进行自增操作，成功后版本号变为1。 |
| ABS | 绝对版本号。设置后，无论key当前的版本号是多少，完成自增并将key的版本号覆盖为该选项中设置的值。 |
| MIN | TairString value的最小值。 |
| MAX | TairString value的最大值。 |

- 返回值
  - 成功：value的当前值。
  - 其它情况返回异常。
- 使用示例

```
127.0.0.1:6379> EXSET foo 100
OK
127.0.0.1:6379> EXINCRBYFLOAT foo 10.123
"110.123"
127.0.0.1:6379> EXINCRBYFLOAT foo 20 MAX 100
(error) ERR increment or decrement would overflow
127.0.0.1:6379> EXINCRBYFLOAT foo 20 MIN 100
"130.123"
127.0.0.1:6379> EXGET foo
1) "130.123"
2) (integer) 3
```

## EXCAS

- 语法

  EXCAS <key> <newvalue> <version>
- 时间复杂度

  O(1)
- 命令描述

  当指定key的版本值和命令中设置的version相等时，将key的value修改为一个新的值；不相等则不修改。
- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairString的key，用于指定作为命令调用对象的TairString。 |

| 参数或选项 | 说明 |
| --- | --- |
| newvalue | 当version和key的现有版本值相等时，将value修改为newvalue。 |
| version | 用于跟key的现有版本值比较的值。 |

- 返回值
  - 成功：["OK", "", version]。返回值中间的""为无意义的空字符串，version为key当前的版本。
  - 失败：["ERR update version is stale", value, version]。value和version为key当前的value和版本。
  - 其它情况返回异常。
- 使用示例

```
127.0.0.1:6379> EXSET foo bar
OK
127.0.0.1:6379> EXCAS foo bzz 1
1) OK
2)
3) (integer) 2
127.0.0.1:6379> EXGET foo
1) "bzz"
2) (integer) 2
127.0.0.1:6379> EXCAS foo bee 1
1) ERR update version is stale
2) "bzz"
3) (integer) 2
```

## EXCAD

- 语法

  EXCAD <key> <version>
- 时间复杂度

  O(1)
- 命令描述

  当指定key的版本值和命令中设置的version相等时删除Key。
- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairString的key，用于指定作为命令调用对象的TairString。 |
| newvalue | 当version和key的现有版本值相等时，将value修改为newvalue。 |
| version | 用于跟key的现有版本值比较的值。 |

- 返回值
  - 成功：1。
  - key不存在：-1。

- 失败：0。
- 其它情况返回异常。

- 使用示例

```
127.0.0.1:6379> EXSET foo bar
OK
127.0.0.1:6379> EXGET foo
1) "bar"
2) (integer) 1
127.0.0.1:6379> EXCAD not-exists 1
(integer) -1
127.0.0.1:6379> EXCAD foo 0
(integer) 0
127.0.0.1:6379> EXCAD foo 1
(integer) 1
127.0.0.1:6379> EXGET foo
(nil)
```

# 3.4. TairHash命令

本文介绍TairHash数据支持的命令。

## TairHash简介

TairHash是一种可为field设置过期时间和版本的hash类型数据结构。TairHash不但和Redis Hash一样支持丰富的数据接口和高处理性能，还改变了hash只能为key设置过期时间的限制，可以为field设置过期时间和版本，极大地提高了hash数据结构的灵活性，简化了很多场景下的业务开发工作。TairHash使用高效的Active Expire算法，可以在不对响应时间造成明显影响的前提下，更高效的完成对field的过期判断和删除。

主要特性：

- field支持单独设置expire和version。
- field支持高效active expire和passivity expire。
- 语法和原生Redis Hash数据类型类似。
- 支持高效的主动淘汰策略，会在一定程度上增加内存消耗。

> **警告** TairHash与Redis中的原生Hash是两种不同的数据，相关命令不可混用。

## 使用前提

请注意，本文介绍的命令只有在满足以下条件时才能生效。

- Redis实例为企业版性能增强型。
- 操作对象为性能增强型实例中的TairHash数据。

> **说明** 性能增强型实例中可同时设置Redis Hash（即Redis原生Hash）和TairHash，本文的命令无法对Redis Hash使用。

## 命令列表

### TairHash命令

| 命令 | 语法 | 说明 |
| --- | --- | --- |
| EXHSET | EXHSET <key> <field> <value> [EX time] [EXAT time] [PX time] [PXAT time] [NX/XX] [VER/ABS version] [NOACTIVE] | 向key指定的TairHash中插入一个field，如果TairHash不存在则自动创建一个，如果field已经存在则覆盖其值。该命令会触发对field的被动淘汰检查。 |
| EXHMSET | EXHMSET <key> <field> <value> [field value...] | 同时向key指定的TairHash中插入多个field，如果TairHash不存在则自动创建一个，如果field已经存在则覆盖其值。该命令会触发对field的被动淘汰。 |
| EXHPEXPIREAT | EXHPEXPIREAT <key> <field> <milliseconds-timestamp> [VER/ABS version] [NOACTIVE] | 在key指定的TairHash中为一个field设置绝对过期时间，单位为毫秒。该命令会触发对field的被动淘汰。 |
| EXHPEXPIRE | EXHPEXPIRE <key> <field> <milliseconds> [NOACTIVE] | 在key指定的TairHash中为一个field设置相对过期时间，单位为毫秒。该命令会触发对field的被动淘汰。 |
| EXHEXPIREAT | EXHEXPIREAT <key> <field> <timestamp> [NOACTIVE] | 在key指定的TairHash中为一个field设置绝对过期时间，单位为秒。该命令会触发对field的被动淘汰。 |
| EXHEXPIRE | EXHEXPIRE <key> <field> <seconds> [NOACTIVE] | 在key指定的TairHash中为一个field设置相对过期时间，单位为秒。该命令会触发对field的被动淘汰。 |
| EXHPTTL | EXHPTTL <key> <field> | 查看key指定的TairHash中一个field的剩余超时时间，单位为毫秒。该命令会触发对field的被动淘汰。 |
| EXHTTL | EXHTTL <key> <field> | 查看key指定的TairHash中一个field的剩余超时时间，单位为秒。该命令会触发对field的被动淘汰。 |
| EXHVER | EXHVER <key> <field> | 查看key指定的TairHash中一个field的当前版本号。该命令会触发对field的被动淘汰。 |
| EXHSETVER | EXHSETVER <key> <field> <version> | 为key指定的TairHash中的一个field设置版本号。该命令会触发对field的被动淘汰。 |
| EXHINCRBY | EXHINCRBY <key> <field> <num> [EX time] [EXAT time] [PX time] [PXAT time] [VER/ABS version] [MIN minval] [MAX maxval] | 将key指定的TairHash中一个field的值加上整型value。如果TairHash不存在则自动新创建一个，如果指定的field不存在，则在加之前先将field的值设置为0。同时还可以使用EX、EXAT、PX或PXAT为field设置超时时间。该命令会触发对field的被动淘汰。<br>**说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。 |

| 命令 | 语法 | 说明 |
| --- | --- | --- |
| EXHINCRBYFLOAT | EXHINCRBYFLOAT <key> <field> <value> [EX time] [EXAT time] [PX time] [PXAT time] [VER/ABS version] [MIN minval] [MAX maxval] | 将key指定的TairHash中一个field的值加上浮点型value。如果TairHash不存在则自动新创建一个，如果指定的field不存在，则在加之前先将field的值设置为0。同时还可以使用EX、EXAT、PX或PXAT为field设置超时时间。该命令会触发对field的被动淘汰。<br>**说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。 |
| EXHGET | EXHGET <key> <field> | 获取key指定的TairHash中一个field的值，如果TairHash不存在或者field不存在，则返回nil。该命令会触发对field的被动淘汰。 |
| EXHGETWITHVER | EXHGETWITHVER <key> <field> | 同时获取key指定的TairHash中一个field的值和版本，如果TairHash不存在或者field不存在，则返回nil。该命令会触发对field的被动淘汰。 |
| EXHMGET | EXHMGET <key> <field> [field ...] | 同时获取key指定的TairHash中多个field的值，如果TairHash不存在或者field不存在，则返回nil。该命令会触发对field的被动淘汰。 |
| EXHMGETWITHVER | EXHMGETWITHVER <key> <field> [field ...] | 同时获取key指定的TairHash中多个field的值和版本，如果TairHash不存在或者field不存在，则返回nil。该命令会触发对field的被动淘汰。 |
| EXHDEL | EXHDEL <key> <field> <field> <field> ... | 删除key指定的TairHash中的一个field，如果TairHash不存在或者field不存在则返回0，成功删除返回1。该命令会触发对field的被动淘汰。 |
| EXHLEN | EXHLEN <key> [noexp] | 获取key指定的TairHash中field个数，该命令不会触发对过期field的被动淘汰，也不会将其过滤掉，所以结果中可能包含已经过期但还未被删除的field。如果只想返回当前没有过期的field个数，可以在命令中设置*noexp*选项。 |
| EXHEXISTS | EXHEXISTS <key> <field> | 查询key指定的TairHash中是否存在对应的field。该命令会触发对field的被动淘汰。 |
| EXHSTRLEN | EXHSTRLEN <key> <field> | 获取key指定的TairHash中一个field对应的value的长度。该命令会触发对field的被动淘汰。 |
| EXHKEYS | EXHKEYS <key> | 获取key指定的TairHash中所有的field。该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。 |
| EXHVALS | EXHVALS <key> | 获取key指定的TairHash中所有field的值 。该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。 |

| 命令 | 语法 | 说明 |
| --- | --- | --- |
| EXHGETALL | EXHGETALL <key> | 获取key指定的TairHash中所有的field及其value。该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。 |
| EXHSCAN | EXHSCAN <key> <op> <subkey> [MATCH pattern] [COUNT count] | 扫描key指定的TairHash。扫描的方式（使用op选项设置）可以为>、>=、<、<=、==、^、$。扫描的同时可以根据MATCH指定的pattern对subkey进行正则过滤，还可以使用COUNT对结果数目进行限制，如果不指定则默认值为10。该命令会过滤掉已经过期的field，但是为了尽快响应客户端，不会执行真正的删除操作。 |
| DEL | DEL <key> [key ...] | 使用原生Redis的DEL命令可以删除一条或多条TairHash数据。 |

## EXHSET

- 语法

  EXHSET <key> <field> <value> [EX time] [EXAT time] [PX time] [PXAT time] [NX | XX] [VER/ABS version] [NOACTIVE]

- 时间复杂度

  O(1)

- 命令描述

  向key指定的TairHash中插入一个field。如果TairHash不存在则自动创建一个，如果field已经存在则覆盖其值。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |
| value | field对应的值，一个field只能有一个value。 |
| EX | 指定field的相对过期时间，单位为秒，为0表示不过期。 |
| EXAT | 指定field的绝对过期时间，单位为秒，为0表示不过期。 |
| PX | 指定field的相对过期时间，单位为毫秒，为0表示不过期。 |
| PXAT | 指定field的绝对过期时间，单位为毫秒 ，为0表示不过期。 |
| NX | 只在field不存在时插入。 |
| XX | 只在field存在时插入。 |

| 参数或选项 | 说明 |
| --- | --- |
| VER | 版本号。<br>○ 如果field存在，和当前版本号做比较：<br>■ 如果相等，继续操作，且版本号加1。<br>■ 如果不相等，返回异常。<br>○ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。 |
| ABS | 绝对版本号，不论field是否存在，可以在插入field时设置为本参数所指定的版本号。 |
| NOACTIVE | 在指定*EX*、*EXAT*、*PX*或*PXAT*选项时，设置*NOACTIVE*表示该field不使用active expiration策略，这样可以降低内存占用。 |

- 返回值
  - 新建field并成功为它设置值：1。
  - field已经存在，成功覆盖旧值：0。
  - 指定了XX且field不存在：-1。
  - 指定了NX且field已经存在：-1。
  - 指定了VER且版本和当前版本不匹配："ERR update version is stale"。
  - 其它情况返回相应的异常信息。

## EXHGET

- 语法

  EXHGET <key> <field>

- 时间复杂度

  O(1)

- 命令描述

  获取指定TairHash一个field的值。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - field存在且操作成功：field对应的值。
  - key不存在或者field不存在：nil。
  - 其它情况返回相应的异常信息。

## EXHMSET

- 语法

EXHMSET <key> <field> <value> [field value...]

- 时间复杂度

  O(1)

- 命令描述

  同时向key指定的TairHash中插入多个field，如果TairHash不存在则自动创建一个，如果field已经存在则覆盖其value。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |
| value | field对应的值，一个field只能有一个value。 |

- 返回值
  - 成功：OK。
  - 其它情况返回相应的异常信息。

## EXHPEXPIREAT

- 语法

  EXHPEXPIREAT <key> <field> <milliseconds-timestamp> [VER/ABS version] [NOACTIVE]

- 时间复杂度

  O(1)

- 命令描述

  在key指定的TairHash中为一个field设置绝对过期时间，精确到毫秒。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |
| milliseconds-timestamp | 精确到毫秒的时间戳。 |
| VER | 版本号。<br>◦ 如果field存在，和当前版本号做比较：<br>▪ 如果相等，继续操作，且版本号加1。<br>▪ 如果不相等，返回异常。<br>◦ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。 |

| 参数或选项 | 说明 |
| --- | --- |
| ABS | 绝对版本号，不论field是否存在，继续操作并将field的版本号设置为本参数所指定的值。 |
| NOACTIVE | 在指定*EX*、*EXAT*、*PX*或*PXAT*选项时，设置*NOACTIVE*表示该field不使用active expiration策略，这样可以降低内存占用。 |

- 返回值
  - field存在且设置成功：1。
  - field不存在：0。
  - 其它情况返回相应的异常信息。

## EXHPEXPIRE

- 语法

  EXHPEXPIRE <key> <field> <milliseconds> [VER/ABS version] [NOACTIVE]

- 时间复杂度

  O(1)

- 命令描述

  在key指定的TairHash中为一个field设置相对过期时间，单位为毫秒。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |
| milliseconds | 相对过期时间，单位为毫秒。 |
| VER | 版本号。<br>◦ 如果field存在，和当前版本号做比较：<br>▪ 如果相等，继续操作，且版本号加1。<br>▪ 如果不相等，返回异常。<br>◦ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。 |
| ABS | 绝对版本号，不论field是否存在，继续操作并将field的版本号设置为本参数所指定的值。 |
| NOACTIVE | 在指定*EX*、*EXAT*、*PX*或*PXAT*选项时，设置*NOACTIVE*表示该field不使用active expiration策略，这样可以降低内存占用。 |

- 返回值
  - field存在且设置成功：1。
  - field不存在：0。
  - 其它情况返回相应的异常信息。

## EXHEXPIREAT

- 语法

  EXHEXPIREAT <key> <field> <timestamp> [VER/ABS version] [NOACTIVE]

- 时间复杂度

  O(1)

- 命令描述

  在key指定的TairHash中为一个field设置绝对过期时间，精确到秒。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |
| timestamp | 精确到秒的时间戳。 |
| VER | 版本号。<br>○ 如果field存在，和当前版本号做比较：<br>■ 如果相等，继续操作，且版本号加1。<br>■ 如果不相等，返回异常。<br>○ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。 |
| ABS | 绝对版本号，不论field是否存在，继续操作并将field的版本号设置为本参数所指定的值。 |
| NOACTIVE | 在指定*EX*、*EXAT*、*PX*或*PXAT*选项时，设置*NOACTIVE*表示该field不使用active expiration策略，这样可以降低内存占用。 |

- 返回值
  - field存在且设置成功：1。
  - field不存在：0。
  - 其它情况返回相应的异常信息。

## EXHEXPIRE

- 语法

  EXHEXPIRE <key> <field> <seconds>

- 时间复杂度

  O(1)

- 命令描述

  在key指定的TairHash中为一个field设置相对过期时间，单位为秒。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |
| seconds | 相对过期时间，单位为秒。 |
| VER | 版本号。<br>○ 如果field存在，和当前版本号做比较：<br>■ 如果相等，继续操作，且版本号加1。<br>■ 如果不相等，返回异常。<br>○ 如果field不存在或者field当前版本为0，忽略传入的版本号并继续操作，成功后版本号变为1。 |
| ABS | 绝对版本号，不论field是否存在，继续操作并将field的版本号设置为本参数所指定的值。 |
| NOACTIVE | 在指定*EX*、*EXAT*、*PX*或*PXAT*选项时，设置*NOACTIVE*表示该field不使用active expiration策略，这样可以降低内存占用。 |

- 返回值
  - field存在且设置成功：1。
  - field不存在：0。
  - 其它情况返回相应的异常信息。

## EXHPTTL

- 语法

  EXHPTTL <key> <field>

- 时间复杂度

  O(1)

- 命令描述

  查看key指定的TairHash中一个field的过期时间，结果精确到毫秒。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - key或者field不存在：-2。
  - field存在但是没有设置过期时间：-1。
  - field存在且设置了过期时间：过期时间，单位为毫秒。
  - 其它情况返回相应的异常信息。

## EXHTTL

- 语法

  EXHTTL <key> <field>

- 时间复杂度

  O(1)

- 命令描述

  查看key指定的TairHash中一个field的过期时间，结果精确到秒。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - key或者field不存在：-2。
  - field存在但是没有设置过期时间：-1。
  - field存在且设置了过期时间：过期时间，单位为秒。
  - 其它情况返回相应的异常信息。

## EXHVER

- 语法

  EXHVER <key> <field>

- 时间复杂度

  O(1)

- 命令描述

  查看key指定的TairHash中一个field的版本号。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - key不存在：-1。
  - field不存：-2。
  - 查询成功：field的版本号。
  - 其它情况返回相应的异常信息。

## EXHSETVER

- 语法

  EXHSETVER <key> <field> <version>

- 时间复杂度

  O(1)

- 命令描述

  设置key指定的TairHash中一个field的版本号。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - TairHash或者field不存在：0。
  - 设置成功：1。
  - 其它情况返回相应的异常信息。

## EXHINCRBY

- 语法

  EXHINCRBY <key> <field> <num> [EX time] [EXAT time] [PX time] [PXAT time] [VER/ABS version] [MIN minval] [MAX maxval]

- 时间复杂度

  O(1)

- 命令描述

  将key指定的TairHash中一个field的value增加num，num为一个整数。如果TairHash不存在则自动新创建一个，如果指定的field不存在，则在加之前插入该field并将其值设置为0。该命令会触发对field的被动淘汰检查。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |
| num | 需要为field的value增加的整数值。 |
| EX | 指定field的相对过期时间，单位为秒，为0表示不过期。 |
| EXAT | 指定field的绝对过期时间，单位为秒，为0表示不过期。 |
| PX | 指定field的相对过期时间，单位为毫秒，为0表示不过期。 |

| 参数或选项 | 说明 |
| --- | --- |
| PXAT | 指定field的绝对过期时间，单位为毫秒 ，为0表示不过期。 |
| VER | 版本号。<br>○ 如果field存在，和当前版本号做比较：<br>■ 如果相等，完成递增，且版本号加1。<br>■ 如果不相等，返回异常。<br>○ 如果*VER*选项所携带的版本号为0，则不进行版本校验。 |
| ABS | 绝对版本号，不论field是否存在，完成递增并将field的版本号设置为本参数所指定的值，该值不能为0。 |
| MIN | value的最小值，小于该值则提示异常。 |
| MAX | value的最大值，大于该值则提示异常。 |
| NOACTIVE | 在设置了*EX*、*EXAT*、*PX*或*PXAT*选项的同时，如果设置了*NOACTIVE*，则该field不使用active expiration策略，内存占用较小。 |

**说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。

- 返回值
  - 成功：与num相加后value的值。
  - 其它情况返回异常。

## EXHINCRBYFLOAT

- 语法

  EXHINCRBYFLOAT <key> <field> <num> [EX time] [EXAT time] [PX time] [PXAT time] [VER/ABS version] [MIN minval] [MAX maxval]

- 时间复杂度

  O(1)

- 命令描述

  将key指定的TairHash中一个field的value增加num，num为一个浮点数。如果TairHash不存在则自动新创建一个，如果指定的field不存在，则在加之前插入该field并将其值设置为0。该命令会触发对field的被动淘汰检查。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |
| num | 需要为field的value增加的值，类型为浮点。 |

| 参数或选项 | 说明 |
| --- | --- |
| EX | 指定field的相对过期时间，单位为秒，为0表示不过期。 |
| EXAT | 指定field的绝对过期时间，单位为秒，为0表示不过期。 |
| PX | 指定field的相对过期时间，单位为毫秒，为0表示不过期。 |
| PXAT | 指定field的绝对过期时间，单位为毫秒 ，为0表示不过期。 |
| VER | 版本号。<br>○ 如果field存在，和当前版本号做比较：<br>■ 如果相等，完成递增，且版本号加1。<br>■ 如果不相等，返回异常。<br>○ 如果*VER*选项所携带的版本号为0，则不进行版本校验。 |
| ABS | 绝对版本号，不论field是否存在，完成递增并将field的版本号设置为本参数所指定的值，该值不能为0。 |
| MIN | value的最小值，小于该值则提示异常。 |
| MAX | value的最大值，大于该值则提示异常。 |
| NOACTIVE | 在设置了*EX*、*EXAT*、*PX*或*PXAT*选项的同时，如果设置了*NOACTIVE*，则该field不使用active expiration策略，内存占用较小。 |

**说明** 为Key的field设置了超时时间后，再次执行该命令时如果没有设置超时时间，该field将被设置为永不过期。

- 返回值
  - 成功：与num相加后value的值。
  - 其它情况返回异常。

## EXHGETWITHVER

- 语法

  EXHGETWITHVER <key> <field>

- 时间复杂度

  O(1)

- 命令描述

  同时获取key指定的TairHash一个field的值和版本。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |

| 参数或选项 | 说明 |
| --- | --- |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - field存在且操作成功：field对应的值和版本。
  - key不存在或者field不存在：nil。
  - 其它情况返回相应的异常信息。

## EXHMGET

- 语法

  EXHMGET <key> <field> [field ...]

- 时间复杂度

  O(1)

- 命令描述

  同时获取key指定的TairHash多个field的值。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - key不存在：nil。
  - key存在且查询的所有field都存在：返回一个数组，数组的每一个元素对应一个field的value。
  - key存在且查询的field中有不存在的：返回一个数组，数组的每一个元素对应一个field的value，不存在的field对应的元素显示为nil。
  - 其它情况返回相应的异常信息。

## EXHMGETWITHVER

- 语法

  EXHMGETWITHVER <key> <field> [field ...]

- 时间复杂度

  O(1)

- 命令描述

  同时获取key指定的TairHash多个field的值和版本。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - key不存在：nil。
  - key存在且查询的所有field都存在：返回一个数组，数组的每一个元素对应一个field的value和version。
  - key存在且查询的field中有不存在的：返回一个数组，数组的每一个元素对应一个field的value和version，不存在的field对应的元素显示为nil。
  - 其它情况返回相应的异常信息。

## EXHDEL

- 语法

  EXHDEL <key> <field> <field> <field> ...

- 时间复杂度

  O(1)

- 命令描述

  删除key指定的TairHash一个field。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - key不存在或者field不存在：0。
  - 删除成功：1。
  - 其它情况返回相应的异常信息。

## EXHLEN

- 语法

  EXHLEN <key> [noexp]

- 时间复杂度

  O(1)

- 命令描述

  获取key指定的TairHash中field的个数，结果中可能存在已经过期但还未被删除的field。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| noexp | EXHLEN默认不会触发对过期field的被动淘汰，也不会将其过滤掉，所以结果中可能包含已经过期但还未被删除的field。如果只想返回当前没有过期的field个数，可以在命令中设置*noexp*选项。在设置*noexp*时：<br>○ 因为要遍历整条TairHash数据，EXHLEN命令的响应时间将受到Tairhash大小的影响。<br>○ EXHLEN命令的返回结果中会过滤掉过期的field，但过期field不会被淘汰。 |

- 返回值
  - key不存在或者field不存在：0。
  - 成功：field个数。
  - 其它情况返回相应的异常信息。

## EXHEXISTS

- 语法

  EXHEXISTS <key> <field>

- 时间复杂度

  O(1)

- 命令描述

  查询key指定的TairHash中是否存在对应的field。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - key不存在或者field不存在：0。
  - field存在：1。
  - 其它情况返回相应的异常信息。

## EXHSTRLEN

- 语法

  EXHSTRLEN <key> <field>

- 时间复杂度

  O(1)

- 命令描述

  获取key指定的TairHash中一个field的value的长度。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |
| field | TairHash中的一个元素，一个TairHash key可以有多个field。 |

- 返回值
  - key不存在或者field不存在：0。
  - 查询成功：value的长度。
  - 其它情况返回相应的异常信息。

## EXHKEYS

- 语法

  EXHKEYS <key>

- 时间复杂度

  O(1)

- 命令描述

  获取key指定的TairHash中所有field。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |

- 返回值
  - key不存在：返回一个空数组。
  - key存在：返回一个数组，数组的每一位对应TairHash中的每一个field。
  - 其它情况返回相应的异常信息。

## EXHVALS

- 语法

  EXHVALS <key>

- 时间复杂度

  O(1)

- 命令描述

  获取key指定的TairHash中所有field的value。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |

- 返回值

- key不存在：返回一个空数组。
- key存在：返回一个数组，数组的每个元素对应TairHash中的一个field的value。
- 其它情况返回相应的异常信息。

## EXHGETALL

- 语法

  EXHGETALL <key>

- 时间复杂度

  O(1)

- 命令描述

  获取key指定的TairHash中所有field和value。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |

- 返回值
  - key不存在：返回一个空数组。
  - key存在：返回一个数组，数组的每个元素对应TairHash中的一对field和value。
  - 其它情况返回相应的异常信息。

## EXHSCAN

- 语法

  EXHSCAN <key> <op> <subkey> [MATCH pattern] [COUNT count]

- 时间复杂度

  O(1)、O(N)

- 命令描述

  扫描key指定的TairHash。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairHash的key，用于指定作为命令调用对象的TairHash。 |

| 参数或选项 | 说明 |
| --- | --- |
| op | 用于定位扫描的起点，可选值：<br>○ >，表示从第一个大于subkey的field开始。<br>○ >=，表示从第一个大于等于subkey的field开始。<br>○ <，表示从第一个小于subkey的field开始。<br>○ <=，表示从第一个小于等于subkey的field开始。<br>○ ==，表示从第一个等于subkey的field开始。<br>○ ^，表示从第一个field开始。<br>○ $，表示从最后一个field开始。 |
| subkey | 用于与op选项搭配，设置扫描起始位置，当op为^或$时该值将被忽略。 |
| MATCH | 用于过滤扫描结果。 |

- 返回值
  - key不存在：返回一个空数组。
  - key存在：返回一个数组，数组的每个元素对应TairHash中的一对field和value。
  - 其它情况返回相应的异常信息。

# 3.5. TairGIS命令

TairGis是一种使用R-Tree做索引，支持地理信息系统GIS（Geographic Information System）相关接口的数据结构。Redis的原生GEO命令是使用GeoHash和Redis Sorted Set结构完成的，主要用于点的查询，TairGIS在此基础上还支持线、面的查询，功能更加强大。

## 前提条件

- Redis实例为企业版性能增强型。
- Redis实例的小版本为最新。关于如何升级，请参见云数据库Redis*用户指南*中的*升级小版本*章节。

## 主要特性

- 使用R-Tree作为索引存储。
- 支持点、线、面的相关查询（含相交查询）。
- 兼容Redis的原生GEO命令。

## 命令列表

### TairGIS命令

| 命令 | 语法 | 说明 |
| --- | --- | --- |

| 命令 | 语法 | 说明 |
| --- | --- | --- |
| GIS.ADD | `GIS.ADD <area> <polygonName> <polygonWkt> [<polygonName> <polygonWkt> ...]` | 在area中添加指定名称的多边形（可添加多个），多边形使用WKT（Well-known text）格式描述。<br>**说明** WKT是一种文本标记语言，用于描述矢量几何对象、空间参照系统及空间参照系统之间的转换。 |
| GIS.GET | `GIS.GET <area> <polygonName>` | 在area中查找指定名称的多边形的WKT信息。 |
| GIS.GETALL | `GIS.GETALL <area> [WITHOUTWKT]` | 返回一个area中所有多边形的名称和WKT信息。 |
| GIS.DEL | `GIS.DEL <area> <polygonName>` | 删除area中指定名称的多边形。 |
| DEL | `DEL <key> [key ...]` | 原生Redis命令，可以删除一条或多条TairGIS数据。 |
| GIS.CONTAINS | `GIS.CONTAINS <area> <polygonWkt> [WITHOUTWKT]` | 判断指定的点、线或面是否包含在area中的某个多边形内。 |
| GIS.INTERSECTS | `GIS.INTERSECTS <area> <polygonWkt>` | 判断给定点、线或面，和area中包含的多边形的相交情况。 |
| GIS.SEARCH | `GIS.SEARCH [RADIUS longitude latitude distance m\|km\|ft\|mi] [MEMBER field distance m\|km\|ft\|mi] [GEOM geom] [COUNT count] [ASC\|DESC] [WITHDIST] [WITHOUTWKT]` | 查找给定经纬度及距离为半径范围内的点。 |
| GIS.WITHIN | `GIS.WITHIN <area> <polygonWkt> [WITHOUTWKT]` | 在area中搜索指定多边形内部的点、线或面。 |

## 通用参数说明

| 参数 | 说明 |
| --- | --- |
| area | 一个几何区域。 |
| PolygonName | 一个多边形的名称。 |

| 参数 | 说明 |
| --- | --- |
| polygonWkt | 一个多边形的描述信息，使用WKT（Well-known text）描述，支持如下类型：<br>• POINT：描述一个点的WKT信息，例如 `'POINT (30 11)'` 。<br>• LINESTRING：描述一条线的WKT信息，例如 `'LINESTRING (30 10, 40 40)'` 。<br>• POLYGON：描述一个多边形的WKT信息，例如 `'POLYGON ((31 20, 29 20, 29 21, 31 31))'` 。<br>**说明** 不支持MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRYCOLLECTION。 |
| WITHOUTWKT | 用于控制是否返回多边形的WKT信息。在使用GIS.GETALL、GIS.CONTAINS、GIS.SEARCH和GIS.WITHIN命令时，如果加上该参数，则不返回多边形的WKT信息。 |

## GIS.ADD

- 语法

  GIS.ADD <area> <polygonName> <polygonWkt> [<polygonName> <polygonWkt>...]

- 时间复杂度（用于表示命令执行时间随数据规模增长的变化趋势）

  O(log n)

- 命令描述

  在area中添加指定名称的多边形（可添加多个），使用WKT（Well-known text）描述。

- 返回值
  - 成功：返回插入和更新成功的个数。
  - 其它情况：返回相应的异常信息。

- 使用示例

```
127.0.0.1:6379> GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'
(integer) 1
```

## GIS.GET

- 语法

```
GIS.GET <area> <polygonName>
```

- 时间复杂度

  O(1)

- 命令描述

  在area中查找指定名称的多边形的WKT信息。

- 返回值
  - 成功：返回查询到的WKT。
  - area或polygonName不存在：返回nil。

- 其它情况：返回相应的异常信息。

- 使用示例

```
127.0.0.1:6379> GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'
(integer) 1
127.0.0.1:6379> GIS.GET hangzhou campus
"POLYGON((30 10,40 40,20 40,10 20,30 10))"
127.0.0.1:6379> GIS.GET hangzhou not-exists
(nil)
127.0.0.1:6379> GIS.GET not-exists campus
(nil)
```

## GIS.GETALL

- 语法

```
GIS.GETALL <area> [WITHOUTWKT]
```

- 时间复杂度

  O(n)

- 命令描述

  返回一个area中所有多边形的名称和WKT信息。如果设置了*WITHOUTWKT*选项，仅返回多边形的名称。

- 返回值
  - 成功：返回多边形的名称和WKT信息。如果设置了*WITHOUTWKT*选项，仅返回多边形的名称。
  - 未查询到：返回nil。
  - 其它情况：返回相应的异常信息。

- 使用示例

```
127.0.0.1:6379> GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'
(integer) 1
127.0.0.1:6379> GIS.GETALL hangzhou
1) "campus"
2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"
127.0.0.1:6379> GIS.GETALL hangzhou WITHOUTWKT
1) "campus"
```

## GIS.DEL

- 语法

  GIS.DEL <area> <polygonName>

- 时间复杂度

  O(log n)

- 命令描述

  删除area中指定名称的多边形。

- 返回值
  - 成功：OK。

- area或polygonName不存在：返回nil。
- 其它情况：返回相应的异常信息。

- 使用示例

```
127.0.0.1:6379> GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'
(integer) 1
127.0.0.1:6379> GIS.GET hangzhou campus
"POLYGON((30 10,40 40,20 40,10 20,30 10))"
127.0.0.1:6379> GIS.DEL hangzhou not-exists
(nil)
127.0.0.1:6379> GIS.DEL not-exists campus
(nil)
127.0.0.1:6379> GIS.DEL hangzhou campus
OK
127.0.0.1:6379> GIS.GET hangzhou campus
(nil)
```

## GIS.CONTAINS

- 语法

  GIS.CONTAINS <area> <polygonWkt>

- 时间复杂度
  - 最理想情况：$O(\log_M n)$
  - 最差情况：log(n)

- 命令描述

  判断指定的点、线或面是否包含在area中的某个多边形内。

- 返回值
  - 成功：返回包含指定点、线或面的多边形的名称和WKT信息。如果设置了*WITHOUTWKT*选项，仅返回多边形的名称。
  - 未查询到：返回nil。
  - 其它情况：返回相应的异常信息。

- 使用示例

```
127.0.0.1:6379> GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'
(integer) 1
127.0.0.1:6379> GIS.CONTAINS hangzhou 'POINT (30 11)'
1) "0"
2) 1) "campus"
   2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"
127.0.0.1:6379> GIS.CONTAINS hangzhou 'LINESTRING (30 10, 40 40)'
1) "0"
2) 1) "campus"
   2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"
127.0.0.1:6379> GIS.CONTAINS hangzhou 'POLYGON ((31 20, 29 20, 29 21, 31 31))'
1) "0"
2) 1) "campus"
   2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"
```

## GIS.INTERSECTS

- 语法

  GIS.INTERSECTS <area> <polygonWkt>

- 时间复杂度
  - 最理想情况：$O(\log_M n)$
  - 最差情况：log(n)
- 命令描述

  判断指定的点、线或面，与area中的某个多边形的相交情况。
- 返回值
  - 成功：返回与指定点、线或面相交的多边形的名称和WKT信息。
  - 未查询到：返回nil。
  - 其它情况：返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'
(integer) 1
127.0.0.1:6379> GIS.INTERSECTS hangzhou 'POINT (30 11)'
1) "0"
2) 1) "campus"
   2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"
127.0.0.1:6379> GIS.INTERSECTS hangzhou 'LINESTRING (30 10, 40 40)'
1) "0"
2) 1) "campus"
   2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"
127.0.0.1:6379> GIS.INTERSECTS hangzhou 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'
1) "0"
2) 1) "campus"
   2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"
127.0.0.1:6379>
```

## GIS.SEARCH

- 语法

```
GIS.SEARCH [RADIUS longitude latitude distance m|km|ft|mi]
[MEMBER field distance m|km|ft|mi]
[GEOM geom]
[COUNT count]
[ASC|DESC]
[WITHDIST]
[WITHOUTWKT]
```

- 时间复杂度
  - 最理想情况：$O(\log_M n)$
  - 最差情况：log(n)
- 命令描述

  查找给定经纬度及距离为半径范围内的点，支持下述可选参数：
  - *RADIUS*：按照参数传递的经度、纬度和半径搜索，取值顺序为经度、纬度、半径、半径单位，例如 `RADIUS 15 37 200 km` 。
  - *MEMBER*：从已经存在的多边形中获取经纬度并根据半径搜索，取值顺序为多边形名称、半径、半径单位，例如 `MEMBER Agrigento 100 km` 。
  - *GEOM*：按照WKT的格式设置搜索范围，可以是任意多边形，例如 `GIS.SEARCH Sicily "POINT (13.361389 38.115556)"` 。
  - *COUNT*：用于限定返回的个数，例如 `COUNT 3` 。
  - *ASC|DESC*：用于控制返回信息按照距离排序，例如ASC表示根据中心位置，由近到远排序。
  - *WITHDIST*：用于控制是否返回距离。
  - *WITHOUTWKT*：用于控制是否返回多边形的WKT信息。
- 返回值
  - 成功：返回查询到的多边形名称和WKT信息。
  - 未查询到：返回nil。
  - 其它情况：返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> GIS.ADD Sicily "Palermo" "POINT (13.361389 38.115556)" "Catania" "POINT(1
5.087269 37.502669)"
(integer) 2
127.0.0.1:6379> GIS.SEARCH Sicily RADIUS 15 37 200 km WITHDIST
1) "2"
2) 1) "Palermo"
   2) "POINT(13.361389 38.115556)"
   3) "190.4424"
   4) "Catania"
   5) "POINT(15.087269 37.502669)"
   6) "56.4413"
127.0.0.1:6379> GIS.SEARCH Sicily RADIUS 15 37 200 km WITHDIST WITHOUTWKT
1) "2"
2) 1) "Palermo"
   2) "190.4424"
   3) "Catania"
   4) "56.4413"
127.0.0.1:6379> GIS.SEARCH Sicily RADIUS 15 37 200 km WITHDIST WITHOUTWKT ASC
1) "2"
2) 1) "Catania"
   2) "56.4413"
   3) "Palermo"
   4) "190.4424"
127.0.0.1:6379> GIS.SEARCH Sicily RADIUS 15 37 200 km WITHDIST WITHOUTWKT ASC COUNT 1
1) "2"
2) 1) "Catania"
   2) "56.4413"
127.0.0.1:6379> GIS.ADD Sicily "Agrigento" "POINT (13.583333 37.316667)"
(integer) 1
127.0.0.1:6379> GIS.SEARCH Sicily MEMBER Agrigento 100 km
1) "2"
2) 1) "Palermo"
   2) "POINT(13.361389 38.115556)"
   3) "Agrigento"
   4) "POINT(13.583333 37.316667)"
```

## GIS.WITHIN

- 语法

```
GIS.WITHIN <area> <polygonWkt> [WITHOUTWKT]
```

- 时间复杂度
  - 最理想情况：$O(\log_M n)$
  - 最差情况：log(n)
- 命令描述

  在area中搜索指定的多边形内部的点、线或面。如果设置了*WITHOUTWKT*选项，仅返回多边形的名称。
- 返回值
  - 成功：成功则返回对应多边形的名称和WKT信息。

- 未查找到：返回null。
- 其它情况：返回相应的异常信息。

- 使用示例

```
127.0.0.1:6379> GIS.ADD hangzhou campus 'POINT (30 10)'
(integer) 1
127.0.0.1:6379> GIS.ADD hangzhou campus1 'LINESTRING (30 10, 40 40)'
(integer) 1
127.0.0.1:6379> GIS.WITHIN hangzhou 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'
1) "2"
2) 1) "campus"
   2) "POINT(30 10)"
   3) "campus1"
   4) "LINESTRING(30 10,40 40)"
```

# 3.6. TairBloom命令

本文介绍TairBloom数据支持的命令。

## TairBloom简介

TairBloom是一种可动态扩容的布隆过滤器。TairBloom首先是一种概率性数据结构（space-efficient probabilistic data structure），主要使用较低的内存消耗来判断一个元素是否存在；其次，TairBloom具有动态扩容的能力，可在扩容的同时维持误判率的稳定。

在传统的Redis数据结构中，可以使用Hash、Set、String的Bitset等实现类似功能，但这些实现方式不是内存占用量非常大，就是无法动态伸缩和保持误判率不变。因此，TairBloom非常适合需要高效判断大量数据是否存在且允许一定误判率的业务场景。业务可以直接使用TairBloom提供的Bloom Filter接口，无需二次封装，更无需在本地实现布隆过滤器的功能。

主要特性：

- 内存占用低。
- 可动态扩容。
- 可自定义的误判率（False Positive Rate）且在扩容时保持不变。

## 使用前提

请注意，本文介绍的命令只有在满足以下条件时才能生效。

- Redis实例为企业版性能增强型。
- 操作对象为性能增强型实例中的TairBloom数据。

## 命令列表

## TairBloom命令

| 命令 | 语法 | 说明 |
| --- | --- | --- |
| BF.RESERVE | BF.RESERVE <key> <error_rate> <capacity> | 创建一个大小为capacity，错误率为error_rate的空的TairBloom。 |
| BF.ADD | BF.ADD <key> <item> | 在key指定的TairBloom中添加一个元素item。 |

| 命令 | 语法 | 说明 |
| --- | --- | --- |
| BF.MADD | BF.MADD <key> <item> [item...] | 在key指定的TairBloom中一次性添加多个元素。 |
| BF.EXISTS | BF.EXISTS <key> <item> | 检查一个元素是否存在于key指定的TairBloom中。 |
| BF.MEXISTS | BF.MEXISTS <key> <item> [item...] | 同时检查多个元素是否存在于key指定的TairBloom中。 |
| BF.INSERT | BF.INSERT <key> [CAPACITY cap] [ERROR error] [NOCREATE] ITEMS <item...> | 在key指定的TairBloom中一次性添加多个元素，添加时可以指定大小和错误率，且可以控制在TairBloom不存在的时候是否自动创建。 |
| BF.DEBUG | BF.DEBUG <key> | 可以查看key指定的TairBloom内部信息，如当前层数和每一层的元素个数、错误率等。 |
| DEL | DEL <key> [key ...] | 使用原生Redis的DEL命令可以删除一条或多条TairBloom数据。<br>**说明** 已加入TairBloom数据中的元素无法单独删除，您可以使用DEL命令删除整条TairBloom数据。 |

## BF.RESERVE

- 语法

  BF.RESERVE <key> <error_rate> <capacity>

- 时间复杂度

  O(1)

- 命令描述

  创建一个大小为capacity，错误率为error_rate的空的TairBloom。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairBloom的key，用于指定作为命令调用对象的TairBloom。 |
| error_rate | 期望的错误率（False Positive Rate），该值必须介于0和1之间。该值越小，TairBloom的内存占用量越大，CPU使用率越高。 |
| capacity | TairBloom的初始容量，即期望添加到TairBloom中的元素的个数。<br>当实际添加的元素个数超过该值时，TairBloom将进行自动的扩容，该过程会导致性能有所下降，下降的程度是随着元素个数的指数级增长而线性下降的，这是因为TairBloom的扩容是通过增加Bloom Filter的层数来完成的。每增加一层，在查询的时候就可能会遍历多层Bloom Filter来完成，每一层的容量都是上一层的两倍。因此，如果对性能非常的敏感，需要在使用前充分评估要添加到TairBloom的元素个数，避免发生扩容操作。 |

- 返回值
  - 成功：OK。

- 其它情况返回相应的异常信息。

## BF.ADD

- 语法

  BF.ADD <key> <item>

- 时间复杂度

  O(log N)，其中N是TairBloom的层数。

- 命令描述

  在key指定的TairBloom中添加一个元素。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairBloom的key，用于指定作为命令调用对象的TairBloom。 |
| item | 需要添加到TairBloom的元素。 |

- 返回值
  - 元素一定不存在：1。
  - 元素可能已经存在：0。
  - 其它情况返回相应的异常信息。

## BF.MADD

- 语法

  BF.MADD <key> <item> [item...]

- 时间复杂度

  O(log N)，其中N是TairBloom的层数。

- 命令描述

  在key指定的TairBloom中添加多个元素。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairBloom的key，用于指定作为命令调用对象的TairBloom。 |
| item | 需要添加到TairBloom的元素，可设置多个。 |

- 返回值
  - 成功：返回一个数组，数组的每一个元素可能为1或0，当item一定不存在时数组元素值为1，当item可能已经存在时数组元素值为0。
  - 其它情况返回相应的异常信息。

## BF.EXISTS

- 语法

  BF.EXISTS <key> <item>

- 时间复杂度

  O(log N)，其中N是TairBloom的层数。

- 命令描述

  检查一个元素是否存在于key指定的TairBloom中。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairBloom的key，用于指定作为命令调用对象的TairBloom。 |
| item | 需要查询的元素。 |

- 返回值
  - 元素一定不存在：0。
  - 元素可能存在：1。
  - 其它情况返回相应的异常信息。

## BF.MEXISTS

- 语法

  BF.MEXISTS <key> <item> [item...]

- 时间复杂度

  O(log N)，其中N是TairBloom的层数。

- 命令描述

  同时检查多个元素是否存在于key指定的TairBloom中。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairBloom的key，用于指定作为命令调用对象的TairBloom。 |
| item | 需要查询的元素，可设置多个。 |

- 返回值
  - 成功：返回一个数组，数组的每一个元素可能为1或0，当item一定不存在时数组元素值为0，当item可能已经存在时数组元素值为1。
  - 其它情况返回相应的异常信息。

## BF.INSERT

- 语法

  BF.INSERT <key> [CAPACITY cap] [ERROR error] [NOCREATE] ITEMS <item...>

- 时间复杂度

O(log N)，其中N是TairBloom的层数。

- 命令描述

  在key指定的TairBloom中一次性添加多个元素，添加时可以指定大小和错误率，且可以控制在TairBloom不存在的时候是否自动创建。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairBloom的key，用于指定作为命令调用对象的TairBloom。 |
| CAPACITY | 指定TairBloom的容量，即期望添加到TairBloom中的元素的个数，当TairBloom已经存在时该值将被忽略。<br>当实际添加的元素个数超过该值时，TairBloom将进行自动的扩容，该过程会导致性能有所下降，下降的程度是随着元素个数的指数级增长而线性下降的，这是因为TairBloom的扩容是通过增加Bloom Filter的层数来完成的。每增加一层，在查询的时候就可能会遍历多层Bloom Filter来完成，每一层的容量都是上一层的两倍。因此，如果对性能非常的敏感，需要在使用前充分评估要添加到TairBloom的元素个数，避免发生扩容操作。 |
| ERROR | 期望的错误率（False Positive Rate），当TairBloom已经存在时该值将被忽略。该值必须介于0和1之间。该值越小，TairBloom的内存占用量越大，CPU使用率越高。 |
| NOCREATE | 设置该选项后，当指定的TairBloom不存在的时候不要自动创建该TairBloom。该参数不能与CAPACITY和ERROR同时设置。 |
| ITEMS | 需要添加到TairBloom中的所有元素。 |

- 返回值
  - 成功：返回一个数组，数组的每一个元素可能为1或0，当item一定不存在时数组元素为1，当item可能已经存在时数组元素值为0。
  - 其它情况返回相应的异常信息。

## BF.DEBUG

- 语法

  BF.DEBUG <key>

- 时间复杂度

  O(log N)，其中N是TairBloom的层数。

- 命令描述

  可以查看key指定的TairBloom内部信息，如当前层数和每一层的元素个数、错误率等。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairBloom的key，用于指定作为命令调用对象的TairBloom。 |

- 返回值

- 成功：返回一个数组，数组的每一个元素可能为1或0，当item一定不存在时数组元素为1，当item可能已经存在时数组元素值为0。
- 其它情况返回相应的异常信息。

### 内存占用测试结果

| 容量（元素的个数） | false positive:0.01 | false positive:0.001 | false positive:0.0001 |
| --- | --- | --- | --- |
| 100000 | 0.12 MB | 0.25 MB | 0.25 MB |
| 1000000 | 2 MB | 2 MB | 4 MB |
| 10000000 | 16 MB | 32 MB | 32 MB |
| 100000000 | 128 MB | 256 MB | 256 MB |
| 1000000000 | 2 GB | 2 GB | 4 GB |

# 3.7. TairDoc命令

本文介绍TairDoc数据支持的命令。

## TairDoc简介

TairDoc是一种文档数据结构，支持JSON数据的增删改查。

主要特性：

- 完整地支持JSON标准。
- 完全兼容RedisJSON。
- 支持JSONPointer和JSONPath两种语法。
- 文档作为二进制树存储，可以快速访问JSON数据的子元素。
- 支持JSON到XML或YAML格式的转换。

## 使用前提

请注意，本文介绍的命令只有在满足以下条件时才能生效。

- Redis实例为企业版性能增强型。
- 操作对象为性能增强型实例中的TairDoc数据。

## 命令列表

### TairDoc命令

| 命令 | 语法 | 说明 |
| --- | --- | --- |
| JSON.SET | JSON.SET <key> <path> <json> [NX or XX] | 存储JSON的值在key的path中。对于不存在的key，path必须是root。对于已经存在的key，当path存在时，替换掉目前的JSON值。 |

| 命令 | 语法 | 说明 |
| --- | --- | --- |
| JSON.GET | JSON.GET <key> [PATH] [FORMAT <XML/YAML>] [ROOTNAME <root>] [ARRNAME <arr>] | 获取一个TairDoc在path中存储的JSON数据。 |
| JSON.DEL | JSON.DEL <key> [path] | 删除path对应的JSON数据。如果未指定path，则删除key。不存在的key和不存在的path将会被忽略。 |
| JSON.TYPE | JSON.TYPE <key> [path] | 获取path对应的值的类型。 |
| JSON.NUMINCRBY | JSON.NUMINCRBY <key> [path] <value> | 将path对应的值增加value。path必须存在且path对应的值和value必须是都是int或double类型的值。 |
| JSON.STRAPPEND | JSON.STRAPPEND <key> [path] <json-string> | 将json-string中的字符串添加到path对应的字符串类型的值中。如果未设置path，默认为root。 |
| JSON.STRLEN | JSON.STRLEN <key> [path] | 获取path对应值的长度。如果未设置path，默认为root。 |
| JSON.ARRAPPEND | JSON.ARRAPPEND <key> <path> <json> [<json> ...] | 将JSON插入到path对应的array最后，JSON可以是多个。 |
| JSON.ARRPOP | JSON.ARRPOP <key> <path> [index] | 移除并返回path对应的数组中index位置的元素。 |
| JSON.ARRINSERT | JSON.ARRINSERT <key> <path> <index> <json> [<json> ...] | 将JSON插入到path对应的数组中，原有元素会往后移动。 |
| JSON.ARRLEN | JSON.ARRLEN <key> [path] | 获取path对应数组的长度。 |
| JSON.ARRTRIM | JSON.ARRTRIM <key> <path> <start> <stop> | 按照start和stop指定的范围修剪数组。 |
| DEL | DEL <key> [key ...] | 使用原生Redis的DEL命令可以删除一条或多条TairDoc数据。 |

## JSON.SET

- 语法

  JSON.SET <key> <path> <json> [NX | XX]

- 时间复杂度

  O(N)

- 命令描述

  存储JSON的值在key的path中。对于不存在的key，path必须是root。对于已经存在的key，当path存在时，替换掉目前的JSON值。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |

| 参数或选项 | 说明 |
| --- | --- |
| path | TairDoc的path。<br>○ 如果key不存在，path必须为root。<br>○ 如果key存在，当path存在时，替换当前的JSON值。 |
| json | 如果key存在，当path存在时，用JSON替换TairDoc中当前的JSON数据。 |
| NX | 当path不存在时写入。 |
| XX | 当path存在时写入。 |

- 返回值
  - 成功：OK。
  - 指定了NX或XX时失败：null。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"foo": "bar", "baz" : 42}'
OK
127.0.0.1:6379> JSON.SET doc .foo '"flower"'
OK
127.0.0.1:6379> JSON.GET doc .foo
"flower"
127.0.0.1:6379> JSON.SET doc .not-exists 123 XX
127.0.0.1:6379> JSON.SET doc .not-exists 123 NX
OK
127.0.0.1:6379> JSON.GET doc .not-exists
123
```

## JSON.GET

- 语法

  JSON.GET <key> <path> [FORMAT <XML | YAML>] [ROOTNAME <root>] [ARRNAME <arr>]
- 时间复杂度

  O(N)
- 命令描述

  获取一个TairDoc在path中存储的JSON数据。
- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |
| FORMAT | 指定返回的JSON格式，支持XML或YAML。 |

| 参数或选项 | 说明 |
| --- | --- |
| ROOTNAME | 指定XML语法ROOT元素的标签。 |
| ARRNAME | 指定XML语法ARRAY元素的标签。 |

- 返回值
  - 成功：返回path中存储的JSON。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"foo": "bar", "baz" : 42}'
OK
127.0.0.1:6379> JSON.GET doc
{"foo":"bar","baz":42}
127.0.0.1:6379> JSON.GET doc .foo
"bar"
127.0.0.1:6379> JSON.GET doc .not-exists
ERR pointer illegal or array index error or object type is not array or map
127.0.0.1:6379> JSON.GET doc . format xml
<?xml version="1.0" encoding="UTF-8"?><root><foo>bar</foo><baz>42</baz></root>
127.0.0.1:6379> JSON.GET doc . format xml rootname ROOT arrname ARRAY
<?xml version="1.0" encoding="UTF-8"?><ROOT><foo>bar</foo><baz>42</baz></ROOT>
127.0.0.1:6379> JSON.GET doc . format yaml
foo: bar
baz: 42
```

## JSON.DEL

- 语法

  JSON.DEL <key> [path]
- 时间复杂度

  O(N)
- 命令描述

  删除path对应的JSON数据。如果未指定path，则删除key。不存在的key和不存在的path将会被忽略。
- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |

- 返回值
  - 成功：1。
  - 失败：0。
  - 其它情况返回相应的异常信息。

- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"foo": "bar", "baz" : 42}'
OK
127.0.0.1:6379> JSON.DEL doc .foo
1
127.0.0.1:6379> JSON.DEL doc .not-exists
ERR old item is null for remove or replace
127.0.0.1:6379> JSON.DEL not-exists
0
127.0.0.1:6379> JSON.GET doc
{"baz":42}
127.0.0.1:6379> JSON.DEL doc
1
127.0.0.1:6379> JSON.GET doc
127.0.0.1:6379>
```

## JSON.TYPE

- 语法

  JSON.TYPE <key> [path]

- 时间复杂度

  O(N)

- 命令描述

  获取path对应的值的类型。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |

- 返回值
  - 成功：返回查询到的类型，可能的结果包括"boolean"、"null"、 "number"、"string"、"array"、"object"、"raw"、"reference"、"const"。
  - key或者path不存在：null。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"foo": "bar", "baz" : 42}'
OK
127.0.0.1:6379> JSON.TYPE doc
object
127.0.0.1:6379> JSON.TYPE doc .foo
string
127.0.0.1:6379> JSON.TYPE doc .baz
number
127.0.0.1:6379> JSON.TYPE doc .not-exists
127.0.0.1:6379>
```

## JSON.NUMINCRBY

- 语法

  JSON.NUMINCRBY <key> [path] <value>

- 时间复杂度

  O(N)

- 命令描述

  将path对应的值增加value。path必须存在且path对应的值和value必须是都是int或double类型的值。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |
| value | 需要为path的值增加的数值。 |

- 返回值
  - 成功：返回操作完成后path对应的值。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"foo": "bar", "baz" : 42}'
OK
127.0.0.1:6379> JSON.NUMINCRBY doc .baz 1
43
127.0.0.1:6379> JSON.NUMINCRBY doc .baz 1.5
44.5
127.0.0.1:6379> JSON.NUMINCRBY doc .foo 1
ERR node not exists or not number type
127.0.0.1:6379> JSON.NUMINCRBY doc .not-exists 1
ERR node not exists or not number type
127.0.0.1:6379>
```

## JSON.STRAPPEND

- 语法

JSON.STRAPPEND <key> [path] <json-string>

- 时间复杂度

  O(N)

- 命令描述

  将json-string中的字符串添加到path对应的字符串类型的值中。如果未设置path，默认为root。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |
| json-string | 需要添加到path对应值的字符串。 |

- 返回值
  - 成功：返回path对应的字符串扩展后的长度。
  - key不存在：-1。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"foo": "bar", "baz" : 42}'
OK
127.0.0.1:6379> JSON.STRAPPEND doc .foo rrrrr
8
127.0.0.1:6379> JSON.GET doc .foo
"barrrrrr"
127.0.0.1:6379> JSON.STRAPPEND doc .not-exists
ERR node not exists or not string type
127.0.0.1:6379> JSON.STRAPPEND not-exists abc
-1
```

## JSON.STRLEN

- 语法

  JSON.STRLEN <key> [path]

- 时间复杂度

  O(N)

- 命令描述

  获取path对应值的长度。如果未设置path，默认为root。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |

| 参数或选项 | 说明 |
| --- | --- |
| path | TairDoc的path。 |

- 返回值
  - 成功：返回path对应值的长度。
  - key不存在：-1。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"foo": "bar", "baz" : 42}'
OK
127.0.0.1:6379> JSON.STRLEN doc .foo
3
127.0.0.1:6379> JSON.STRLEN doc .baz
ERR node not exists or not string type
127.0.0.1:6379> JSON.STRLEN not-exists
-1
```

## JSON.ARRAPPEND

- 语法

  JSON.ARRAPPEND <key> <path> <json> [<json> ...]
- 时间复杂度

  O(M*N)，M是需要插入的元素（json）数量，N是数组元素数量。
- 命令描述

  将JSON插入到path对应的array最后，JSON可以是多个。
- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |
| json | 需要插入的数据。 |

- 返回值
  - 成功：返回操作完成后array中元素数量。
  - key不存在：-1。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"id": [1,2,3]}'
OK
127.0.0.1:6379> JSON.GET doc .id
[1,2,3]
127.0.0.1:6379> JSON.ARRAPPEND doc .id null false true
6
127.0.0.1:6379> JSON.GET doc .id
[1,2,3,null,false,true]
127.0.0.1:6379> JSON.GET doc .id.2
3
127.0.0.1:6379> JSON.ARRAPPEND not-exists .a 1
-1
```

## JSON.ARRPOP

- 语法

  JSON.ARRPOP <key> <path> [index]

- 时间复杂度

  O(M*N)，M是key包含的子元素，N是数组元素数量。

- 命令描述

  移除并返回path对应的数组中index位置的元素。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |
| index | 数组的索引，不传默认为最后一个元素，负数表示反向取值。 |

- 返回值
  - 成功：返回数组相应的节点。
  - 数组为空数组：‘ERR array index outflow’。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"id": [1,2,3]}'
OK
127.0.0.1:6379> JSON.ARRPOP doc .id 1
2
127.0.0.1:6379> JSON.GET doc .id
[1,3]
127.0.0.1:6379> JSON.ARRPOP doc .id -1
3
127.0.0.1:6379> JSON.GET doc .id
[1]
127.0.0.1:6379> JSON.ARRPOP doc .id 10
ERR array index outflow
127.0.0.1:6379> JSON.ARRPOP doc .id
1
127.0.0.1:6379> JSON.ARRPOP doc .id
ERR array index outflow
127.0.0.1:6379>
```

## JSON.ARRINSERT

- 语法

  JSON.ARRINSERT <key> <path> <index> <json> [<json> ...]

- 时间复杂度

  O(M*N)，M是要插入的元素（json）数量，N是数组元素数量。

- 命令描述

  将JSON插入到path对应的数组中，原有元素会往后移动。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |
| index | 数组的索引，不传默认为最后一个元素，负数表示反向取值。 |
| json | 需要插入的数据。 |

- 返回值
  - 成功：返回操作完成后数组中元素的个数。
  - 数组为空数组：‘ERR array index outflow’。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"id": [2,3,5]}'
OK
127.0.0.1:6379> JSON.ARRINSERT doc .id 0 0 1
5
127.0.0.1:6379> JSON.GET doc .id
[0,1,2,3,5]
127.0.0.1:6379> JSON.ARRINSERT doc .id 4 4
6
127.0.0.1:6379> JSON.GET doc .id
[0,1,2,3,4,5]
127.0.0.1:6379>
```

## JSON.ARRLEN

- 语法

  JSON.ARRLEN <key> [path]

- 时间复杂度

  O(N)

- 命令描述

  获取path对应数组的长度。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |

- 返回值
  - 成功：数组的长度。
  - key不存在：-1。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"id": [2,3,5]}'
OK
127.0.0.1:6379> JSON.ARRLEN doc .id
3
127.0.0.1:6379> JSON.ARRLEN not-exists
-1
```

## JSON.ARRTRIM

- 语法

  JSON.ARRTRIM <key> <path> <start> <stop>

- 时间复杂度

  O(N)

- 命令描述

  按照start和stop指定的范围修剪数组。

- 参数及选项说明

| 参数或选项 | 说明 |
| --- | --- |
| key | TairDoc的key，用于指定作为命令调用对象的TairDoc。 |
| path | TairDoc的path。 |
| start | 修剪的开始位置，取值为从0开始的一个索引值，修剪后的数组包含该位置的元素。 |
| stop | 修剪的结束位置，取值为从0开始的一个索引值，修剪后的数组包含该位置的元素。 |

- 返回值
  - 成功：操作完成后数组的长度。
  - key不存在：-1。
  - 其它情况返回相应的异常信息。
- 使用示例

```
127.0.0.1:6379> JSON.SET doc . '{"id": [1,2,3,4,5,6]}'
OK
127.0.0.1:6379> JSON.ARRTRIM doc .id 3 4
2
127.0.0.1:6379> JSON.GET doc .id
[4,5]
127.0.0.1:6379> JSON.ARRTRIM doc .id 3 4
ERR array index outflow
127.0.0.1:6379> JSON.ARRTRIM doc .id -2 -5
ERR array index outflow
127.0.0.1:6379>
```

## JSONPointer和JSONPath

TairDoc在支持JSONPointer的基础上支持了JSONPath的部分语法，示例如下：

| JSONPointer | JSONPath |
| --- | --- |
| 127.0.0.1:6379> JSON.SET doc . '{"foo": "bar", "baz" : [1,2,3]}'<br>OK<br>127.0.0.1:6379> JSON.GET doc .foo<br>"bar"<br>127.0.0.1:6379> JSON.GET doc .baz[0]<br>1 | 127.0.0.1:6379> JSON.SET doc "" '{"foo": "bar", "baz" : [1,2,3]}'<br>OK<br>127.0.0.1:6379> JSON.GET doc /foo<br>"bar"<br>127.0.0.1:6379> JSON.GET doc /baz/0<br>1 |

具体的兼容方式如下表所示。

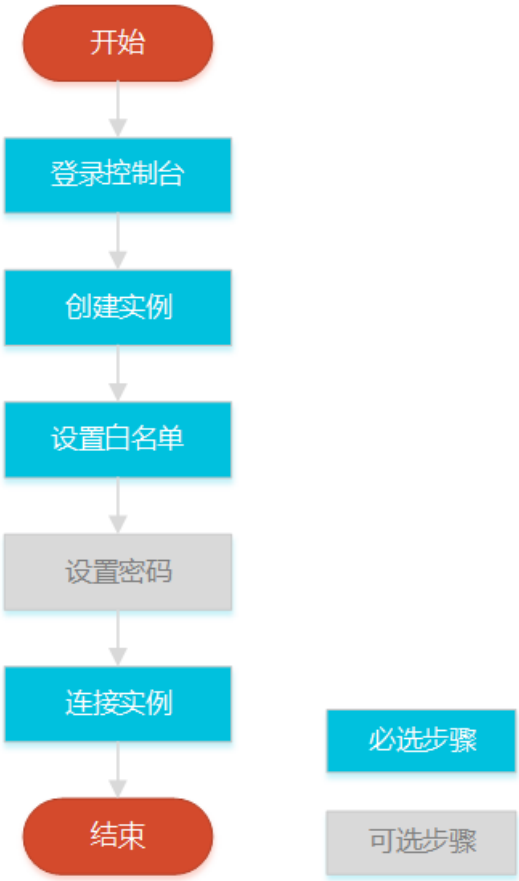
| 兼容项 | JSONPath | JSONPointer |
| --- | --- | --- |
| 根元素 | . | “” |
| 获取元素 | .a.b.c | /a/b/c |
| 数组 | .a[2] | /a/2 |
| 整体获取 | .a["b.c"] | /a/b.c |
| | .a['b.c'] | /a/b.c |

# 4.快速入门

## 4.1. 开始使用云数据库Redis

本章节介绍如何从创建实例到登录数据库的一系列操作，帮助您快速地了解Redis实例的操作流程。

Redis实例操作步骤流程图如下所示。




- 登录云数据库Redis控制台

  介绍如何登录到云数据库Redis控制台。

- 创建实例

  阿里云数据库支持经典网络和专有网络两种网络类型，您可以创建不同网络下的Redis实例。

- 设置白名单

  为了数据库的安全稳定，在开始使用 Redis 实例前，您需要将访问数据库的 IP 地址或者 IP 段加到目标实例的白名单中。

- 如果在创建实例时，没有设置实例的密码，用户需要在**实例信息**页面设置实例的密码。

- 连接实例

  用户可以使用支持Redis协议的客户端连接实例，也可以使用Redis原生的工具redis-cli连接实例。

## 4.2. 登录云数据库Redis控制台

本章节介绍如何登录到云数据库Redis控制台。

## 前提条件

- 登录Apsara Uni-manager运营控制台前，确认您已从部署人员处获取Apsara Uni-manager运营控制台的服务域名地址。
- 推荐使用Chrome浏览器。

## 操作步骤

1. 在浏览器地址栏中，输入Apsara Uni-manager运营控制台的服务域名地址，按回车键。
2. 输入正确的用户名及密码。

   请向运营管理员获取登录控制台的用户名和密码。

   > **说明** 首次登录Apsara Uni-manager运营控制台时，需要修改登录用户名的密码，请按照提示完成密码修改。为提高安全性，密码长度必须为8~20位，且至少包含以下两种类型：
   > - 英文大写或小写字母（A~Z、a~z）
   > - 阿拉伯数字（0~9）
   > - 特殊符号（感叹号（！）、at（@）、井号（#）、美元符号（$）、百分号（%）等）

3. 单击**登录**。
4. 如果账号已激活MFA多因素认证，请根据以下两种情况进行操作：
   - 管理员强制开启MFA后的首次登录：
     a. 在绑定虚拟MFA设备页面中，按页面提示步骤绑定MFA设备。
     b. 按照步骤2重新输入账号和密码，单击**登录**。
     c. 输入6位MFA码后单击**认证**。
   - 您已开启并绑定MFA：

     输入6位MFA码后单击**认证**。

     > **说明** 绑定并开启MFA的操作请参见*Apsara Uni-manager运营控制台用户指南*中的章节*绑定并开启虚拟MFA设备*。

5. 在页面顶部的菜单栏中，选择**产品 > 数据库 > 云数据库 Redis**。

# 4.3. 创建实例

本章节介绍如何通过Redis控制台创建实例。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 单击页面右上角**创建实例**。
3. 设置以下参数。

### 设置新实例参数

| 类别 | 配置 | 说明 |
| --- | --- | --- |
| **基本配置** | **组织** | 选择Redis实例所属组织。 |
| | **资源集** | 选择Redis实例所属资源集。<br>**注意** 选择项目后，只有属于该资源集的成员才能使用该实例。 |
| **区域** | **地域** | 选择Redis实例所属地域。 |
| | **可用区** | 选择Redis实例所属可用区，若选择多可用区即可实现同城容灾。<br>**说明** 多可用区仅支持创建**标准版**、**集群版**类型。 |
| **规格配置** | **版本类型** | ○ **社区版**：兼容开源Redis协议标准、高性能的数据库服务。<br>○ **企业版**：基于社区版开发的Redis产品，读写性能可达到社区版的3倍，同时集成多个自研的Redis模块集成，扩展了Redis的适用性。更多信息，请参见*产品简介*手册中的*企业版性能增强型介绍*章节。 |
| | **芯片架构** | 选择Redis实例所属机器的芯片架构。<br>**说明** 选择该配置时如提示未授权，请联系运营管理员对该账号执行授权。 |
| | **引擎版本** | 选择Redis的引擎版本，支持Redis 4.0（社区版）和5.0。 |
| | **架构类型** | ○ **标准版**：采用主从架构，能提供高性能的缓存服务，同时保证数据的高可用。<br>○ **集群版**：可轻松突破Redis自身单线程瓶颈，满足大容量、高性能的业务需求。<br>○ **读写分离**：可提供高可用、高性能、高灵活的读写分离服务，解决热点数据集中及高并发读取的业务需求，最大化地节约用户运维成本。<br>**说明** 仅社区版支持**读写分离**架构。 |
| | **节点类型** | 固定为**双副本**，即一主一从共两个节点，双机热备保障可用性。 |
| | **实例规格** | 选择实例的规格。<br>不同的实例规格提供的最大连接数和最大内网带宽不同，更多信息，请参见*产品简介*中的*实例规格*章节。 |

| 类别 | 配置 | 说明 |
| --- | --- | --- |
| **网络** | **网络类型** | ○ 经典网络：经典网络中的云服务在网络上不进行隔离，只能依靠云服务自身的安全组或白名单策略来阻挡非法访问。<br>○ 专有网络VPC（Virtual Private Cloud）：专有网络帮助您在阿里云上构建出一个隔离的网络环境。您可以自定义专有网络里面的路由表、IP地址范围和网关。此外还可以通过专线或者VPN的方式将自建机房与阿里云专有网络内的云资源组合成一个虚拟机房，实现应用平滑上云。<br>**说明** 若选择专有网络，请先创建好VPC，如何创建VPC请参见*VPC用户指南*中的*创建专有网络*和*创建交换机*章节。 |
| **配置密码** | **实例名称** | 设置实例的名称，便于后续业务识别。<br>○ 长度为2~128个字符。<br>○ 以大小写字母或中文开头，可包含字母数字、下划线（_）和短划线（-）。 |
| | **设置密码** | 您可以选择**立即设置**或选择**创建后设置**。 |
| | **登录密码** | 设置访问实例的密码，密码要求如下：<br>○ 长度为8~30个字符。<br>○ 必须同时包含大小写字母和数字，不支持特殊符号。 |
| | **确认密码** | 再次输入访问实例的密码。 |

4. 完成上述参数配置后，单击**提交**。

# 4.4. 设置白名单

为了数据库的安全稳定，在开始使用Redis实例前，您需要将访问数据库的IP地址或者IP段加到目标实例的白名单中。

## 背景信息

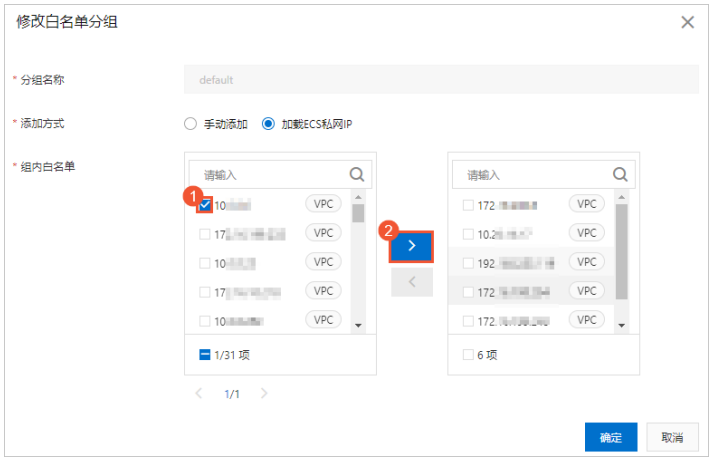
**说明** 正确使用白名单可以让Redis得到高级别的访问安全保护，建议您定期维护白名单。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**白名单设置**。
4. 找到目标白名单分组，单击**修改**。

   **说明** 您也可以单击**添加白名单分组**创建一个新的分组。分组名称长度为2~32个字符，由小写字母、数字或下划线组成，需以小写字母开头，以小写字母或数字结尾。

5. 在弹出的对话框中，根据要执行的操作，选择下述操作步骤：

| 要执行的操作 | 操作步骤 |
| --- | --- |
| 手动修改白名单 | 手动填写IP地址或IP地址段。<br>手动修改白名单<br>  <br><br>**说明**<br>○ IP地址以英文逗号分隔，不可重复，最多1000个。支持格式为10.23.12.24（具体IP地址）、10.23.12.24/24（CIDR模式，即无类域间路由，/24表示地址中前缀的长度，范围为1~32）。<br>○ 地址中的前缀长度设置为0（例如0.0.0.0/0、127.0.0.1/0）表示允许所有IP地址访问该实例，存在高安全风险，请谨慎设置。 |
| 加载ECS实例的内网IP地址为白名单 | i. 单击**加载ECS私网IP**。<br>ii. 根据业务需求选中对应的IP地址。<br>加载ECS内网IP地址<br>  <br><br>**说明** 为便于定位目标IP地址，您可以将鼠标指针悬浮在对应的IP地址上，系统将展示该IP地址所属的ECS实例ID和名称。 |
| 清除白名单分组 | 当某个白名单分组中的所有IP地址均需要移除且需要保留该分组时，您可以单击**清除白名单分组**来完成该操作。 |

6. 单击**确定**。

# 4.5. 连接实例

## 4.5.1. 使用Redis客户端程序连接

云数据库Redis与原生Redis完全兼容，连接数据库的方式也基本相同，您可以根据自身应用特点选用任何兼容Redis协议的客户端程序。本文列举一些常见的客户端程序的代码示例，帮助您快速连接。

### 前提条件

已将ECS实例的内网IP地址或本地主机的外网IP地址加入Redis白名单，具体操作，请参见设置白名单。

### 获取连接信息

在使用客户端程序连接Redis实例时，通常您需要获取以下信息并设置在代码中：

| 需获取的信息 | 获取方式 |
| --- | --- |
| 实例的连接地址 | 可在**实例信息**页面中的**连接信息**区域框中获取到。<br>**说明** Redis实例支持多种连接地址，推荐使用专有网络连接，可获得更高的安全性和更低的网络延迟。 |
| 端口号 | 端口号默认为6379。 |
| 实例的账号（部分客户端程序无需设置） | Redis实例默认会创建一个以实例ID命名的账号（例如r-bp10noxlhcoim2****）。 |
| 账号的密码 | 在创建实例时已设置，如果忘记密码，可重置密码，具体操作，请参见修改密码。 |

### 常见客户端程序

关于Redis支持的客户端列表，请参见Redis Clients。

- Jedis客户端
- PhpRedis客户端
- redis-py客户端
- C或C++客户端
- .net客户端
- node-redis客户端
- C#客户端StackExchange.Redis

### Jedis客户端

1. 下载并安装Jedis客户端。具体操作，请参见Jedis使用说明。
2. 根据业务需求选择连接方式。
   - JedisPool连接池连接（推荐）

a. 打开Eclipse客户端，创建一个Project并配置pom文件，具体内容如下：

```
<dependency>
<groupId>redis.clients</groupId>
<artifactId>jedis</artifactId>
<version>2.7.2</version>
<type>jar</type>
<scope>compile</scope>
</dependency>
```

b. 在Project中输入下述代码添加相关应用。

```
import org.apache.commons.pool2.PooledObject;
import org.apache.commons.pool2.PooledObjectFactory;
import org.apache.commons.pool2.impl.DefaultPooledObject;
import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;
```

c. 根据Jedis客户端版本，在Project中输入下述代码，然后根据注释提示修改代码。

> **说明** 关于如何获取Redis实例的连接地址和密码，请参见获取连接信息。

- Jedis 2.7.2版本

```
JedisPoolConfig config = new JedisPoolConfig();
//最大空闲连接数，需自行评估，不超过Redis实例的最大连接数
config.setMaxIdle(200);
//最大连接数，需自行评估，不超过Redis实例的最大连接数
config.setMaxTotal(300);
config.setTestOnBorrow(false);
config.setTestOnReturn(false);
String host = "*.aliyuncs.com";
String password = "密码";
JedisPool pool = new JedisPool(config, host, 6379, 3000, password);
Jedis jedis = null;
try {
jedis = pool.getResource();
/// ... do stuff here ... for example
jedis.set("foo", "bar");
String foobar = jedis.get("foo");
jedis.zadd("sose", 0, "car");
jedis.zadd("sose", 0, "bike");
Set<String> sose = jedis.zrange("sose", 0, -1);
} finally {
if (jedis != null) {
jedis.close();
}
}
/// ... when closing your application:
pool.destroy();
```

- Jedis 2.6或Jedis 2.5版本

```
JedisPoolConfig config = new JedisPoolConfig();
//最大空闲连接数，需自行评估，不超过Redis实例的最大连接数
config.setMaxIdle(200);
//最大连接数，需自行评估，不超过Redis实例的最大连接数
config.setMaxTotal(300);
config.setTestOnBorrow(false);
config.setTestOnReturn(false);
String host = "*.aliyuncs.com";
String password = "密码";
JedisPool pool = new JedisPool(config, host, 6379, 3000, password);
Jedis jedis = null;
boolean broken = false;
try {
     jedis = pool.getResource();
     /// ... do stuff here ... for example
     jedis.set("foo", "bar");
     String foobar = jedis.get("foo");
     jedis.zadd("sose", 0, "car");
     jedis.zadd("sose", 0, "bike");
     Set<String> sose = jedis.zrange("sose", 0, -1);
}
catch(Exception e)
{
     broken = true;
} finally {
if (broken) {
     pool.returnBrokenResource(jedis);
} else if (jedis != null) {
     pool.returnResource(jedis);
 }
}
```

- Jedis单连接（不推荐，单次超时后无法自动恢复）

  打开Eclipse客户端，创建一个Project，输入下述代码，然后根据注释提示修改代码。

  > **说明** 关于如何获取Redis实例的连接地址和密码，请参见获取连接信息。

```
import redis.clients.jedis.Jedis;
public class jedistest {
public static void main(String[] args) {
try {
     String host = "xx.kvstore.aliyuncs.com";//控制台显示访问地址
     int port = 6379;
     Jedis jedis = new Jedis(host, port);
     //鉴权信息
     jedis.auth("password");//password
     String key = "redis";
     String value = "aliyun-redis";
     //select db默认为0
     jedis.select(1);
     //set一个key
     jedis.set(key, value);
     System.out.println("Set Key " + key + " Value: " + value);
     //get 设置进去的key
     String getvalue = jedis.get(key);
     System.out.println("Get Key " + key + " ReturnValue: " + getvalue);
     jedis.quit();
     jedis.close();
}
catch (Exception e) {
 e.printStackTrace();
 }
}
}
```

3. 运行上述Project，在Eclipse的控制台输出如下运行结果则表示您已成功连接至云数据库Redis。

```
Set Key redis Value aliyun-redis
Get Key redis ReturnValue aliyun-redis
```

## PhpRedis客户端

1. 下载并安装PhpRedis客户端。具体操作，请参见PhpRedis使用说明。
2. 在PHP编辑器中输入下述代码，然后根据注释提示修改代码。

> **说明** 关于如何获取Redis实例的连接地址、账号和密码，请参见获取连接信息。

```
<?php
 /* 这里替换为连接的实例连接地址和端口 */
 $host = "r-bp10noxlhcoim2****.redis.rds.aliyuncs.com";
 $port = 6379;
 /* 这里替换为实例ID和实例的密码 */
 $user = "test_username";
 $pwd = "test_password";
 $redis = new Redis();
 if ($redis->connect($host, $port) == false) {
        die($redis->getLastError());
   }
 if ($redis->auth($pwd) == false) {
        die($redis->getLastError());
  }
  /* 认证后就可以进行数据库操作，详情请参见https://github.com/phpRedis/phpredis */
 if ($redis->set("foo", "bar") == false) {
        die($redis->getLastError());
 }
 $value = $redis->get("foo");
 echo $value;
 ?>
```

3. 执行上述代码即可完成连接。

   更多信息，请参见官方文档 。

## redis-py客户端

1. 下载并安装redis-py客户端。具体操作，请参见redis-py使用说明。
2. 在Python编辑器中输入下述代码，然后根据注释提示修改代码。

   > **说明** 关于如何获取Redis实例的连接地址和密码，请参见获取连接信息。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
import redis
#替换host的值为实例的连接地址，替换port的值为端口。
host = 'localhost'
port = 6379
#这里替换为实例password
pwd = 'test_password'
r = redis.StrictRedis(host=host, port=port, password=pwd)
#连接建立后就可以进行数据库操作，详情请参见https://github.com/andymccurdy/redis-py
r.set('foo', 'bar');
print r.get('foo')
```

3. 执行上述代码即可完成连接。

## C或C++客户端

1. 执行下述命令，下载并编译安装 C 客户端。

```
git clone https://github.com/redis/hiredis.git
cd hiredis
make
sudo make install
```

2. 在C或C++编辑器中输入下述代码，然后根据注释提示修改代码。

> **说明** 关于如何获取Redis实例的连接地址和密码，请参见获取连接信息。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hiredis.h>
int main(int argc, char **argv) {
unsigned int j;
redisContext *c;
redisReply *reply;
if (argc < 4) {
        printf("Usage: example xxx.kvstore.aliyuncs.com 6379 instance_id password\n");
        exit(0);
}
const char *hostname = argv[1];
const int port = atoi(argv[2]);
const char *instance_id = argv[3];
const char *password = argv[4];
struct timeval timeout = { 1, 500000 }; // 1.5 seconds
c = redisConnectWithTimeout(hostname, port, timeout);
if (c == NULL || c->err) {
if (c) {
        printf("Connection error: %s\n", c->errstr);
        redisFree(c);
} else {
        printf("Connection error: can't allocate redis context\n");
}
exit(1);
}
/* AUTH */
reply = redisCommand(c, "AUTH %s", password);
printf("AUTH: %s\n", reply->str);
freeReplyObject(reply);
/* PING server */
reply = redisCommand(c,"PING");
printf("PING: %s\n", reply->str);
freeReplyObject(reply);
/* Set a key */
reply = redisCommand(c,"SET %s %s", "foo", "hello world");
printf("SET: %s\n", reply->str);
freeReplyObject(reply);
/* Set a key using binary safe API */
reply = redisCommand(c,"SET %b %b", "bar", (size_t) 3, "hello", (size_t) 5);
printf("SET (binary API): %s\n", reply->str);
freeReplyObject(reply);
/* Try a GET and two INCR */
```

```
reply = redisCommand(c,"GET foo");
printf("GET foo: %s\n", reply->str);
freeReplyObject(reply);
reply = redisCommand(c,"INCR counter");
printf("INCR counter: %lld\n", reply->integer);
freeReplyObject(reply);
/* again ... */
reply = redisCommand(c,"INCR counter");
printf("INCR counter: %lld\n", reply->integer);
freeReplyObject(reply);
/* Create a list of numbers, from 0 to 9 */
reply = redisCommand(c,"DEL mylist");
freeReplyObject(reply);
for (j = 0; j < 10; j++) {
        char buf[64];
        snprintf(buf,64,"%d",j);
        reply = redisCommand(c,"LPUSH mylist element-%s", buf);
        freeReplyObject(reply);
    }
/* Let's check what we have inside the list */
reply = redisCommand(c,"LRANGE mylist 0 -1");
if (reply->type == REDIS_REPLY_ARRAY) {
        for (j = 0; j < reply->elements; j++) {
        printf("%u) %s\n", j, reply->element[j]->str);
}
}
freeReplyObject(reply);
/* Disconnects and frees the context */
redisFree(c);
return 0;
}
```

3. 编译上述代码。

```
gcc -o example -g example.c -I /usr/local/include/hiredis -lhiredis
```

4. 测试运行，完成连接。

```
example xxx.kvstore.aliyuncs.com 6379 instance_id password
```

## .net客户端

**警告** 如果您的Redis实例为集群版，且需要执行切换或选择数据库的操作（即使用多数据库功能），您必须先将cluster_compat_enable参数设置为*0*（即关闭原生Redis Cluster语法兼容），然后重启客户端应用，否则将提示报错：`Multiple databases are not supported on this server; cannot switch to database`。具体操作，请参见参数设置。

1. 执行下述命令，下载.net客户端。

```
git clone https://github.com/ServiceStack/ServiceStack.Redis
```

2. 在.net 客户端中新建.net项目。
3. 添加客户端引用，引用文件在库文件的ServiceStack.Redis/lib/tests中。

4. 在新建的.net项目中输入如下代码，然后根据注释提示修改代码。更多信息，请参见接口说明 。

> **说明** 关于如何获取Redis实例的连接地址和密码，请参见获取连接信息。

```
using System;
 using System.Collections.Generic;
 using System.Linq;
 using System.Text;
 using System.Threading.Tasks;
 using ServiceStack.Redis;
 namespace ServiceStack.Redis.Tests
 {
         class Program
 {
 public static void RedisClientTest()
 {
         string host = "127.0.0.1";/*访问host地址*/
         string password = "password";/*密码*/
         RedisClient redisClient = new RedisClient(host, 6379, password);
         string key = "test-aliyun";
         string value = "test-aliyun-value";
         redisClient.Set(key, value);
         string listKey = "test-aliyun-list";
         System.Console.WriteLine("set key " + key + " value " + value);
         string getValue = System.Text.Encoding.Default.GetString(redisClient.Get(key))
;
         System.Console.WriteLine("get key " + getValue);
         System.Console.Read();
 }
 public static void RedisPoolClientTest()
 {
         string[] testReadWriteHosts = new[] {
         "redis://password@127.0.0.1:6379"/*redis://密码@访问地址:端口*/
 };
 RedisConfig.VerifyMasterConnections = false;//需要设置
 PooledRedisClientManager redisPoolManager = new PooledRedisClientManager(10/*连接池个数
*/, 10/*连接池超时时间*/, testReadWriteHosts);
 for (int i = 0; i < 100; i++){
         IRedisClient redisClient = redisPoolManager.GetClient();//获取连接
         RedisNativeClient redisNativeClient = (RedisNativeClient)redisClient;
         redisNativeClient.Client = null;//ApsaraDB for Redis不支持client setname所以这里
需要显示的把client对象置为null
 try
 {
         string key = "test-aliyun1111";
         string value = "test-aliyun-value1111";
         redisClient.Set(key, value);
         string listKey = "test-aliyun-list";
         redisClient.AddItemToList(listKey, value);
         System.Console.WriteLine("set key " + key + " value " + value);
         string getValue = redisClient.GetValue(key);
         System.Console.WriteLine("get key " + getValue);
         redisClient.Dispose();//
```

```
}catch (Exception e)
{
        System.Console.WriteLine(e.Message);
}
}
        System.Console.Read();
}
static void Main(string[] args)
{
        //单链接模式
        RedisClientTest();
        //连接池模式
        RedisPoolClientTest();
}
}
}
```

## node-redis客户端

1. 下载并安装node-redis。

```
npm install hiredis redis
```

2. 在node-redis客户端中输入下述代码，然后根据注释提示修改代码。

> **说明** 关于如何获取Redis实例的连接地址和密码，请参见获取连接信息。

```
var redis = require("redis"),
client = redis.createClient(<port>, <"host">, {detect_buffers: true});
client.auth("password", redis.print)
```

参数说明：

- <port>：Redis实例的服务端口，默认为6379。
- <"host">：Redis实例的连接地址。

配置示例：

```
var redis = require("redis"),
client = redis.createClient(6379, "r-abcdefg.redis.rds.aliyuncs.com", {detect_buffers:
true});
client.auth("password", redis.print)
```

3. 执行上述代码完成连接。
4. 使用云数据库Redis。

```
// 写入数据
client.set("key", "OK");
// 获取数据，返回String
client.get("key", function (err, reply) {
console.log(reply.toString()); // print `OK`
});
// 如果传入一个Buffer，返回也是一个Buffer
client.get(new Buffer("key"), function (err, reply) {
console.log(reply.toString()); // print `<Buffer 4f 4b>`
});
client.quit();
```

## C#客户端StackExchange.Redis

**警告** 如果您的Redis实例为集群版，且需要执行切换或选择数据库的操作（即使用多数据库功能），您必须先将cluster_compat_enable参数设置为*0*（即关闭原生Redis Cluster语法兼容），然后重启客户端应用，否则将提示报错：`RedisCommandException: Multiple databases are not supported on this server; cannot switch to database: 1`。具体操作，请参见参数设置。

1. 下载并安装StackExchange.Redis。
2. 添加引用。

```
using StackExchange.Redis;
```

3. 初始化ConnectionMultiplexer。

   ConnectionMultiplexer是StackExchange.Redis的核心，它被整个应用程序共享和重用，应该设置为单例，它的初始化如下：

   **说明**
   - 关于如何获取Redis实例的连接地址和密码，请参见获取连接信息。
   - ConfigurationOptions包含很多选项，例如keepAlive、connectRetry、name，详情请参见ConfigurationOptions。

```
// redis config
private static ConfigurationOptions configurationOptions = ConfigurationOptions.Parse(
"127.0.0.1:6379,password=xxx,connectTimeout=2000");
 //the lock for singleton
private static readonly object Locker = new object();
 //singleton
private static ConnectionMultiplexer redisConn;
//singleton
public static ConnectionMultiplexer getRedisConn()
{
    if (redisConn == null)
    {
        lock (Locker)
        {
            if (redisConn == null || !redisConn.IsConnected)
            {
                redisConn = ConnectionMultiplexer.Connect(configurationOptions);
            }
        }
    }
    return redisConn;
}
```

4. 由于GetDatabase()返回的对象是轻量级的，每次用的时候从ConnectionMultiplexer对象中获取即可。

```
redisConn = getRedisConn();
var db = redisConn.GetDatabase();
```

5. 通过客户端程序操作Redis数据库。

> **说明** 下文列出常见数据接口的demo示例，和原生API略有不同。

   - String

```
//set get
string strKey = "hello";
string strValue = "world";
bool setResult = db.StringSet(strKey, strValue);
Console.WriteLine("set " + strKey + " " + strValue + ", result is " + setResult);
//incr
string counterKey = "counter";
long counterValue = db.StringIncrement(counterKey);
Console.WriteLine("incr " + counterKey + ", result is " + counterValue);
//expire
db.KeyExpire(strKey, new TimeSpan(0, 0, 5));
Thread.Sleep(5 * 1000);
Console.WriteLine("expire " + strKey + ", after 5 seconds, value is " + db.StringGet(
strKey));
//mset mget
KeyValuePair<RedisKey, RedisValue> kv1 = new KeyValuePair<RedisKey, RedisValue>("key1
", "value1");
KeyValuePair<RedisKey, RedisValue> kv2 = new KeyValuePair<RedisKey, RedisValue>("key2
", "value2");
db.StringSet(new KeyValuePair<RedisKey, RedisValue>[] {kv1,kv2});
RedisValue[] values = db.StringGet(new RedisKey[] {kv1.Key, kv2.Key});
Console.WriteLine("mget " + kv1.Key.ToString() + " " + kv2.Key.ToString() + ", result
is " + values[0] + "&&" + values[1]);
```

- Hash

```
string hashKey = "myhash";
//hset
db.HashSet(hashKey,"f1","v1");
db.HashSet(hashKey,"f2", "v2");
HashEntry[] values = db.HashGetAll(hashKey);
//hgetall
Console.Write("hgetall " + hashKey + ", result is");
for (int i = 0; i < values.Length;i++)
{
  HashEntry hashEntry = values[i];
  Console.Write(" " + hashEntry.Name.ToString() + " " + hashEntry.Value.ToString());
}
Console.WriteLine();
```

- List

```
//list key
string listKey = "myList";
//rpush
db.ListRightPush(listKey, "a");
db.ListRightPush(listKey, "b");
db.ListRightPush(listKey, "c");
//lrange
RedisValue[] values = db.ListRange(listKey, 0, -1);
Console.Write("lrange " + listKey + " 0 -1, result is ");
for (int i = 0; i < values.Length; i++)
{
 Console.Write(values[i] + " ");
}
Console.WriteLine();
```

- Set

```
//set key
string setKey = "mySet";
//sadd
db.SetAdd(setKey, "a");
db.SetAdd(setKey, "b");
db.SetAdd(setKey, "c");
//sismember
bool isContains = db.SetContains(setKey, "a");
Console.WriteLine("set " + setKey + " contains a is " + isContains );
```

- Sorted Set

```
string sortedSetKey = "myZset";
//sadd
db.SortedSetAdd(sortedSetKey, "xiaoming", 85);
db.SortedSetAdd(sortedSetKey, "xiaohong", 100);
db.SortedSetAdd(sortedSetKey, "xiaofei", 62);
db.SortedSetAdd(sortedSetKey, "xiaotang", 73);
//zrevrangebyscore
RedisValue[] names = db.SortedSetRangeByRank(sortedSetKey, 0, 2, Order.Ascending);
Console.Write("zrevrangebyscore " + sortedSetKey + " 0 2, result is ");
for (int i = 0; i < names.Length; i++)
{
  Console.Write(names[i] + " ");
}
Console.WriteLine();
```

# 4.5.2. 使用redis-cli连接

redis-cli是原生Redis自带的命令行工具，您可以使用redis-cli连接云数据库Redis，并进行数据管理。

## 前提条件

- ECS实例与Redis实例同属于经典网络或属于同一专有网络。
- 已将ECS实例的内网IP地址添加至Redis实例的白名单中。具体操作，请参见设置白名单。
- ECS实例的操作系统为Linux，且已安装原生Redis，安装方法，请参见Redis社区版官网。

## 操作步骤

1. 登录ECS设备的命令行，然后执行如下格式的命令连接至Redis实例：

```
src/redis-cli -h <hostname> -p <port>
```

### 参数说明

| 参数 | 获取方式 |
| --- | --- |
| <hostname> | Redis实例的内网连接地址，可在**实例信息**页面中的**连接信息**区域框中获取到。 |
| <port> | Redis实例的端口号，默认为6379。 |

命令示例

```
src/redis-cli -h r-bp1zxszhcgatnx****.redis.rds.aliyuncs.com -p 6379
```

2. 执行下述格式的命令完成密码验证：

```
AUTH <password>
```

<password>：账号的密码，在创建实例时已设置，如果忘记密码，可重置密码，具体操作，请参见修改密码。

命令示例：

```
AUTH testaccount:Rp829dlwa
```

验证成功后，返回如下结果：

```
OK
```

# 5.实例管理

## 5.1. 修改密码

如果您忘记密码、需要修改旧密码，或者在创建实例时没有设置密码，您可以重新设置实例的密码。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在**基本信息**页，单击右上角**修改密码**。
4. 在弹出的对话框中，输入旧密码和新密码。

   > **说明**
   > - 如果您忘记旧密码，在该对话框中单击**忘记旧密码**，然后设置新密码。
   > - 密码长度为8~32位。
   > - 密码需包含大写字母、小写字母、数字和特殊字符中的至少三种，支持的特殊字符为 `!@#$%^&*()_+-=` 。

5. 单击**确定**。

## 5.2. 设置白名单

为了数据库的安全稳定，在开始使用Redis实例前，您需要将访问数据库的IP地址或者IP段加到目标实例的白名单中。

### 背景信息

> **说明** 正确使用白名单可以让Redis得到高级别的访问安全保护，建议您定期维护白名单。

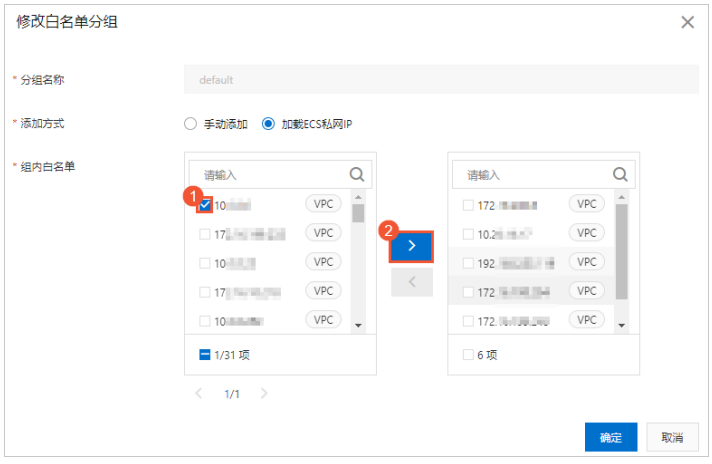
### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**白名单设置**。
4. 找到目标白名单分组，单击**修改**。

   > **说明** 您也可以单击**添加白名单分组**创建一个新的分组。分组名称长度为2~32个字符，由小写字母、数字或下划线组成，需以小写字母开头，以小写字母或数字结尾。

5. 在弹出的对话框中，根据要执行的操作，选择下述操作步骤：

| 要执行的操作 | 操作步骤 |
| --- | --- |

| 要执行的操作 | 操作步骤 |
| --- | --- |
| 手动修改白名单 | 手动填写IP地址或IP地址段。<br>手动修改白名单<br>  <br><br>**说明**<br>○ IP地址以英文逗号分隔，不可重复，最多1000个。支持格式为10.23.12.24（具体IP地址）、10.23.12.24/24（CIDR模式，即无类域间路由，/24表示地址中前缀的长度，范围为1~32）。<br>○ 地址中的前缀长度设置为0（例如0.0.0.0/0、127.0.0.1/0）表示允许所有IP地址访问该实例，存在高安全风险，请谨慎设置。 |
| 加载ECS实例的内网IP地址为白名单 | i. 单击**加载ECS私网IP**。<br>ii. 根据业务需求选中对应的IP地址。<br>加载ECS内网IP地址<br>  <br><br>**说明** 为便于定位目标IP地址，您可以将鼠标指针悬浮在对应的IP地址上，系统将展示该IP地址所属的ECS实例ID和名称。 |

| 要执行的操作 | 操作步骤 |
| --- | --- |
| 清除白名单分组 | 当某个白名单分组中的所有IP地址均需要移除且需要保留该分组时，您可以单击**清除白名单分组**来完成该操作。 |

6. 单击**确定**。

# 5.3. 变更配置

本文介绍如何变更Redis实例配置。

## 注意事项

变配操作完成后系统将进行数据迁移和切换操作，切换操作将出现秒级闪断，请您尽量在业务低峰执行本操作。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在跳转到的页面中，选择需要的配置。

| 配置 | 说明 |
| --- | --- |
| **架构类型** | 选择Redis实例的架构类型。<br>○ **标准版**：采用主从架构，不仅能提供高性能的缓存服务，还支持数据高可靠。<br>○ **集群版**：可轻松突破Redis自身单线程瓶颈，满足大容量、高性能的业务需求。<br>○ **读写分离**：可提供高可用、高性能、高灵活的读写分离服务，解决热点数据集中及高并发读取的业务需求，最大化地节约用户运维成本。 |
| **实例规格** | 选择实例的规格。<br>不同的实例规格提供的最大连接数和最大内网带宽不同。 |

4. 单击**提交**。

# 5.4. 设置可维护时间段

您可以修改默认的可维护时间段，使云数据库Redis实例在业务低峰期间进行维护。

## 背景信息

在阿里云平台上，为保障云数据库Redis实例的稳定性，后端系统会不定期对实例、机器进行维护操作。

实例维护当天，为保障整个维护过程的稳定性，实例会在所设置的可运维时间前一段时间，进入实例维护中的状态，当实例处于该状态时，数据库本身正常的数据访问不会受到任何影响，但控制台上涉及该实例的变更类的功能均暂无法使用（如变更配置），查询类如性能监控等可以正常查阅。

> **说明** 在进入实例所设置的可运维时间后，实例在维护期间可能会发生闪断，建议您尽量选择业务低峰期为运维时间段。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在**基本信息**区域框中，单击**可维护时间段**右侧的**设置**。
4. 选择可维护时间段，单击**保存**。

> **说明** 时间段以北京时间为基准。

# 5.5. 升级小版本

云数据库Redis版不断对内核进行深度优化，修复安全漏洞，提升服务稳定性。您可以在控制台上一键将内核升级至最新版本。

### 背景信息

> **说明**
> - 请在业务低峰期进行升级，并确保应用程序具备重连机制。
> - 系统会自动检测实例的内核版本，如果当前版本已经是最新版本，控制台**基本信息**页的**小版本升级**按钮将呈现为灰色状态。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在**实例信息**页，单击**基本信息**框右上角的**小版本升级**。
4. 在**小版本升级**对话框，单击**立即升级**。

   操作完成后，您可以在**基本信息**页中查看实例状态，实例的状态显示为小版本升级中。等状态变回使用中，即表示升级完成。

# 5.6. 设置SSL加密

为提高链路的安全性，您可以启用SSL（Secure Sockets Layer）加密，然后安装SSL CA证书到您的应用服务。SSL加密功能在传输层对网络连接进行加密，在提升通信数据安全性的同时，保证数据的完整性。

### 前提条件

- Redis实例为2.8版本（标准版或集群版）。
- Redis实例为4.0或5.0版本（集群版）。

### 背景信息

> **说明** 开启SSL会增加实例的网络响应时间，建议必要时才开启该功能。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**SSL设置**。
4. 在**SSL设置**页，单击页面右上角**设置SSL**。
5. 在弹出的对话框中，单击**开通SSL证书信息**右侧的滑块。

   > **说明** 如果提示版本不支持，请升级小版本。具体操作，请参见升级小版本。

6. 单击**确定**。

   > **说明**
   > - 完成操作后，需等待一段时间后，系统才能显示操作结果。
   > - 您还可通过右上方的**更新有效期**和**下载CA证书**按键执行相关的操作。

# 5.7. 开启透明数据加密TDE

云数据库Redis支持透明数据加密TDE（Transparent Data Encryption），可对RDB数据文件执行加密和解密。您可以通过控制台启用TDE功能，对RDB数据进行自动加密和解密，以满足提升数据安全性及合规需要。

## 前提条件

- Redis实例为企业版。
- Redis实例的小版本为最新，升级方案请参见升级小版本。

## 背景信息

云数据库Redis的TDE功能可以将RDB数据文件在写入磁盘之前进行加密，从磁盘读入内存时进行解密，具有不额外占用存储空间、无需更改客户端应用程序等优势。

TDE加密


## 影响

由于开启TDE功能后无法关闭，在开启前，需要评估对业务的影响，具体如下：

- 暂不支持缓存分析操作。
- 暂不支持通过DTS或Redis-shake执行迁移或同步数据。

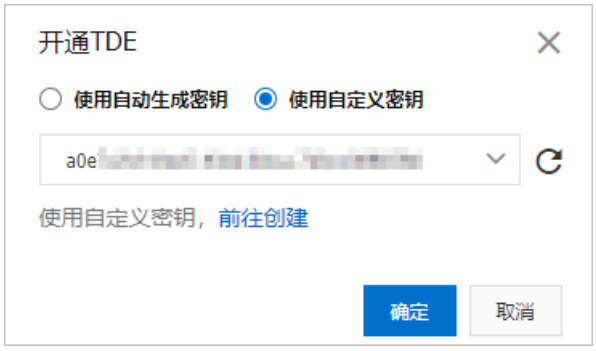
## 注意事项

- TDE的开启粒度为实例级别，不支持Key（键）或DB（库）粒度的控制。
- TDE加密对象为数据落盘文件（即RDB备份文件，如*dump.rdb*）。
- TDE所使用的密钥，由密钥管理服务KMS（Key Management Service）统一生成和管理，云数据库Redis不提供加密所需的密钥和证书。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**TDE**。
4. 打开**TDE状态**右侧的开关。
5. 在弹出的对话框中，选择**使用自动生成密钥**或**使用自定义密钥**，然后单击**确定**。

   开启TDE选择密钥




> **说明**
>
> - 如果您的阿里云账号首次为Redis实例开启TDE功能，请根据页面弹出的提示完成授权（授权的角色为AliyunRdsInstanceEncryptionDefaultRole），授权完成后才可以使用相关密钥服务。
> - 关于自定义密钥的创建方法，请参见*密钥管理服务*用户手册的**创建密钥**章节。

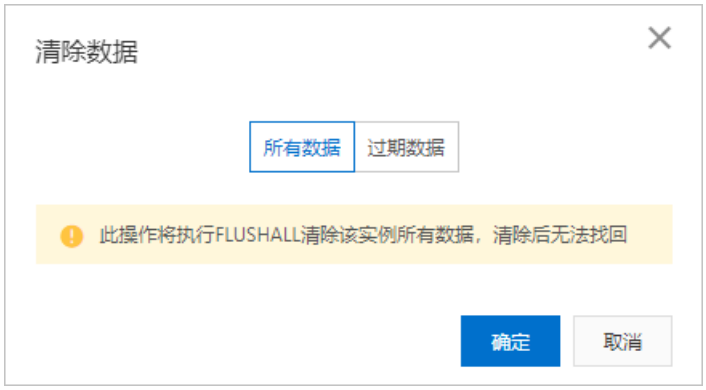
   设置完成后，实例状态改为**TDE修改中**，当实例状态转变为**运行中**表示操作完成。

# 5.8. 清除数据

您可以在控制台清除Redis实例中的数据，可选的清除对象为所有数据或过期数据。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在**实例信息**页，单击页面右上角**清除数据**。
4. 在弹出的对话框中，选择待清除的对象：




- **所有数据**：此操作将执行**FLUSHALL**命令清除该实例所有数据，清除后数据无法找回。
- **过期数据**：此操作将执行**SCAN**命令批量清除该实例的所有过期数据，清除后数据无法找回。您可选择**立即执行**或在可维护时间段执行。更多信息，请参见设置可维护时间段。

> **警告** 通过该功能清除的数据无法找回且即刻生效，同时会对线上业务造成影响，请谨慎执行。推荐在执行清除前对Redis实例的数据进行备份，具体操作，请参见手动备份。

5. 单击**确定**。

> **说明** 如选择为清除**所有数据**，单击**确定**后，您还可以选择是否备份数据。

# 5.9. 释放实例

根据业务需求，您可以随时释放Redis实例，本文介绍如何释放Redis实例。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，选择目标实例**操作**列的 > **释放**。

> **警告** 实例释放后不可恢复，请谨慎进行后续步骤。建议您在释放实例前先备份数据。

3. 在**释放实例**对话框中单击**确定**。

# 5.10. 账号管理

云数据库Redis版支持在一个实例中创建20个账号，您可以根据使用情况给这些账号设置相应的权限，从而更加灵活地管理实例，最大限度地避免误操作。

## 前提条件

实例的引擎版本为Redis 4.0或以上。

> **说明** 非Redis4.0版本实例只有默认账号，默认账号在创建实例时同时创建，如果需要修改默认账号密码，请参见修改密码。

## 背景信息

账号管理支持创建账号、删除账号、重置密码、修改权限等功能。创建账号后，当连接上数据库时，即可在命令行中使用账号及相应权限对数据库进行操作。

### 创建账号

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在**实例信息**页，单击左侧导航栏中**账号管理**。

> **说明** 若发现Redis 4.0或以上版本的实例左侧导航栏没有账号管理，请尝试升级小版本。

4. 在**账号管理**页，单击右侧的**创建账号**。
5. 在弹出的对话框中，进行如下设置并单击**确定**。

| 设置 | 说明 |
| --- | --- |
| **数据库账号** | 账号需以字母开头，由小写字母、数字、下划线组成，长度不超过16个字符。 |
| **权限设置** | 设置该账号所拥有的权限，分为只读、读写和复制三种。拥有复制权限的账号在连接Redis实例时可以使用SYNC/PSYNC命令。<br>**说明** 目前仅4.0以上的Redis标准版实例支持创建有复制权限的账号。 |
| **密码** | 设置该账号的密码。密码长度为8~32个字符，需至少包含大写字母、小写字母、数字和特殊字符中的三种。支持的特殊字符为：`!@#$%^&*()+-=_`。 |
| **确认密码** | 再次输入密码进行确认。 |
| **备注说明** | 填入账号的备注信息。 |

# 5.11. 重启实例

您可以在控制台的实例列表中重启Redis实例。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在目标实例的**操作**列，选择 > **重启实例**。

> **警告** 重启过程中实例会发生秒级闪断，请您在业务低峰期重启，并确保应用程序具备重连机制。

3. 在弹出的对话框中，选择重启时间并单击**确认**。
   - 立即重启：单击确认后实例重启。
   - 可维护时间窗重启：在设置的可维护时间段重启。

# 5.12. 导出实例列表

您可以在Redis控制台导出实例列表，在线下管理云上的实例资源。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击右上方的**导出实例列表**  图标。
3. 在**导出资源列表**对话框勾选需要导出的信息，之后单击**确认**。

> **说明** 单击**确认**后，浏览器开始下载.csv格式的实例列表，您可以用文本编辑工具查看该文件。

# 5.13. Lua脚本支持与限制

云数据库Redis版的各版本实例都支持Lua相关命令。

## Lua命令支持

Lua脚本的使用进一步提升了Redis的性能。通过内嵌对Lua环境的支持，Redis解决了长久以来不能高效地处理CAS（check-and-set）命令的缺点， 并且可以通过组合使用多个命令， 轻松实现以前很难实现或者不能高效实现的模式。

> **说明** 如果发现无法执行Eval相关命令，例如提示 `ERR command eval not support for normal user` ，请尝试升级小版本。升级过程中会出现短暂的闪断和实例只读，建议在业务低峰期进行。

## Lua使用限制

为了保证脚本里面的所有操作都在相同slot进行，云数据库Redis集群版本会对Lua脚本做如下限制：

- 所有key都应该由KEYS数组来传递，**redis.call/pcall**中调用的Redis命令，key的位置必须是KEYS array（不能使用Lua变量替换KEYS），否则直接返回错误信息：

  ```
  -ERR bad lua script for redis cluster, all the keys that the script uses should be passed using the KEYS array\r\n
  ```

- 所有key必须在一个slot上，否则返回错误信息：

  ```
  -ERR eval/evalsha command keys must be in same slot\r\n
  ```

- 调用必须要带有key，否则直接返回错误信息：

  ```
  -ERR for redis cluster, eval/evalsha number of keys can't be negative or zero\r\n
  ```

- 不支持发布订阅命令，包括**PSUBSCRIBE**、**PUBSUB**、**PUBLISH**、**PUNSUBSCRIBE**、**SUBSCRIBE**和**UNSUBSCRIBE**。
- 不支持UNPACK函数。

**说明** 如果用户能够在代码中确保所有操作都在相同slot，且希望打破Redis集群的Lua限制，可以在控制台将script_check_enable修改为0，则后端不会对脚本进行校验。

# 6.连接管理

## 6.1. 查看连接地址

您可以在Redis管理控制台上查看Redis实例的内网和外网连接地址。

### 背景信息

> 说明
> - Redis实例的VIP地址在服务维护或者变动时可能发生变化，建议在业务中使用连接地址访问Redis实例，确保连接的可用性。
> - 外网连接地址的申请方式参见申请外网连接地址。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在**连接信息**区域框，可查看到实例的私网连接地址和公网连接地址。

> 说明 Redis实例默认提供私网连接地址，如需公网连接地址，请手动申请。具体操作，请参见申请公网连接地址。

## 6.2. 申请公网连接地址

本文介绍如何申请公网连接地址。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在**连接信息**区域框中，单击**申请公网连接**。
4. 在弹出的对话框中，输入自定义的连接地址和端口，然后单击**确定**。

> 说明
> - 自定义连接地址由字母，数字组成，小写字母开头，长度为8~64个字符。
> - 自定义端口范围为1024~65535，默认为6379。
> - 申请公网地址后需要将公网访问的IP地址添加到白名单才可以访问Redis实例，添加白名单请参见设置白名单。

5. 在**实例信息**页，查看**连接信息**的**公网连接地址**。

> 说明 如果不再需要公网连接地址，单击**公网连接地址**右侧的**释放公网连接**按钮释放该地址。

# 6.3. 修改连接地址

云数据库Redis版支持修改实例的内网和外网连接地址。更换Redis实例时，您可以将新实例的连接地址修改为原实例的连接地址，无需对应用程序进行其它修改。

## 前提条件

实例状态为运行中。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在**实例信息**页，单击**连接信息**右侧的**修改连接地址**。
4. 在**修改连接地址**对话框，配置以下信息。

| 配置 | 说明 |
| --- | --- |
| 连接类型 | 选择**私网地址**和**公网地址**。 |
| 连接地址 | 设置连接地址前缀。<br>○ 由小写字母、数字组成。<br>○ 以小写字母开头。<br>○ 长度为8~64个字符。 |
| 端口 | 设置端口，可设置范围为1024~65535。<br>**说明** 公网地址的端口号修改生效大约需要10分钟，请生效后刷新页面查看最新端口号信息。 |

5. 在**修改连接地址**对话框，选择**连接类型**，修改**连接地址**的前缀和**端口**，之后单击**确定**。

> **说明**
> ○ 自定义前缀需由小写英文字母和数字组成，以小写字母开头，长度为8~64字符。
> ○ 自定义端口范围为1024~65535，默认为6379。

# 7.性能监控

## 7.1. 查看监控数据

您可以查询过去一个月内指定时间段的Redis监控数据。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**性能监控**。
4. 选择查询的起止时间，然后单击**确定**。

> **说明** 监控指标请参见监控指标说明。

## 7.2. 自定义监控项

您可以根据需要选择显示在Redis控制台性能监控页面的监控指标。

### 背景信息

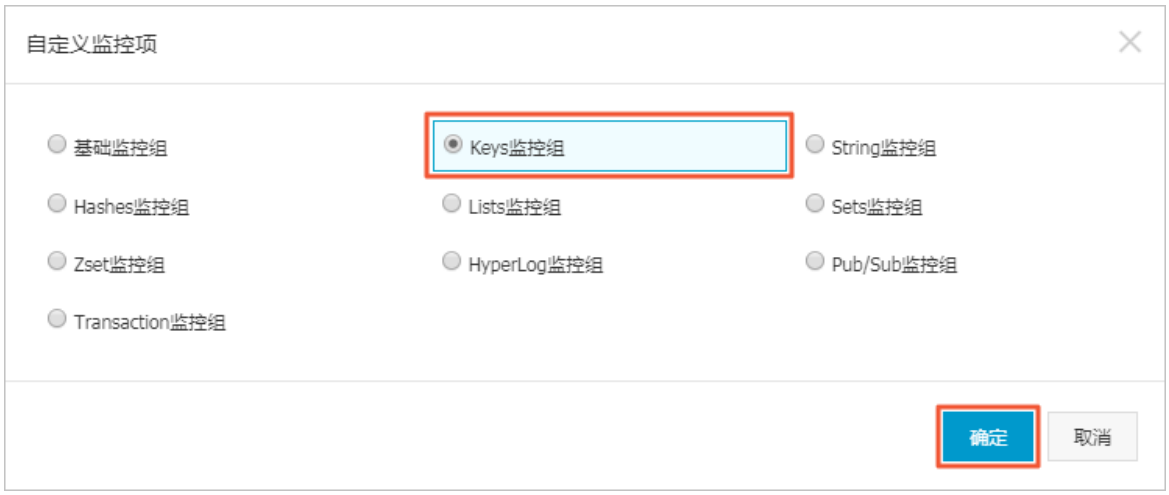
云数据库Redis版支持十余组监控指标，性能页默认显示基础监控组的监控指标，您可以通过**自定义监控项**按钮切换为显示其它监控组的指标。各监控组的说明如下：

| 监控组 | 说明 |
| --- | --- |
| 基础监控组 | 实例信息等基本监控信息，包含QPS、带宽及内存使用情况等。 |
| Keys监控组 | 使用键值相关命令的监控信息，例如DEL、EXITS等命令的调用次数。 |
| String监控组 | 使用string数据类型相关命令的监控信息，例如APPEND、MGET等命令的调用次数。 |
| Hashes监控组 | 使用hash数据类型相关命令的监控信息，例如HGET、HDEL等命令的调用次数。 |
| Lists监控组 | 使用set数据类型相关命令的监控信息，例如BLPOP、BRPOP等命令的调用次数。 |
| Sets监控组 | 使用list数据类型相关命令的监控信息，例如SADD、SCARD等命令的调用次数。 |
| Zset监控组 | 使用zset数据类型相关命令的监控信息，例如ZADD、ZCARD等命令的调用次数。 |
| HyperLog监控组 | 使用HyperLogLog数据类型相关命令的监控信息，例如PFADD、PFCOUNT等命令的调用次数。 |
| Pub/Sub监控组 | 使用发布与订阅（Pub/Sub）功能相关命令的监控信息，例如PUBLISH、SUBSCRIBE等命令的调用次数。 |
| Transaction监控组 | 使用事务（transaction）相关命令的监控信息，例如WATCH、MULTI、EXEC等命令的调用次数。 |
| Lua脚本监控组 | 使用Lua脚本相关命令的监控信息，例如EVAL、SCRIPT等命令的调用次数。 |

如需了解各监控组中监控指标的作用，请参见监控指标说明。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**性能监控**。
4. 单击**数据指标**栏末尾的**自定义监控项**。
5. 在弹出的对话框中，选择需要的监控组并单击**确定**。




监控界面将会显示该监控组的指标。

# 7.3. 修改监控频率

您可以在控制台调整云数据库Redis版采集监控数据的频率，即监控频率。

### 背景信息

云数据库Redis版提供两种监控频率供您选择：5秒/次和60秒/次。默认的监控频率为60秒/次，可满足常规监控需求，如果需要对某些指标进行高频率、低延迟的观察，您可以按照本文的步骤将监控频率修改为5秒/次。监控数据不占用实例存储空间，监控数据的采集不影响实例的正常运行。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**性能监控**。
4. 在**性能监控**页，单击右上方**监控频率**。
5. 在**监控频率**对话框，选择需要的监控频率，单击**确定**。

# 7.4. 监控指标说明

云数据库Redis版对十余组指标进行实时监控，帮助您掌握Redis服务的运行状况。您可以在本文中了解每个监控指标的作用。

### 基础监控组指标说明

| 监控指标 | 单位 | 说明 | 统计方法 |
| --- | --- | --- | --- |
| CpuUsage | % | CPU使用率 | 采集时的CPU使用率 |
| UsedMemory | Bytes | 已使用的内存 | 采集时的内存使用量 |
| TotalQps | Counts/s | 实例每秒接收的请求次数 | 监控周期内请求数÷监控周期 |
| ConnCount | Counts | 连接数 | 采集时的连接数 |
| InFlow | KBps | 实例每秒接收的数据量 | 监控周期内接收数据的总量÷监控周期 |
| OutFlow | KBps | 实例每秒发送的数据量 | 监控周期内发送数据的总量÷监控周期 |
| FailedCount | Counts/s | 平均每秒产生的异常请求数 | 监控周期内总的异常请求数÷监控周期 |
| AvgRt | us | 所有请求的平均响应时间<br>说明 详细说明请参见响应时间（RT）指标。 | 监控周期内所有请求的处理时间÷监控周期内的请求个数 |
| MaxRt | us | 请求的最大响应时间<br>说明 详细说明请参见响应时间（RT）指标。 | 监控周期处理单个请求的最长耗时 |
| Keys | Counts | key的总数 | 采集时key的个数 |
| Expires | Counts | 设置了过期时间的key总数 | 采集时设置了过期时间的key总数 |
| ExpiredKeys | Counts | 已过期的key总数 | 截止到采集时的累计值，实例重启后重新计算。 |
| EvictedKeys | Counts | 因内存满而淘汰的key总数 | 截止到采集时的累计值，实例重启后重新计算。 |
| request | Bytes | 一个监控周期内Redis节点接收到的请求数据的总量 | 参见左侧说明 |
| response | Bytes | 一个监控周期内Redis节点发出的响应数据的总量 | 参见左侧说明 |
| request_max | Bytes | 一个监控周期内单次请求数据大小的最大值 | 参见左侧说明 |
| response_max | Bytes | 一个监控周期内单次响应数据大小的最大值 | 参见左侧说明 |
| traffic_control_input | Counts | 触发下行流量控制的次数 | 采集周期内的累计值 |

| 监控指标 | 单位 | 说明 | 统计方法 |
| --- | --- | --- | --- |
| traffic_control_output | Counts | 触发上行流量控制的次数 | 采集周期内的累计值 |
| traffic_control_input_status | Counts | 监控周期内是否触发过下行流量控制，0表示未触发，1表示曾经触发。 | 参见左侧说明 |
| traffic_control_output_status | Counts | 监控周期内是否触发过上行流量控制，0表示未触发，1表示曾经触发。 | 参见左侧说明 |
| hit_rate | % | 请求的命中率，即尝试访问数据时，数据存在于Redis实例中的概率。 | 监控周期内命中的请求在总请求数中所占的百分比 |
| hit | Counts | 命中的请求数 | 监控周期内命中的请求数 |
| miss | Counts | 未命中的请求数 | 监控周期内未命中的请求数 |
| evicted_keys_per_sec | Counts/s | 每秒淘汰key的个数 | 监控周期内淘汰key总数÷监控周期 |

## 其它监控组指标说明

除基础监控组外，其它监控组包含的都是有关特定类型的数据或者特定功能的监控指标，这些监控组中的指标分为如下两类：

- 反映命令使用次数的指标。例如Keys监控组中的del、dump、exists指标监控的是DEL、DUMP、EXISTS命令的使用次数。
- 命令响应时间（RT）指标。例如Keys监控组中的del_avg_rt、dump_avg_rt、exists_avg_rt等以avg_rt结尾的指标监控的是DEL、DUMP、EXISTS命令在一个监控时间段中的平均响应时间。

## 响应时间（RT）指标

所有监控组中都有关于响应时间的指标，这些指标一般以Rt或者rt结尾，例如基础监控组中的AvgRt、MaxRt或者Keys监控组的del_avg_rt、exists_avg_rt。

基础监控组中的AvgRt和MaxRt在所有响应时间类指标中最为常用。这两个指标在proxy节点和数据节点中代表的意义不同，详细说明如下。

- 集群版或读写分离版实例中，proxy节点的AvgRt指标反映了proxy节点处理所有命令平均消耗的时间，处理过程包含以下阶段：
  i. proxy节点接收到请求，转发给数据节点。
  ii. 数据节点处理请求并回复proxy节点。
  iii. proxy节点转发出处理结果。

  proxy的AvgRt时间包含了数据节点处理命令的时间、等待时间以及proxy节点和数据节点之间的网络消耗时间。
- 在集群版与读写分离版实例的数据节点或者标准版实例中，AvgRt指标反映了数据节点处理所有命令平均消耗的时间，即从数据节点接收命令到返回处理结果所花费时间。该时间与proxy节点的行为以及网络耗时无关。
- MaxRt代表请求的最大响应时间，在各版本Redis实例中的统计方式与上述AvgRt的统计方式相似。

# 8.参数设置

云数据库Redis版允许用户自定义部分实例参数。通过本文，您可以了解相关参数的说明以及在Redis控制台修改各参数值的通用方法。

## 注意事项

- 为最大程度保障Redis实例的稳定运行，目前仅开放部分参数，如果某个参数未在本文中列出，即不支持设置该参数。
- 部分参数在提交修改后会自动重启实例（重启过程中实例会发生秒级闪断）。在控制台设置参数时，您需要关注目标参数的**重启生效**列。

## 参数说明

## 参数说明

| 参数 | 说明 |
| --- | --- |
| #no_loose_check-whitelist-always | 开启VPC免密访问后默认不检查客户端IP是否在Redis白名单中，此时如果将该参数的值设置为yes，则白名单在VPC免密模式下依然生效。可选值：<br>• yes，表示开启。<br>• no，默认值，表示关闭。 |
| #no_loose_disabled-commands | 设置禁用命令，多个命令通过英文逗号（,）隔开，目前可禁用的命令有：**FLUSHALL**、**FLUSHDB**、**KEYS**、**HGETALL**、**EVAL**、**EVALSHA**、**SCRIPT**。 |
| #no_loose_ssl-enabled | 开启或关闭SSL加密连接，可选值：<br>• yes，表示开启。<br>• no，默认值，表示关闭。 |
| #no_loose_sentinel-enabled | 开启或关闭哨兵（Sentinel）兼容模式，可选值：<br>• yes，表示开启。<br>• no，默认值，表示关闭。 |
| client-output-buffer-limit pubsub | 限制对发布订阅客户端的输出缓冲，参数值格式为 `<hard limit> <soft limit> <soft seconds>` 。<br>• hard limit：当某客户端的输出缓冲区占用内存达到或超过hard limit的限制时，断开该客户端的连接。hard limit值的单位为Byte。<br>• soft limit和soft seconds：当某客户端的输出缓冲区占用内存达到或超过soft limit的限制，且该状态持续时间大于等于soft seconds限定的秒数，断开该客户端的连接。soft limit值的单位为Byte，soft seconds值的单位为s。 |
| dynamic-hz | 开启或关闭动态hz，可选值：<br>• yes，默认值，表示开启。<br>• no，表示关闭。 |

| 参数 | 说明 |
| --- | --- |
| hash-max-ziplist-entries | 哈希对象同时满足以下两个条件时， 使用ziplist编码。<br>1. 哈希对象保存的所有键值对的键和值的字符串长度的字节数都小于hash-max-ziplist-value的值。<br>2. 哈希对象保存的键值对数量小于hash-max-ziplist-entries的值。 |
| hash-max-ziplist-value | 哈希对象同时满足以下两个条件时， 使用ziplist编码。<br>1. 哈希对象保存的所有键值对的键和值的字符串长度的字节数都小于hash-max-ziplist-value的值。<br>2. 哈希对象保存的键值对数量小于hash-max-ziplist-entries的值。 |
| hz | 设置Redis后台任务执行频率，例如清除过期键任务。取值范围为1~500，默认值为10，即每秒执行10次。该值越大，CPU资源消耗越多，但在过期键较多的情况下清理频率也更高，同时Redis能够更精确地处理超时。建议取值不要超过100。 |
| lazyfree-lazy-eviction | 是否开启基于lazyfree的驱逐功能，可选值：<br>• yes，表示开启。<br>• no，默认值，表示不开启。 |
| lazyfree-lazy-expire | 是否开启基于lazyfree的过期key删除功能，可选值：<br>• yes，默认值，表示开启。<br>• no，表示不开启。 |
| lazyfree-lazy-server-del | DEL命令是否基于lazyfree异步删除数据，可选值：<br>• yes，默认值，表示开启。<br>• no，表示不开启。 |
| list-compress-depth | 列表中两端不被压缩的节点个数，取值范围0~65535。<br>• 0：默认值，表示都不压缩。<br>• 1：表示list两端各有1个节点不压缩，中间的节点压缩。<br>• 2：表示list两端各有2个节点不压缩，中间的节点压缩。<br>• 3：表示list两端各有3个节点不压缩，中间的节点压缩。<br>• 其后依此类推。 |

| 参数 | 说明 |
| --- | --- |
| list-max-ziplist-size | • 取正值表示按照数据项个数来限定每个quicklist节点上的ziplist长度。例如，当该参数配置为5时，每个quicklist节点的ziplist最多包含5个数据项。<br>• 取负值表示按照占用字节数来限定每个quicklist节点上的ziplist长度。此时，该值只能取-1到-5这五个值，每个值含义如下。<br>  ◦ -5：每个quicklist节点上的ziplist大小不能超过64Kb（注：1kb = 1024 bytes）。<br>  ◦ -4：每个quicklist节点上的ziplist大小不能超过32Kb。<br>  ◦ -3：每个quicklist节点上的ziplist大小不能超过16Kb。<br>  ◦ -2：每个quicklist节点上的ziplist大小不能超过8Kb（Redis默认值）。<br>  ◦ -1：每个quicklist节点上的ziplist大小不能超过4 Kb。 |
| maxmemory-policy | 设置缓存满后Redis删除内容的策略，您可以在如下八种策略中进行选择。LRU表示最近最少使用的。LFU表示最不常用的。LRU，LFU和volatile-ttl都是使用近似随机算法实现的。<br>• volatile-lru：只从设置失效（expire set）的key中选择最近最少使用的key进行删除。<br>• allkeys-lru：优先删除掉最近最少使用的key。<br>• volatile-lfu：只从设置失效（expire set）的key中选择最不常用的key进行删除。<br>• allkeys-lfu：优先删除掉最不常用的key。<br>• volatile-random：只从设置失效（expire set）的key中，随机选择一些key进行删除。<br>• allkeys-random：随机选择一些key进行删除。<br>• volatile-ttl：只从设置失效（expire set）的key中，选出存活时间（TTL）最短的key进行删除。<br>• noeviction：不删除任何key，只是在写操作时返回错误。 |
| notify-keyspace-events | notify-keyspace-events的参数可以是以下字符的任意组合，它指定了服务器该发送哪些类型的通知。<br>• 字符：发送的通知。<br>• K：键空间通知，所有通知以__keyspace@<db>__为前缀。<br>• E：键事件通知，所有通知以__keyevent@<db>__为前缀。<br>• g：DEL、EXPIRE、RENAME等类型无关的通用命令的通知。<br>• l：列表命令的通知。<br>• s：集合命令的通知。<br>• h：哈希命令的通知。<br>• z：有序集合命令的通知。<br>• x：过期事件。每当有过期键被删除时发送。<br>• e：驱逐（evict）事件。每当有键因为maxmemory政策而被删除时发送。<br>• A：参数g$lshzxe的别名。 |

| 参数 | 说明 |
| --- | --- |
| set-max-intset-entries | 当Set集合内的数据符合以下条件时，会使用intset编码。<br>1. 当集合内所有数据都是字符对象。<br>2. 都是基数为10的整数，范围为64位有符号整数。 |
| slowlog-log-slower-than | 设置是否记录慢查询日志：<br>• 当取值为负时，不记录任何操作。<br>• 当取值为0时，记录所有操作。<br>• 当取值为正时，只有当操作执行时间大于设置值，操作才被记录。<br>单位为微秒（ms），取值范围0~10000000，默认值10000。 |
| slowlog-max-len | 慢日志最多保存记录条数，取值范围100~10000，默认值1024。 |
| stream-node-max-bytes | Stream中每个宏节点（Macro Node）能够占用的最大内存，取值范围：0~999,999,999,999,999。0表示无限制。 |
| stream-node-max-entries | Stream中每个宏节点中可存储条目的最大数量，取值范围：0~999,999,999,999,999。0表示无限制。 |
| timeout | 在客户端连接空闲时长达到指定值时关闭连接。单位为秒（s），取值范围：0~100000。0表示不开启。 |
| zset-max-ziplist-entries | 排序集合对象同时满足以下两个条件时， 使用ziplist编码。<br>1. 排序集合对象保存的所有键值对的键和值的字符串长度的字节数都小于zset-max-ziplist-value的值。<br>2. 排序集合对象保存的键值对数量小于zset-max-ziplist-entries的值。 |
| zset-max-ziplist-value | 排序集合对象同时满足以下两个条件时， 使用ziplist编码。<br>1. 排序集合对象保存的所有键值对的键和值的字符串长度的字节数都小于zset-max-ziplist-value的值。<br>2. 排序集合对象保存的键值对数量小于zset-max-ziplist-entries的值。 |
| list-max-ziplist-entries | 链表对象同时满足以下两个条件时， 使用ziplist编码。<br>1. 链表对象保存的所有元素的字符串长度的字节数都小于list-max-ziplist-value的值。<br>2. 链表集合对象保存的元素数量小于list-max-ziplist-entries的值。 |
| list-max-ziplist-value | 链表对象同时满足以下两个条件时， 使用ziplist编码。<br>1. 链表对象保存的所有元素的字符串长度的字节数都小于list-max-ziplist-value的值。<br>2. 链表集合对象保存的元素数量小于list-max-ziplist-entries的值。 |

| 参数 | 说明 |
| --- | --- |
| cluster_compat_enable | 开启或关闭原生Redis cluster语法兼容，可选值：<br>• 0，表示关闭。<br>• 1，默认值，表示开启。 |
| script_check_enable | 检查Lua脚本涉及的key是否在相同slot，可选值：<br>• 0，表示不检查。<br>• 1，默认值，表示检查。 |

**说明** 用于调整Redis数据节点最大连接数的maxclients参数不支持自定义，默认值为10000。

## 在Redis管理控制台设置参数

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**参数设置**。
4. 单击目标参数**操作**列的**修改**。
5. 在弹出的对话框中，修改参数的值并单击**确定**。

# 9.备份与恢复

## 9.1. 自动备份

由于越来越多的应用将Redis作为持久化存储，所以需要常规的备份机制保证数据误操作之后，具备数据快速恢复的能力。阿里云采用在备节点上执行RDB快照备份，备份期间对您的实例访问不会产生性能影响，并且提供了控制台的快捷操作可以让您进行个性化的备份设置。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**备份与恢复**。
4. 单击**备份设置**页签。
5. 单击**编辑**，设置备份周期和备份时间段。
   - **保留天数**：备份文件的保留天数固定为7天，不可变更。
   - **备份周期**：可以设置为一星期中的某一天或者某几天，默认为每天备份一次。
   - **备份时间**：可以设置为任意时段，以小时为单位，建议设置为业务的低峰期。
6. 单击**确定**。

## 9.2. 手动备份

您可以在控制台上随时发起手动备份。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**备份与恢复**。
4. 单击页面右上角的**创建备份**。
5. 单击**确定**。

> **说明** 您可以在**备份数据**页签中，查看到已完成的备份数据（备份数据的保留时间固定为7天）。

## 9.3. 下载备份

如果您需要更长时间的数据存档，可以直接在控制台上复制链接将数据库备份文件自行下载到本地进行长时间存储。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**备份与恢复**。

4. 找到目标备份文件，单击其**操作**列的**下载**。

> **说明** 当Redis实例为集群架构时，为保障数据一致性，您需要下载同一时间点中该实例每个数据分片的备份文件。

# 9.4. 数据恢复

云数据库Redis支持将指定备份集的数据恢复至当前实例。

## 前提条件

Redis实例为主从或集群架构。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**备份与恢复**。
4. 根据实例的架构，选择下述操作步骤：
   - 主从架构：找到目标备份集，单击其**操作**列的**数据恢复**。
   - 集群架构：选择同一时间点中，所有数据分片对应的备份集，然后单击右上角的**数据恢复**。

> **警告** 由于数据恢复操作有较高的风险，请充分验证数据正确性之后，再进行数据恢复操作。

5. 在弹出的对话框中，阅读提示并单击**继续执行恢复操作**。

   您也可以执行克隆实例操作，将备份数据恢复到新创建的实例上，具体操作，请参见克隆实例。

# 9.5. 克隆实例

云数据库Redis支持从指定的备份集创建新实例，新实例中的数据将和该备份集中的数据一致，可用于数据恢复、快速部署业务或数据验证等场景。

## 前提条件

Redis实例为主从或集群架构。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**备份与恢复**。
4. 找到目标备份集，单击其**操作**列的**克隆实例**。

> **说明** 实例为集群架构时，您需要选择同一时间点中，所有数据分片对应的备份集。

5. 在弹出的对话框中，单击**确定**。
6. 在跳转到的页面中，设置新实例的相关配置并单击**提交**。

> **说明** 新实例的相关配置说明，请参见创建实例。

# 10.CloudDBA

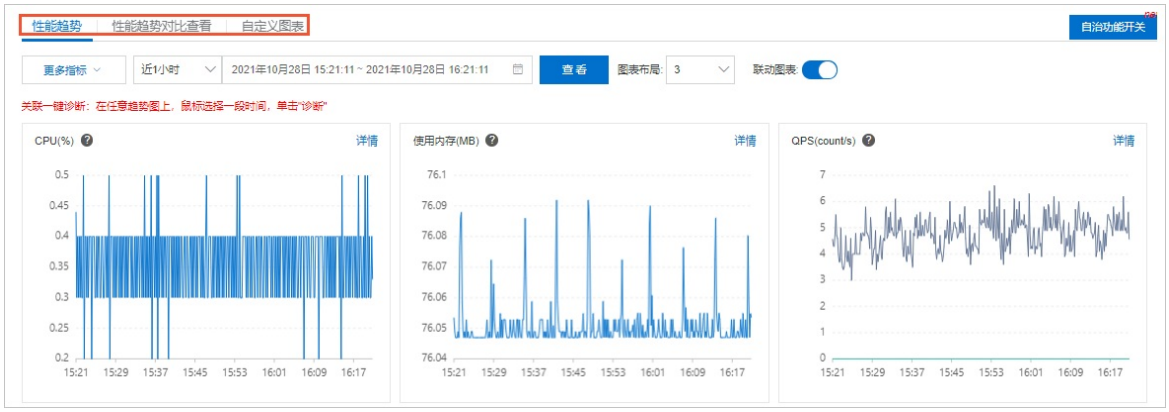
## 10.1. 性能趋势

CloudDBA的性能趋势功能可以监控Redis实例在某个时间段的基础性能及其运行趋势，包括CPU使用率、使用内存量、QPS（每秒访问次数）、总连接数、响应时间、网络流量、Key命中信息等。

### 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**CloudDBA > 性能趋势**。
4. 性能趋势目前支持如下查看方式，您可以根据需求选择：

> **说明** 如果实例为集群版，本页面还会展示节点列表信息（展示近1小时性能数据），单击节点ID可展示指定节点的详细数据。




- **性能趋势**

  单击**性能趋势**页签，设置时间段和更多指标，然后单击**查看**。

  > **说明**
  > - **联动图表**默认开启，如果您将光标悬置于CPU图表上来查看Redis实例9点时的CPU性能指标，其他多个图表也会显示该实例在9点时的性能指标数据。
  > - 单击图表右上方的**含义？**和**详情**，查看性能指标含义和性能趋势大图。

- **性能趋势对比查看**

  单击**性能趋势对比查看**页签，设置任意两个时间段，配置更多指标，然后单击**查看**，对比两个时间段的性能趋势。

- **自定义图表**

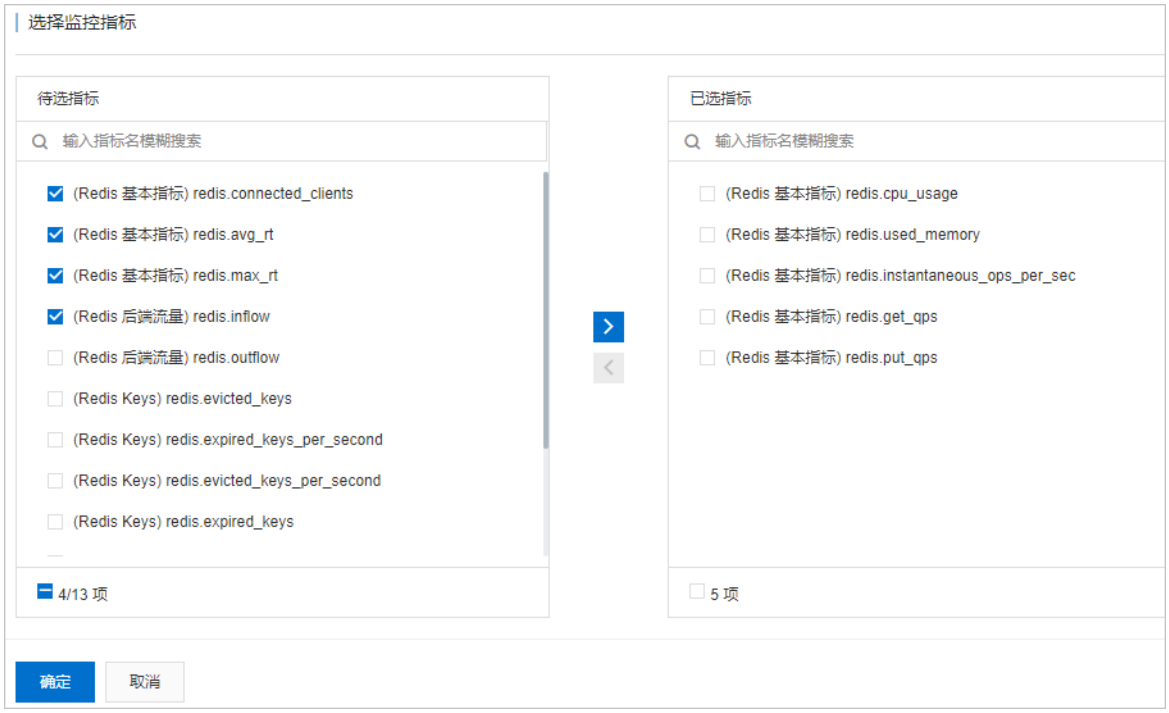
  以上两种查看方式显示了Redis实例的基础性能指标，如仅需展现其中部分基础指标，您可以自定义性能趋势图表。详情请参见自定义性能趋势图表。

## 10.2. 自定义性能趋势图表

默认性能趋势显示了Redis实例的基础性能指标，您也可以在自定义性能趋势图表中，选择仅展示部分基础性能指标，以便有针对性地监控和分析实例的性能及运行趋势。本文档介绍了如何在监控大盘中自定义创建Redis实例的性能趋势图表。

## 操作步骤

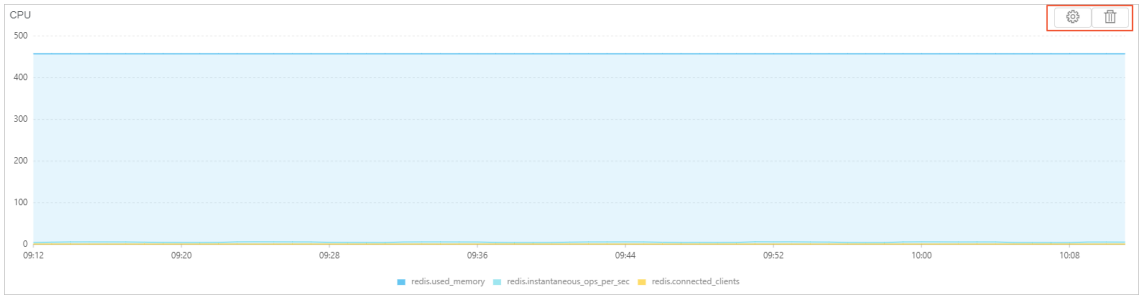
1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**CloudDBA > 性能趋势**。
4. 单击**自定义图表**页签。
5. 单击**新增监控大盘**，填写大盘名称并单击**确认**。
6. 单击**+添加图表**或**新增监控图表**。如下图所示，选中需要查看的指标，单击**确定**。




7. （可选）通过查看、修改和删除操作，管理监控大盘。
   - 查看监控大盘

     选择目标监控大盘并设置时间段，然后单击**查看**。
   - 修改监控大盘

     单击如下所示图标，根据需求进行修改或删除图表。




   - 删除监控大盘

单击**操作大盘 > 删除监控大盘**。

# 10.3. 实时性能

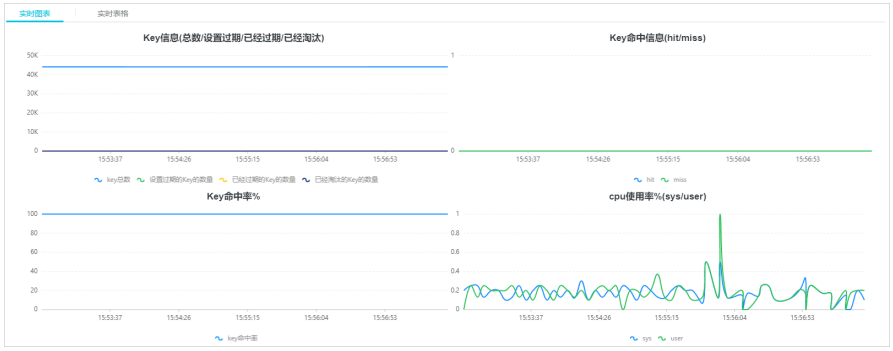
在CloudDBA中可以实时查看Redis实例的性能，包括CPU使用率、内存使用信息、QPS（每秒访问次数）、网络流量、Server信息、Key信息、Client信息、连接信息等。

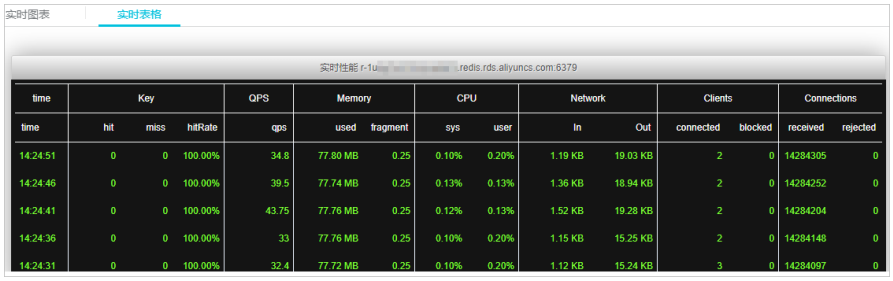
## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**CloudDBA > 实时性能**。
4. 根据业务需求，选择实时性能的展示方式：

   页面上方实时显示实例性能指标，包括Server信息、Key信息、内存信息、Client信息、连接信息；页面下方以**实时图表**和**实时表格**的形式显示详细的性能指标。


> **说明** 为方便您实时查看性能的变化，性能指标的值每5秒自动刷新一次，剩余刷新次数可在右上角显示。您也可以单击**暂停**停止刷新。

| 展示方式 | 介绍 |
| --- | --- |
| **实时图表** | 以曲线图的形式显示实例的实时性能指标，如Key信息、Key命中信息、Key命中率、CPU使用率、内存信息、QPS、网络流量。  |

| 展示方式 | 介绍 |
| --- | --- |
| 实时表格 | 以表格的形式显示实例的Key命中信息、Key命中率、QPS、Memory（内存信息）、CPU使用率、Network（网络流量）、Client信息、连接信息等性能。实时的性能值每5秒新增一行，最多显示999行。  |

实时图表 实时表格

实时性能 r-1u[redacted].redis.rds.aliyuncs.com:6379

| time | Key | | | QPS | Memory | | CPU | | Network | | Clients | | Connections | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| time | hit | miss | hitRate | qps | used | fragment | sys | user | In | Out | connected | blocked | received | rejected |
| 14:24:51 | 0 | 0 | 100.00% | 34.8 | 77.80 MB | 0.25 | 0.10% | 0.20% | 1.19 KB | 19.03 KB | 2 | 0 | 14284305 | 0 |
| 14:24:46 | 0 | 0 | 100.00% | 39.5 | 77.74 MB | 0.25 | 0.13% | 0.13% | 1.36 KB | 18.94 KB | 2 | 0 | 14284252 | 0 |
| 14:24:41 | 0 | 0 | 100.00% | 43.75 | 77.76 MB | 0.25 | 0.12% | 0.13% | 1.52 KB | 19.28 KB | 2 | 0 | 14284204 | 0 |
| 14:24:36 | 0 | 0 | 100.00% | 33 | 77.76 MB | 0.25 | 0.10% | 0.20% | 1.15 KB | 15.25 KB | 2 | 0 | 14284148 | 0 |
| 14:24:31 | 0 | 0 | 100.00% | 32.4 | 77.72 MB | 0.25 | 0.10% | 0.20% | 1.12 KB | 15.24 KB | 3 | 0 | 14284097 | 0 |

# 10.4. 实例会话

通过实例会话，您可以实时查看Redis实例与客户端间的会话信息，包括客户端信息，所执行的命令和已连接的时长等，还可以根据业务需求终止异常会话。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**CloudDBA > 实例会话**。
4. 根据业务需求，选择下述步骤对实例会话进行管理：
   - 查看会话：页面默认展示了所有会话的详细信息，您可以将鼠标放置在对应的参数名称上查看其含义。

     > **说明**
     > - 您可以在**搜索会话**文本框中输入关键字来过滤会话信息。
     > - 如需刷新实例会话信息，可以单击左上角的**刷新**或者开启**自动刷新**（每30秒自动刷新一次）。

   - 终止会话：选择目标会话或按shif键可以选择多个目标会话，单击右上角的**kill选中会话**或者直接点击**kill全部会话**，可以终止选中或全部会话。

     > **警告** 请勿终止系统级的会话，否则可能引发一些不可预料的结果。

   - 查看会话统计：会话统计记录了实例会话的总客户端数，活跃客户端数，来源（客户端的地址）数量。

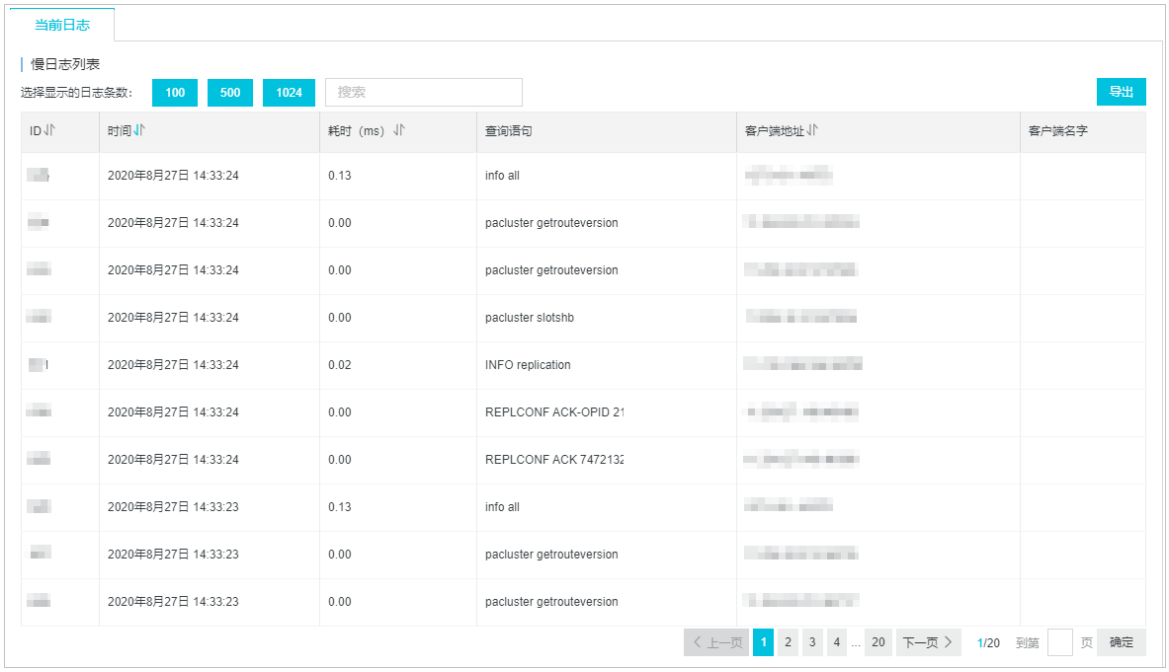
     > **说明** 在**按照来源统计**区域框的表格中，单击客户端地址右侧的编辑图标即可修改来源的别名。在总数栏中单击数值，可查看访问来源地址的详细信息。

# 10.5. 慢请求

慢请求问题会极大地影响Redis服务的稳定性。您可在CloudDBA中查看慢日志详细信息，来监控和分析慢请求问题。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**CloudDBA > 慢请求**。
4. 查看实例的慢请求日志信息。




> **说明** 您可以选择要展示的日志条数或在搜索文本框中输入关键字进行过滤。

# 10.6. 缓存分析

通过缓存分析功能来分析Redis的备份文件，可以快速发现实例中的BigKey（大Key），帮助您掌握Key在内存中的占用和分布、Key过期时间等信息，为您的优化操作提供数据支持，帮助您避免因Key倾斜引发的内存不足、性能下降等问题。

## 操作步骤

1. 登录云数据库Redis控制台。
2. 在**实例列表**页，单击目标实例ID。
3. 在左侧导航栏，单击**CloudDBA > 缓存分析**。
   **缓存分析**页签默认展示最近一天缓存分析结果列表，您可以根据需求选择其他时间段。
4. 在**缓存分析**页签，单击页面右侧的**立即分析**。


5. 在弹出的对话框中，设置分析的节点和方式。


| 参数 | 说明 |
| --- | --- |
| **选择分析节点** | 选择需要执行缓存分析的节点ID。<br>**说明** 您可以选择分析整个实例，也可以只选中某个节点进行分析。当选择分析整个实例时，若该实例的节点数超过8个，系统仅对内存使用量前8的节点进行分析。 |
| **分析方式** | 您可以选择分析方式为**使用上一个备份文件**或**新建备份，并使用最新的备份进行分析**。 |

6. 单击**确定**。
   系统执行分析并展示分析状态，您可以单击**刷新**以更新分析状态。
7. 找到已完成的分析任务，单击其**操作**列的**详情**展示详细的分析结果。
   - **基本信息**：展示实例基本属性和缓存分析方法等信息。


   - **相关节点**：展示实例内各节点的内存情况和Key统计信息。


> **说明** 当实例为集群，且选择的分析节点为整个实例时，**详情**页才会展示**相关节点**信息并提供节点选择的功能。

- **详情**：展示实例或节点的Key内存占有情况、Key数量分布情况、Key中元素的内存占用和分布情况、Key过期时间分布、大Key排名等信息。