ALIBABA CLOUD

Alibaba Cloud

ApsaraDB for Redis
Product Introduction

Document Version: 20201014

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.What is ApsaraDB for Redis?

ApsaraDB for Redis is a database service that is compatible with native Redis protocols. It supports a hybrid of memory and hard disks for storage. ApsaraDB for Redis provides a highly available hot standby architecture, and can scale to meet requirements for high-performance and low-latency read/write operations.

Benefits

- Hardware and data are deployed in the cloud. ApsaraDB for Redis is a fully-managed cloud database service provided by Alibaba Cloud. Alibaba Cloud manages infrastructure planning, network security and system maintenance. This allows you to focus on business development.
- ApsaraDB for Redis supports various data types, such as strings, lists, sets, sorted sets, hash tables, and streams. The service also provides advanced features, such as transactions, message subscription, and message publishing.
- ApsaraDB for Redis Enhanced Edition (Tair) is a key-value pair cloud cache service that is an upgraded version of ApsaraDB for Redis Community Edition. ApsaraDB for Redis Enhanced Edition (Tair) supports two series of instances: performance-enhanced instances and hybrid-storage instances.

For more information, see Comparison between ApsaraDB for Redis and on-premises Redis and Scenarios.

Editions

ApsaraDB for Redis provides Community Edition and Enhanced Edition.

Edition	Description
ApsaraDB for Redis Community Edition	ApsaraDB for Redis Community Edition is compatible with the data cache service of native Redis engines. It supports master-replica instances, cluster instances, and read/write splitting instances.
ApsaraDB for Redis Enhanced Edition (Tair)	ApsaraDB for Redis Enhanced Edition (Tair) is developed based on Community Edition. It supports the following instance types: • Performance Enhanced instances: Performance-enhanced instances provide a multi-threading model and integrate with some features from Tair of Alibaba Group. These instances are fully compatible with native Redis databases and suitable for some scenarios. • Hybrid-storage instances: Hybrid-storage instances store all data in disks and hot data in memory. This ensures high read/write performance. ApsaraDB for Redis Enhanced Edition (Tair) also uses Redis Database (RDB) and Append Only File (AOF) to perform high speed read/write operations and meet data persistence requirements.

Architectures

ApsaraDB for Redis supports multiple deployment architectures that are applicable in different scenarios.

Architecture type	Description
Standard master-replica instances	The system synchronizes data between the master node and replica node in real time. If the master node fails, the system automatically fails over to the replica node and restores services in seconds. This process is transparent to users and does not affect your business. The master-replica architecture ensures high availability of system services.
Cluster master-replica instances	Cluster instances use a distributed architecture. Each shard works as a master-replica HA node and provides automatic disaster recovery and failover. Multiple cluster specifications are available. You can select a specification based on your business requirements. The cluster architecture supports the following connection modes: • Proxy mode is the default endpoint of a cluster instance. You can configure automatic connection to reduce application development costs. • Direct connect mode allows the client to bypass the proxy server and directly access backend shards to reduce network overhead and service response time. This mode is suitable for business scenarios that are latency-sensitive.
Read/write splitting instances	A read/write splitting instance uses a master-replica architecture to provide high availability. Read replicas are attached to the primary node to facilitate data replication and implement linear scaling of read performance. Read replicas can alleviate performance issues caused by hot keys. Read/write splitting instances are suitable for business scenarios that require high read/write ratios. Read/write splitting instances are classified into non-cluster instances and cluster instances. • A non-cluster read/write splitting instance can have one, three, or five read replicas. • For a read/write splitting cluster instance, a read replica is attached to each shard to achieve automatic read/write splitting on individual shards. Read/write splitting cluster instances are suitable for large-scale business scenarios that require high read/write ratios.

Instance types

ApsaraDB for Redis Community Edition and Enhanced Edition provide different instance types for each architecture. For more information, see Overview.

2.Overview and selection of ApsaraDB for Redis

3.ApsaraDB for Redis Enhanced Edition (Tair)

3.1. Overview

ApsaraDB for Redis Enhanced Edition (Tair) is a key-value pair cloud cache service based on the Tair used by Alibaba Group. ApsaraDB for Redis Enhanced Edition (Tair) has officially handled the cache business of Alibaba Group since 2009 and has proven its outstanding performance in scenarios such as Double 11, Youku Spring Festival Gala, Cainiao, and AMAP.

Emergence of Tair

In 2004, Taobao began using caching technology to support its business operations. This is the first time that Frontend page caching was used. This technology uses ESI to identify Web content segments that can or cannot be accelerated. Frontend page caching provides effective methods to reduce the number of entire pages that are captured from the server.

The rapid growth of network traffic to Taobao has exerted more pressure on databases. To reduce the load on databases, backend caching becomes an effective countermeasure. Backend caching evolved over many iterations from Tbstore to TDBM. Tbstore provided services such as Taobao details and verification codes. TDBM was initially used in Taobao User Center. In 2009, the large-scale and high-speed Tair was released. This solution is developed based on the success of previous systems and technical experience.

Milestones of Tair

Date	Event
2009.04	Tair 1.0 was released and used in services such as the Taobao core system, MDB cache, and User Center.
2009.11	The first year that Tair started to provide support for high traffic during Double 11.
2011.06	The LDB persistence engine was released to meet key-value storage requirements on the Internet.
2012.10	The RDB cache engine was released with APIs that are similar to that used in Redis. This provides support for more flexible and complex data structures.
2013.04	The Fastdump service was released. It can reduce import time and access latency. Tair was implemented on a large scale in Alimama.
2014.05	After OCS was released, Tair became one of basic Alibaba Cloud products and started to provide services to Memcache users.
2015.03	Tair entered the cloud era with the release of KVStore.
2016.08	Tair Smart O&M Platform was released. The release of the platform contributed to a significant boost in sales during 2016 Double 11.

Date	Event
2017.04	Tair 2.0 was released and started to provide services to the AMAP and Youku business units. OCS was upgraded to KVStore.
2017.11	Tair dynamic hashing delivered successful support for 2017 Double 11 and was able to fix several cache hotspot issues within the industry.
2018.08	KVStore started to offer hybrid storage instances to separate cold data from hot data and reduce costs for key customers.
2019.04	The KVStore team was among the top three contributors in the Redis open source community and delivered a public speech at RedisConf 2019.
2019.11	Tair 3.0, or ApsaraDB for Redis Enterprise Edition, was officially released.

Tair types and features

As a distributed NoSQL database with high availability and performance, Tair focuses on caching and high-speed storage for multiple data structures. It is fully compatible with the Redis protocol. Tair is one of the most popular systems of Alibaba Group and has provided core access acceleration during the Double 11 for many years. Tair can handle hundreds of millions of calls per second. Compared to ApsaraDB for Redis Community Edition, Tair provides higher performance, more data structures, and more storage methods. For more information, see the following table.

Tair type	Feature
Performance-enhanced series	• Performance-enhanced series uses multithreading and provides about three times the performance of ApsaraDB for Redis Community Edition instances of the same specifications. • Performance-enhanced series provides multiple enhanced data structure modules such as TairString (including CAS and CAD commands), TairHash, TairGIS, TairBloom, and TairDoc. Performance-enhanced series can improve the efficiency of business development. You no longer need to be concerned about storage structures and timeliness.
Hybrid-storage series	Hybrid-storage series adopts the memory and disk storage method. During off-peak hours, this method can separate hot data from cold data to ensure a high memory access speed. Hybrid-storage series also balances performance and costs by providing a larger storage capacity than ApsaraDB for Redis Community Edition.

3.2. Enhanced multi-threading performance

Compared with ApsaraDB for Redis Community Edition, performance-enhanced instances of ApsaraDB for Redis Enhanced Edition (Tair) provide more benefits. For example, these instances provide enhanced multi-threading performance and integrate multiple Redis modules. This topic describes enhanced multi-threading performance of ApsaraDB for Redis Enhanced Edition (Tair).

> **Note** For more information about Redis modules, see Integration of multiple Redis modules.

Benefits

- Supports full compatibility with native Redis databases. You do not need to modify application code when you manage on-premises Redis databases by using ApsaraDB for Redis.
- Provides read/write performance three times that of native Redis databases or ApsaraDB for Redis Community Edition when the same specifications are used. This eliminates the performance limits on high-frequency read/write requests for hot data.
- Requires less response time to process a large number of queries per second (QPS) when compared with native Redis databases.
- Ensures stable performance in high-concurrency scenarios and eliminates connection problems caused by a sudden rise in the number of requests during peak hours.
- Runs full and incremental synchronization tasks in input and output (I/O) threads to speed up the synchronization.
- Supports standard, cluster, and read/write splitting architectures.

How performance-enhanced instances work

In a process of handling requests, native Redis databases and ApsaraDB for Redis Community Edition must go through the following steps: read requests, parse requests, process data, and then send responses. During the process, network I/O operations and request parsing consume most resources. Performance-enhanced instances use multiple threads to process the tasks in these steps in parallel.

- I/O threads are used to read requests, send responses, and parse commands.
- Worker threads are used to process commands and timer events.
- Auxiliary threads are used to monitor the statuses of nodes and heartbeats.

The following figures show the differences between a single-threading model and a multi-threading model.

Single-threading model of ApsaraDB for Redis

Multi-threading model of ApsaraDB for Redis

ApsaraDB for Redis reads and parses requests in I/O threads, and sends commands through the queues of the parsed requests to worker threads. Afterward, worker threads run the commands to process the requests and send the responses to I/O threads through other queues.

Performance-enhanced instances support a maximum of four parallel I/O threads. To improve the multi-threading performance, unlocked queues and pipelines are used to transmit data between I/O threads and worker threads.

Performance comparison

ApsaraDB for Redis Community Edition and native Redis databases support the single-threading model. The query rate on each data shard reaches 80,000 QPS to 100,000 QPS. Performance-enhanced instances support the multi-threading model. In this model, I/O threads, worker threads, and auxiliary threads process requests in parallel. Each data shard of ApsaraDB for Redis Enhanced Edition (Tair) provides three times the performance of each data shard of ApsaraDB for Redis Community Edition. The performance is compared in details as follows:

- ApsaraDB for Redis Community Edition
  - Standard instances of ApsaraDB for Redis Community Edition are not suitable for processing more than 100,000 QPS on a single data shard.
  - A cluster instance of ApsaraDB for Redis Community Edition contains multiple data shards. Each data shard provides performance similar to that of a standard instance. If one of the data shards stores hot data and receives a large number of concurrent requests for hot data, the access to other data of this data shard may be delayed, which results in a performance bottleneck.
  - Read/write splitting instances of ApsaraDB for Redis Community Edition provide high read performance. Instances of this type have outstanding performance in scenarios where more reads are requested than writes. These instances cannot support a large number of concurrent writes.
- Performance-enhanced standard instances of ApsaraDB for Redis Enhanced Edition (Tair)
  - Performance-enhanced standard instances are suitable for processing more than 100,000 QPS on a single data shard.
  - Performance-enhanced cluster instances can provide high performance to read and write hot data and reduce maintenance costs.
  - Performance-enhanced read/write splitting instances can provide high read performance and process a large number of concurrent writes. Instances of this type are suitable for processing a large number of reads and writes, where more reads are requested than writes.

Scenarios

- Live video streaming
- Flash sales promotion systems
- Online education

Performance-enhanced instances can support a great number of concurrent reads and writes and process large amounts of hot data. You can use ApsaraDB for Redis Enhanced Edition (Tair) if you require higher performance than ApsaraDB for Redis Community Edition. The following section describes some typical scenarios where performance-enhanced instances are used.

- Scenario 1: A flash sales promotion system may process 200,000 QPS or higher for some cached hot keys. Standard master-replica instances of ApsaraDB for Redis Community Edition cannot provide high performance during peak hours. However, standard master-replica instances of ApsaraDB for Redis Enhanced Edition (Tair) can process requests for popular commodities and provide excellent user experience. The performance bottleneck is resolved.
- Scenario 2: Cluster instances of ApsaraDB for Redis Community Edition have limits on database transactions and Lua scripts. However, performance-enhanced cluster instances of ApsaraDB for Redis Enhanced Edition (Tair) can provide required performance and eliminate the limits on cluster commands.

- Scenario 3: If you build an on-premises Redis service with one master node and multiple replica nodes, the number of replica nodes increases when your workloads increase. This raises your management and maintenance costs. However, performance-enhanced read/write splitting instances of ApsaraDB for Redis Enhanced Edition (Tair) can provide one data shard and up to five read replicas. This architecture helps you process one million or more QPS.
- Scenario 4: If you build an on-premises Redis cluster to process ten million or more QPS, the number of shards increases when your workloads increase. This raises your management and maintenance costs. However, performance-enhanced cluster instances of ApsaraDB for Redis Enhanced Edition (Tair) can serve your workloads and reduce the cluster size by two thirds. This instance type is a cost-effective choice.

Specifications

For more information about the specifications of performance-enhanced instances, see Overview.

Purchase and activate an instance

For more information, see Step 1: Create an ApsaraDB for Redis instance.

3.3. Hybrid-storage instances

ApsaraDB for Redis Enhanced Edition (Tair) supports two instance types: performance-enhanced instances and hybrid-storage instances. Hybrid-storage instances store data in memory and disks whereas in-memory instances store all data in memory. Hybrid-storage instances combine the benefits of memory and disks to provide high read/write performance and persistent data storage.

Introduction

Hybrid-storage instances of ApsaraDB for Redis Enhanced Edition (Tair), which are independently developed by Alibaba Cloud, are compatible with the Redis protocol. These instances store all data in disks and hot data in memory to ensure high read/write performance. Hybrid-storage instances deliver a cost-effective solution to provide high-performance queries for frequently accessed data without the limit of memory capacity.

- In-memory storage: Keys and values of hot data are stored in memory. The information about keys is also cached in memory. This allows you to check whether a specified key exists.
- Disk storage: All keys and values are stored in disks. Redis data structures such as Hash are also stored in disks in a certain format.

  Architecture

  

Scenarios

- Live video streaming

  Live video streams generate a large amount of hot data. This type of data is from popular live streaming rooms. You can use hybrid-storage instances to store data from popular live streams in memory and data from less popular live streams in disks. This allows you to make full use of the limited memory capacity.

- E-commerce

E-commerce applications generate a large amount of data. You can use hybrid-storage instances to store data and optimize memory usage. The data of popular items is stored in memory and data of less popular items is stored in disks.

- Online education

  Online education applications generate a large amount of data. Such data includes online courses, question libraries, and messages between teachers and students. Only popular courses and the latest question libraries are frequently queried. You can use hybrid-storage instances to store the data of online courses in disks, and store the data of popular courses and question libraries in memory. ApsaraDB for Redis provides a cost-effective solution to ensure high read/write performance for frequently queried data.

The following examples show the benefits of hybrid-storage instances:

- Example 1: A native Redis cluster is used to store 100 GB of data. The query rate is less than 20,000 queries per second (QPS) at peak hours and 80% of the data is infrequently accessed. In this case, you can use a hybrid-storage instance of ApsaraDB for Redis Enhanced Edition (Tair) that has 32 GB memory and 128 GB disk capacity. This decreases memory usage by about 70 GB and reduces storage costs by more than 50%.
- Example 2: An on-premises Pika instance is used to reduce the storage costs of a native Redis deployment. The total size of the data is about 400 GB and only about 10% of the data is frequently accessed. High costs are incurred for cluster operations and maintenance (O&M). In this case, you can use a hybrid-storage instance of ApsaraDB for Redis Enhanced Edition (Tair) that has 64 GB memory and 512 GB disk capacity. This minimizes your O&M investment and ensures high availability.

Select instances based on scenarios

Hybrid-storage instances of ApsaraDB for Redis Enhanced Edition (Tair) and cache-only instances of ApsaraDB for Redis Community Edition are applicable to different scenarios, as shown in the following table.

Scenario	Hybrid-storage instance of ApsaraDB for Redis Enhanced Edition (Tair)	ApsaraDB for Redis Community Edition instance
Data size and budget	• You need to store a large amount of data at a lower cost. • You use Pika, SSDB, or Antibiotic Resistance Genes Database (ARDB) database services. • You use master-replica or cluster instances of ApsaraDB for Redis to store a large amount of data.	You need to store a small amount of data, or your budget is sufficient for storage costs.
Separation of hot and cold data	Your business data is divided into hot data and cold data. Note Cold data is the data that is infrequently accessed.	Your business data is accessed at random.

Scenario	Hybrid-storage instance of ApsaraDB for Redis Enhanced Edition (Tair)	ApsaraDB for Redis Community Edition instance
QPS performance and latency requirements for hot and cold data	• High QPS performance is required for hot data. • The access to cold data is not affected by latency.	• High QPS performance is required for all data. • The access to all data is affected by latency.
Access frequency of big keys	• No big key exists in your business data. • Big keys are frequently accessed and need to be stored in memory. • Big keys are infrequently accessed and are not affected by latency.	Big keys are irregularly accessed and are highly sensitive to latency.

Select instances based on specifications

When you create a hybrid-storage instance of ApsaraDB for Redis Enhanced Edition (Tair), you must specify the memory and disk capacity. The memory capacity determines the maximum amount of hot data that can be stored and the disk capacity determines the total amount of data that can be stored. ApsaraDB for Redis allocates CPU resources to the instance based on the specified memory and disk capacity. For example, if you specify a **master-replica instance that has 64 GB memory and 256 GB disk capacity**, the instance can store up to 256 GB of data and can cache 64 GB of data in memory.

For more information, see Overview.

> **Note**
> - You cannot customize the number of CPU cores.
> - Hybrid-storage instances store the metadata of keys in memory and disks. The metadata includes the time-to-live (TTL) value, the least recently used (LRU) clock value, and the type of each key. When you specify the memory and disk capacity, you must reserve the storage capacity for metadata. For more information, see the following description in this topic.

- Memory capacity

  To support more native Redis features, hybrid-storage instances store all keys in memory. These instances store the values of frequently accessed keys in memory and the values of infrequently accessed keys in disks. Therefore, make sure that the specified memory capacity is sufficient to store all keys and related metadata. The following table lists the recommended memory capacity based on the total number of keys.

Total number of keys	Recommended memory capacity
Less than 20 million	64 GB, 32 GB, and 16 GB
20 million to 50 million	64 GB and 32 GB

Total number of keys	Recommended memory capacity
50 million to 100 million	128 GB, 64 GB, and 32 GB
More than 100 million	128 GB and 64 GB

> **Note** The memory capacity determines the CPU resources that are allocated to the instance when you create an instance. A higher memory capacity indicates higher performance of the instance.

- Disk capacity

  Hybrid storage instances store all data in disks, including the metadata generated for each key. The metadata occupies extra storage. Therefore, we recommend that you select a disk capacity that is 20% to 50% more than the required storage.

Purchase an instance

For more information, see Step 1: Create an ApsaraDB for Redis instance.

Instance performance

The performance of a hybrid-storage instance depends on the instance type and memory hit ratio. This hit ratio indicates the probability that the requested data is found in memory. Higher specifications and a higher memory hit ratio indicate higher performance. If all requested data is found in the memory of the hybrid-storage instance, the performance is the same as that of a high-performance in-memory instance. A lower memory hit ratio indicates lower performance of the hybrid-storage instance. If all requested data is found in disks, the hybrid-storage instance delivers the lowest performance.

In the following performance tests, the ApsaraDB for Redis instance stores 20 million keys and the size of each value is 1 KB. The **GET** command is used to read values. Different types of keys are accessed in the following scenarios.

Test results

Test scenario	QPS of a Community Edition instance	QPS of a hybrid-storage instance
Keys are accessed at random.	123,000	15,000
Based on the Gaussian distribution, 20% of the keys are accessed at a probability of 80%.	120,000	54,000
Based on the Gaussian distribution, 1% of the keys are accessed at a probability of 99%.	135,000	114,000

Command limits

Hybrid-storage instances of ApsaraDB for Redis Enhanced Edition (Tair) support most native Redis commands. To ensure high performance, hybrid-storage instances impose limits on the use of some commands. For more information, see Compatibility in commands.

FAQ

- What is the threshold of hot data in memory? What may happen after the threshold is exceeded?

  The threshold of hot data in memory is 90% of the memory capacity of the instance. For example, for a hybrid-storage instance with a memory capacity of 16 GB, if the amount of used memory exceeds 14.4 GB, ApsaraDB for Redis evicts values of cold data from memory. However, values of cold data are still stored in disks.

- What is the granularity of data eviction if the amount of used memory exceeds the threshold?

  Keys are the smallest unit for eviction. For example, if the memory usage of a hash exceeds the threshold, the hash table is evicted.

- How is a key-value pair stored in memory?

  Compared with instances of Community Edition, hybrid-storage instances store additional metadata. The metadata is used to record whether the values of keys are cached in memory, and check whether the data is hot or cold when it is evicted.

- How is data written to disks?

  To avoid direct access to disks, data is first modified in memory, which is similar to page cache. Then, background threads continuously synchronize data modifications to the disks.

  > **Note** Hot data can be directly modified in memory. Cold data must be loaded to memory before modification.

- How does a hybrid-storage instance work if the data to be accessed is not cached in memory?

  ApsaraDB for Redis checks whether the data to be accessed is cached in memory before it runs a command. If the data is not cached in memory, ApsaraDB for Redis caches the data from disks to memory before it runs the command.

  > **Note** It takes longer to load data with more complex data structures from disks. We recommend that you evaluate access models and data models to avoid frequent requests to cold data.

- Why does a write timeout occur?

  The write timeout may occur due to the following causes:

  - When the cold data is requested, it takes a long time to cache data from disks to the memory.
  - A large number of concurrent write requests are processed. After the data is modified in memory, background threads synchronize the modification to disks. However, the write speed of disks is less than that of the memory. When the data is written to memory at a much faster rate than to disks, a write speed limit must be set on the memory to prevent write timeouts.

Other FAQ

- An `OOM error` occurred when I wrote data to a hybrid-storage instance. All memory is consumed, but the disks still have some remaining capacity. How can I resolve this issue?

  The memory capacity is insufficient to store all keys and related metadata. Therefore, the memory capacity must be increased. For more information, see Change specifications.

- Can I change configurations from hybrid-storage instances to Community Edition instances or performance-enhanced instances.

  No, you cannot change the configurations from hybrid-storage instances to Community Edition instances or performance-enhanced instances.

- How do I migrate data from hybrid-storage instances to Community Edition instances or performance-enhanced instances?

  You can use the redis-shake tool to migrate full data. For more information, see Use redis-shake to migrate data.

  > **Note**
  > - Hybrid storage instances use different encoding formats and replication protocols from Community Edition instances. You can use only the rump or sync mode to migrate full data. The migration of incremental data is not supported.
  > - You cannot use Data Transmission Service (DTS) to migrate data.

- How do I export data from a hybrid storage instance to my local device?

  You can use the dump mode of the redis-shake tool to export Redis Database Backup (RDB) files. If you use the Community Edition replication protocol to connect to a hybrid-storage instance, the hybrid-storage instance converts data to the RDB format and sends the data to the redis-shake tool. For more information, see Use the redis-shake tool to back up data.

- Can I mount a Community Edition instance or a performance-enhanced instance of Enhanced Edition (Tair) to a hybrid-storage instance as a secondary instance?

  Yes, you can perform this operation. This operation is applicable only in specific scenarios because hybrid-storage instances support only full data synchronization.

4.Architectures

4.1. Overview

ApsaraDB for Redis supports the standard, cluster, and read/write splitting architectures. You can select the instances suitable for your business requirements.

Architectures

The following table lists the supported architectures. You can click the architecture name to view more details.

Warning

Architecture	Edition	Description	Scenario
Standard master-replica instances	• Enhanced Edition • Community Edition	Master-replica architecture to ensure high availability	• Support for more native Redis features. • Persistent storage on ApsaraDB for Redis instances. • Stable query rate on a single node of ApsaraDB for Redis. • Use of simple Redis commands, when only a few sorting and computing commands are used.
Cluster master-replica instances	• Enhanced Edition • Community Edition	Uses a cluster architecture with multiple data shards. Each data shard runs in the master-replica mode and supports connections through proxy servers and internal endpoints.	• Large data volume. • High queries per second (QPS). • Throughput-intensive workloads.
Read/write splitting instances	• Enhanced Edition • Community Edition	Consists of multiple proxy servers, one master node, one replica node, and one or more read replicas.	• High QPS. • Support for more native Redis features. • Persistent storage on ApsaraDB for Redis instances.

Documentation applicability

You must understand the following concepts in ApsaraDB for Redis: architectures, editions such as Community Edition and Enhanced Edition, series types such as performance-enhanced instances, and engine versions such as Redis 4.0 or 5.0. The descriptions and topics listed in the preceding table are applicable to related editions, series types, and engine versions. For standard instances, you can view architecture information in Standard master-replica instances. This rule also applies to cluster instances and read/write splitting instances.

4.2. Standard master-replica instances

Standard master-replica instances of ApsaraDB for Redis provide high-performance caching services and support high data reliability.

Overview

Standard master-replica instances of ApsaraDB for Redis run in a master-replica architecture. The master node serves your workloads and the replica node stays in a hot standby state to ensure high availability. If the master node fails, the system switches the workloads to the replica node within 30 seconds after the failure occurs. This mechanism guarantees the high availability for your workloads.

Features

- Reliability
  - Service reliability

    Standard master-replica instances adopt a master-replica architecture, with master and replica nodes deployed on different physical machines. The master node serves your workloads. You can use the Redis command-line interface (CLI) and common clients to add, delete, modify, and query data on the master node. Alibaba Cloud has developed a high-availability (HA) system for standard master-replica instances. If the master node fails, the HA system performs a failover to guarantee the high availability for your workloads.
  - Data reliability

    By default, data persistence is enabled for standard master-replica instances. Instances support data backup. You can clone or roll back an instance based on a specified backup set to restore data after misoperations. Instances created in zones that provide disaster recovery, such as Hangzhou Zones H and I, also support zone-disaster recovery.
- Compatibility

  Standard master-replica instances of ApsaraDB for Redis are developed based on Redis 2.8 and compatible with all Redis commands. You can migrate your workloads on an on-premises Redis database to a standard master-replica instance of ApsaraDB for Redis without service disruption. Alibaba Cloud also provides Data Transmission Service (DTS) to support incremental migration to ApsaraDB for Redis. This ensures a stable transition for your business.
- Proprietary systems developed by Alibaba Cloud
  - HA system

    ApsaraDB for Redis uses the HA system to detect failures on the master node, such as disk input and output (I/O) failures and CPU failures, and performs failovers to ensure high availability.

- Master-replica replication mechanism

  Alibaba Cloud has customized the master-replica replication mechanism of ApsaraDB for Redis. You can replicate data in the format of incremental logs between the master node and the replica node. If the replication is interrupted, system performance and stability is unchanged and free from issues caused by the master-replica replication mechanism of native Redis databases.

  Some issues caused by the master-replica replication mechanism of native Redis databases are described as follows:

  - If the replication is interrupted, the replica node runs the Partial Resynchronization (PSYNC) command to resynchronize partial data. During this process, the resynchronization fails. Then, the master node synchronizes all RDB files to the replica node.
  - To synchronize all RDB files, the master node must perform full replication first due to the single-threading model. As a result, the master node has a latency of several milliseconds or seconds.
  - Child processes are created to perform copy-on-write (COW) tasks. The child processes consume memory on the master node. The master node may run out of memory and cause the application to exit abnormally.
  - The replica files that the master node generates consume disk I/O and CPU resources.
  - The replication of GB-level files may lead to outbound traffic bursts on the server and increase the sequential I/O throughput of disks. This delays responses and causes more issues.

Scenarios

- Support for more native Redis features

  Standard instances are compatible with all Redis commands. You can migrate your workloads to instances without service disruption.

- Persistent storage on an ApsaraDB for Redis instance

  Standard instances support data persistence, backup, and recovery features to ensure data reliability.

- Stable query rate on a single node of ApsaraDB for Redis.

  Due to the single-threading model of native Redis databases, we recommend that you use a standard instance if your workloads support a query rate lower than 100,000 QPS. To achieve higher performance, select a cluster instance.

- Use of simple Redis commands, where only a few sorting and computing commands are used

  CPU performance is the main bottleneck due to the single-threading model of native Redis databases. We recommend that you use a cluster instance of ApsaraDB for Redis to process a large number of sorting and computing workloads.

4.3. Cluster master-replica instances

Cluster master-replica instances of ApsaraDB for Redis can resolve the performance bottleneck that is caused by a single-threading model. You can use the instances to process large-capacity or high-performance workload. Cluster master-replica instances can run in proxy mode or direct connect mode. You can select one based on your business requirements.

Proxy mode

By default, cluster instances use the proxy mode. In this mode, you can access ApsaraDB for Redis clusters by using a unified endpoint (domain). Proxy server forwards requests from clients to each data shard. Proxy servers, data shards, and config servers do not provide separate endpoints. This simplifies application development and code. The following sections describe the architecture and components in proxy mode.

Architecture of an ApsaraDB for Redis cluster master-replica instance in proxy mode

Components of an ApsaraDB for Redis cluster master-replica instance in proxy mode

Component	Description
Proxy server	Each proxy server has a single node configured. A cluster instance contains multiple proxy servers. The system implements load balancing and failover for the proxy servers.
Data shard	Each data shard runs in a high-availability master-replica architecture. If the master node fails, the system switches the workload to the replica node to ensure high availability.
Config server	A config server stores configuration data and sharding policies, and runs in a high-availability master-replica architecture.

The number and specifications for each instance components depend on the specifications that you select when you create a cluster instance. You can use Change specifications to modify the size of the cluster instance or select another architecture type. For more information about architecture types, see Overview.

Direct connect mode

The proxy mode simplifies business development but slightly deteriorates the service response speed, because all requests are forwarded by proxy servers. If you require high response speed, use the direct connect mode. In this mode, you can directly connect to backend data shards without the need to use proxy servers. This reduces network overheads and service response time. The following sections describe the architecture and components in direct connect mode.

Architecture of an ApsaraDB for Redis cluster master-replica instance in direct connect mode

To use the direct connect mode, you must apply for a private endpoint to obtain a connection string. Then, you can connect to the cluster in the same way you connect to a native Redis cluster. When a client connects to the cluster for the first time, DNS resolves the connection string to the virtual IP address of a random shard. Then, the client can use the Redis cluster protocol to access each data shard. The direct connect mode and proxy mode are different from each other. For more information about relevant precautions and connection examples, see Use a private endpoint to connect to an ApsaraDB for Redis instance.

Scenarios

- Large data volume

  Cluster instances of ApsaraDB for Redis support the scaling of storage capacity. Compared with standard instances, cluster instances support a storage capacity of up to 4,098 GB. You can scale out storage capacity based on your business requirements.

- High QPS

  Standard instances of ApsaraDB for Redis cannot support high-QPS scenarios. You must deploy multiple data shards to resolve the performance bottleneck that is caused by the single-threading model. For more information, see Master-replica cluster instances.

- Throughput-intensive applications

  Compared with standard instances, cluster instances provide higher throughput over internal networks. You can read hot data with high performance and manage high-throughput workload.

- Applications that do not require high compatibility with the Redis protocol

  A cluster instance has multiple components. Therefore, compared with a standard instance, a cluster instance has more limits when it runs Redis commands. For more information, see Limits on the commands supported by cluster instances.

References

- To troubleshoot excessive memory consumption of shards in a cluster instance, see How do I search for large keys?.
- You can analyze data distribution in memory. For more information, see Analyze memory usage of ApsaraDB for Redis.

FAQ

Can I use the direct connect mode and proxy mode together?

Yes, you can use these modes together.

4.4. Read/write splitting instances

In the scenarios where reads are more than writes, read/write splitting instances of ApsaraDB for Redis support multiple specifications to meet various business requirements.

Overview

Read/write splitting instances of ApsaraDB for Redis are suitable for workloads where reads are more than writes. These instances ensure high availability (HA) and high performance, and support multiple specifications. The read/write splitting architecture allows a large number of concurrent requests to read hot data from read replicas. This can reduce the loads on the master node and minimize your costs of operations and maintenance (O&M).

Components

A read/write splitting instance of ApsaraDB for Redis consists of multiple proxy servers, a master node, a replica node, and one or more read replicas.

As a hot standby node, the replica node does not provide services. Read replicas only process read requests. Proxy servers forward write requests to the master node and forward read requests to the master node or one of the read replicas based on weights. The weight of each node is determined by the system and cannot be customized.

> **Note** The system evenly distributes read requests among the master node and read replicas. For example, if you purchase an instance with three read replicas, the read weight is 25% for the master node and each read replica.

The HA system monitors the status of each node. If a master node fails, the HA system performs a failover between the master node and the replica node. If a read replica fails, the HA system creates a new read replica to process read requests. During this process, the HA system updates the routing and weighting information.

The read/write splitting architecture supports chain replication. This allows you to scale out read replicas to increase the read capacity. The source code of the replication process has been customized and optimized by specialists of Alibaba Cloud to maximize workload stability during replication.

When an application is connected to the read/write splitting instance, a proxy server identifies read and write requests, performs load balancing, and forwards the requests to different nodes based on weights. Write requests are forwarded to the master node, and read requests are evenly distributed among the master node and read replicas.

Cluster instances of ApsaraDB for Redis are developed based on the native Redis protocol and are compatible with all Redis commands. You can upgrade a standard master-replica instance to a read/write splitting instance with a few clicks. You can also migrate data from an on-premises Redis database to a read/write splitting instance of ApsaraDB for Redis. Both instance upgrades and data migrations can be performed without the need to disrupt services.

Key Features

- High availability
  - Alibaba Cloud has developed an HA system for read/write splitting instances. The HA system monitors the status of all nodes on an instance to guarantee high availability. If a master node fails, the HA system will switch the workload from the master node to the replica node and update the instance topology. If a read replica fails, the HA system will enable another read replica. It will then synchronize data and forward read requests to this enabled read replica and suspend the failed read replica.
  - The proxy server module detects the service status of each read replica in real time. If a read replica is unavailable due to an exception, the proxy server module reduces the weight of this read replica. After a read replica cannot be connected for a specified number of times, the system suspends the read replica and forwards read requests to available read replicas. If the unavailable read replica recovers, the system enables and continues to monitor the read replica.
- High performance

  The read/write splitting instance supports chained replication. You can scale out the read replicas to increase the read capacity and optimize the usage of physical resources for each read replica.

Scenarios

- High QPS

  Standard instances of ApsaraDB for Redis cannot support high queries per second (QPS) scenarios. If reads of your workloads are more than writes, you must deploy multiple read replicas. This allows you to fix the performance bottleneck issue caused by a single-threading model. You can attach one, three, or five read replicas to a cluster instance of ApsaraDB for Redis. Compared with a standard instance, the cluster instance has the QPS performance improved by about five times.

- More native Redis features

  Read/write splitting instances are compatible with all Redis commands. You can migrate data to these instances without disrupting services.

- Persistent storage on ApsaraDB for Redis instances

  Read/write splitting instances support data persistence, backup, and recovery to ensure data reliability.

4.5. Instructions of read/write splitting instances

Read/write splitting instances of ApsaraDB for Redis are different from standard instances and cluster instances of ApsaraDB for Redis in terms of architectures and data processing methods. This topic describes how each component of a read/write splitting instance works and provides guidelines about how to use them.

Architecture description

- A read/write splitting instance of ApsaraDB for Redis provides one or more read replicas.
- One-way chained replication is used between the master node and the read replicas.
- A proxy server forwards requests to the master node and read replicas in the following ways:
  - Forwards write requests to the master node.
  - Forwards read requests to the master node and read replicas in an equal-weight manner.
- Requests from Lua scripts and database transactions are forwarded to the master node.
- The high-availability (HA) module monitors the statuses of the master node and read replicas and performs a failover if one of the components fails.

Architecture of a read/write splitting instance of ApsaraDB for Redis

Notes

- If a read replica fails, requests are forwarded to another read replica. If all read replicas are unavailable, requests are forwarded to the master node. Read replica failures may result in increases of the workloads on the master node and the response time. We recommend that you use multiple read replicas to process a large number of read requests.
- If an error occurs on a read replica, the HA module suspends the read replica and forwards requests to an available read replica. This failover process involves resource allocation, data synchronization, and service loading. The amount of time that it takes to perform a failover depends on the system workloads and data size. ApsaraDB for Redis does not guarantee the

time when the faulty read replica recovers.

- Full synchronization between read replicas is triggered in certain scenarios, for example, when a failover occurs on the master node. During full synchronization, read replicas are not available. If your requests are forwarded to the read replicas, the following error message is returned: `-LOADING Redis is loading the dataset in memory\r\n`.
- The master node conforms to the Service Level Agreement of ApsaraDB for Redis.

5.Instance specifications

5.1. Overview

ApsaraDB for Redis provides multiple editions, series types, and architectures.

This topic describes the edition, series type, and architecture specifications of ApsaraDB for Redis instances. You can refer to the following table for more information about the required instance specifications.

Edition	Series Type	Architecture type	Description
Community Edition	None	Standard master-replica instance	A master-replica instance of ApsaraDB for Redis. The maximum memory capacity is 64 GB and the QPS reference value is approximately 80,000.
		Master-replica cluster instances	A cluster instance of ApsaraDB for Redis. Each data shard is deployed in a master-replica architecture. The instance with the highest capacity and performance contains 32 data shards and supports a memory capacity of 512 GB. The QPS reference value is approximately 2,560,000.
		Read/Write Splitting	An ApsaraDB for Redis instance that contains a master-replica architecture and one or more read replicas. The instance with the highest capacity and performance contains five read replicas and supports a memory capacity of 64 GB.
		Read/write splitting cluster instances	A read/write splitting instance of ApsaraDB for Redis that contains a cluster instance and one or more read replicas. Each shard in the cluster is attached to one read replica. The instance with the highest capacity and performance contains 32 shards and 32 read replicas, and supports a memory capacity of 512 GB.
		Standard master-replica instance	A standard master-replica instance that uses multi-thread model provides the performance approximately three times that of an ApsaraDB for Redis Community Edition instance of the same specifications. The maximum memory capacity is 64 GB and the QPS reference value is approximately 240,000.

Edition	Series Type	Architecture type	Description
ApsaraDB for Redis Enhanced Edition (Tair)	Performance enhanced	Cluster instances	A cluster instance of ApsaraDB for Redis that uses multi-thread model supports several types of new data structures. Each data shard uses a master-replica architecture, and the performance is approximately three times that of an ApsaraDB for Redis Community Edition instance of the same specifications. The maximum memory capacity is 4,096 GB and the QPS reference value is approximately 61,440,000.
		Read/Write Splitting	A read/write splitting instance of ApsaraDB for Redis uses multi-thread model and contains a master-replica architecture and one or more read replicas. It provides the performance approximately three times that of an ApsaraDB for Redis Community Edition instance of the same specifications. The instance with the highest capacity and performance contains five read replicas and supports a memory capacity of 64 GB.
	Hybrid storage	Standard master-replica instances	A hybrid-storage instance of ApsaraDB for Redis that is developed based on standard master-replica instances. The hybrid-storage instance supports storage in memory and disks. The instance with the highest capacity and performance supports a memory capacity of 64 GB and disk storage of up to 512 GB.
		Cluster instances	A cluster instance of ApsaraDB for Redis that supports storage in memory and disks. The instance with the highest capacity and performance supports a memory capacity of 1,024 GB and disk storage of up to 8,192 GB, and contains 16 shards.
Phased-out instance types	N/A	N/A	This topic lists the ApsaraDB for Redis instances that are no longer available for purchase. If you have already purchased one or more of these instances, you can continue to use them. You can view the information about these specifications such as connection limit, bandwidth, and QPS reference value in Phased-out instance types.

FAQ

- Q: Do I need to reserve snapshot memory resources as I do with AWS ElastiCache when I choose a specification?

A: No. Alibaba Cloud provides the instances of ApsaraDB for Redis Community Edition and Enhanced Edition (Tair). Therefore, you do not need to reserve the memory required for snapshots. The memory capacity corresponding to the specifications is the maximum available memory under an account. The memory capacity includes the memory occupied by user data, the static memory consumed by the database, and the memory occupied by the network link. In the future, Alibaba Cloud may provide hosts to support your ApsaraDB for Redis service. You may need to follow the instructions in related documentation to reserve memory when you purchase the hosts.

- Q: Why are some instance types not available?

  A: These instances may no longer be available to purchase. For more information, see Phased-out instance types.

- Q: How can I check the specifications of an ApsaraDB for Redis instance by using InstanceClass?

  A: You can enter the value of InstanceClass in the search bar at the top of an Alibaba Cloud document.

- Q: How do I test the performance of ApsaraDB for Redis instances?

  A: You can test the performance of ApsaraDB for Redis instances by using the methods described in Performance White Paper.

5.2. Community Edition

5.2.1. Standard master-replica instance

This topic describes the specifications of standard master-replica cluster instances of ApsaraDB for Redis Community Edition. These specifications include the memory capacity, the maximum number of connections to each instance, maximum internal bandwidth, and queries per second (QPS) reference value.

Instance types

Instance type	InstanceClass (used in API)	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
256 MB master-replica instance	redis.master.micro.default	10000	10000	10	80000
1 GB master-replica instance	redis.master.small.default	10000	10000	10	80000
2 GB master-replica instance	redis.master.mid.default	10000	10000	16	80000
4 GB master-replica instance	redis.master.stand.default	10000	10000	24	80000

Instance type	InstanceClass (used in API)	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
8 GB master-replica instance	redis.master.large.default	10000	10000	24	80000
16 GB master-replica instance	redis.master.2xlarge.default	10000	10000	32	80000
32 GB master-replica instance	redis.master.4xlarge.default	10000	10000	32	80000
64 GB master-replica instance	redis.master.8xlarge.default	10000	10000	48	80000

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if the number of shards or nodes is increased.
- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively. If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.
- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).
- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

FAQ

- Why is the actual bandwidth of my instance different from the bandwidth described in this topic?

The bandwidth of some instance types may be adjusted after the upgrades are complete. If the bandwidth of your instance is different from that described in this topic, you can change configurations and select the same specifications as that of your instance to update the bandwidth. You are not charged for the same specifications. For more information, see Change specifications.

- How do I create an instance with 256 MB of memory?

  On the Subscription tab of the buy page, you can find this instance type.

5.2.2. Master-replica cluster instances

This topic describes the specifications of master-replica cluster instances of ApsaraDB for Redis Community Edition. These specifications include the memory capacity, number of concurrent connections to each instance, maximum internal bandwidth, and queries per second (QPS) reference value.

Instance specifications

Instance type	InstanceClass (API operation)	Number of shards	Maximum number of new connections per second	Maximum number of concurrent connections	Bandwidth (Mbit/s)	QPS reference value
16 GB cluster instance	redis.logic.sharding.2g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
32 GB cluster instance	redis.logic.sharding.4g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
64 GB cluster instance	redis.logic.sharding.8g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
128 GB cluster instance	redis.logic.sharding.8g.16db.0rodb.16proxy.default	16	50000	160000	1536	1280000
256 GB cluster instance	redis.logic.sharding.16g.16db.0rodb.16proxy.default	16	50000	160000	1536	1280000

Instance type	InstanceClass (API operation)	Number of shards	Maximum number of new connections per second	Maximum number of concurrent connections	Bandwidth (Mbit/s)	QPS reference value
512 GB cluster instance	redis.logic.sharding.16g.32db.0rodb.32proxy.default	32	50000	320000	2048	2560000

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if the number of shards or nodes is increased.
- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively. If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.
- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).
- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

FAQ

Why does the bandwidth of my instance differ from that of the same instance described in this topic?

The bandwidth of some instance types may be adjusted after service upgrades. If the bandwidth of your instance is different from those described in this topic, you can change configurations and select the same specifications as that of your current instance. (You are not charged for the same instance type). For more information, see Modify instance configurations.

5.2.3. Read/Write Splitting

This topic describes the specifications of read/write splitting instances of ApsaraDB for Redis Community Edition. These specifications include the memory capacity, maximum number of concurrent connections to each instance, maximum bandwidth, and QPS reference value.

Instance specifications

Instance type	InstanceClass (API operation)	Number of read replicas	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
1 GB read/write splitting instance (1 primary node, 1 read replica)	redis.logic.splitrw.small.1db.1rodb.4proxy.default	1	192	20000	20000	200000
1 GB read/write splitting instance (1 primary node, 3 read replicas)	redis.logic.splitrw.small.1db.3rodb.4proxy.default	3	384	40000	40000	400000
1 GB read/write splitting instance (1 primary node, 5 read replicas)	redis.logic.splitrw.small.1db.5rodb.6proxy.default	5	576	50000	60000	600000
2 GB read/write splitting instance (1 primary node, 1 read replica)	redis.logic.splitrw.mid.1db.1rodb.4proxy.default	1	192	20000	20000	200000
2 GB read/write splitting instance (1 primary node, 3 read replicas)	redis.logic.splitrw.mid.1db.3rodb.4proxy.default	3	384	40000	40000	400000
2 GB read/write splitting instance (1 primary node, 5 read replicas)	redis.logic.splitrw.mid.1db.5rodb.6proxy.default	5	576	50000	60000	600000
4 GB read/write splitting instance (1 primary node, 1 read replica)	redis.logic.splitrw.stand.1db.1rodb.4proxy.default	1	192	20000	20000	200000

Instance type	InstanceClass (API operation)	Number of read replicas	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
4 GB read/write splitting instance (1 primary node, 3 read replicas)	redis.logic.splitrw.stand.1db.3rodb.4proxy.default	3	384	40000	40000	400000
4 GB read/write splitting instance (1 primary node, 5 read replicas)	redis.logic.splitrw.stand.1db.5rodb.6proxy.default	5	576	50000	60000	600000
8 GB read/write splitting instance (1 primary node, 1 read replica)	redis.logic.splitrw.large.1db.1rodb.4proxy.default	1	192	20000	20000	200000
8 GB read/write splitting instance (1 primary node, 3 read replicas)	redis.logic.splitrw.large.1db.3rodb.4proxy.default	3	384	40000	40000	400000
8 GB read/write splitting instance (1 primary node, 5 read replicas)	redis.logic.splitrw.large.1db.5rodb.6proxy.default	5	576	50000	60000	600000
16 GB read/write splitting instance (1 primary node, 1 read replica)	redis.logic.splitrw.2xlarge.1db.1rodb.4proxy.default	1	192	20000	20000	200000
16 GB read/write splitting instance (1 primary node, 3 read replicas)	redis.logic.splitrw.2xlarge.1db.3rodb.4proxy.default	3	384	40000	40000	400000
16 GB read/write splitting instance (1 primary node, 5 read replicas)	redis.logic.splitrw.2xlarge.1db.5rodb.6proxy.default	5	576	50000	60000	600000
32 GB read/write splitting instance (1 primary node, 1 read replica)	redis.logic.splitrw.4xlarge.1db.1rodb.4proxy.default	1	192	20000	20000	200000

Instance type	InstanceClass (API operation)	Number of read replicas	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
32 GB read/write splitting instance (1 primary node, 3 read replicas)	redis.logic.splitrw.4xlarge.1db.3rodb.4proxy.default	3	384	40000	40000	400000
32 GB read/write splitting instance (1 primary node, 5 read replicas)	redis.logic.splitrw.4xlarge.1db.5rodb.6proxy.default	5	576	50000	60000	600000
64 GB read/write splitting instance (1 primary node, 1 read replica)	redis.logic.splitrw.8xlarge.1db.1rodb.4proxy.default	1	192	20000	20000	200000
64 GB read/write splitting instance (1 primary node, 3 read replicas)	redis.logic.splitrw.8xlarge.1db.3rodb.4proxy.default	3	384	40000	40000	400000
64 GB read/write splitting instance (1 primary node, 5 read replicas)	redis.logic.splitrw.8xlarge.1db.5rodb.6proxy.default	5	576	50000	60000	600000

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if the number of shards or nodes is increased.
- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively. If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.
- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).
- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

5.2.4. Read/write splitting cluster instances

This topic describes the specifications of read/write splitting cluster instances of ApsaraDB for Redis Community Edition. These specifications include the memory capacity, maximum number of concurrent connections to each instance, maximum bandwidth, and QPS reference value for these instances.

Instance specifications

Instance type	InstanceClass (API operation)	Number of shards	Number of read replicas per shard	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
4 GB read/write splitting instance (2 primary nodes, 1 read replica)	redis.logic.splitrw.sharding2g.2db.1rodb.4proxy.default	2	1	384	50000	40000	400000
8 GB read/write splitting instance (2 primary nodes, 1 read replica)	redis.logic.splitrw.sharding4g.2db.1rodb.4proxy.default	2	1	384	50000	40000	400000
16 GB read/write splitting instance (2 primary nodes, 1 read replica)	redis.logic.splitrw.sharding8g.2db.1rodb.8proxy.default	2	1	384	50000	40000	400000
32 GB read/write splitting instance (8 primary nodes, 1 read replica)	redis.logic.splitrw.sharding4g.8db.1rodb.16proxy.default	8	1	1536	50000	160000	1600000

Instance type	InstanceClass (API operation)	Number of shards	Number of read replicas per shard	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
64 GB read/write splitting instance (16 primary nodes, 1 read replica)	redis.logic.splitrw.sharding4g.16db.1rodb.32proxy.default	16	1	2048	50000	320000	3200000
128 GB read/write splitting instance (16 primary nodes, 1 read replica)	redis.logic.splitrw.sharding8g.16db.1rodb.32proxy.default	16	1	2048	50000	320000	3200000
256 GB read/write splitting instance (32 primary nodes, 1 read replica)	redis.logic.splitrw.sharding8g.32db.1rodb.64proxy.default	32	1	2048	50000	500000	6400000
512 GB read/write splitting instance (32 primary nodes, 1 read replica)	redis.logic.splitrw.sharding16g.32db.1rodb.64proxy.default	32	1	2048	50000	500000	6400000

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if the number of shards or nodes is increased.
- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively. If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.
- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).
- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

FAQ

Q: How can I create a read/write splitting cluster instance in the console?

A: To create a read/write splitting cluster instance in the console, follow these steps:

1. Log on to the ApsaraDB for Redis console. In the left-side navigation pane, click Instances, and click Create Instance. On the buy page that appears, set Architecture Type to **Cluster Edition**.
2. Set **Shards** to the number of shards on the read-write splitting instance.
3. Select **Master-Replica** from the **Node Type** drop-down list.
4. Select the specifications from the **Instance Class** drop-down list.

5.3. Enhanced Edition

5.3.1. Performance-enhanced cluster instances

This topic describes the specifications of performance-enhanced cluster instances of ApsaraDB for Redis Enterprise Edition. These specifications include the memory capacity, number of concurrent connections to each instance, maximum bandwidth, and queries per second (QPS) reference value.

Instance specifications

Instance type	InstanceClass (API operation)	Number of I/O threads	Maximum number of new connections per second	Maximum number of concurrent connections	Bandwidth (MB/s)	QPS reference value
1 GB master-replica performance-enhanced instance	redis.amber.master.small.multithread	4	10000	30000	96	240000

Instance type	InstanceClass (API operation)	Number of I/O threads	Maximum number of new connections per second	Maximum number of concurrent connections	Bandwidth (MB/s)	QPS reference value
2 GB master-replica performance-enhanced instance	redis.amber.master.mid.multithread	4	10000	30000	96	240000
4 GB master-replica performance-enhanced instance	redis.amber.master.stand.multithread	4	10000	30000	96	240000
8 GB master-replica performance-enhanced instance	redis.amber.master.large.multithread	4	10000	30000	96	240000
16 GB master-replica performance-enhanced instance	redis.amber.master.2xlarge.multithread	4	10000	30000	96	240000
32 GB master-replica performance-enhanced instance	redis.amber.master.4xlarge.multithread	4	10000	30000	96	240000
64 GB master-replica performance-enhanced instance	redis.amber.master.8xlarge.multithread	4	10000	30000	96	240000

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if the number of shards or nodes is increased.
- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively.

If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.

- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).
- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

FAQ

Q: Why does the maximum number of concurrent connections of my instance differ from that of the same instance described in this topic?

A: The maximum number of concurrent connections of some instance types may be adjusted after service upgrades. If the maximum number of concurrent connections of your instance is different than those described in this topic, refer to Change specifications and modify your instance configurations. You are not charged for the same instance type.

5.3.2. Performance-enhanced cluster instances

This topic describes the specifications of performance-enhanced cluster instances of ApsaraDB for Redis Enhanced Edition. These specifications include the memory capacity, number of concurrent connections to each instance, bandwidth, and queries per second (QPS) reference value.

Instance specifications

Instance type	InstanceClass (API operation)	Number of I/O threads	Number of shards	Maximum number of new connections per second	Maximum number of concurrent connections	Bandwidth (Mbit/s)	QPS reference value

Instance type	InstanceClass (API operation)	Number of I/O threads	Number of shards	Maximum number of new connections per second	Maximum number of concurrent connections	Bandwidth (Mbit/s)	QPS reference value
4 GB performance-enhanced cluster instance	redis.amber.logic.sharding.2g.2db.0rodb.6proxy.multithread	4	2	40000	60000	192	480000
8 GB performance-enhanced cluster instance	redis.amber.logic.sharding.2g.4db.0rodb.12proxy.multithread	4	4	40000	120000	384	960000
16 GB performance-enhanced cluster instance	redis.amber.logic.sharding.2g.8db.0rodb.24proxy.multithread	4	8	50000	240000	768	1920000
32 GB performance-enhanced cluster instance	redis.amber.logic.sharding.4g.8db.0rodb.24proxy.multithread	4	8	50000	240000	768	1920000
64 GB performance-enhanced cluster instance	redis.amber.logic.sharding.8g.8db.0rodb.24proxy.multithread	4	8	50000	240000	768	1920000
128 GB performance-enhanced cluster instance	redis.amber.logic.sharding.8g.16db.0rodb.48proxy.multithread	4	16	50000	480000	1536	3840000

Instance type	InstanceClass (API operation)	Number of I/O threads	Number of shards	Maximum number of new connections per second	Maximum number of concurrent connections	Bandwidth (Mbit/s)	QPS reference value
256 GB performance-enhanced cluster instance	redis.amber.logic.sharding.16g.16db.0rodb.48proxy.multithread	4	16	50000	480000	1536	3840000
512 GB performance-enhanced cluster instance	redis.amber.logic.sharding.16g.32db.0rodb.96proxy.multithread	4	32	50000	500000	2048	7680000
1,024 GB performance-enhanced cluster instance	redis.amber.logic.sharding.16g.64db.0rodb.192proxy.multithread	4	64	50000	500000	2048	15360000
2,048 GB performance-enhanced cluster instance	redis.amber.logic.sharding.16g.128db.0rodb.384proxy.multithread	4	128	50000	500000	2048	30720000
4,096 GB performance-enhanced cluster instance	redis.amber.logic.sharding.16g.256db.0rodb.768proxy.multithread	4	256	50000	500000	2048	61440000

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased

even if the number of shards or nodes is increased.

- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively. If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.
- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).
- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

FAQ

Q: Why does the maximum number of concurrent connections of my instance differ from that of the same instance described in this topic?

A: The maximum number of concurrent connections of some instance types may be adjusted after service upgrades. If the maximum number of concurrent connections of your instance is different than those described in this topic, refer to Change specifications and modify your instance configurations. You are not charged for the same instance type.

5.3.3. Performance-enhanced read/write splitting instances

This topic describes the specifications of read/write splitting instances of ApsaraDB for Redis Enhanced Edition (Tair). These specifications include the number of read-only replicas, memory capacity, maximum number of concurrent connections to each instance, bandwidth, and queries per second (QPS) reference value.

Instance specifications

Instance type	InstanceClass (used in API operations)	Number of I/O threads	Number of read/write nodes	Number of read-only replicas	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
1 GB performance-enhanced read/write splitting instance (1 shard, 1 read-only replica)	redis.amber.logic.splitrw.small.1db.1rodb.6proxy.multithread	4	1	1	192	20,000	60,000	480,000
2 GB performance-enhanced read/write splitting instance (1 shard, 1 read-only replica)	redis.amber.logic.splitrw.mid.1db.1rodb.6proxy.multithread	4	1	1	192	20,000	60,000	480,000
4 GB performance-enhanced read/write splitting instance (1 shard, 1 read-only replica)	redis.amber.logic.splitrw.stand.1db.1rodb.6proxy.multithread	4	1	1	192	20,000	60,000	480,000
8 GB performance-enhanced read/write splitting instance (1 shard, 1 read-only replica)	redis.amber.logic.splitrw.large.1db.1rodb.6proxy.multithread	4	1	1	192	20,000	60,000	480,000
16 GB performance-enhanced read/write splitting instance (1 shard, 1 read-only replica)	redis.amber.logic.splitrw.2xlarge.1db.1rodb.6proxy.multithread	4	1	1	192	20,000	60,000	480,000

Instance type	InstanceClass (used in API operations)	Number of I/O threads	Number of read/write nodes	Number of read-only replicas	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
32 GB performance-enhanced read/write splitting instance (1 shard, 1 read-only replica)	redis.amber.logic.splitrw.4xlarge.1db.1rodb.6proxy.multithread	4	1	1	192	20,000	60,000	480,000
64 GB performance-enhanced read/write splitting instance (1 shard, 1 read-only replica)	redis.amber.logic.splitrw.8xlarge.1db.1rodb.6proxy.multithread	4	1	1	192	20,000	60,000	480,000
1 GB performance-enhanced read/write splitting instance (1 shard, 3 read-only replicas)	redis.amber.logic.splitrw.small.1db.3rodb.12proxy.multithread	4	1	3	384	40,000	120,000	960,000
2 GB performance-enhanced read/write splitting instance (1 shard, 3 read-only replicas)	redis.amber.logic.splitrw.mid.1db.3rodb.12proxy.multithread	4	1	3	384	40,000	120,000	960,000
4 GB performance-enhanced read/write splitting instance (1 shard, 3 read-only replicas)	redis.amber.logic.splitrw.stand.1db.3rodb.12proxy.multithread	4	1	3	384	40,000	120,000	960,000

Instance type	InstanceClass (used in API operations)	Number of I/O threads	Number of read/write nodes	Number of read-only replicas	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
8 GB performance-enhanced read/write splitting instance (1 shard, 3 read-only replicas)	redis.amber.logic.splitrw.large.1db.3rodb.12proxy.multithread	4	1	3	384	40,000	120,000	960,000
16 GB performance-enhanced read/write splitting instance (1 shard, 3 read-only replicas)	redis.amber.logic.splitrw.2xlarge.1db.3rodb.12proxy.multithread	4	1	3	384	40,000	120,000	960,000
32 GB performance-enhanced read/write splitting instance (1 shard, 3 read-only replicas)	redis.amber.logic.splitrw.4xlarge.1db.3rodb.12proxy.multithread	4	1	3	384	40,000	120,000	960,000
64 GB performance-enhanced read/write splitting instance (1 shard, 3 read-only replicas)	redis.amber.logic.splitrw.8xlarge.1db.3rodb.12proxy.multithread	4	1	3	384	40,000	120,000	960,000
1 GB performance-enhanced read/write splitting instance (1 shard, 5 read-only replicas)	redis.amber.logic.splitrw.small.1db.5rodb.18proxy.multithread	4	1	5	576	50,000	180,000	1,440,000

Instance type	InstanceClass (used in API operations)	Number of I/O threads	Number of read/write nodes	Number of read-only replicas	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
2 GB performance-enhanced read/write splitting instance (1 shard, 5 read-only replicas)	redis.amber.logic.splitrw.mid.1db.5rodb.18proxy.multithread	4	1	5	576	50,000	180,000	1,440,000
4 GB performance-enhanced read/write splitting instance (1 shard, 5 read-only replicas)	redis.amber.logic.splitrw.stand.1db.5rodb.18proxy.multithread	4	1	5	576	50,000	180,000	1,440,000
8 GB performance-enhanced read/write splitting instance (1 shard, 5 read-only replicas)	redis.amber.logic.splitrw.large.1db.5rodb.18proxy.multithread	4	1	5	576	50,000	180,000	1,440,000
16 GB performance-enhanced read/write splitting instance (1 shard, 5 read-only replicas)	redis.amber.logic.splitrw.2xlarge.1db.5rodb.18proxy.multithread	4	1	5	576	50,000	180,000	1,440,000
32 GB performance-enhanced read/write splitting instance (1 shard, 5 read-only replicas)	redis.amber.logic.splitrw.4xlarge.1db.5rodb.18proxy.multithread	4	1	5	576	50,000	180,000	1,440,000

Instance type	InstanceClass (used in API operations)	Number of I/O threads	Number of read/write nodes	Number of read-only replicas	Bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of concurrent connections	QPS reference value
64 GB performance-enhanced read/write splitting instance (1 shard, 5 read-only replicas)	redis.amber.logic.splitrw.8xlarge.1db.5rodb.18proxy.multithread	4	1	5	576	50,000	180,000	1,440,000

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if the number of shards or nodes is increased.
- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively. If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.
- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).
- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

5.3.4. Standard hybrid-storage instances

This topic describes the specifications of standard hybrid-storage instances of ApsaraDB for Redis Enhanced Edition (Tair), such as the memory capacity, maximum number of concurrent connections to each instance, maximum internal bandwidth, and queries per second (QPS) reference value.

Instance specifications

Note The maximum internal bandwidth is applicable to both upstream bandwidth and downstream bandwidth. If network resources are sufficient, the bandwidth is unlimited for ApsaraDB for Redis instances. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect for the instances.

Instance type	InstanceClass (used in API operations)	Number of new connections per second	Maximum number of concurrent connections	Maximum internal bandwidth (MB/s)	QPS reference value	Description
Master-replica instance with 16 GB memory and 32 GB disk storage	redis.amber.master.16g.2x.ext4.default	10,000	50,000	48	40,000	Master-replica hybrid-storage instance
Master-replica instance with 16 GB memory and 64 GB disk storage	redis.amber.master.16g.4x.ext4.default	10,000	50,000	48	40,000	Master-replica hybrid-storage instance
Master-replica instance with 16 GB memory and 128 GB disk storage	redis.amber.master.16g.8x.ext4.default	10,000	50,000	48	40,000	Master-replica hybrid-storage instance
Master-replica instance with 32 GB memory and 64 GB disk storage	redis.amber.master.32g.2x.ext4.default	10,000	50,000	48	40,000	Master-replica hybrid-storage instance
Master-replica instance with 32 GB memory and 128 GB disk storage	redis.amber.master.32g.4x.ext4.default	10,000	50,000	48	40,000	Master-replica hybrid-storage instance

Instance type	InstanceClass (used in API operations)	Number of new connections per second	Maximum number of concurrent connections	Maximum internal bandwidth (MB/s)	QPS reference value	Description
Master-replica instance with 32 GB memory and 256 GB disk storage	redis.amber.master.32g.8x.ext4.default	10,000	50,000	48	40,000	Master-replica hybrid-storage instance
Master-replica instance with 64 GB memory and 128 GB disk storage	redis.amber.master.64g.2x.ext4.default	10,000	50,000	48	40,000	Master-replica hybrid-storage instance
Master-replica instance with 64 GB memory and 256 GB disk storage	redis.amber.master.64g.4x.ext4.default	10,000	50,000	48	40,000	Master-replica hybrid-storage instance
Master-replica instance with 64 GB memory and 512 GB disk storage	redis.amber.master.64g.8x.ext4.default	10,000	50,000	48	40,000	Master-replica hybrid-storage instance

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if the number of shards or nodes is increased.
- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively. If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.
- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added

per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).

- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

5.3.5. Hybrid-storage cluster instances

This topic describes the specifications of hybrid-storage cluster instances of ApsaraDB for Redis Enhanced Edition (Tair). These specifications include the memory and disk capacity, maximum number of concurrent connections to each instance, maximum internal bandwidth, and queries per second (QPS) reference value.

Instance types

Instance type	InstanceClass (used in API)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
64 GB memory and 256 GB disk storage (4 shards)	redis.amber.sharding.16g.4db.0rodb.12proxy.4x.ext4.default	4	40000	200000	384	120000
64 GB memory and 512 GB disk storage (4 shards)	redis.amber.sharding.16g.4db.0rodb.12proxy.8x.ext4.default	4	40000	200000	384	120000
128 GB memory and 512 GB disk storage (4 shards)	redis.amber.sharding.32g.4db.0rodb.12proxy.4x.ext4.default	4	40000	200000	384	120000
128 GB memory and 1,024 GB disk storage (4 shards)	redis.amber.sharding.32g.4db.0rodb.12proxy.8x.ext4.default	4	40000	200000	384	120000
256 GB memory and 1,024 GB disk storage (4 shards)	redis.amber.sharding.64g.4db.0rodb.12proxy.4x.ext4.default	4	40000	200000	384	120000

Instance type	InstanceClass (used in API)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
256 GB memory and 2,048 GB disk storage (4 shards)	redis.amber.sharding.64g.4db.0rodb.12proxy.8x.ext4.default	4	40000	200000	384	120000
128 GB memory and 512 GB disk storage (8 shards)	redis.amber.sharding.16g.8db.0rodb.24proxy.4x.ext4.default	8	50000	400000	768	240000
128 GB memory and 1,024 GB disk storage (8 shards)	redis.amber.sharding.16g.8db.0rodb.24proxy.8x.ext4.default	8	50000	400000	768	240000
256 GB memory and 1,024 GB disk storage (8 shards)	redis.amber.sharding.32g.8db.0rodb.24proxy.4x.ext4.default	8	50000	400000	768	240000
256 GB memory and 2,048 GB disk storage (8 shards)	redis.amber.sharding.32g.8db.0rodb.24proxy.8x.ext4.default	8	50000	400000	768	240000
512 GB memory and 2,048 GB disk storage (8 shards)	redis.amber.sharding.64g.8db.0rodb.24proxy.4x.ext4.default	8	50000	400000	768	240000
512 GB memory and 4,096 GB disk storage (8 shards)	redis.amber.sharding.64g.8db.0rodb.24proxy.8x.ext4.default	8	50000	400000	768	240000
256 GB memory and 1,024 GB disk storage (16 shards)	redis.amber.sharding.16g.16db.0rodb.48proxy.4x.ext4.default	16	50000	500000	1536	480000
256 GB memory and 2,048 GB disk storage (16 shards)	redis.amber.sharding.16g.16db.0rodb.48proxy.8x.ext4.default	16	50000	500000	1536	480000

Instance type	InstanceClass (used in API)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
512 GB memory and 2,048 GB disk storage (16 shards)	redis.amber.sharding.32g.16db.0rodb.48proxy.4x.ext4.default	16	50000	500000	1536	480000
512 GB memory and 4,096 GB disk storage (16 shards)	redis.amber.sharding.32g.16db.0rodb.48proxy.8x.ext4.default	16	50000	500000	1536	480000
1,024 GB memory and 4,096 GB disk storage (16 shards)	redis.amber.sharding.64g.16db.0rodb.48proxy.4x.ext4.default	16	50000	500000	1536	480000
1,024 GB memory and 8,192 GB disk storage (16 shards)	redis.amber.sharding.64g.16db.0rodb.48proxy.8x.ext4.default	16	50000	500000	1536	480000

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if the number of shards or nodes is increased.
- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively. If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.
- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).

- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

5.4. Phased-out instance types

This topic describes the ApsaraDB for Redis instances that are phased out. If you have purchased these instances, you can continue to use them. The following table lists the maximum number of connections to each instance, maximum bandwidth, and QPS reference value for these instances.

Standard zone-disaster recovery instances of Community Edition

Instance type	InstanceClass (used in API)	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
1 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb1g.1db.0rodb.4proxy.default	10000	10000	10	80000
2 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb2g.1db.0rodb.4proxy.default	10000	10000	16	80000
4 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb4g.1db.0rodb.4proxy.default	10000	10000	24	80000
8 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb8g.1db.0rodb.4proxy.default	10000	10000	24	80000
16 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb16g.1db.0rodb.4proxy.default	10000	10000	32	80000
32 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb32g.1db.0rodb.4proxy.default	10000	10000	32	80000

Instance type	InstanceClass (used in API)	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
64 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb64g.1db.0rodb.4proxy.default	10000	10000	48	80000

Master-replica cluster instances of Community Edition

Instance type	InstanceClass (used in API)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
1 GB cluster instance (2 shards)	redis.logic.sharding.512m.2db.0rodb.4proxy.default	2	20000	20000	48	200000
1 GB cluster instance (4 shards)	redis.logic.sharding.256m.4db.0rodb.4proxy.default	4	40000	40000	96	400000
2 GB cluster instance (4 shards)	redis.logic.sharding.512m.4db.0rodb.4proxy.default	4	40000	40000	96	400000
2 GB cluster instance (8 shards)	redis.logic.sharding.256m.8db.0rodb.8proxy.default	8	50000	80000	192	800000
4 GB cluster instance (8 shards)	redis.logic.sharding.512m.8db.0rodb.8proxy.default	8	50000	80000	192	800000
4 GB cluster instance (16 shards)	redis.logic.sharding.256m.16db.0rodb.16proxy.default	16	50000	160000	384	1600000

Instance type	InstanceClass (used in API)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
8 GB cluster instance (16 shards)	redis.logic.sharding.512m.16db.0rodb.16proxy.default	16	50000	160000	384	1600000
8 GB cluster instance (32 shards)	redis.logic.sharding.256m.32db.0rodb.32proxy.default	32	50000	320000	768	3200000
16 GB cluster instance (32 shards)	redis.logic.sharding.512m.32db.0rodb.32proxy.default	32	50000	320000	768	3200000
16 GB cluster instance	redis.sharding.small.default	8	50000	80000	768	640000
32 GB cluster instance	redis.sharding.mid.default	8	50000	80000	768	640000
64 GB cluster instance	redis.sharding.large.default	8	50000	80000	768	640000
128 GB cluster instance	redis.sharding.2xlarge.default	16	50000	160000	1536	1280000
256 GB cluster instance	redis.sharding.4xlarge.default	16	50000	160000	1536	1280000
512 GB cluster instance	redis.sharding.8xlarge.default	32	50000	320000	2048	2560000

Zone-disaster recovery cluster instances of Community Edition

Instance type	InstanceClass (used in API)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Bandwidth (MB/s)	QPS reference value
16 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb16g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
32 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb32g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
64 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb64g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
128 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb128g.16db.0rodb.16proxy.default	16	50000	160000	1536	1280000
256 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb256g.16db.0rodb.16proxy.default	16	50000	160000	1536	1280000
512 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb512g.32db.0rodb.32proxy.default	32	50000	320000	2048	2560000

Calculation rules for bandwidth

- If network resources are sufficient, the bandwidth is unlimited. However, if network resources are insufficient, the limit on maximum internal bandwidth takes effect.
- The bandwidth value in the table refers to the bandwidth of the instance. It is the sum of bandwidth of all shards or nodes on the instance.
- The upper limit of total bandwidth for a cluster or read/write splitting instance (as shown in the table) is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if the number of shards or nodes is increased.
- The bandwidth applies to the upstream bandwidth and downstream bandwidth respectively.

If the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidth of the instance are both 10 MB/s.

- The bandwidth listed in the table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the instance and a client. We recommend that you connect to the instance over an internal network to maximize the performance.

Calculation rules for connections

- The following example explains the maximum number of new connections that can be added per second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the number of connections is 22,000 (12,000 + 10,000).
- The maximum number of new connections per second and the maximum number of connections are the sum of the corresponding values of all shards or nodes on the instance. After 500,000 is reached, the maximum number of connections cannot be increased even if the number of shards or nodes is increased.

FAQ

Zone-disaster recovery instances of ApsaraDB for Redis are phased out. How can I create a zone-disaster recovery instance when I use ApsaraDB for Redis?

Zone-disaster recovery instances of ApsaraDB for Redis are upgraded. To use the zone-disaster recovery feature, you can select a zone that supports zone-disaster recovery when you create an ApsaraDB for Redis instance.

6.Commands

6.1. Overview

ApsaraDB for Redis provides instances of multiple editions, series, and architectures. The supported commands vary based on different instance types. This topic describes the commands supported and unsupported by ApsaraDB for Redis instances. You can refer to the topics in the following tables to view details of commands.

Common commands

> **Note** The common commands are applicable to the Community Edition and Enhanced Edition.

Topic	Description
Commands supported by Redis 2.8	This topic describes the commands supported by ApsaraDB for Redis instances that use Redis 2.8. These instances include standard instances, cluster instances, and read/write splitting instances. ApsaraDB for Redis instances that use Redis 2.8 support the commands supported by native Redis 3.0.
Commands supported by Redis 4.0	This topic describes the commands supported by ApsaraDB for Redis instances that use Redis 4.0. These instances include standard instances, cluster instances, and read/write splitting instances.
Unsupported commands	This topic describes the native Redis commands that are not supported by ApsaraDB for Redis.
Limits on the commands supported by cluster instances	This topic describes the limits on the commands supported by cluster instances. Cluster instances and standard instances are deployed in different architectures of ApsaraDB for Redis. These types of instances follow different rules to run Redis commands.
Redis commands developed by Alibaba Cloud	ApsaraDB for Redis also supports certain Redis commands developed by Alibaba Cloud. You can use these commands to manage cluster instances or read/write splitting instances of ApsaraDB for Redis.

Commands for Enhanced Edition

Topic	Description
Commands supported by performance-enhanced instances	This topic describes the commands that are supported by ApsaraDB for Redis performance-enhanced instances. Performance-enhanced instances of ApsaraDB for Redis integrate with certain features of Tair, a distributed key-value storage system developed by Alibaba Group. Performance-enhanced instances support the commands supported by Community Edition as well as some new commands. This topic describes these new Redis commands.
CAS and CAD commands	This topic describes the enhanced commands that you can run to process strings on performance-enhanced instances of ApsaraDB for Redis Enhanced Edition (Tair). These commands include Compare And Set (CAS) and Compare And Delete (CAD).
TairString commands	This topic describes the commands supported by a TairString.
TairHash commands	This topic describes the commands supported by a TairHash.
TairGIS commands	This topic describes the commands supported by a TairGIS.
TairBloom commands	This topic describes the commands supported by a TairBloom.
TairDoc commands	This topic describes the commands supported by a TairDoc.
Compatibility in commands	ApsaraDB for Redis hybrid-storage instances support most native Redis commands. To guarantee high performance, hybrid-storage instances have limits on the use of certain Redis commands. This topic describes these limited commands.

6.2. Community Edition

6.2.1. Commands supported by Redis 2.8

This topic describes the commands supported by ApsaraDB for Redis instances that use Redis 2.8. These instances include standard instances, cluster instances, and read/write splitting instances. ApsaraDB for Redis instances that use Redis 2.8 support the commands supported by native Redis 3.0.

Supported Redis commands

Supported Redis commands 1

Key	String	Hash	List	Set	SortedSet
DEL	APPEND	HDEL	BLPOP	SADD	ZADD
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD
EXISTS	BITOP	HGET	BRPOPLPUSH	SDIFF	ZCOUNT
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY
EXPIREAT	DECR	HINCRBY	LINSERT	SINTER	ZRANGE
MOVE	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE
PERSIST	GET	HKEYS	LPOP	SISMEMBER	ZRANK
PEXPIRE	GETBIT	HLEN	LPUSH	SMEMBERS	ZREM
PEXPIREAT	GETRANGE	HMGET	LPUSHX	SMOVE	ZREMRANGEBYRANK
PTTL	GETSET	HMSET	LRANGE	SPOP	ZREMRANGEBYSCORE
RANDOMKEY	INCR	HSET	LREM	SRANDMEMBER	ZREVRANGE
RENAME	INCRBY	HSETNX	LSET	SREM	ZREVRANGEBYSCORE
RENAMENX	INCRBYFLOAT	HVALS	LTRIM	SUNION	ZREVRANK
RESTORE	MGET	HSCAN	RPOP	SUNIONSTORE	ZSCORE
SORT	MSET	None	RPOPLPUSH	SSCAN	ZUNIONSTORE
TTL	MSETNX	None	RPUSH	None	ZINTERSTORE
TYPE	PSETEX	None	RPUSHX	None	ZSCAN
SCAN	SET	None	None	None	ZRANGEBYLEX
OBJECT	SETBIT	None	None	None	ZLEXCOUNT
None	SETEX	None	None	None	ZREMRANGEBYLEX
None	SETNX	None	None	None	None

Key	String	Hash	List	Set	SortedSet
None	SETRANGE	None	None	None	None
None	STRLEN	None	None	None	None

Supported Redis commands 2

HyperLogLog	Pub/Sub	Transaction	Connection	Server	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	FLUSHALL	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	FLUSHDB	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	DBSIZE	SCRIPT EXISTS	GEOPOS
None	PUNSUBSCRIBE	UNWATCH	QUIT	TIME	SCRIPT FLUSH	GEODIST
None	SUBSCRIBE	WATCH	SELECT	INFO	SCRIPT KILL	GEORADIUS
None	UNSUBSCRIBE	None	None	KEYS	SCRIPT LOAD	GEORADIUSBYMEMBER
None	None	None	None	CLIENT KILL	None	None
None	None	None	None	CLIENT LIST	None	None
None	None	None	None	CLIENT GETNAME	None	None
None	None	None	None	CLIENT SETNAME	None	None
None	None	None	None	CONFIG GET	None	None
None	None	None	None	MONITOR	None	None
None	None	None	None	SLOWLOG	None	None

Note

- In an ApsaraDB for Redis cluster instance:
  - You can run the CLIENT LIST command to retrieve information about the connections to the specified proxy server. The fields including id, age, idle, addr, fd, name, db, multi, omem, and cmd indicate the same meanings as those in the native Redis kernel. The values of the sub and psub fields are either both 1 or both 0 for all proxy servers. The qbuf, qbuf-free, obl, and oll fields have no meanings and can be disregarded.
  - You can run the CLIENT KILL command by using the following two methods: `client kill ip:port` and `client kill addr ip:port`.
- The **CLIENT ID** command is not supported by read/write splitting instances and cluster instances.

Other Redis commands

- For more information about other supported commands, see Overview.
- For more information about unsupported commands, see Unsupported commands.

6.2.2. Commands supported by Redis 4.0

This topic describes the commands supported by ApsaraDB for Redis instances that use Redis 4.0. These instances include standard instances, cluster instances, and read/write splitting instances.

Supported Redis commands

Supported Redis commands 1

Key	String	Hash	List	Set	SortedSet
DEL	APPEND	HDEL	BLPOP	SADD	ZADD
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD
EXISTS	BITOP	HGET	BRPOPLPUSH	SDIFF	ZCOUNT
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY
EXPIREAT	DECR	HINCRBY	LINSERT	SINTER	ZRANGE
MOVE	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE
PERSIST	GET	HKEYS	LPOP	SISMEMBER	ZRANK
PEXPIRE	GETBIT	HLEN	LPUSH	SMEMBERS	ZREM
PEXPIREAT	GETRANGE	HMGET	LPUSHX	SMOVE	ZREMRANGEBYRANK

Key	String	Hash	List	Set	SortedSet
PTTL	GETSET	HMSET	LRANGE	SPOP	ZREMRANGEBYSCORE
RANDOMKEY	INCR	HSET	LREM	SRANDMEMBER	ZREVRANGE
RENAME	INCRBY	HSETNX	LSET	SREM	ZREVRANGEBYSCORE
RENAMENX	INCRBYFLOAT	HVALS	LTRIM	SUNION	ZREVRANK
RESTORE	MGET	HSCAN	RPOP	SUNIONSTORE	ZSCORE
SORT	MSET	None	RPOPLPUSH	SSCAN	ZUNIONSTORE
TTL	MSETNX	None	RPUSH	None	ZINTERSTORE
TYPE	PSETEX	None	RPUSHX	None	ZSCAN
SCAN	SET	None	None	None	ZRANGEBYLEX
OBJECT	SETBIT	None	None	None	ZLEXCOUNT
UNLINK	SETEX	None	None	None	ZREMRANGEBYLEX
None	SETNX	None	None	None	None
None	SETRANGE	None	None	None	None
None	STRLEN	None	None	None	None

Supported Redis commands 2

HyperLogLog	Pub/Sub	Transaction	Connection	Server	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	FLUSHALL	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	FLUSHDB	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	DBSIZE	SCRIPT EXISTS	GEOPOS
None	PUNSUBSCRIBE	UNWATCH	QUIT	TIME	SCRIPT FLUSH	GEODIST

HyperLogLog	Pub/Sub	Transaction	Connection	Server	Scripting	Geo
None	SUBSCRIBE	WATCH	SELECT	INFO	SCRIPT KILL	GEORADIUS
None	UNSUBSCRIBE	None	None	KEYS	SCRIPT LOAD	GEORADIUSBYMEMBER
None	None	None	None	CLIENT KILL	None	None
None	None	None	None	CLIENT LIST	None	None
None	None	None	None	CLIENT GETNAME	None	None
None	None	None	None	CLIENT SETNAME	None	None
None	None	None	None	CONFIG GET	None	None
None	None	None	None	MONITOR	None	None
None	None	None	None	SLOWLOG	None	None
None	None	None	None	SWAPDB	None	None
None	None	None	None	MEMORY	None	None

Note

- In an ApsaraDB for Redis cluster instance:
  - You can run the CLIENT LIST command to retrieve information about the connections to the specified proxy server. The fields including id, age, idle, addr, fd, name, db, multi, omem, and cmd indicate the same meanings as those in the native Redis kernel. The values of the sub and psub fields are either both 1 or both 0 for all proxy servers. The qbuf, qbuf-free, obl, and oll fields have no meanings and can be disregarded.
  - You can run the CLIENT KILL command by using the following two methods: `client kill ip:port` and `client kill addr ip:port`.
- The **CLIENT ID** command is not supported by read/write splitting instances and cluster instances.

Other Redis commands

- For more information about other supported commands, see Overview.
- For more information about unsupported commands, see Unsupported commands.

6.2.3. Commands supported by Redis 5.0

This topic describes the commands supported by the engine version Redis 5.0 of ApsaraDB for Redis instances. The commands include standard instances, cluster instances, and read/write splitting instances.

Supported Redis commands

Supported Redis commands 1

Key	String	Hash	List	Set	SortedSet	Stream
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	XINFO
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	XADD
EXISTS	BITOP	HGET	BRPOPLPUSH	SDIFF	ZCOUNT	XTRIM
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	XDEL
EXPIREAT	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	XRANGE
MOVE	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	XREVRANGE
PERSIST	GET	HKEYS	LPOP	SISMEMBER	ZRANK	XLEN
PEXPIRE	GETBIT	HLEN	LPUSH	SMEMBERS	ZREM	XREAD
PEXPIREAT	GETRANGE	HMGET	LPUSHX	SMOVE	ZREMRANGEBYRANK	XGROUP
PTTL	GETSET	HMSET	LRANGE	SPOP	ZREMRANGEBYSCORE	XREADGROUP
RANDOMKEY	INCR	HSET	LREM	SRANDMEMBER	ZREVRANGE	XACK
RENAME	INCRBY	HSETNX	LSET	SREM	ZREVRANGEBYSCORE	XCLAIM
RENAMENX	INCRBYFLOAT	HVALS	LTRIM	SUNION	ZREVRANK	XPENDING
RESTORE	MGET	HSCAN	RPOP	SUNIONSTORE	ZSCORE	
SORT	MSET		RPOPLPUSH	SSCAN	ZUNIONSTORE	
TTL	MSETNX		RPUSH		ZINTERSTORE	

Key	String	Hash	List	Set	SortedSet	Stream
TYPE	PSETEX		RPUSHX		ZSCAN	
SCAN	SET				ZRANGEBYLEX	
OBJECT	SETBIT				ZLEXCOUNT	
UNLINK	SETEX				ZREMRANGEBYLEX	
	SETNX				ZPOPMAX	
	SETRANGE				ZPOPMIN	
	STRLEN				BZPOPMIN	
					BZPOPMAX	

Supported Redis commands 2

HyperLogLog	Pub/Sub	Transaction	Connection	Server	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	FLUSHALL	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	FLUSHDB	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	DBSIZE	SCRIPT EXISTS	GEOPOS
	PUNSUBSCRIBE	UNWATCH	QUIT	TIME	SCRIPT FLUSH	GEODIST
	SUBSCRIBE	WATCH	SELECT	INFO	SCRIPT KILL	GEORADIUS
	UNSUBSCRIBE			KEYS	SCRIPT LOAD	GEORADIUSBYMEMBER
				CLIENT KILL		
				CLIENT LIST		
				CLIENT GETNAME		
				CLIENT SETNAME		

HyperLogLog	Pub/Sub	Transaction	Connection	Server	Scripting	Geo
				CONFIG GET		
				MONITOR		
				SLOWLOG		
				SWAPDB		
				MEMORY		
				CLIENT UNBLOCK		
				CLIENT ID		
				LOLWUT		

Note

- On ApsaraDB for Redis cluster instances:
  - When you run the **CLIENT LIST** command, you can retrieve information about all connections to the specified proxy server. In this list, the fields including id, age, idle, addr, fd, name, db, multi, omem, and cmd indicate the same meanings as those in the native Redis kernel. The values of the sub and psub fields are either both 1 or both 0 for all proxy servers. The qbuf, qbuf-free, obl, and oll fields have no meanings and can be ignored.
  - You can run the **CLIENT KILL** command in two ways: `client kill ip:port` and `client kill addr ip:port`.
- On ApsaraDB for Redis read-write-splitting and cluster instances:**CLIENT ID** is not available.

6.2.4. Unsupported commands

This topic describes the native Redis commands that are not supported by ApsaraDB for Redis.

Key	Server
MIGRATE	BGREWRITEAOF
None	BGSAVE
None	CONFIG REWRITE
None	CONFIG SET

Key	Server
None	CONFIG RESETSTAT
None	COMMAND
None	COMMAND COUNT
None	COMMAND GETKEYS
None	COMMAND INFO
None	DEBUG OBJECT
None	DEBUG SEGFAULT
None	LASTSAVE
None	ROLE
None	SAVE
None	SHUTDOWN
None	SLAVEOF
None	SYNC

6.2.5. Limits on the commands supported by cluster instances

This topic describes the limits on the commands supported by cluster instances. Cluster instances and standard instances are deployed in different architectures of ApsaraDB for Redis. These types of instances follow different rules to run Redis commands.

Limited commands

Key	String	List	HyperLogLog	Transaction	Scripting
RENAME	MSETNX	RPOPLPUSH	PFMERGE	DISCARD	EVAL
RENAMENX	None	BRPOP	PFCOUNT	EXEC	EVALSHA
SORT	None	BLPOP	None	MULTI	SCRIPT EXISTS
None	None	BRPOPLPUSH	None	UNWATCH	SCRIPT FLUSH
None	None	None	None	WATCH	SCRIPT KILL
None	None	None	None	None	SCRIPT LOAD

Unsupported commands

SWAP

Limits

- To run limited commands in cluster instances, you can use hash tags to make sure that target keys are distributed in a hash slot. For more information, see Limited commands.

  For example, if you process key1, aakey, and abkey3, you must store them as {key}1, aa{key}, and ab{key}3 to run the limited commands. For more information, visit Official Redis Documentation.

- If you do not run the WATCH command before a transaction is executed, and each command in the transaction processes only one key, the keys processed by all commands can be distributed in different slots. You can run these commands in the same way as you run them in a database that is connected without a proxy server. In other scenarios, all keys that all commands process in a transaction must be in the same slot.
  - The commands that process multiple keys include: DEL, SORT, MGET, MSET, BITOP, EXISTS, MSETNX, RENAME, RENAMENX, BLPOP, BRPOP, RPOPLPUSH, BRPOPLPUSH, SMOVE, SUNION, SINTER, SDIFF, SUNIONSTORE, SINTERSTORE, SDIFFSTORE, ZUNIONSTORE, ZINTERSTORE, PFMERGE, and PFCOUNT.
  - The commands that do not support transactions include: WATCH, UNWATCH, RANDOMKEY, KEYS, SUBSCRIBE, UNSUBSCRIBE, PSUBSCRIBE, PUNSUBSCRIBE, PUBLISH, PUBSUB, SCRIPT, EVAL, EVALSHA, SCAN, ISCAN, DBSIZE, ADMINAUTH, AUTH, PING, ECHO, FLUSHDB, FLUSHALL, MONITOR, IMONITOR, RIMONITOR, INFO, IINFO, RIINFO, CONFIG, SLOWLOG, TIME, and CLIENT.
- When you run Lua scripts on a cluster instance, follow these limits:
  - You must use KEYS arrays to pass all keys. For Redis commands in redis.call() and redis.pcall(), keys must be KEYS arrays. You cannot replace KEYS with Lua variables. Otherwise, the system returns the following error: " `-ERR bad lua script for redis cluster, all the keys that the script uses should be passed using the KEYS array\r\n` ".
  - All keys must be in the same slot. Otherwise, the system returns an error: " `-ERR eval/evalsh a command keys must be in same slot\r\n` ".
  - You must use keys when you run Redis commands. Otherwise, the system returns an error: " `-ERR for redis cluster, eval/evalsha number of keys can't be negative or zero\r\n` ".
  - Pub/Sub commands are not supported, including PSUBSCRIBE, PUBSUB, PUBLISH, PUNSUBSCRIBE, SUBSCRIBE, and UNSUBSCRIBE.

  > **Note** You can ignore these limits when you run Lua scripts on a standard ApsaraDB for Redis instance.

6.2.6. Commands supported by read/write splitting instances

Read/write splitting instances of ApsaraDB for Redis are classified into non-cluster read/write splitting instances and read/write splitting cluster instances. The supported commands vary depending on the type of instances.

Instance type	Supported command
Non-cluster read/write splitting instance	A non-cluster read/write splitting instance is a single-shard read/write splitting instance. The architecture consists of one read/write shard and one, three, or five read replicas. The read/write shard runs in a master-replica structure. The instance supports the commands that are supported by standard instances of ApsaraDB for Redis. The commands supported by an instance depend on the engine version of the instance. For example, to check the commands supported by the instances that use Redis 2.8, see Commands supported by Redis 2.8. A non-cluster read/write splitting instance supports certain commands developed by Alibaba Cloud, such as **RIINFO** and **RIMONITOR**. These commands are only applicable to read/write splitting instances. For more information, see Redis commands developed by Alibaba Cloud.
Read/write splitting cluster instance	A read/write splitting cluster instance consists of multiple read/write shards. Each shard runs in a master-replica structure and has one read replica attached. The instance supports the commands that are supported by cluster instances of ApsaraDB for Redis. For more information, see Limits on the commands supported by cluster instances. A read/write splitting cluster instance supports certain commands developed by Alibaba Cloud, such as **RIINFO** and **RIMONITOR**. These commands are only applicable to read/write splitting instances. For more information, see Redis commands developed by Alibaba Cloud.

6.2.7. Redis commands developed by Alibaba Cloud

Besides native Redis commands, ApsaraDB for Redis supports certain Redis commands developed by Alibaba Cloud. You can use these commands to manage cluster instances or read/write splitting instances of ApsaraDB for Redis.

Description

- INFO KEY: You can run this command to query slots and databases (DBs) to which keys belong. The native Redis command INFO can only contain one optional section by following this syntax: `info [section]`. When you run certain commands for cluster instances of ApsaraDB for Redis, all keys must be in the same slot. The `INFO KEY` command allows you to check whether keys are in the same slot or DB. You can run this command by following this syntax:

```
127.0.0.1:6379> info key test_key
slot:15118 node_index:0
```

> **Notice**
>
> - In earlier versions, the `INFO KEY` command may return a `node index` different from the `node index` in the topology of an instance. This issue has been fixed in the latest version. If your instance is an earlier version, you can upgrade the minor version. For more information, see Upgrade the minor version.
> - The `INFO KEY` command returns the node indexes of shard servers on cluster instances. These shard servers are different from the DBs used in the SELECT command.

- IINFO: You can run this command to specify the node of ApsaraDB for Redis to run the INFO command. This command is similar to the INFO command. You can run this command by following this syntax:

  ```
  iinfo db_idx [section]
  ```

  In this command, db_idx supports the range of [0, nodecount]. You can obtain the nodecount value by running the INFO command, and specify the section option in the same way as you specify this option for a native Redis database. To view a node of ApsaraDB for Redis, you can run the IINFO command or check the instance topology in the console.

- RIINFO: You can run this command in a similar way as you run IINFO, but only in read/write splitting scenarios. This command specifies the idx value as the identifier of the read replica where you want to run the INFO command. If you use this command on instances other than cluster read/write splitting instances, the system returns an error. You can run this command by following this syntax:

  ```
  riinfo db_idx ro_slave_idx [section]
  ```

- ISCAN: You can run this command to specify the DB of a cluster where you want to run the SCAN command. This command provides the db_idx parameter based on SCAN. The db_idx parameter supports the range of [0, nodecount]. You can obtain the nodecount value by running the INFO command or by checking the instance topology in the console. You can run this command by following this syntax:

  ```
  iscan db_idx cursor [MATCH pattern] [COUNT count]
  ```

- IMONITOR: Similar to IINFO and ISCAN, this parameter provides the db_idx parameter based on the MONITOR command. The db_idx parameter specifies the node where you want to run MONITOR. The db_idx parameter supports the range of [0, nodecount). You can obtain the nodecount value by running the INFO command or by checking the instance topology in the console. You can run this command by following this syntax:

  ```
  imonitor db_idx
  ```

- RIMONITOR: Similar to RIINFO, you can run this command to specify the read replica in a specified shard where you want to run the MONITOR command. This command supports read/write splitting scenarios. You can run this command by following this syntax:

  ```
  rimonitor db_idx ro_slave_idx
  ```

> **Note** Before you run IMONITOR and RIMONITOR, use telnet to make sure that your application is connected with the target ApsaraDB for Redis instance. To terminate these commands, run the **QUIT** command.

6.3. Enhanced Edition

6.3.1. Commands supported by performance-enhanced instances

This topic describes the commands that are supported by ApsaraDB for Redis performance-enhanced instances. This type of instances integrate with certain features of Tair and support the commands supported by Community Edition as well as some new commands.

Supported new command types

The following table lists the new commands that are exclusively supported by performance-enhanced instances. For more information about how to use these commands, see the related topic of each command.

Supported new command types

Command type	Description
Enhanced string commands	These commands are developed to enhance the performance of Redis strings. You can run these commands to change or delete a value if it matches a specified value.
TairString commands	Performance-enhanced instances integrate Tair and support the TairString data type and related commands.
TairHash commands	Performance-enhanced instances integrate Tair and support the TairHash data type and related commands.
TairGIS commands	Performance-enhanced instances integrate Tair and support the TairGIS data type and related commands.
TairBloom commands	Performance-enhanced instances integrate Tair and support the TairBloom data type and related commands.
TairDoc commands	Performance-enhanced instances integrate Tair and support the TairDoc data type and related commands.

Other commands

Performance-enhanced instances of ApsaraDB for Redis Enhanced Edition support the same commands as Community Edition when the same architecture is used. For more information about the commands supported by different ApsaraDB for Redis editions and architectures, see Overview.

6.3.2. CAS and CAD commands

This topic describes the enhanced commands that you can run to process strings on performance-enhanced instances of ApsaraDB for Redis Enhanced Edition (Tair). These commands include Compare And Set (CAS) and Compare And Delete (CAD).

Prerequisites

The commands described in this topic take effect only if the following conditions are met:

- A performance-enhanced instance of ApsaraDB for Redis Enhanced Edition (Tair) is used.
- The native Redis string data to be managed is stored on the performance-enhanced instance.

> Note You can manage both native Redis string data and TairString data on a performance-enhanced instance. Only native Redis string data supports the commands described in this topic.

Commands

Enhanced commands supported by Redis strings

Command	Syntax	Description
CAS	CAS <key> <oldvalue> <newvalue>	Changes the value of a specified key to newvalue if the current value of the key matches the oldvalue parameter. If the current value of the key does not match the oldvalue parameter, the value is not changed. Note CAS is only applicable to Redis strings. To change TairString values, use the EXCAS command.
CAD	CAD <key> <value>	Deletes a specified key if the current value of the key matches the oldvalue parameter. The key is not deleted if the current value of the key does not match the oldvalue parameter. Note CAD is only applicable to Redis strings. To delete TairString keys, use the EXCAD command.

CAS

- Syntax

  CAS <key> <oldvalue> <newvalue>

- Time complexity

O(1)

- Description

  This command can be used to change the value of a specified key to a new value if the current value of the key matches a specified value. The value is not changed if the current value of the key does not match the specified value.

- Parameters and options

Parameter	Description
key	The key of the native Redis string data that you want to manage with the command.
oldvalue	The value that you compare with the current value of the specified key.
newvalue	Changes the value of the specified key to the value of this parameter if the current value of the key matches the specified value.

- Returned values
  - Expected result: 1.
  - A value of -1 is returned if the specified key does not exist.
  - A value of 0 is returned if the change fails.
  - An error message is returned as another unexpected result.
- Examples

```
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

- Syntax

  CAD <key> <value>

- Time complexity

  O(1)

- Description

This command can be used to delete a specified key if the current value of the key matches a specified value. The key is not deleted if the current value of the key does not match the specified value.

- Parameters and options

Parameter	Description
key	The key of the native Redis string data that you want to manage with the command.
value	The value that you compare with the current value of the specified key.

- Returned values
  - Expected result: 1.
  - A value of -1 is returned if the specified key does not exist.
  - A value of 0 is returned if the deletion fails.
  - An error message is returned as another unexpected result.
- Examples

```
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)
```

6.3.3. TairString commands

This topic describes the commands supported by a TairString.

Overview

A TairString is a string that includes a version number. The string data of native Redis uses a key-value pair structure and only contains keys and values. However, TairStrings consist of keys, values, and version numbers. TairStrings can be used in scenarios where optimistic locking occurs. The **INCRBY** and **INCRBYFLOAT** commands are used to increase or decrease values of Redis strings. You can use TairStrings to limit the range of results returned by these commands. If a result is out of range, an error message is returned.

Key features:

- Each TairString value includes a version number.
- TairStrings can be used to limit the range of results returned by the **INCRBY** and **INCRBYFLOAT** commands when you run these commands to increase values.

Warning TairStrings are different from native Redis strings. The commands that are supported by TairStrings and native Redis strings are not interchangeable.

Prerequisites

The commands described in this topic take effect only if the following conditions are met:

- A performance-enhanced instance of ApsaraDB for Redis Enterprise Edition is used.
- The TairString data to be managed is stored on the performance-enhanced instance.

Note You can manage both native Redis string data and TairStrings on a performance-enhanced instance. However, Redis strings do not support the commands described in this topic.

Commands supported by TairStrings

TairString commands

Command	Syntax	Description
EXSET	EXSET <key> <value> [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER version | ABS version]	Writes a value to a key.
EXGET	EXGET <key>	Retrieves the value and version number of a TairString.
EXSETVER	EXSETVER <key> <version>	Specifies the version number of a key.
EXINCRBY	EXINCRBY <key> <num> [EX time] [PX time] [EXAT time] [EXAT time] [PXAT time] [NX | XX] [VER version | ABS version] [MIN minval] [MAX maxval]	Increases or decreases the value of a TairString. The value of the num parameter must be of long type.
EXINCRBYFLOAT	EXINCRBYFLOAT <key> <num> [EX time] [PX time] [EXAT time] [EXAT time] [PXAT time] [NX | XX] [VER version | ABS version] [MIN minval] [MAX maxval]	Increases the value of a TairString that you want to manage. The num parameter specifies a value of double type.
EXCAS	EXCAS <key> <newvalue> <version>	Changes the value of a specified key only if the current version number of the key matches the specified version number. The current value and version number of the key are returned if the update fails.

Command	Syntax	Description
EXCAD	EXCAD <key> <version>	Deletes a key if the current version number of the key matches the specified version number. An error message is returned if the operation fails.
DEL	DEL <key> [key ...]	Deletes one or more TairStrings.

EXSET

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to write a value to a key.

- Parameters and options

Parameter/option	Description
key	The key of the TairString that you want to manage.
value	The value that you want to write to the specified key.
EX	The relative timeout of the specified key in seconds. A value of 0 specifies that the key immediately expires.
EXAT	The absolute timeout of the specified key in seconds. A value of 0 specifies that the key immediately expires.
PX	The relative timeout of the specified key in milliseconds. A value of 0 specifies that the key immediately expires.
PXAT	The absolute timeout of the specified key in milliseconds. A value of 0 specifies that the key immediately expires.
NX	Specifies that the value is written only if the specified key does not exist.
XX	Specifies that the value is written only if the specified key exists.

Parameter/option	Description
VER	The version number of the specified key. ○ If the specified key exists, the version number specified by this parameter is compared with the current version number: ■ If the version numbers match, the specified value is written to the key and the version number is increased by 1. ■ If the version numbers do not match, an error message is returned. ○ If the specified key does not exist or the current version number of the key is 0, this parameter is ignored. The specified value is written to the key, and then the version number is set to 1.
ABS	The absolute version number of the key. Writes the specified value to the key disregard of the current version number of the key, and then overwrites the version number with the ABS value.

- Returned values
  - OK: the operation is successful.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

  EXGET <key>

- Time complexity

  O(1)

- Description

  This command is used to retrieve the value and version number of a TairString.

- Parameters and options

  key: the key of the TairString that you want to manage.

- Returned values
  - The value and version number of the TairString is returned if the operation is successful.
  - Otherwise, an exception is returned.

- Example

```
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

- Syntax

  EXSETVER <key> <version>

- Time complexity

  O(1)

- Description

  This command is used to specify the version number of a key.

- Parameters and options

Parameter/option	Description
key	The key of the TairString that you want to manage.
version	The version number that you want to specify.

- Returned values
  - 1: the operation is successful.
  - 0: the specified key does not exist.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to increase or decrease the value of a TairString. The value of the num parameter must be of long type.

- Parameters and options

Parameter/option	Description
key	The key of the TairString that you want to manage.
num	The value by which the specified TairString is increased. The value must be an integer.
EX	The relative timeout of the specified key in seconds. A value of 0 specifies that the key immediately expires.
EXAT	The absolute timeout of the specified key in seconds. A value of 0 specifies that the key immediately expires.

Parameter/option	Description
PX	The relative timeout of the specified key in milliseconds. A value of 0 specifies that the key immediately expires.
PXAT	The absolute timeout of the specified key in milliseconds. A value of 0 specifies that the key immediately expires.
NX	Specifies that the value is written only if the specified key does not exist.
XX	Specifies that the value is written only if the specified key exists.
VER	The version number of the specified key. ◦ If the specified key exists, the version number specified by this parameter is compared with the current version number: ▪ If the version numbers match, the value of the TairString is increased by num and the version number is increased by 1. ▪ If the version numbers do not match, an error message is returned. ◦ If the specified key does not exist or the current version number of the key is 0, the specified version number does not take effect. The TairString data is increased by num, and then the version number is set to 1.
ABS	The absolute version number of the key. Increases the value of the TairString disregard of the current version number of the key, and then overwrites the version number with the ABS value.
MIN	The minimum value of the TairString.
MAX	The maximum value of the TairString.

- Returned values
  - The current value of the TairString is returned if the operation is successful.
  - Otherwise, an exception is returned.
- Example

```
127.0.0.1:6379> EXINCRBY foo 100
(integer) 200
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

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to increase or decrease the value of a TairString. The value of the num parameter must be of double type.

- Parameters and options

Parameter/option	Description
key	The key of the TairString that you want to manage.
num	The value by which the specified TairString is increased. The value must be a floating-point number.
EX	The relative timeout of the specified key in seconds. A value of 0 specifies that the key immediately expires.
EXAT	The absolute timeout of the specified key in seconds. A value of 0 specifies that the key immediately expires.

Parameter/option	Description
PX	The relative timeout of the specified key in milliseconds. A value of 0 specifies that the key immediately expires.
PXAT	The absolute timeout of the specified key in milliseconds. A value of 0 specifies that the key immediately expires.
NX	Specifies that the value is written only if the specified key does not exist.
XX	Specifies that the value is written only if the specified key exists.
VER	The version number of the specified key. ◦ If the specified key exists, the version number specified by this parameter is compared with the current version number: ▪ If the version numbers match, the value of the TairString is increased by num and the version number is increased by 1. ▪ If the version numbers do not match, an error message is returned. ◦ If the specified key does not exist or the current version number of the key is 0, the specified version number does not take effect. The TairString data is increased by num, and then the version number is set to 1.
ABS	The absolute version number of the key. Increases the value of the TairString disregard of the current version number of the key, and then overwrites the version number with the ABS value.
MIN	The minimum value of the TairString.
MAX	The maximum value of the TairString.

- Returned values
  - The current value of the TairString is returned if the operation is successful.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

  EXCAS <key> <newvalue> <version>

- Time complexity

  O(1)

- Description

  This command is used to change the value of a specified key only if the current version number of the key matches the specified version number.

- Parameters and options

Parameter/option	Description
key	The key of the TairString that you want to manage.
newvalue	The value of the newvalue parameter overwrites the value of the specified key if the current version number of the key matches the specified version number.
version	The version number to be matched against the current version number of the specified key.

- Returned values
  - ["OK", "", version] is returned if the operation is successful. The quotation marks ("") represent an empty string and version represents the current version number of the key.
  - If the operation fails, an error message is returned: ["ERR update version is stale", value, version]. Value represents the current value of the key and version represents the current version number of the key.
  - Otherwise, an exception is returned.

- Example

```
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

- Syntax

  EXCAD <key> <version>

- Time complexity

  O(1)

- Description

  This command is used to delete a key if the current version number of the key matches the specified version number.

- Parameters and options

Parameter/option	Description
key	The key of the TairString that you want to manage.
newvalue	The value of the newvalue parameter overwrites the value of the specified key if the current version number of the key matches the specified version number.
version	The version number to be matched against the current version number of the specified key.

- Returned values
  - 1: the operation is successful.
  - -1: the specified key does not exist.
  - 0: the operation fails.
  - Otherwise, an exception is returned.

- Example

```
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)
```

6.3.4. TairHash commands

This topic describes the commands supported by a TairHash.

Overview

A TairHash is a hash that allows you to specify the expiration time and version number of a field. Similar to native Redis hashes, TairHash supports multiple commands and high performance in data processing. For native Redis hashes, you can only specify the expiration time of keys. However, TairHash allows you to specify the expiration time of both keys and fields and specify versions of fields. The improved TairHash features help you simplify the development in most scenarios. TairHashes use an efficient Active Expire algorithm to check the expiration time of fields and delete expired fields. This process does not increase the database response time.

Key features:

- You can set the expiration time and version number for each field.
- Fields support the Active Expire and Passive Expire algorithms.
- TairHash and native Redis hash use similar syntax.

> **Warning** TairHashes are different from native Redis hashes. The commands that are supported by TairHashes and native Redis hashes are not interchangeable.

Prerequisites

The commands described in this topic take effect only if the following conditions are met:

- A **performance-enhanced** instance of ApsaraDB for Redis Enhanced Edition is used.
- The TairHash to be managed is stored on the performance-enhanced instance.

> **Note** You can manage both native Redis hashes and TairHashes on a performance-enhanced instance. The commands described in this topic are not applicable to native Redis hashes.

Memory consumption and expiration policies of TairHash

TairHash supports efficient and active expiration policies. However, this can increase the memory consumption to some extent. For more information, see TairHash memory consumption and expiration policies.

Commands

TairHash commands

Command	Syntax	Description

Command	Syntax	Description
EXHSET	EXHSET <key> <field> <value> [EX time] [EXAT time] [PX time] [PXAT time] [NX/XX] [VER/ABS version] [NOACTIVE]	Adds a field to a specified TairHash. If the key does not exist, a new key holding a TairHash is created. If the specified field exists, this command overwrites the value of the field. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHMSET	EXHMSET <key> <field> <value> [field value...]	Sets specified fields to respective values in the TairHash. If the key does not exist, a new key holding a TairHash is created. If the field already exists, this command overwrites the value of the field. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHPEXPIREAT	EXHPEXPIREAT <key> <field> <milliseconds-timestamp> [VER/ABS version] [NOACTIVE]	Specifies the absolute expiration time of a field in a specified TairHash. Unit: milliseconds. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHPEXPIRE	EXHPEXPIRE <key> <field> <milliseconds> [NOACTIVE]	Specifies the absolute expiration time of a field in a specified TairHash. Unit: milliseconds. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHEXPIREAT	EXHEXPIREAT <key> <field> <timestamp> [NOACTIVE]	Specifies the absolute relative expiration time of a filed in a specified TairHash. Unit: seconds. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHEXPIRE	EXHEXPIRE <key> <field> <seconds> [NOACTIVE]	Specifies the relative expiration time of a filed in a specified TairHash. Unit: seconds. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHPTTL	EXHPTTL <key> <field>	Retrieves the remaining expiration time of a field in a specified TairHash. Unit: milliseconds. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.

Command	Syntax	Description
EXHTTL	EXHTTL <key> <field>	Retrieves the remaining expiration time of a field in a specified TairHash. Unit: seconds. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHVER	EXHVER <key> <field>	Retrieves the current version number of a field in a specified TairHash if the key matches the specified key. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHSETVER	EXHSETVER <key> <field> <version>	Sets the current version number of a field in the specified TairHash if the key matches the specified key. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHINCRBY	EXHINCRBY <key> <field> <num> [EX time] [EXAT time] [PX time] [PXAT time] [VER/ABS version] [MIN minval] [MAX maxval]	Increases the value of a specified field by an integer if the key matches the specified key. If the specified key does not exist, a TairHash is created. If the specified field does not exist, this command adds the field and sets the value of the field to 0 before creating a TairHash. You can also run the EX, EXAT, PX, or PXAT command to specify the expiration time for the field. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHINCRBY FLOAT	EXHINCRBYFLOAT <key> <field> <value> [EX time] [EXAT time] [PX time] [PXAT time] [VER/ABS version] [MIN minval] [MAX maxval]	Increases a specified field value by a floating-point number in a TairHash if the key matches the specified key. If the specified key does not exist, a TairHash is created. If the specified field does not exist, this command adds the field and sets the value of the field to 0 before creating a TairHash. You can also run the EX, EXAT, PX, or PXAT command to specify the expiration time for the field. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.

Command	Syntax	Description
EXHGET	EXHGET <key> <field>	Retrieves a value associated with the specified field in a TairHash. A value of nil is returned if the specified key or field does not exist. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHGETWITHVER	EXHGETWITHVER <key> <field>	Retrieves the value and version number of a field in the specified TairHash if the key matches the specified key. A value of nil is returned if the specified key or field does not exist. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHMGET	EXHMGET <key> <field> [field ...]	Retrieves multiple field values in TairHash data in each query if the key of the TairHash data matches the specified key. A value of nil is returned if the specified key or fields do not exist. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHMGETWITHVER	EXHMGETWITHVER <key> <field> [field ...]	Retrieves the values and version numbers of multiple fields in a specified TairHash if the key matches the specified key. A value of nil is returned if the specified key or fields do not exist. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHDEL	EXHDEL <key> <field> <field> <field> ...	Deletes a field from a specified TairHash. A value of 0 is returned if the specified key or field does not exist. A value of 1 is returned if the field is deleted. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHLEN	EXHLEN <key> [noexp]	Retrieves the number of fields in a TairHash if the key matches the specified key. The returned value may include the number of expired fields that have not been deleted. If you want to query only the number of fields that are not expired, you can specify the *oexp* parameter.

Command	Syntax	Description
EXHEXISTS	EXHEXISTS <key> <field>	Checks whether a field exists in a TairHash if the key matches the specified key. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHSTRLEN	EXHSTRLEN <key> <field>	Retrieves the length of a field value in the TairHash if the key matches the specified key. When you run this command, the system uses the Passive Expire algorithm to expire and delete fields.
EXHKEYS	EXHKEYS <key>	Retrieves all fields in a TairHash if the key matches the specified key. The returned result filters out expired fields. To reduce response time, the system does not delete these expired fields when running this command.
EXHVALS	EXHVALS <key>	Retrieves all field values in the TairHash if the key matches the specified key. The returned result filters out expired fields. To reduce response time, the system does not delete these expired fields when running this command.
EXHGETALL	EXHGETALL <key>	Retrieves all fields and associated values in a TairHash if the key matches the specified key. The returned result filters out expired fields. To reduce response time, the system does not delete these expired fields when running this command.
EXHSCAN	EXHSCAN <key> <op> <subkey> [MATCH pattern] [COUNT count]	Scans TairHashes if the key matches the specified key. You can set the op parameter to values, such as >, >=, <, <=, ==, ^, or $. This op parameter specifies a scan method. You can also set the MATCH parameter to specify a regular expression and filter subkeys. The COUNT parameter limits the number of returned values. If you do not specify the COUNT parameter, the default value is set to 10. The returned result filters out expired fields. To reduce response time, the system does not delete these expired fields when running this command.
DEL	DEL <key> [key ...]	Deletes one or more TairHashes.

EXHSET

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to add a field to a specified TairHash. If the key does not exist, a new key holding a TairHash is created. If the specified field exists, this command overwrites the value of the field.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.
value	The value of the specified field. Each field can have only one value.
EX	The relative expiration time of the specified field in seconds. A value of 0 specifies that the field does not expire.
EXAT	The absolute expiration time of the specified field in seconds. A value of 0 specifies that the field does not expire.
PX	The relative expiration time of the specified field in milliseconds. A value of 0 specifies that the field does not expire.
PXAT	The absolute expiration time of the specified field in milliseconds. A value of 0 specifies that the field does not expire.
NX	Specifies that the value is written only if the field does not exist.
XX	Specifies that the value is written only if the field exists.
VER	The version number of the specified field. ◦ If the specified field exists, the version number specified by this parameter is matched against the current version number: ▪ If the version numbers match, the TairHash is increased by num and the version number is increased by 1. ▪ If the version numbers do not match, an error message is returned. ◦ If the specified field does not exist or the current version number of the field is 0, this parameter is ignored. The specified value is written to the field, and then the version number is set to 1.

Parameter/option	Description
ABS	The absolute version number of the field. If you specify this parameter, the system forcibly writes the specified value to the field regardless whether the field exists. Then, the version number is overwritten with the specified ABS value.
NOACTIVE	When you specify the *EX/EXAT/PX/PXAT* parameters, you can specify the *NOACTIVE* parameter to disable the active expiration policy for the field. This allows you to reduce the memory consumption.

- Returned values
  - 1: a new field is created and a value is set.
  - 0: the field exists and the specified value overwrites the current value.
  - -1: the XX parameter is specified and the specified field does not exist.
  - -1: the NX parameter is specified and the specified field exists.
  - An error message of "ERR update version is stale" is returned. The message indicates that the value of the VER parameter does not match the current version number.
  - Otherwise, an exception is returned.

EXHGET

- Syntax

  EXHGET <key> <field>

- Time complexity

  O(1)

- Description

  This command is used to retrieve a value associated with the specified field in a TairHash.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - The value of the field is returned if the field exists and the operation is successful.
  - nil: the key or field does not exist.
  - Otherwise, an exception is returned.

EXHMSET

- Syntax

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

- Time complexity

O(1)

- Description

  This command is used to set specified fields to respective values in the TairHash. If the key does not exist, a new key holding a TairHash is created. If the field already exists, this command overwrites the value of the field.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.
value	The value of the specified field. Each field can have only one value.

- Returned values
  - OK: the operation is successful.
  - Otherwise, an exception is returned.

EXHPEXPIREAT

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to specify the absolute expiration time of a field in a specified TairHash. Unit: milliseconds.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.
milliseconds-timestamp	The Unix timestamp. Unit: milliseconds.

Parameter/option	Description
VER	The version number of the specified field. ○ If the specified field exists, the version number specified by this parameter is matched against the current version number: ■ If the version numbers match, the TairHash is increased by num and the version number is increased by 1. ■ If the version numbers do not match, an error message is returned. ○ If the specified field does not exist or the current version number of the field is 0, this parameter is ignored. The specified value is written to the field, and then the version number is set to 1.
ABS	The absolute version number of the field. If you specify this parameter, the system forcibly writes the specified value to the field regardless whether the field exists. Then, the version number is overwritten with the specified ABS value.
NOACTIVE	When you specify the *EX/EXAT/PX/PXAT* parameters, you can specify the *NOACTIVE* parameter to disable the active expiration policy for the field. This allows you to reduce the memory consumption.

- Returned values
  - 1: the field exists and a value is set.
  - 0: the field does not exist.
  - Otherwise, an exception is returned.

EXHPEXPIRE

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to specify the relative expiration time of a filed in a specified TairHash. Unit: milliseconds.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.
milliseconds	The relative expiration time of the specified field in milliseconds.

Parameter/option	Description
VER	The version number of the specified field. ○ If the specified field exists, the version number specified by this parameter is matched against the current version number: ■ If the version numbers match, the TairHash is increased by num and the version number is increased by 1. ■ If the version numbers do not match, an error message is returned. ○ If the specified field does not exist or the current version number of the field is 0, this parameter is ignored. The specified value is written to the field, and then the version number is set to 1.
ABS	The absolute version number of the field. If you specify this parameter, the system forcibly writes the specified value to the field regardless whether the field exists. Then, the version number is overwritten with the specified ABS value.
NOACTIVE	When you specify the *EX/EXAT/PX/PXAT* parameters, you can specify the *NOACTIVE* parameter to disable the active expiration policy for the field. This allows you to reduce the memory consumption.

- Returned values
  - 1: the field exists and a value is set.
  - 0: the field does not exist.
  - Otherwise, an exception is returned.

EXHEXPIREAT

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to specify the absolute relative expiration time of a filed in a specified TairHash. Unit: seconds.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.
timestamp	The Unix timestamp. Unit: seconds.

Parameter/option	Description
VER	The version number of the specified field. ○ If the specified field exists, the version number specified by this parameter is matched against the current version number: ■ If the version numbers match, the TairHash is increased by num and the version number is increased by 1. ■ If the version numbers do not match, an error message is returned. ○ If the specified field does not exist or the current version number of the field is 0, this parameter is ignored. The specified value is written to the field, and then the version number is set to 1.
ABS	The absolute version number of the field. If you specify this parameter, the system forcibly writes the specified value to the field regardless whether the field exists. Then, the version number is overwritten with the specified ABS value.
NOACTIVE	When you specify the *EX/EXAT/PX/PXAT* parameters, you can specify the *NOACTIVE* parameter to disable the active expiration policy for the field. This allows you to reduce the memory consumption.

- Returned values
  - 1: the field exists and a value is set.
  - 0: the field does not exist.
  - Otherwise, an exception is returned.

EXHEXPIRE

- Syntax

  EXHEXPIRE <key> <field> <seconds>

- Time complexity

  O(1)

- Description

  This command is used to specify the relative expiration time of a filed in a specified TairHash. Unit: seconds.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.
seconds	The TTL value of the specified field. Unit: seconds.

Parameter/option	Description
VER	The version number of the specified field. ○ If the specified field exists, the version number specified by this parameter is matched against the current version number: ■ If the version numbers match, the TairHash is increased by num and the version number is increased by 1. ■ If the version numbers do not match, an error message is returned. ○ If the specified field does not exist or the current version number of the field is 0, this parameter is ignored. The specified value is written to the field, and then the version number is set to 1.
ABS	The absolute version number of the field. If you specify this parameter, the system forcibly writes the specified value to the field regardless whether the field exists. Then, the version number is overwritten with the specified ABS value.
NOACTIVE	When you specify the *EX/EXAT/PX/PXAT* parameters, you can specify the *NOACTIVE* parameter to disable the active expiration policy for the field. This allows you to reduce the memory consumption.

- Returned values
  - 1: the field exists and a value is set.
  - 0: the field does not exist.
  - Otherwise, an exception is returned.

EXHPTTL

- Syntax

  EXHPTTL <key> <field>

- Time complexity

  O(1)

- Description

  This command is used to retrieve the remaining expiration time of a field in a specified TairHash. Unit: milliseconds.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - -2: the specified key or field does not exist.
  - -1: the specified field exists but the TTL value is not specified.

- The expiration time of the field in milliseconds is returned if the field exists and the expiration time of the field is specified.
- Otherwise, an exception is returned.

EXHTTL

- Syntax

  EXHTTL <key> <field>

- Time complexity

  O(1)

- Description

  This command is used to retrieve the remaining expiration time of a field in a specified TairHash. Unit: seconds.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - -2: the specified key or field does not exist.
  - -1: the specified field exists but the TTL value is not specified.
  - The expiration time of the field in seconds is returned if the field exists and the expiration time of the field is specified.
  - Otherwise, an exception is returned.

EXHVER

- Syntax

  EXHVER <key> <field>

- Time complexity

  O(1)

- Description

  This command is used to retrieve the current version number of a field in a specified TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - -1: the specified key does not exist.
  - -2: the specified field does not exist.
  - The version number of the specified field is returned if the operation is successful.
  - Otherwise, an exception is returned.

EXHSETVER

- Syntax

  EXHSETVER <key> <field> <version>

- Time complexity

  O(1)

- Description

  This command is used to set the current version number of a field in the specified TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - 0: the specified TairHash or field does not exist.
  - 1: the version number is specified.
  - Otherwise, an exception is returned.

EXHINCRBY

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to increase the value of a field by num in a TairHash if the key matches the specified key. The value of the num parameter must be an integer. If the specified TairHash does not exist, a TairHash is created. If the specified field does not exist, this command adds the field and sets the value of the field to 0 before creating a TairHash. When you run this command, the system uses a passive expiration algorithm to delete expired fields.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.
num	The integer by which you want to increase a specified field value.
EX	The relative expiration time of the specified field in seconds. A value of 0 specifies that the field does not expire.
EXAT	The absolute expiration time of the specified field in seconds. A value of 0 specifies that the field does not expire.
PX	The relative expiration time of the specified field in milliseconds. A value of 0 specifies that the field does not expire.
PXAT	The absolute expiration time of the specified field in milliseconds. A value of 0 specifies that the field does not expire.
VER	The version number of the specified field. ○ If the specified field exists, the version number specified by this parameter is matched against the current version number: ■ If the version numbers match, the TairHash is increased by num and the version number is increased by 1. ■ If the version numbers do not match, an error message is returned. ○ If the value of the *VER* parameter is 0, you do not need to check the version number.
ABS	The absolute version number of the field. If you specify this parameter, the system forcibly increases the TairHash by num regardless whether the field exists. Then, the version number is overwritten with the specified ABS value. The value of this parameter must not be 0.
MIN	The minimum value of the field value. If the specified value is smaller than this lower limit, an exception is returned.
MAX	The maximum value of the field value. If the specified value is larger than this upper limit, an exception is returned.
NOACTIVE	When you specify the *EX/EXAT/PX/PXAT* parameters, you can specify the *NOACTIVE* parameter to disable the active expiration policy for the field. This allows you to reduce the memory consumption.

- Returned values
  - The value increased by num is returned if the operation is successful.
  - Otherwise, an exception is returned.

EXHINCRBYFLOAT

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to increase a specified field value by num in a TairHash if the key matches the specified key. The value of the num parameter must be a floating-point number. If the specified TairHash does not exist, a TairHash is created. If the specified field does not exist, this command adds the field and sets the value of the field to 0 before creating a TairHash. When you run this command, the system uses a passive expiration algorithm to delete expired fields.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.
num	The increment (a floating-point number) to be added to the specified field value.
EX	The relative expiration time of the specified field in seconds. A value of 0 specifies that the field does not expire.
EXAT	The absolute expiration time of the specified field in seconds. A value of 0 specifies that the field does not expire.
PX	The relative expiration time of the specified field in milliseconds. A value of 0 specifies that the field does not expire.
PXAT	The absolute expiration time of the specified field in milliseconds. A value of 0 specifies that the field does not expire.
VER	The version number of the specified field. ◦ If the specified field exists, the version number specified by this parameter is matched against the current version number: ▪ If the version numbers match, the TairHash is increased by num and the version number is increased by 1. ▪ If the version numbers do not match, an error message is returned. ◦ If the value of the *VER* parameter is 0, you do not need to check the version number.
ABS	The absolute version number of the field. If you specify this parameter, the system forcibly increases the TairHash by num regardless whether the field exists. Then, the version number is overwritten with the specified ABS value. The value of this parameter must not be 0.
MIN	The minimum value of the field value. If the specified value is smaller than this lower limit, an exception is returned.

Parameter/option	Description
MAX	The maximum value of the field value. If the specified value is larger than this upper limit, an exception is returned.
NOACTIVE	When you specify the *EX/EXAT/PX/PXAT* parameters, you can specify the *NOACTIVE* parameter to disable the active expiration policy for the field. This allows you to reduce the memory consumption.

- Returned values
  - The value increased by num is returned if the operation is successful.
  - Otherwise, an exception is returned.

EXHGETWITHVER

- Syntax

  EXHGETWITHVER <key> <field>

- Time complexity

  O(1)

- Description

  This command is used to retrieve the value and version number of a field in the specified TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - The value and version number of the field is returned if the field exists and the operation is successful.
  - nil: the key or field does not exist.
  - Otherwise, an exception is returned.

EXHMGET

- Syntax

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

- Time complexity

  O(1)

- Description

This command is used to retrieve multiple field values in a TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - nil: the key does not exist.
  - An array is returned if the specified key and fields exist. Each element in the array corresponds to a field value.
  - An array is returned if the specified key exists but some fields do not exist. Each element in the array corresponds to a field value. The elements of the non-existing fields are displayed as nil.
  - Otherwise, an exception is returned.

EXHMGETWITHVER

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to retrieve the values and version numbers of multiple fields in a specified TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - nil: the key does not exist.
  - An array is returned if the specified key and fields exist. Each element in the array corresponds to a field value and a version number.
  - An array is returned if the specified key exists but some fields do not exist. Each element in the array corresponds to a field value and a version number. The elements of the non-existing fields are displayed as nil.
  - Otherwise, an exception is returned.

EXHDEL

- Syntax

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

- Time complexity

  O(1)

- Description

  This command is used to delete a field from a specified TairHash.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - 0: the specified key or field does not exist.
  - 1: the operation is successful.
  - Otherwise, an exception is returned.

EXHLEN

- Syntax

  EXHLEN <key> [noexp]

- Time complexity

  O(1)

- Description

  This command is used to retrieve the number of fields in a TairHash if the key matches the specified key. The returned value may include the number of expired fields that have not been deleted.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.

Parameter/option	Description
noexp	By default, the **EXHLEN** command does not delete or filter expired fields. Therefore, the results may include the number of expired fields that have not been deleted. If you only want to query the number of fields that are not expired, you can specify the *noexp* parameter. When you specify the *noexp* parameter, ○ the response time of the **EXHLEN** command is based on the size of the Tairhash, because the system scans all TairHashes. ○ The result of the **EXHLEN** command does not include the number of expired fields that are not deleted.

- Returned values
  - 0: the specified key or field does not exist.
  - The number of fields in the TairHash is returned if the operation is successful.
  - Otherwise, an exception is returned.

EXHEXISTS

- Syntax

  EXHEXISTS <key> <field>

- Time complexity

  O(1)

- Description

  This command is used to check whether a field exists in a TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - 0: the specified key or field does not exist.
  - 1: the specified field exists.
  - Otherwise, an exception is returned.

EXHSTRLEN

- Syntax

  EXHSTRLEN <key> <field>

- Time complexity

  O(1)

- Description

  This command is used to retrieve the length of a field value in the TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
field	An element of the TairHash. A TairHash key can be mapped to multiple fields.

- Returned values
  - 0: the specified key or field does not exist.
  - The length of the specified field value is returned if the operation is successful.
  - Otherwise, an exception is returned.

EXHKEYS

- Syntax

  EXHKEYS <key>

- Time complexity

  O(1)

- Description

  This command is used to retrieve all fields in a TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.

- Returned values
  - An empty array is returned if the specified key does not exist.
  - An array is returned if the specified key exists. Each element in the array corresponds to a field.
  - Otherwise, an exception is returned.

EXHVALS

- Syntax

  EXHVALS <key>

- Time complexity

  O(1)

- Description

This command is used to retrieve all field values in the TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.

- Returned values
  - An empty array is returned if the specified key does not exist.
  - An array is returned if the specified key exists. Each element in the array corresponds to a field value.
  - Otherwise, an exception is returned.

EXHGETALL

- Syntax

  EXHGETALL <key>

- Time complexity

  O(1)

- Description

  This command is used to retrieve all fields and their values in a TairHash if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.

- Returned values
  - An empty array is returned if the specified key does not exist.
  - An array is returned if the specified key exists. Each element in the array corresponds to a field value.
  - Otherwise, an exception is returned.

EXHSCAN

- Syntax

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

- Time complexity

  O(1) and O(N)

- Description

  This command is used to scan TairHashes if the key matches the specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairHash that you want to manage.
op	The position from which a scan starts. Valid values: ○ >, specifies that the scan starts from the first field with the key greater than subkey. ○ >=, specifies that the scan starts from the first field with the key greater than or equal to subkey. ○ <, specifies that the scan starts from the first field with the key less than subkey. ○ <=, specifies that the scan starts from the first field with the key less than or equal to subkey. ○ ==, specifies that the scan starts from the first field with the key equal to subkey. ○ ^, specifies that the scan starts from the first field. ○ $, specifies that the scan starts from the last field.
subkey	Specifies the position from which a scan starts. This parameter is specified together with the op parameter. If op is set to ^ or $, this parameter does not take effect.
MATCH	The criteria used to filter the scanning result.

- Returned values
  - An empty array is returned if the specified key does not exist.
  - An array is returned if the specified key exists. Each element in the array corresponds to a field value.
  - Otherwise, an exception is returned.

6.3.5. TairGIS commands

This topic describes the commands supported by TairGISs.

Overview

A TairGIS is a data structure that uses R-tree indexes and supports Geographic Information System (GIS) API operations. The GEO commands of Redis allows you to use GeoHash and Redis SortedSet to query points. Compared with GEO commands, TairGIS allows you to query points, lines, and planes, and provides more features.

Key features:

- Supports R-tree indexing.
- Allows you to query points, lines, and planes (including querying intersection relationships).

Prerequisites

The commands described in this topic take effect only if the following conditions are met:

- A performance-enhanced instance of ApsaraDB for Redis Enterprise Edition is used.
- The TairGIS data to be managed is stored on the performance-enhanced instance.

Commands

TairGIS commands

Command	Syntax	Description
GIS.ADD	GIS.ADD <area> <PolygonName> <POLYGON> [<PolygonName2> <POLYGON> ...]	Adds one or more polygons to a specified area. The polygons are described in well-known text (WKT). You can specify one or more polygons. Note WKT is a text markup language that you can use to represent vector geometry objects on a map and the spatial reference systems of spatial objects. WKT also allows you to perform transformations between spatial reference systems.
GIS.GET	GIS.GET <area> <PolygonName>	Retrieves the WKT information about a polygon in a specified area. The PolygonName parameter specifies the polygon name.
GIS.DEL	GIS.DEL <area> <PolygonName>	Deletes a polygon in a specified area. The PolygonName parameter specifies the polygon name.
GIS.SEARCH	GIS.SEARCH <area> <POINT | LINESTRING | POLYGON>	Queries a polygon that contains a specified point, line, or plane in a specified area. The name and WKT information of the polygon are returned.
GIS.CONTAINS	GIS.CONTAINS <area> <POINT | LINESTRING | POLYGON> [WITHOUTWKT]	Checks whether a polygon in a specified area contains a specified point, line, or plane. If you specify the WITHOUTWKT parameter, the WKT information of the polygon is not returned.
GIS.INTERSECTS	GIS.INTERSECTS <area> <POINT | LINESTRING | POLYGON>	Queries the intersection relationship between a polygon in a specified area and a specified point, line, or plane.
GIS.GETALL	GIS.GETALL <area> [WITHOUTWKT]	Queries all polygons in a specified area. If you specify the WITHOUTWKT parameter, the WKT information of the polygons is not returned.
DEL	DEL <key> [key ...]	Deletes one or more TairGISs.

Parameters

Parameter	Description
area	The geometric area in which you want to manage the data.
PolygonName	The name of the polygon that you want to manage.
POINT	The well-known text (WKT) information that describes a point.
LINESTRING	The WKT information that describes a line.
POLYGON	The WKT information that describes a polygon.

GIS.ADD

- Syntax

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

- Time complexity

  O(log n)

- Description

  This command is used to add one or more polygons to a specified area. The polygons are described in WKT. For example, `'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` . You can specify one or more polygons.

- Returned values
  - 1: the operation is successful.
  - Expected result: If one or more polygon names exist in the area, the specified WKT information overwrites the current WKT information and a value of 1 is returned.
  - Otherwise, an exception is returned.

- Example

```
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.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'
(integer) 0
127.0.0.1:6379> GIS.GET hangzhou campus
"POLYGON((30 10,40 40,20 40,10 20,30 10))"
```

GIS.GET

- Syntax

  GIS.GET <area> <polygonName>

- Time complexity

  O(1)

- Description

  This command is used to retrieve the WKT information about a polygon in a specified area. The PolygonName parameter specifies the polygon name.

- Returned values
  - The WKT information is returned if the operation is successful.
  - nil: the specified area or polygon name does not exist.
  - Otherwise, an exception is returned.

- Example

```
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.DEL

- Syntax

  GIS.DEL <area> <polygonName>

- Time complexity

  O(log n)

- Description

  This command is used to delete a polygon in a specified area. The PolygonName parameter specifies the polygon name.

- Returned values
  - OK: the operation is successful.
  - nil: the specified area or polygon name does not exist.
  - Otherwise, an exception is returned.

- Example

```
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.SEARCH

- Syntax

  GIS.SEARCH <area> <POINT/LINESTRING/POLYGONNAME>

- Time complexity
  - Most desired time complexity: O(logM n).
  - Least desired time complexity: log(n).
- Description

  This command is used to query a polygon that contains a specified point, line, or plane in a specified area. The name and WKT information of the polygon are returned.

- Returned values
  - The WKT information is returned if the operation is successful.
  - nil: no data is found.
  - Otherwise, an exception is returned.
- Example

```
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.SEARCH 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.SEARCH 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.SEARCH 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.CONTAINS

- Syntax

  GIS.CONTAINS <area> <POINT | LINESTRING | POLYGON> [WITHOUTWKT]

- Time complexity
  - Most desired time complexity: O(logM n).
  - Least desired time complexity: log(n).
- Description

  This command is used to check whether a polygon in a specified area contains a specified point, line, or plane.

- Returned values
  - The name and WKT information about a polygon that contains the specified points, lines, or planes are returned if the operation is successful. If you specify the WITHOUTWKT parameter, the WKT information of the polygon is not returned.
  - nil: no data is found.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

  GIS.INTERSECTS <area> <POINT/LINESTRING/POLYGONNAME>

- Time complexity
  - Most desired time complexity: O(logM n).
  - Least desired time complexity: log(n).
- Description

  This command is used to query the intersection relationship between a polygon in a specified area and a specified point, line, or plane.

- Returned values
  - The WKT information of the specified polygon is returned if the operation is successful.
  - nil: no data is found.
  - Otherwise, an exception is returned.
- Example

```
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.GETALL

- Syntax

  GIS.GETALL <area> [WITHOUTWKT]

- Time complexity

  O(n)

- Description

  This command is used to query all polygons in a specified area. If you specify the WITHOUTWKT parameter, the WKT information of the polygons is not returned.

- Returned values
  - The name and WKT information of the polygon are returned if the operation is successful. If you specify the WITHOUTWKT parameter, the WKT information of the polygons is not returned.
  - nil: no data is found.
  - Otherwise, an exception is returned.

- Example

```
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"
127.0.0.1:6379>
```

6.3.6. TairBloom commands

This topic describes the commands supported by TairBlooms.

Overview

TairBloom is a Bloom filter that supports dynamic scaling. TairBloom is a space-efficient probabilistic data structure that consumes minimal memory to check whether an element exists. TairBloom supports dynamic scaling and maintains a stable false positive rate during scaling.

For Redis data structures, such as hashes, sets, and strings, you can use bitmaps to achieve similar features as the TairBloom. However, these data structures may consume a large amount of memory, or fail to maintain a stable false positive rate during dynamic scaling. TairBloom can be used to check whether large volumes of data exists. In this case, a certain false positive rate is allowed. You can use the Bloom filter built in the TairBloom without further development or the need to create an extra Bloom filter.

Key features:

- Consumes minimal memory.
- Enables dynamic scaling.
- Maintains a stable custom false positive rate during scaling.

Prerequisites

The commands described in this topic take effect only if the following conditions are met:

- A performance-enhanced instance of ApsaraDB for Redis Enhanced Edition is used.
- The TairBloom to be managed is stored on the performance-enhanced instance.

Commands

TairBloom commands

Command	Syntax	Description
BF.RESERVE	BF.RESERVE <key> <error_rate> <capacity>	Creates an empty TairBloom filter with a specified capacity. The error_rate parameter specifies the false positive rate of the TairBloom filter.

Command	Syntax	Description
BF.ADD	BF.ADD <key> <item>	Adds an item to a TairBloom filter.
BF.MADD	BF.ADD <key> <item> [item...]	Adds multiple items to a TairBloom filter specified by the key.
BF.EXISTS	BF.EXISTS <key> <item>	Checks whether an item exists in a TairBloom filter specified by the key.
BF.MEXISTS	BF.EXISTS <key> <item> [item...]	Checks whether multiple items exist in a TairBloom filter.
BF.INSERT	BF.INSERT <key> [CAPACITY cap] [ERROR error] [NOCREATE] ITEMS <item... >	Adds multiple items to a TairBloom filter. You can specify the capacity and false positive rate and specify whether to create a TairBloom filter if the TairBloom filter does not exist.
BF.DEBUG	BF.DEBUG <key>	Retrieves the information about a TairBloom filter. The information includes the number of layers, the number of items at each layer, and the false positive rate.
DEL	DEL <key> [key ...]	Deletes one or more TairBlooms. Note The items added to a TairBloom cannot be deleted from the TairBloom. You can run the DEL command to delete the TairBloom.

BF.RESERVE

- Syntax

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

- Time complexity

O(1)

- Description

  This command is used to create an empty TairBloom filter with a specified capacity. The error_rate parameter specifies the false positive rate of the TairBloom filter.

- Parameters and options

Parameter/option	Description
key	The key of the TairBloom filter that you want to manage.
error_rate	The required false positive rate that must be between 0 and 1. A lower value indicates higher memory and CPU usage by the TairBloom filter.
capacity	The initial capacity of the TairBloom filter. This is the maximum number of items that can be added to the TairBloom filter. If the number of items that have been added to the TairBloom filter exceeds the specified capacity, TairBloom expands the capacity by increasing the layers of the Bloom filter. During the scaling process, the number of items in the Tairbloom filter is increased exponentially while the performance is deceased on a linear scale. After a layer is added to the filter, to query a specified item, TairBloom may iterate through multiple layers of the filter. The capacity of each new layer is double that of the previous layer. If your workloads require high performance, we recommend that you add proper number of items to the TairBloom to avoid automatic scaling.

- Returned values
  - Expected result: OK.
  - An error message is returned for other unexpected results each.

BF.ADD

- Syntax

  BF.ADD <key> <item>

- Time complexity

  O(log N). N specifies the number of layers of the TairBloom.

- Description

  This command is used to add an item to a TairBloom filter.

- Parameters and options

Parameter/option	Description
key	The key of the TairBloom filter that you want to manage.
item	The item that you want to add to the TairBloom filter.

- Returned values

- 1: the specified item does not exist in the filter.
- 0: the specified item may exist in the filter.
- An error message is returned for other unexpected results each.

BF.MADD

- Syntax

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

- Time complexity

  O(log N). N specifies the number of layers of the TairBloom.

- Description

  This command is used to add multiple items to a TairBloom filter specified by the key.

- Parameters and options

Parameter/option	Description
key	The key of the TairBloom filter that you want to manage.
item	The items that you want to add to the TairBloom filter. You can specify multiple items.

- Returned values
  - Expected result: An array is returned. In the returned array, each value can be 1 or 0. If a specified item does not exist, the value is 1. If a specified item may exist, the value is 0.
  - An error message is returned for other unexpected results each.

BF.EXISTS

- Syntax

  BF.EXISTS <key> <item>

- Time complexity

  O(log N). N specifies the number of layers of the TairBloom.

- Description

  This command is used to check whether an item exists in a TairBloom filter specified by the key.

- Parameters and options

Parameter/option	Description
key	The key of the TairBloom filter that you want to manage.
item	The item that you want to query.

- Returned values

- 0: the specified item does not exist in the filter.
- 1: the specified item may exist in the filter.
- An error message is returned for other unexpected results each.

BF.MEXISTS

- Syntax

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

- Time complexity

  O(log N). N specifies the number of layers of the TairBloom.

- Description

  This command is used to check whether multiple items exist in a TairBloom filter.

- Parameters and options

Parameter/option	Description
key	The key of the TairBloom filter that you want to manage.
item	The items that you want to query in the TairBloom filter. You can specify multiple items.

- Returned values
  - Expected result: An array is returned. In the returned array, each value can be 1 or 0. If a specified item does not exist, the value is 0. If a specified item may exist, the value is 1.
  - An error message is returned for other unexpected results each.

BF.INSERT

- Syntax

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

- Time complexity

  O(log N). N specifies the number of layers of the TairBloom.

- Description

  This command is used to add multiple items to a TairBloom filter. You can specify the capacity and false positive rate and specify whether to create a TairBloom filter if the TairBloom filter does not exist.

- Parameters and options

Parameter/option	Description
key	The key of the TairBloom filter that you want to manage.

Parameter/option	Description
CAPACITY	The initial capacity of the TairBloom filter. This is the maximum number of items that can be added to the TairBloom filter. If the filter exists, you do not need to specify this parameter. If the number of items that have been added to the TairBloom filter exceeds the specified capacity, TairBloom expands the capacity by increasing the layers of the Bloom filter. During the scaling process, the number of items in the Tairbloom filter is increased exponentially while the performance is deceased on a linear scale. After a layer is added to the filter, to query a specified item, TairBloom may iterate through multiple layers of the filter. The capacity of each new layer is double that of the previous layer. If your workloads require high performance, we recommend that you add proper number of items to the TairBloom to avoid automatic scaling.
ERROR	The required false positive rate of the TairBloom filter. If the filter exists, you do not need to specify this parameter. The value must be between 0 and 1. A lower value indicates higher memory and CPU usage by the TairBloom filter.
NOCREATE	Specifies that a TairBloom filter is not automatically created if the filter does not exist. This parameter cannot be specified together with CAPACITY or ERROR.
ITEMS	All items that you want to add to the TairBloom filter.

- Returned values
  - Expected result: An array is returned. In the returned array, each value can be 1 or 0. If a specified item does not exist, the value is 1. If a specified item may exist, the value is 0.
  - An error message is returned for other unexpected results each.

BF.DEBUG

- Syntax

  BF.DEBUG <key>

- Time complexity

  O(log N). N specifies the number of layers of the TairBloom.

- Description

  This command is used to retrieve the information about a TairBloom filter. The information includes the number of layers, the number of items at each layer, and the false positive rate.

- Parameters and options

Parameter/option	Description
key	The key of the TairBloom filter that you want to manage.

- Returned values

- Expected result: An array is returned. In the returned array, each value can be 1 or 0. If a specified item does not exist, the value is 1. If a specified item may exist, the value is 0.
- An error message is returned for other unexpected results each.

Memory usage test result

Capacity	false positive:0.01	false positive:0.001	false positive:0.0001
100000	0.12MB	0.25MB	0.25MB
1000000	2MB	2MB	4MB
10000000	16MB	32MB	32MB
100000000	128MB	256MB	256MB
1000000000	2GB	2GB	4GB

6.3.7. TairDoc commands

This topic describes the commands supported by TairDocs.

Overview

A TairDoc is a document data structure. You can use TairDoc to add, modify, query, or delete JavaScript Object Notation (JSON) data.

Key features:

- Supports JSON standards.
- Fully compatible with RedisJSON.
- Supports the syntax of JSONPath and JSON Pointer.
- Stores data in a binary tree and simplifies the retrieval of child elements.
- Supports conversion from the JSON format to the Extensible Markup Language (XML) or YAML Ain't Markup Language (YAML) format.

Prerequisites

The commands described in this topic take effect only if the following conditions are met:

- A performance-enhanced instance of ApsaraDB for Redis Enterprise Edition is used.
- The TairDoc to be managed is stored on the performance-enhanced instance.

Commands

TairDoc commands

Command	Syntax	Description

Command	Syntax	Description
JSON.SET	JSON.SET <key> <path> <json> [NX or XX]	Writes a JSON value to a TairDoc path of a specified key. If the specified key does not exist, the path must be the root directory. If the specified key and path exist, the specified JSON value overwrites the current JSON value in the path.
JSON.GET	JSON.GET <key> [PATH] [FORMAT <XML/YAML>] [ROOTNAME <root>] [ARRNAME <arr>]	Retrieves JSON data from a TairDoc path of a specified key.
JSON.DEL	JSON.DEL <key> [path]	Deletes JSON data from a TairDoc path of a specified key. If the path is not specified, the key is deleted. This command does not take effect if the key or path does not exist.
JSON.TYPE	JSON.TYPE <key> [path]	Retrieves the type of JSON data from a TairDoc path of a specified key.
JSON.NUMINCRBY	JSON.NUMINCRBY <key> [path] <value>	Increases JSON data in a TairDoc path by a specified value. The path must exist, and both the JSON data and increased value must be of int or double type.
JSON.STRAPPEND	JSON.STRAPPEND <key> [path] <json-string>	Appends a string specified in json-string to the end of the string in a TairDoc path. If you do not specify the path, the root directory is used.
JSON.STRLEN	JSON.STRLEN <key> [path]	Retrieves the JSON value length in a TairDoc path. If you do not specify the path, the root directory is used.
JSON.ARRAPPEND	JSON.ARRAPPEND <key> <path> <json> [<json> ...]	Appends one or more JSON values to the end of an array in a TairDoc path.
JSON.ARRPOP	JSON.ARRPOP <key> <path> [index]	Removes an element specified by index from an array in a specified TairDoc path and return the removed element.

Command	Syntax	Description
JSON.ARRINSERT	JSON.ARRINSERT <key> <path> <index> <json> [<json> ...]	Adds one or more JSON elements to an array in a TairDoc path. The index parameter specifies the position to which the JSON elements are added.
JSON.ARRLEN	JSON.ARRLEN <key> [path]	Retrieves the length of the array in a TairDoc path.
JSON.ARRTRIM	JSON.ARRTRIM <key> <path> <start> <stop>	Trims a JSON array in a TairDoc path. The start value and the stop value specify the range in which the JSON data is retained.
DEL	DEL <key> [key ...]	Deletes one or more TairDocs.

JSON.SET

- Syntax

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

- Time complexity

  O(N)

- Description

  This command is used to write a JSON value to the path of a specified key. If the specified key does not exist, the path must be the root directory. If the specified key and path exist, the specified JSON value overwrites the current JSON value in the path.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data. ◦ If the specified key does not exist, the path must be the root directory. ◦ If the specified key and path exist, the specified JSON value overwrites the current JSON value in the path.
json	If the specified key and path exist, the specified JSON value overwrites the current JSON value in the TairDoc.
NX	Specifies that a JSON value is written only if the required path does not exist.
XX	Specifies that a JSON value is written only if the required path exists.

- Returned values

- OK: the operation is successful.
- null: The operation fails. This occurs when you specify the NX or XX parameter.
- Otherwise, an exception is returned.

- Example

```
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

- Syntax

  JSON.GET <key> <path> [FORMAT <XML | YAML>] [ROOTNAME <root>] [ARRNAME <arr>]

- Time complexity

  O(N)

- Description

  This command is used to retrieve JSON data from a TairDoc path of a specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.
FORMAT	The format of the JSON data to be returned. Valid values: XML and YAML.
ROOTNAME	The tag that specifies a root element in an XML document.
ARRNAME	The tag that specifies an array element in an XML document.

- Returned values
  - The JSON data stored in the path is returned if the operation is successful.
  - Otherwise, an exception is returned.

- Example

```
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

- Syntax

  JSON.DEL <key> [path]

- Time complexity

  O(N)

- Description

  This command is used to delete JSON data from a TairDoc path of a specified key. If the path is not specified, the key is deleted. This command does not take effect if the key or path does not exist.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.

- Returned values
  - 1: the operation is successful.
  - 0: the operation fails.
  - Otherwise, an exception is returned.

- Example

```
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

- Syntax

  JSON.TYPE <key> [path]

- Time complexity

  O(N)

- Description

  This command is used to retrieve the type of JSON data from a TairDoc path of a specified key.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.

- Returned values
  - The type of JSON data is returned if the operation is successful. The type includes boolean, null, number, string, array, object, raw, reference, or const.
  - null: the specified key or path does not exist.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

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

- Time complexity

  O(N)

- Description

  This command is used to increase JSON data in a TairDoc path by a specified value. The path must exist, and the JSON data and increased value must be both of the type of int or double.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.
value	The increment to be added to the JSON data in the specified path.

- Returned values
  - The increased value in the specified path is returned if the operation is successful.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

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

- Time complexity

  O(N)

- Description

  This command is used to append a string specified in json-string to the end of the string in a TairDoc path. If you do not specify the path, the root directory is used.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.
json-string	The string to be appended to the specified path.

- Returned values
  - The length of the increased value in the path is returned if the operation is successful.
  - -1: the specified key does not exist.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

  JSON.STRLEN <key> [path]

- Time complexity

  O(N)

- Description

  This command is used to retrieve the JSON value length in a TairDoc path. If you do not specify the path, the root directory is used.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.

- Returned values
  - The length of the value in the path is returned if the operation is successful.
  - -1: the specified key does not exist.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

  JSON.ARRAPPEND <key> <path> <json> [<json> ...]

- Time complexity

  O(M×N). M specifies the number of JSON elements to be appended and N specifies the number of elements in the array.

- Description

  This command is used to append one or more JSON values to the end of an array in a TairDoc path.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.
json	The JSON value to be appended to a specified array.

- Returned values
  - The number of elements in the array is returned if the operation is successful. The added elements are included.
  - -1: the specified key does not exist.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

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

- Time complexity

  O(M×N). M specifies the child elements that the specified key contains and N specifies the number of elements in the array.

- Description

  This command is used to remove an element specified by index from an array in a specified TairDoc path and return the removed element.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.
index	The index of the array, which specifies the value to be removed. If you do not specify this parameter, the last value in the array is removed. A negative value specifies reverse numbering from the end of the array.

- Returned values
  - The removed element is returned if the operation is successful.
  - An error message is returned if the array is empty: 'ERR array index outflow'.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

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

- Time complexity

  O(M×N). M specifies the number of JSON elements to be appended and N specifies the number of elements in the array.

- Description

  This command is used to add one or more JSON elements to an array in a TairDoc path. The index parameter specifies the position to which the JSON elements are added.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.
index	The index of the array, which specifies the value to be removed. If you do not specify this parameter, the last value in the array is removed. A negative value specifies reverse numbering from the end of the array.
json	The JSON value to be inserted to a specified array.

- Returned values
  - The number of elements in the array is returned if the operation is successful. The added elements are included.
  - An error message is returned if the array is empty: 'ERR array index outflow'.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

  JSON.ARRLEN <key> [path]
- Time complexity

  O(N)
- Description

  This command is used to retrieve the length of the array in a TairDoc path.
- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.

- Returned values
  - The length of the queried array is returned if the operation is successful.
  - -1: the specified key does not exist.
  - Otherwise, an exception is returned.
- Example

```
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

- Syntax

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

- Time complexity

  O(N)

- Description

  This command is used to trim a JSON array in a TairDoc path. The start value and the stop value specify the range in which the JSON data is retained.

- Parameters and options

Parameter/option	Description
key	The key of the TairDoc that you want to manage.
path	The TairDoc path where you want to manage JSON data.
start	The start of the range in which elements are retained after a trim. The value is an index that starts from 0. The element at the start position is retained.
stop	The end of the range in which elements are retained after a trim. The value is an index that starts from 0. The element at the end position is retained.

- Returned values
  - The length of the trimmed array is returned if the operation is successful.
  - -1: the specified key does not exist.
  - Otherwise, an exception is returned.
- Example

```
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>
```

JSON Pointer and JSONPath

TairDoc supports the JSONPointer syntax and also supports some of the JSONPath syntax. The following table shows the syntax examples.

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

The following table shows how TairDoc supports JSONPath and JSON Pointer.

Compatible item	JSONPath	JSONPointer
Root element	.	""
An individual element in a path	.a.b.c	/a/b/c
Array	.a[2]	/a/2
Multiple elements in a path	.a["b.c"]	/a/b.c
Multiple elements in a path	.a['b.c']	/a/b.c

7.Version description

7.1. Feature updates of ApsaraDB for Redis 5.0

ApsaraDB for Redis 5.0 significantly optimizes kernel performance and system stability. It currently supports new features, such as the Redis Streams data type, account management, and audit logging, to meet the needs in diverse scenarios.

Update description

- Supports a new data type: Redis Streams. For more information, see Redis Streams.
- Supports Manage database accounts.
- Supports the log management feature. You can manage audit logs, operational logs, and slow logs. You can query the records of commands used to manage databases, read/write operations, and sensitive operations such as KEYS and FLUSHALL operations. You can also monitor slow logs.
- Supports timer, cluster, and dictionary APIs.
- Supports the least frequently used (LFU) and least recently used (LRU) caching strategies for Redis database (RDB) files.
- Supports the Redis command line interface (redis-cli) instead of redis-trib.rb to manage clusters. You need to write command code by using C language instead of Ruby language.
- Supports the following commands for sorted sets: ZPOPMIN, ZPOPMAX, BZPOPMIN, and BZPOPMAX.
- Supports Active Defragmentation v2.
- Supports enhanced performance of HyperLogLog.
- Supports optimized statistical reports of the memory.
- Supports the HELP subcommand for various commands that can include subcommands.
- Supports performance stability when connections between databases and clients are frequently disabled and enabled.
- Supports Jemalloc 5.1.0.
- Supports the CLIENT ID and CLIENT UNBLOCK commands.
- Supports the LOLWUT command that is used to produce interesting outputs.
- Discards the term "slave" in all scenarios unless you need to ensure backward compatibility of APIs.
- Optimizes the network layer.
- Improves Lua script-related configuration.
- Supports the dynamic-hz parameter to optimize CPU usage and response performance.
- Reconstructs and improves Redis core code.

> **Note** ApsaraDB for Redis 5.0 only supports the Standard Edition. We are working on support for other editions.

7.2. Features of engine version 4.0 of ApsaraDB for Redis

Alibaba Cloud has developed engine version 4.0 of ApsaraDB for Redis based on Redis 4.0 and fixed several bugs to provide you with excellent performance. Engine version 4.0 of ApsaraDB for Redis has all benefits of engine version 2.8 of ApsaraDB for Redis, and supports the following features.

Lazyfree

Engine version 4.0 supports the Lazyfree feature. This feature can avoid congestion on Redis-server caused by the `DEL` , `FLUSHDB` , `FLUSHALL` and `RENAME` commands and ensure service stability. This feature is described as follows.

UNLINK

For engine versions earlier than 4.0, when ApsaraDB for Redis runs the `DEL` command, the command returns `OK` only after releasing the memory of the target key. If the key contains large amounts of data, for example, 10 million items of data in a hash table, other connections have to wait a long time. To be compatible with the existing `DEL` syntax, engine version 4.0 uses the `UNLINK` command. The UNLINK command has the same effect and usage as the `DEL` command, but the background thread releases memory when engine version 4.0 runs the UNLINK command.

```
UNLINK key [key ...]
```

FLUSHDB/FLUSHALL

The `FLUSHDB` and FLUSHALL commands in engine version 4.0 allow you to specify whether to use the Lazyfree feature to clear all memory.

```
FLUSHALL [ASYNC]
FLUSHDB [ASYNC]
```

RENAME

When ApsaraDB for Redis runs the `RENAME OLDKEY NEWKEY` command, if the specified new key already exists, ApsaraDB for Redis deletes the existing new key first. If the key contains large amounts of data, other connections have to wait a long time. To use the Lazyfree feature to delete the key in ApsaraDB for Redis, apply the following configuration in the console:

```
lazyfree-lazy-server-del yes/no
```

> **Note**
>
> This parameter is not available in the console.

Expire or evict data

You can specify data expiration time and allow ApsaraDB for Redis to delete expired data. However, when ApsaraDB for Redis deletes a large expired key, CPU jitter may occur. Engine version 4.0 allows you to specify whether to use the Lazyfree feature to expire or evict data.

```
lazyfree-lazy-eviction yes/no
lazyfree-lazy-expire yes/no
```

New commands

SWAPDB

The `SWAPDB` command is used to exchange data between two databases. After ApsaraDB for Redis runs the `SWAPDB` command, you can connect to the target database and check new data directly without running the `SELECT` command.

```
127.0.0.1:6379> select 0
OK
127.0.0.1:6379> set key value0
OK
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> set key value1
OK
127.0.0.1:6379[1]> swapdb 0 1
OK
127.0.0.1:6379[1]> get key
"value0"
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> get key
"value1"
```

ZLEXCOUNT

The `ZLEXCOUNT` command is used for sorted sets and similar to the `ZRANGEBYLEX` command. However, the `ZRANGEBYLEX` command returns the target members, and the `ZLEXCOUNT` command returns the number of target members.

MEMORY

Engine versions earlier than 4.0 support the `INFO MEMORY` command to provide limited memory information. Engine version 4.0 allows you to use the `MEMORY` command to obtain comprehensive memory status of ApsaraDB for Redis.

```
127.0.0.1:6379> memory help
1) "MEMORY DOCTOR                - Outputs memory problems report"
2) "MEMORY USAGE <key> [SAMPLES <count>] - Estimate memory usage of key"
3) "MEMORY STATS                 - Show memory usage details"
4) "MEMORY PURGE                 - Ask the allocator to release memory"
5) "MEMORY MALLOC-STATS          - Show allocator internal stats"
```

- `MEMORY USAGE`

  The `USAGE` child command is used to check the memory usage of a specified key in ApsaraDB for Redis.

  > **Notice**
  > - Key-value pairs in ApsaraDB for Redis use memory. ApsaraDB for Redis also uses memory when managing these key-value pairs.
  > - The memory usage of keys such as hash tables, lists, sets, and sorted sets is calculated based on sampling. The `SAMPLES` command controls the number of samples.

- `MEMORY STATS`

```
27.0.0.1:6379> memory stats
 1) "peak.allocated"    // The maximum memory that ApsaraDB for Redis has used since startup.
 2) (integer) 423995952
 3) "total.allocated"    //The current memory usage.
 4) (integer) 11130320
 5) "startup.allocated"    //The memory that ApsaraDB for Redis uses after startup and initializati
on.
 6) (integer) 9942928
 7) "replication.backlog"    //The memory of the backlog used in resuming an interrupted master-r
eplica replication. Default value: 10 MB.
 8) (integer) 1048576
 9) "clients.slaves"    // The memory used in a master-replica replication.
10) (integer) 16858
11) "clients.normal"    //The memory used by read and write buffers for common clients.
12) (integer) 49630
13) "aof.buffer"    //The sum of the cache used for append-only file (AOF) persistence and the ca
che generated during the AOF rewrite operation.
14) (integer) 3253
15) "db. 0"    //The memory used by metadata in each database.
16) 1) "overhead.hashtable.main"
   2) (integer) 5808
   3) "overhead.hashtable.expires" //The memory used for managing data that has TTL configur
ed.
   4) (integer) 104
17) "overhead.total"    //The total memory usage for the preceding items.
18) (integer) 11063904
19) "keys.count"    //The total number of keys in the current storage.
20) (integer) 94
21) "keys.bytes-per-key"    //The average size of each key in the current memory.
22) (integer) 12631
23) "dataset.bytes"       //The memory used by user data (= Total memory - Memory used by met
adata of ApsaraDB for Redis).
24) (integer) 66416
25) "dataset.percentage"    //100 * dataset.bytes / (total.allocated - startup.allocated)
26) "5.5934348106384277"
27) "peak.percentage"    // 100 * total.allocated / peak_allocated
28) "2.6251003742218018"
29) "fragmentation"    //The memory fragmentation ratio.
30) "1.1039986610412598"
```

- `MEMORY DOCTOR`

  This command is used to provide diagnostics and indicate hidden issues.

  ```
  Peak memory: peak.allocated/total.allocated > 1.5. This indicates a possibly high memory fragmenta
  tion ratio.
   High fragmentation: fragmentation > 1.4. This indicates a high memory fragmentation ratio.
   Big slave buffers: the average memory for each replica buffer is more than 10 MB. This may be cau
  sed by high traffic of write operations on the master node.
   Big client buffers: the average memory for a common client buffer is more than 200 KB. This may be
  caused by the improper use of pipelining or caused by Pub/Sub clients that delay processing messa
  ges.
  ```

- `MEMORY STATS` and `MALLOC PURGE`

  Both commands are valid only when you use jemalloc.

Least Frequently Used (LFU) mechanism and hotkeys

Engine version 4.0 supports allkey-lfu and volatile-lfu data eviction policies, and allows you to use the `OBJECT` command to obtain the frequency that a specified key is used.

```
object freq user_key
```

Based on the LFU mechanism, you can use a combination of the `SCAN` and OBJECT FREQ commands to find hotkeys. You can also use the Redis command line interface (redis-cli) program as follows:

```
$./redis-cli --hotkeys
# Scanning the entire keyspace to find hot keys as well as
# average sizes per key type.  You can use -i 0.1 to sleep 0.1 sec
# per 100 SCAN commands (not usually needed).
[00.00%] Hot key 'counter:000000000002' found so far with counter 87
[00.00%] Hot key 'key:000000000001' found so far with counter 254
[00.00%] Hot key 'mylist' found so far with counter 107
[00.00%] Hot key 'key:000000000000' found so far with counter 254
[45.45%] Hot key 'counter:000000000001' found so far with counter 87
[45.45%] Hot key 'key:000000000002' found so far with counter 254
[45.45%] Hot key 'myset' found so far with counter 64
[45.45%] Hot key 'counter:000000000000' found so far with counter 93
-------- summary -------
Sampled 22 keys in the keyspace!
hot key found with counter: 254 keyname: key:000000000001
hot key found with counter: 254 keyname: key:000000000000
hot key found with counter: 254 keyname: key:000000000002
hot key found with counter: 107 keyname: mylist
hot key found with counter: 93  keyname: counter:000000000000
hot key found with counter: 87  keyname: counter:000000000002
hot key found with counter: 87  keyname: counter:000000000001
hot key found with counter: 64  keyname: myset
```

7.3. ApsaraDB for Redis 4.0 release notes

This topic describes the feature updates in ApsaraDB for Redis 4.0.

Updates in 2019

Date	Description
July 30, 2019	• Physical IP addresses are not restricted by the value of the maxclients parameter.
July 08, 2019	• The following issue is fixed: The memory used by audit logs is released repeatedly.
July 04, 2019	• The RPOPLPUSH coredump issue is fixed. • The latency record is added for event looping.
June 20, 2019	• The audit log feature is optimized to prevent the loss of operations logs and frequent startup of the bio_audit thread.

Date	Description
June 13, 2019	• Key expiration and eviction are optimized. • When password-free access is enabled, you still need to pass password verification if you access ApsaraDB for Redis instances over the public network. • The DEL command is replaced with the UNLINK command.
May 05, 2019	• The page cache is released automatically after the AOF file is written or the RDB or AOF file is loaded. • The master role is distinguished from the replica role, and management connections are distinguished from user connections in audit logs.
March 11, 2019	• The following issue is fixed: The connection counter for a database in a cluster does not work properly in some scenarios.
February 21, 2019	• The following issue is fixed: Slow logs are not collected in some scenarios.
January 24, 2019	• Jemalloc is upgraded from 4.0.3 to 5.0.
January 24, 2019	• The audit log feature is supported. This feature records the write operations and other sensitive operations in logs. • Multithreading protection is added for some resources, and resource isolation is optimized. • The exception handling feature of Redis is optimized to make sure that exception logs are generated properly. • The following issue is fixed: An exception occurs due to a null pointer when you run commands with long parameters.
January 07, 2019	• The issue of abnormal log rotation is fixed. • The exception of the slow log feature caused by an initialization error is fixed.

Updates in 2018

Date	Description
December 11, 2018	• The debug assert command is optimized. • The following issue is fixed: The master node is affected when a replica client runs commands. • The issue that may cause memory leakage is fixed.

Date	Description
December 11, 2018	• The ROLE command is available, which is used to view the role of the current node. • The SENTINEL command is available.
November 15, 2018	• Issues related to initializing and writing the AOF file are fixed.
October 17, 2018	• If the check-whitelist-always parameter is set to yes for an ApsaraDB for Redis instance, the whitelist is checked when other services in the same VPC connect to the instance even if password-free access is enabled. • The following issue is fixed: The output of the INFO command is abnormal.
October 10, 2018	• The configuration items related to server startup are optimized. • The system checks whether the AOF file list is NULL when obtaining the last AOF binlog during startup.
September 28, 2018	• The timeout issue for status management during cluster migration is fixed.
September 28, 2018	• The following issue is fixed: An exception occurs under special circumstances when the BRPOPLPUSH command in the AOF file is loaded during startup.

7.4. ApsaraDB for Redis 5.0 release notes

This topic describes the feature updates in ApsaraDB for Redis 5.0.

Updates in 2019

Date	Description
June 30, 2019	• Physical IP addresses are not restricted by the value of the maxclients parameter. • When password-free access is enabled, you still need to pass password verification if you access ApsaraDB for Redis instances over the public network.
June 20, 2019	• The adverse impacts on the expiration and eviction algorithms when the number of databases reaches 256 are reduced.

Date	Description
March 22, 2019	• Compatible with open-source Redis 5.0.3. • The features of ApsaraDB for Redis 4.0 are supported. • The following client connection types are supported: dbaas, user, and slave. • The memory leakage issue that occurs at an extremely low probability is fixed. • The print issue is fixed for keys that are larger than 128 bytes.

8.Disaster recovery

Data serves as a core element for most businesses. Databases are used to store data and play a key role in data management. ApsaraDB for Redis is a high-availability (HA) key-value database service. This service helps you store large amounts of important data in various scenarios. This topic describes the disaster recovery solutions provided by ApsaraDB for Redis.

Evolution of disaster recovery solutions

A variety of problems may occur during data management, such as software bugs, device malfunctions, or power failures at data centers. A disaster recovery solution guarantees data consistency and service availability. ApsaraDB for Redis provides optimized disaster recovery solutions to achieve high availability in different scenarios.

The following figure shows how the disaster recovery solutions have evolved.

Disaster recovery solutions

All these solutions are available in ApsaraDB for Redis to meet different requirements. The following sections describe these solutions in details.

Single-zone high availability

All ApsaraDB for Redis instances support a single-zone HA architecture. The HA system runs on an independent platform to guarantee high availability across zones. Compared with on-premises Redis databases, ApsaraDB for Redis enables more stable database management.

Standard master-replica instance

A standard master-replica instance runs in a master-replica architecture. If the HA system detects a failure on the master node, the system switches the workloads from the master node to the replica node and the replica node takes over the role of the master node. The original master node works as the replica node after recovery. By default, data persistence is enabled for the instance. The system automatically creates data backups on the instance. You can use the backups to roll back or clone the instance. This mechanism avoids data loss caused by user mistakes, and enables data reliability and disaster recovery.

High-availability solution of a standard master-replica instance

Master-replica cluster instance

A master-replica cluster instance consists of a configuration server, multiple proxy servers, and multiple data shards.

- The configuration server is a cluster management tool that provides global routing and configuration information. This server uses a cluster architecture with three nodes and follows the Raft protocol.
- A proxy server runs in a standalone architecture. A cluster contains multiple proxy servers. The cluster automatically balances loads and performs failovers among these proxy servers.
- A data shard runs in a master-replica high-availability architecture. Similar to a standard master-replica instance, if the master node fails, the HA system performs a failover to ensure high availability, and updates the information on the proxy servers and configuration server.

High-availability solution of a master-replica cluster

Zone-disaster recovery

Standard instances and cluster instances support zone-disaster recovery across two data centers. If your workloads are deployed in a single region and require disaster recovery, you can select the zones that support zone-disaster recovery when you create an ApsaraDB for Redis instance. For example, you can select **China (Hangzhou) Zone (B+F)** or **China (Hangzhou) (G+H)** from the Zone drop-down list in the console.

Create a zone-disaster recovery instance

When you create a multi-zone instance, the master node and replica node are deployed in different zones and provided with the same specifications. The master node synchronizes data to the replica node through a dedicated channel.

If a power failure or a network error occurs on the master node, the replica node takes over the role of the master node. The system calls an API operation on the configuration server to update routing information for proxy servers. The underlying network performs a failover based on the precision of the routing information available in a backbone network. The master node provides more precise Classless Inter-Domain Routing (CIDR) blocks than the replica node. In normal conditions, the system transmits requests to the master node through precise CIDR blocks. If the master node fails, the master node does not upload routing information to the backbone network. The backbone network only provides less precise CIDR blocks of the replica node. The system routes requests to the replica node according to the available routing information.

ApsaraDB for Redis provides an optimized Redis synchronization mechanism. Similar to global transaction identifiers (GTIDs) of MySQL, ApsaraDB for Redis uses global operation identifiers (OpIDs) to indicate synchronization offsets and runs lock-free threads in the background to search OpIDs. The system synchronizes append-only file (AOF) binary logs (binlogs) asynchronously from the master node to the replica node. You can throttle the synchronization to ensure service performance.

Cross-region disaster recovery

ApsaraDB for Redis supports the Redis Global Replica solution. You can use a global replica instance of ApsaraDB for Redis to run multiple child instances simultaneously across regions worldwide. These child instances exchange data in real time. Different from earlier disaster recovery solutions, child instances work as master nodes at the same time in this solution.

> **Note** Redis Global Replica is tested on the Alibaba Cloud China site. Other Alibaba Cloud sites do not support this solution.

The global replica instance of ApsaraDB for Redis consists of multiple child instances, multiple synchronization channels, and a channel manager.

- A child instance is a basic service unit on the global replica instance. All child instances can process read and write requests.
- The synchronization channels enable two-way synchronizations between child instances in real time. If a synchronization is interrupted, the system can resume the synchronization from the last breakpoint within a few days after the interruption.
- The channel manager controls the lifecycle of the synchronization channels. If a child instance fails, the channel manager switches the workloads from the failed child instance to another child instance, and creates a new child instance. This failover process guarantees high

availability for your workloads.

> **Note** The global replica instance synchronizes data asynchronously among child instances to minimize the impact on the service performance.

When you manage the global replica instance, you can set the failover feature on your application. If a failure occurs on a child instance in a region, the system switches the workloads from the failed child instance to a child instance in another region to ensure service availability.

ApsaraDB for Redis provides multiple disaster recovery solutions to enable instance-level, zone-level, and region-level high availability. You can select a solution to meet business requirements.

9.Features

ApsaraDB for Redis supports multiple architectures, persistent data storage, high availability, auto scaling, and intelligent operations and maintenance.

Flexible architecture

- Dual-node hot standby architecture

  The system synchronizes data between the master node and replica node in real time. If the master node fails, the system automatically fails over to the replica node and restores services within a few seconds. The replica node takes over services. The master-replica architecture ensures high availability of system services. For more information, see Standard master-replica instances.

- Cluster architecture

  Cluster instances run in a distributed architecture. Each node uses a high-availability master-replica structure to automatically perform the failover and disaster recovery. Multiple types of cluster instances are applicable to various businesses. You can scale the database to improve performance as needed. For more information, see Cluster master-replica instances.

- Read/write splitting architecture

  Read/write splitting instances consist of proxy servers, master-replica nodes, and read replicas. Read/write splitting instances provide high availability (HA) and high performance. Multiple specifications are available. The read/write splitting architecture allows a large number of concurrent requests to read hot data from read replicas. This mechanism can reduce the load on the master node and minimize operation and maintenance costs. For more information, see Read/write splitting instances.

Data security

- Persistent data storage

  Based on the hybrid storage of memory and hard disks, ApsaraDB for Redis provides high-speed data read/write capability and enables data persistence.

- Replication and easy recovery

  The system automatically replicates data every day and provides the powerful disaster recovery solution. You can restore data in case of accidental data operations to minimize your business losses.

- Multi-layer network security protection
  - A Virtual Private Cloud (VPC) network isolates network transmission at the transport layer.
  - Anti-DDoS monitors and protects against Distributed-Denial-of-Service (DDoS) attacks.
  - The system supports more than 1,000 IP whitelists configured to control access risks from requests.
  - Password authentication ensures secure and reliable access.

- In-depth kernel optimization

  Alibaba Cloud has performed in-depth engine optimization for the Redis source code to prevent running out of memory, fix security vulnerabilities, and protect your business.

High availability

- Master-replica structure

  Standard and cluster instances support the master-replica structure. They can prevent service interruptions caused by a single point of failure (SPOF).

- Automatic failure detection and recovery

  The system automatically detects hardware failures. In the case of failures, the system performs the failover operation and restores services within a few seconds.

- Resource isolation

  Instance-level resource isolation: ApsaraDB for Redis isolates resources among different instances. A failure on an instance does not affect other instances. This improves the service stability. Instance-level resource isolation is a better measure to ensure stability of individual services.

Scalability

- Data capacity scaling

  ApsaraDB for Redis supports multiple types of memory. You can upgrade the memory based on your needs.

- Performance scaling

  The cluster architecture supports auto scaling of the storage space and throughput performance of the database system. This can eliminate the performance bottlenecks caused by large amounts of data and high-QPS requirements. Therefore, you can easily handle millions of read and write requests per second.

Intelligent O&M

- Monitoring platform

  This platform provides real-time monitoring information about the CPU usage, connections, and disk utilization, and generates alerts. This allows you to learn about the up-to-date instance status.

- Graphical O&M platform

  This platform allows you to perform frequent and high risk operations, such as instance cloning, backup, and data restoration with one click.

- Visualized DMS platform

  The specialized data management service (DMS) platform supports visual data management. This improves the efficiency of comprehensive research and development (R&D) and operations and maintenance (O&M) efforts.

- Database kernel version management

  ApsaraDB for Redis supports automatic upgrades to fix vulnerabilities. This way, you do not have to manage each kernel version. Parameter settings are optimized to maximize the utilization of system resources.

10.Scenarios

This topic describes how to use ApsaraDB for Redis in different scenarios to meet diverse business demands, especially for high concurrency scenarios.

Gaming

ApsaraDB for Redis can serve as an important architecture component in the gaming industry.

- Scenario 1: use ApsaraDB for Redis as a storage service

  Gaming applications can be deployed in a simple architecture, where the main program runs an Elastic Compute Service (ECS) instance, and the business data is stored in ApsaraDB for Redis. ApsaraDB for Redis can be used for persistent storage. It uses a master-replica deployment model to implement redundancy.

- Scenario 2: use ApsaraDB for Redis as a caching service to accelerate connections to applications

  ApsaraDB for Redis can serve as a caching service to accelerate connections to applications. Data is only stored on backend databases (RDS instances).

The high availability of ApsaraDB for Redis is essential to your business. If your ApsaraDB for Redis service becomes unavailable, the RDS instances may be overwhelmed by the requests sent from your applications. ApsaraDB for Redis adopts the master-replica architecture to ensure high availability. The master is responsible for handling requests. When the master fails, the replica takes control of the workloads. The failover is completely transparent to users.

E-commerce

ApsaraDB for Redis is widely used in the E-commerce industry for business such as commodity presentation and recommendation.

- Scenario 1: online shopping systems

  An online shopping system is overwhelmed by user traffic during large promotional activities such as flash sales. Most databases are incapable of handling the heavy load. To resolve this issue, you can choose ApsaraDB for Redis for persistent storage.

- Scenario 2: inventory management systems that support stock taking

  ApsaraDB for Redis is used to count the inventory and RDS is used to store information about the quantities of items. ApsaraDB for Redis instances are deployed on physical servers that use SSD disks. Therefore, data can be synchronized to RDS at high speed.

Live streaming applications

Live streaming is strongly reliant on ApsaraDB for Redis, which is used to store user data and chat records.

- High availability

  ApsaraDB for Redis can be deployed in a master-replica architecture to significantly improve service availability.

- High performance

ApsaraDB for Redis provides cluster instances to eliminate the performance bottleneck that is caused by the single-thread mechanism of native Redis. Cluster instances can handle traffic spikes during live streaming and meet high-performance requirements.

- High scalability

  ApsaraDB for Redis allows you to deal with traffic spikes during peak hours by scaling out an instance with a few clicks. The upgrade is completely transparent to users.

11.Terms

You can find instructions for ApsaraDB for Redis related terms in this topic.

Term	Description
ApsaraDB for Redis	ApsaraDB for Redis is a high-performance key-value storage system that supports caching and storage. The system is developed on the basis of BSD open-source protocols.
Instance identifier (ID)	An instance corresponds to a user space, and serves as the basic unit of the ApsaraDB for Redis service. ApsaraDB for Redis has restrictions on instance configurations, such as connections, bandwidth, and CPU processing capacity. These restrictions vary according to different instance types. You can view the list of instance identifiers that you have purchased in the console.
Master-replica instance	This is an ApsaraDB for Redis instance that contains a master-replica structure. The master-replica instance provides limited capacity and performance.
High-performance cluster instance	This is an ApsaraDB for Redis instance that runs in a scalable cluster architecture. Cluster instances provide better scalability and performance, but they still have limited features.
Connection address	This is the host address for connecting to ApsaraDB for Redis. The connection address is displayed as a domain name. To obtain the connection address, choose **Instance Information** > **Connection Information**.
Connection password	This is the password used to connect to ApsaraDB for Redis. The password is in the format of `Instance ID:custom password`. For example, if you set the password as 1234 when you purchase an instance and the allocated instance ID is xxxx, the connection password is `xxxx:1234`.
Eviction policy	This is consistent with the Redis eviction policy. For more information, see Using Redis as an LRU cache.
DB	This is the abbreviation of the word database to indicate a database in ApsaraDB for Redis. Each ApsaraDB for Redis instance supports 256 databases numbered DB 0 to DB 255. By default, ApsaraDB for Redis writes data to DB 0.

12.Comparison between ApsaraDB for Redis and on-premises Redis

Compared with on-premises Redis databases deployed on your own servers, ApsaraDB for Redis has many advantages, such as high data security, easy O&M, and kernel optimization.

Item	ApsaraDB for Redis	On-premises Redis
Security protection	• Supports Virtual Private Cloud (VPC) for network isolation. • Supports whitelists for access control. • Supports Secure Socket Layer (SSL) encryption. • Allows you to create multiple accounts and grant different permissions to them. • Supports audit logs, which allow you to view request records. The data sampling rate is 100%, and the performance consumption is less than 5%.	• Requires a self-built network security system. It is difficult to construct such a system and brings high costs. • Has data leakage risks owing to security vulnerabilities in the default access configuration of open-source Redis. • Requires a third-party tool to implement SSL encryption. • Has no account authentication system. • Does not support audit logs.
Backup and recovery	• Supports backing up data to RDB and AOF files. • Archives the incremental data in the AOF file, which avoids the adverse impact of rewriting the AOF file.	• Does not support backup and recovery. • Rewrites AOF files irregularly, which deteriorates service performance.
O&M	• Supports over ten groups of monitoring metrics and a minimum monitoring frequency of 5 seconds/time. • Supports settings alerts based on monitoring metrics. • Allows you to create instances of different architectures as needed and change the instance configuration. • Provides accurate big key analysis based on the snapshots without performance consumption.	• Requires a complex third-party monitoring tool to implement service monitoring. • Stops services when you change the specification or the architecture. In addition, the specification or architecture change operation is complex. • Supports big key analysis based on sampling, which is inaccurate.

Item	ApsaraDB for Redis	On-premises Redis
High availability	• Supports single-zone high availability. • Supports zone-based disaster recovery. • Uses an independent and central module to guarantee high availability, which is stable and highly efficient in decision-making. This module also prevents the split-brain issue.	• Allows you to deploy high-availability architecture in an IDC based on the Sentinel mode. • Allows you to deploy zone-based disaster recovery architecture based on the Sentinel mode. • Depends on the Sentinel mode to guarantee high availability. The construction cost is high and the decision-making efficiency is low during service peak hours. Split-brain may occur, which brings loss to your business.
Kernel optimization	• Provides performance-enhanced instances based on multithreading. The performance of a performance-enhanced instance is three times that of a standard-performance instance with the same configuration. • Provides hybrid storage instances that store data in both the memory and disks. Hybrid storage instances can manage hot and cold data at the field level and effectively transfer data between the memory and disks. • Supports cross-slot multi-key operations in the cluster edition.	• Does not have a performance-enhanced edition. • Supports systems such as SSDB or Pika as the persistent storage. However, these systems are not well compatible with the Redis protocol. They can manage hot and cold data only at the key level. Transferring big keys between the memory and disks is costly. These systems are difficult to manage. • Does not support cross-slot multi-key operations in an open-source Redis cluster.