Alibaba Cloud
ApsaraDB for Redis

Best Practices

Issue: 20191119

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 due to product version upgrades , adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall 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 the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. 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 bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequent

ial, exemplary, incidental, 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 intellectu al 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 contact Alibaba Cloud directly if you discover any errors in this document .**

Document conventions

Style	Description	Example
	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.
	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.
	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.
	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]`

Style	Description	Example
{} or {a|b}	**This format is used for a required value, where only one item can be selected.**	`switch {active|stand}`

Contents

1 Migrate MySQL data to ApsaraDB for Redis

You can efficiently migrate data from RDS for MySQL or on-premises MySQL databases to ApsaraDB for Redis by using the pipeline feature of ApsaraDB for Redis. You can also migrate data from RDS databases that use other engines to ApsaraDB for Redis based on the steps described in this topic.

Scenarios

You can use ApsaraDB for Redis as a cache between your applications and databases to expand the service capabilities of traditional relational databases. In this way, you can optimize the business ecosystem. This is one of the classic application scenarios of ApsaraDB for Redis. This service stores hot data in the business. You can easily obtain common data in ApsaraDB for Redis from your applications, or use ApsaraDB for Redis to save sessions of active users in interactive applications. This service can greatly reduce the load on the backend relational database and improve the user experience.

To use ApsaraDB for Redis as a cache, you must first transmit data from a relational database to ApsaraDB for Redis. You cannot directly transmit tables in a relational database to the ApsaraDB for Redis database that stores data in a key-value structure. Before the migration, you must convert the source data to a special structure. This topic describes how to use the native tool to easily and efficiently migrate tables from MySQL databases to ApsaraDB for Redis. You can use the pipeline feature of ApsaraDB for Redis to transmit data in MySQL tables to hash tables of ApsaraDB for Redis.

> **Note:**
> This topic describes Alibaba Cloud RDS for MySQL instance as the migration source and ApsaraDB for Redis instance as the migration destination. In this example, you install the Linux environment that runs the migration command on the ECS instance. These instances run in the same VPC, so they can interconnect with each other.

Similarly, you can migrate data from other relational databases to ApsaraDB for Redis. In this migration, you need to extract data from the source database, convert the data format, and then transmit the data to the heterogeneous database. This migration method is also suitable for data migration between other heterogeneous databases.

Prerequisites

- You have created an RDS for MySQL instance as the source where table data is available for migration.
- You have created an ApsaraDB for Redis instance as the destination.
- You have created an ECS instance that runs the Linux system.
- These instances run in the same VPC of the same region.
- You have added the internal IP address of the ECS instance to the whitelists of RDS for MySQL and ApsaraDB for Redis instances.
- You have installed MySQL and Redis databases on the ECS instance to extract, convert, and transmit data.

Note:
These prerequisites apply only when you migrate data in Alibaba Cloud. If you want to migrate data in your on-premises environment, make sure that the Linux server that performs migration can connect to the source relational database and the destination ApsaraDB for Redis database.

Data before migration

This topic describes how to migrate the test data stored in the company table of the custm_info database. The company table contains test data as shown in the following table.

MySQL [custm_info]> SELECT * FROM company;

id	name	sdate	email	domain	city
[redacted]d96b5	[redacted]Hunter	1986-05-23	[redacted].com	[redacted]@example.net	Michaelborough
[redacted]662a7	[redacted]	1970-12-11	[redacted]	[redacted]@example.com	Pamelaborough
[redacted]db6c1	[redacted]	2001-09-06	[redacted]	[redacted]example.net	Port Melodybury
[redacted]38c2a	[redacted]	1979-06-08	[redacted]th.org	[redacted]ple.org	New Tracymouth
[redacted]65613	[redacted] Hernandez	1975-11-01	[redacted]	[redacted]@example.net	North Matthewhaven
[redacted]bb993	[redacted]d Wagner	2004-06-29	[redacted]	[redacted]le.net	East Angelamouth
[redacted]132b1	[redacted]	2018-01-03	[redacted]mith.com	[redacted]ple.org	Bradleychester
[redacted]0898c	[redacted]	1971-12-07	[redacted]	[redacted]e.org	East Christianhaven
[redacted]a5882	[redacted]	1976-07-14	[redacted]	[redacted]e.org	Port Christopherberg

This table includes six columns. After the migration, the value of the id column in the MySQL table changes to the key of the hash table in ApsaraDB for Redis. The column names of other columns change to the fields of the hash table, and the values of these columns change to the values of the corresponding fields. You can modify the scripts and commands for the migration according to actual scenarios.

Procedure

1. **Analyze the source data structure, create the following migration script on the ECS instance, and then save the script to the mysql_to_redis.sql file.**

```
SELECT CONCAT(
  "*12\r\n", #The number 12 specifies the number of the following
fields, and depends on the data structure of the MySQL table.
  '$', LENGTH('HMSET'), '\r\n', #HMSET specifies the command that
you use when writing data to ApsaraDB for Redis.
  'HMSET', '\r\n',
  '$', LENGTH(id), '\r\n', #id specifies the first field after you
run the HMSET command for fields. This field changes to the key of
the hash table in ApsaraDB for Redis.
  id, '\r\n',
  '$', LENGTH('name'), '\r\n', #'name' is passed to the hash table
as a string field. Other fields such as 'sdate' are processed in the
 same way.
  'name', '\r\n',
  '$', LENGTH(name), '\r\n', #The name variable specifies the
company name in the MySQL table. This variable changes to the value
of the field generated by the 'name' parameter. Other fields such as
 'sdate' are processed in the same way.
  name, '\r\n',
  '$', LENGTH('sdate'), '\r\n',
  'sdate', '\r\n',
  '$', LENGTH(sdate), '\r\n',
  sdate, '\r\n',
  '$', LENGTH('email'), '\r\n',
  'email', '\r\n',
  '$', LENGTH(email), '\r\n',
  email, '\r\n',
  '$', LENGTH('domain'), '\r\n',
  'domain', '\r\n',
  '$', LENGTH(domain), '\r\n',
  domain, '\r\n',
  '$', LENGTH('city'), '\r\n',
  'city', '\r\n',
  '$', LENGTH(city), '\r\n',
  city, '\r\n'
)
```

```
FROM company AS c
```

2. **Run the following command on the ECS instance to migrate data.**

```
mysql -h <MySQL host> -P <MySQL port> -u <MySQL username> -D <MySQL
 database name> -p --skip-column-names --raw < mysql_to_redis.sql |
redis-cli -h <Redis host> --pipe -a <Redis password>
```

Table 1-1: Options

Option	Description	Example
-h	The endpoint of the RDS for MySQL database. **Note:** This is the `-h` option following `mysql`.	rm-bp1xxxxxxxxxxxx.mysql.rds.aliyuncs **Note:** Use the endpoint for connecting the Linux server to the RDS for MySQL database.
-P	The service port of the RDS for MySQL database.	3306
-u	The username of the RDS for MySQL database.	testuser
-D	The database where the MySQL table that you want to migrate is located.	mydatabase
-p	The password for connecting to the RDS for MySQL database. **Note:** · If you do not have any password, you do not need to set this option. · To improve security, you can enter `-p` and do not have the password following this option. You can run the command and then enter the password according to the command-line interface (CLI) prompt.	Mysqlpwd233
--skip-column-names	The column name is not written into the query result.	No value is required.

Option	Description	Example
--raw	The output column value is not escaped.	No value is required.
-h	Specifies the endpoint of ApsaraDB for Redis. **Note:** This is the `-h` option following `redis-cli`.	r-bp1xxxxxxxxxxxxx.redis.rds.aliyuncs.com **Note:** Use the endpoint for connecting the Linux server to the ApsaraDB for Redis database.
--pipe	Use the pipeline feature of ApsaraDB for Redis to transmit data.	No value is required.
-a	The password for connecting to ApsaraDB for Redis. **Note:** If you do not have any password or do not need a password, you do not need to set this option.	Redispwd233

Figure 1-1: Sample code

```
[root@[redacted] ~]# mysql -h rm-bp1[redacted].mysql.rds.aliyuncs.com -P 3306 -u [redacted] -D custm_info -p --skip-column-names --raw < mysql_to_redis.sql | redis-cli -h r-bp1[redacted].redis.rds.aliyuncs.com --pipe -a [redacted]
Enter password:
All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 100
```

Note:
In the result, `errors` indicates the number of errors that occur when the system runs the command, and `replies` indicates the number of responses the system returns. If the value of `errors` is 0 and the value of `replies` equals the number of items in the MySQL table, the migration is completed.

Data after migration

After the migration, one item in the MySQL table corresponds to one item in the hash table of ApsaraDB for Redis. You can run the `HGETALL` command to query an item and view the following result.

```
r-bp1[redacted].redis.rds.aliyuncs.com:6379> HGETALL [redacted]6b132b1
 1) "name"
 2) "[redacted]ons"
 3) "sdate"
 4) "2018-01-03"
 5) "email"
 6) "[redacted]@[redacted] smith.com"
 7) "domain"
 8) "[redacted]@example.org"
 9) "city"
10) "Bradleychester"
```

You can adjust the migration solution based on the query method required in actual scenarios. For example, you can convert other columns in the MySQL table to the keys in the hash table and convert the id column to a field, or ignore the id column.

2 Online player score ranking

Compared with Redis, the features of ApsaraDB for Redis are similar. You can use ApsaraDB for Redis to create a ranking list for an online game.

Sample code

```
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.UUID;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Tuple;
public class GameRankSample {
    static int TOTAL_SIZE = 20;
    public static void main(String[] args)
    {
        //Connection information. This information can be obtained
from the console
        String host = "xxxxxxxxxx.m.cnhz1.kvstore.aliyuncs.com";
        int port = 6379;
        Jedis jedis = new Jedis(host, port);
        try {
            //Instance password
            String authString = jedis.auth("password");//password
            if (! authString.equals("OK"))
            {
                System.err.println("AUTH Failed: " + authString);
                return;
            }
            //Key
            String key = "Game name: Keep Running, Ali!" ; ;
            //Clears any existing data
            jedis.del(key);
            //Generates several simulated players
            List<String> playerList = new ArrayList<String>();
            for (int i = 0; i < TOTAL_SIZE; ++i) 
            {
                //Randomly generates an ID for each player
                playerList.add(UUID.randomUUID().toString());
            }
            System.out.println("Inputs all players ");
            //Records the score for each player
            for (int i = 0; i < playerList.size(); i++) 
            {
                //Randomly generates numbers as the scores of each
simulated player
                int score = (int)(Math.random()*5000); 
                String member = playerList.get(i);
                System.out.println("Player ID:" + member + ", Player
Score: " + score);
                //Adds the player ID and score to SortedSet of the
corresponding key
                jedis.zadd(key, score, member);
            }
            //Prints out the ranking list of all players
            System. out. println ();
            System.out.println("       "+key);
            System.out.println(" Ranking list of all players");
```

```
            //Obtains the sorted list of players from SortedSet of the
 corresponding key
            Set<Tuple> scoreList = jedis.zrevrangeWithScores(key, 0, -
1);
            for (Tuple item : scoreList) {
                System.out.println("Player ID:"+item.getElement()+",
Player Score:"+Double.valueOf(item.getScore()).intValue());
            }
            //Prints out the top five players
            System. out. println ();
            System.out.println("       "+key);
            System.out.println("       Top players");
            scoreList = jedis.zrevrangeWithScores(key, 0, 4);
            for (Tuple item : scoreList) {
                System.out.println("Player ID:"+item.getElement()+",
Player Score:"+Double.valueOf(item.getScore()).intValue());
            }
            //Prints out a list of specific players
            System. out. println ();
            System.out.println("         "+key);
            System.out.println(" Players with scores from 1,000 to 2,
000");
            //Obtains a list of players with scores from 1,000 to 2,
000 from SortedSet of the corresponding key
            scoreList = jedis.zrangeByScoreWithScores(key, 1000, 2000
);
            for (Tuple item : scoreList) {
                System.out.println("Player ID: "+item.getElement()+",
Player Score:"+Double.valueOf(item.getScore()).intValue()); 
            }
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            jedis.quit();
            jedis.close();
        }
    }
}
```

Results

After you access the ApsaraDB for Redis instance with the correct address and password and run the Java code, the following output is displayed:

```
Input all players
Player ID: 9193e26f-6a71-4c76-8666-eaf8ee97ac86. Score: 3,860
Player ID: db03520b-75a3-48e5-850a-071722ff7afb. Score: 4,853
Player ID: d302d24d-d380-4e15-a4d6-84f71313f27a. Score: 2,931
Player ID: bee46f9d-4b05-425e-8451-8aa6d48858e6. Score: 1,796
Player ID: ec24fb9e-366e-4b89-a0d5-0be151a8cad0. Score: 2,263
Player ID: e11ecc2c-cd51-4339-8412-c711142ca7aa. Score: 1,848
Player ID: 4c396f67-da7c-4b99-a783-25919d52d756. Score: 958
Player ID: a6299dd2-4f38-4528-bb5a-aa2d48a9f94a. Score: 2,428
Player ID: 2e4ec631-1e4e-4ef0-914f-7bf1745f7d65. Score: 4,478
Player ID: 24235a85-85b9-476e-8b96-39f294f57aa7. Score: 1,655
Player ID: e3e8e1fa-6aac-4a0c-af80-4c4a1e126cd1. Score: 4,064
Player ID: 99bc5b4f-e32a-4295-bc3a-0324887bb77e.  Score: 4,852
Player ID: 19e2aa6b-a2d8-4e56-bdf7-8b59f64bd8e0. Score: 3,394
Player ID: cb62bb24-1318-4af2-9d9b-fbff7280dbec. Score: 3,405
Player ID: ec0f06da-91ee-447b-b935-7ca935dc7968. Score: 4,391
Player ID: 2c814a6f-3706-4280-9085-5fe5fd56b71c. Score: 2,510
Player ID: 9ee2ed6d-08b8-4e7f-b52c-9adfe1e32dda. Score: 63
```

```
Player ID: 0293b43a-1554-4157-a95b-b78de9edf6dd. Score: 1,008
Player ID: 674bbdd1-2023-46ae-bbe6-dfcd8e372430. Score: 2,265
Player ID: 34574e3e-9cc5-43ed-ba15-9f5405312692. Score: 3,734
              Game name: Keep Running, Ali!
              Ranking list of all players
Player ID: db03520b-75a3-48e5-850a-071722ff7afb. Score: 4,853
Player ID: 99bc5b4f-e32a-4295-bc3a-0324887bb77e. Score: 4,852
Player ID: 2e4ec631-1e4e-4ef0-914f-7bf1745f7d65. Score: 4,478
Player ID: ec0f06da-91ee-447b-b935-7ca935dc7968. Score: 4,391
Player ID: e3e8e1fa-6aac-4a0c-af80-4c4a1e126cd1. Score: 4,064
Player ID: 9193e26f-6a71-4c76-8666-eaf8ee97ac86. Score: 3,860
Player ID: 34574e3e-9cc5-43ed-ba15-9f5405312692. Score: 3,734
Player ID: cb62bb24-1318-4af2-9d9b-fbff7280dbec. Score: 3,405
Player ID: 19e2aa6b-a2d8-4e56-bdf7-8b59f64bd8e0. Score: 3,394
Player ID: d302d24d-d380-4e15-a4d6-84f71313f27a. Score: 2,931
Player ID: 2c814a6f-3706-4280-9085-5fe5fd56b71c. Score: 2,510
Player ID: a6299dd2-4f38-4528-bb5a-aa2d48a9f94a. Score: 2,428
Player ID: 674bbdd1-2023-46ae-bbe6-dfcd8e372430. Score: 2,265
Player ID: ec24fb9e-366e-4b89-a0d5-0be151a8cad0. Score: 2,263
Player ID: e11ecc2c-cd51-4339-8412-c711142ca7aa. Score: 1,848
Player ID: bee46f9d-4b05-425e-8451-8aa6d48858e6. Score: 1,796
Player ID: 24235a85-85b9-476e-8b96-39f294f57aa7. Score: 1,655
Player ID: 0293b43a-1554-4157-a95b-b78de9edf6dd. Score: 1,008
Player ID: 4c396f67-da7c-4b99-a783-25919d52d756. Score: 958
Player ID:9ee2ed6d-08b8-4e7f-b52c-9adfe1e32dda. Score:63
      Game name: Keep Running, Ali!
         Top players
Player ID: db03520b-75a3-48e5-850a-071722ff7afb. Score: 4,853
Player ID: 99bc5b4f-e32a-4295-bc3a-0324887bb77e. Score: 4,852
Player ID: 2e4ec631-1e4e-4ef0-914f-7bf1745f7d65. Score: 4,478
Player ID: ec0f06da-91ee-447b-b935-7ca935dc7968. Score: 4,391
Player ID: e3e8e1fa-6aac-4a0c-af80-4c4a1e126cd1. Score: 4,064
          Game name: Keep Running, Ali!
          Players scored between 1,000 and 2,000
Player ID: 0293b43a-1554-4157-a95b-b78de9edf6dd. Score: 1,008
Player ID: 24235a85-85b9-476e-8b96-39f294f57aa7. Score:1,655
Player ID: bee46f9d-4b05-425e-8451-8aa6d48858e6. Score: 1,796
Player ID: e11ecc2c-cd51-4339-8412-c711142ca7aa. Score: 1,848
```

3 Correlation analysis on E-commerce store items

You can use ApsaraDB for Redis to perform a correlation analysis on E-commerce store items.

Scenario introduction

The correlation between items is the case where multiple items are added to the same shopping cart. The analysis results are crucial for the E-commerce industry and can be used to analyze shopping behaviors. For example:

- **On the details page of a specific item, recommend related items to the user who is browsing this page.**
- **Recommend related items to a user who just added an item to the shopping cart.**
- **Place highly correlated items together on the shelf.**

You can use ApsaraDB for Redis to create a sorted set for each item. For a specific item, the set consists of items that are added with this item to the shopping cart. Members of the set are scored based on how often they appear in the same cart with that specific item. Each time item A and item B appear in the same shopping cart, the respective sorted sets for item A and item B in ApsaraDB for Redis are updated.

Sample code

```
package shop.kvstore.aliyun.com;
import java.util.Set;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Tuple;
    public class AliyunShoppingMall {
        public static void main(String[] args)
        {
            //ApsaraDB for Redis connection. This information can be
obtained from the console
            String host = "xxxxxxxx.m.cnhza.kvstore.aliyuncs.com";
            int port = 6379;
            Jedis jedis = new Jedis(host, port);
            try {
                //ApsaraDB for Redis instance password
                String authString = jedis.auth("password");//password
                if (! authString.equals("OK"))
                {
                    System.err.println("AUTH Failed: " + authString);
                    return;
                }
                //Products
                String key0="Alibaba Cloud: Product: Beer";
                String key1 = "Alibaba Cloud: Product: Chocolate";
                String key2 = "Alibaba Cloud: Product: Cola";
                String key3 = "Alibaba Cloud: Product: Gum";
                String key4 = "Alibaba Cloud: Product: Beef Jerky";
```

```
                String key5="Alibaba Cloud: Product: Chicken Wings";
                final String[] aliyunProducts=new String[]{key0,key1,
key2,key3,key4,key5};
                //Initialize to clear the possible existing data
                for (int i = 0; i < aliyunProducts.length; i++) {
                    jedis.del(aliyunProducts[i]);
                }
                //Simulated shopping behaviors
                for (int i = 0; i < 5; i++) { //Simulates the shopping
 behaviors of multiple customers
                    customersShopping(aliyunProducts,i,jedis);
                }
                System.out.println();
                //Uses ApsaraDB for Redis to generate the correlated
relationship between items
                for (int i = 0; i < aliyunProducts.length; i++) {
                    System.out.println(">>>>>>>>>>and"+aliyunProducts[
i]+"was purchased with <<<<<<<<<<<<<<<");
                    Set<Tuple> relatedList = jedis.zrevrangeWithScores
(aliyunProducts[i], 0, -1);
                    for (Tuple item : relatedList) {
                        System.out.println("Item name:"+item.
getElement()+", Purchased together times:"+Double.valueOf(item.
getScore()).intValue());
                    }
                    System.out.println();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }finally{
                jedis.quit();
                jedis.close();
            }
        }
        private static void customersShopping(String[] products, int i
, Jedis jedis) {
            //Simulates three simple shopping behaviors and randomly
selects one as the behavior of the user
            int bought=(int)(Math.random()*3);
            if(bought==1){
                //Simulated business logic: the user has purchased the
 following products:
                System.out.println("User"+i+"purchased"+products[0
]+","+products[2]+","+products[1]);
                //Records the correlations between the items to
SortSet in ApsaraDB for Redis
                jedis.zincrby(products[0], 1, products[1]);
                jedis.zincrby(products[0], 1, products[2]);
                jedis.zincrby(products[1], 1, products[0]);
                jedis.zincrby(products[1], 1, products[2]);
                jedis.zincrby(products[2], 1, products[0]);
                jedis.zincrby(products[2], 1, products[1]);
            }else if(bought==2){
                //Simulated business logic: the user has purchased the
 following products
                System. out. println ("user" + i + "purchased" +
products [4] + ", "+ products [2] +", "+ products [3]);
                //Records the correlations between the items to
SortSet in ApsaraDB for Redis
                jedis.zincrby(products[4], 1, products[2]);
                jedis.zincrby(products[4], 1, products[3]);
                jedis.zincrby(products[3], 1, products[4]);
                jedis.zincrby(products[3], 1, products[2]);
                jedis.zincrby(products[2], 1, products[4]);
```

```
                jedis.zincrby(products[2], 1, products[3]);
            }else if(bought==0){
                //Simulated business logic: the user has purchased the
 following products:
                System.out.println("user"+i+"purchased"+products[1
]+","+products[5]);
                //Records the correlations between the items to
SortSet in ApsaraDB for Redis
                jedis.zincrby(products[5], 1, products[1]);
                jedis.zincrby(products[1], 1, products[5]);
            }
        }
    }
```

Results

After you access the ApsaraDB for Redis instance with the correct address and password and run the Java code, the following output is displayed:

```
User 0 purchased Alibaba Cloud: Product: Chocolate, Alibaba Cloud:
Product: Chicken Wings
User 1 purchased Alibaba Cloud: Product: Beef Jerky, Alibaba Cloud:
Product: Cola, Alibaba Cloud: Product: Gum
User 2 purchased Alibaba Cloud: Product: Beer, Alibaba Cloud: Product
: Cola, Alibaba Cloud: product: Chocolate
User 3 purchased Alibaba Cloud: Product: Beef Jerky, Alibaba Cloud:
Product: Cola, Alibaba Cloud: Product: Gum
User 4 purchased Alibaba Cloud: Product: Chocolate, Alibaba Cloud:
Product: Chicken Wings
>>>>>>>>>>Alibaba Cloud: Product: Beer was purchased with
<<<<<<<<<<<<<<<
Item Name: Alibaba Cloud: Product: Chocolate. Purchased together times
: 1
Item name: Alibaba Cloud: Product:Cola. Purchased together times: 1
>>>>>>>>>>Alibaba Cloud: Product: Chocolate was purchased with
<<<<<<<<<<<<<<<<<<<<
Item name: Alibaba Cloud: Product: Chicken Wings. Purchased together
times: 2
Item name: Alibaba Cloud: Product: Beer. Purchased together times: 1
Item name: Alibaba Cloud: Product: Cola. Purchased together times: 1
>>>>>>>>>>Alibaba Cloud: Product: Cola was purchased with
<<<<<<<<<<<<<<<<<
Item name: Alibaba Cloud:Product: Beef Jerky. Purchased together times
: 2
Item name: Alibaba Cloud: Product: Gum. Purchased together times: 2
Item name: Alibaba Cloud: Product: Chocolate. Purchased together times
: 1
Item name: Alibaba Cloud: Product: Beer. Purchased together times: 1
>>>>>>>>>>Alibaba Cloud: Product: Gum was purchased with
<<<<<<<<<<<<<<<<<<<<
Item name: Alibaba Cloud: Product: Beef Jerky. Purchased together
times: 2
Item name: Alibaba Cloud: Product: Cola. Purchased together times: 2
>>>>>>>>>>Alibaba Cloud: Product: Beef Jerky was purchased with
<<<<<<<<<<<<<<<
Item name: Alibaba Cloud: Product: Cola. Purchased together times: 2
Item name: Alibaba Cloud: Product: Gum. Purchased together times: 2
>>>>>>>>>>Alibaba Cloud: Product: Chicken Wings was purchased with
<<<<<<<<<<<<<<
Item name: Alibaba Cloud: Product: Chocolate. Purchased together times
: 2
```

4 Publish and subscribe to messages

Similar to Redis, ApsaraDB for Redis provides publishing (pub) and subscription (sub) features. ApsaraDB for Redis allows multiple clients to subscribe to messages published by a client.

Scenario introduction

It must be noted that messages published using ApsaraDB for Redis are "non-persistent". This means the message publisher is only responsible for publishing a message and does not save previously sent messages, whether or not these messages were received. Thus, messages are "lost once published". Message subscribers can only receive messages that are subscribed. They will not receive the earlier messages in the channel.

In addition, the message publisher (publish client) does not necessarily connect to a server exclusively. While publishing messages, you can perform other operations (for example, the List operations) from the same client at the same time. However, the message subscriber (subscribe client) needs to connect to a server exclusivel y. That is, during the subscription period, the client may not perform any other operations. Rather, the operations are blocked while the client is waiting for messages in the channel. Therefore, message subscribers must use a dedicated server connection or thread (see the following example).

Sample code

For the message publisher (publish client)

```
package message.kvstore.aliyun.com;
import redis.clients.jedis.Jedis;
public class KVStorePubClient {
    private Jedis jedis;
    public KVStorePubClient(String host,int port, String password){
        jedis = new Jedis(host,port);
        //KVStore instance password
        String authString = jedis.auth(password);
        if (! authString.equals("OK"))
        {
            System.err.println("AUTH Failed: " + authString);
            return;
        }
    }
    public void pub(String channel,String message){
        System. out. println ("> Publish> channel: "+ channel +">
message sent: "+ message );
        jedis.publish(channel, message);
    }
```

```
    public void close(String channel){
         System.out.println(" >>> PUBLISH End > Channel:"+channel+" >
Message:quit");
         //The message publisher stops sending by sending a “quit”
message
        jedis.publish(channel, "quit");
    }
}
```

For the message subscriber (subscribe client)

```
package message.kvstore.aliyun.com;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPubSub;
public class KVStoreSubClient extends Thread{
    private Jedis jedis;
    private String channel;
    private JedisPubSub listener;
    public KVStoreSubClient(String host,int port, String password){
        jedis = new Jedis(host,port);
                //ApsaraDB for Redis instance password
                String authString = jedis.auth(password); //password
                if (! authString.equals("OK"))
                {
                    System.err.println("AUTH Failed: " + authString);
                    return;
                }
    }
    public void setChannelAndListener(JedisPubSub listener,String
channel){
        this.listener=listener;
        this.channel=channel;
    }
    private void subscribe(){
        if(listener==null || channel==null){
            System.err.println("Error:SubClient> listener or channel
is null");
        }
        System.out.println("  >>> SUBSCRIBE > Channel:"+channel);
        System.out.println();
         //When the recipient is listening for subscribed messages,
the process is blocked until the quit message is received (passively)
or the subscription is canceled actively
        jedis.subscribe(listener, channel);
    }
    public void unsubscribe(String channel){
        System.out.println(" >>> UNSUBSCRIBE > Channel:"+channel);
        System.out.println();
        listener.unsubscribe(channel);
    }
    @Override
    public void run() {
        try{
            System.out.println();
            System.out.println("---------Subscription begins-------");
            subscribe();
            System.out.println("----------Subscription ends-------");
            System.out.println();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
```

```
}
```

For the message listener

```
package message.kvstore.aliyun.com;
import redis.clients.jedis.JedisPubSub;
public class KVStoreMessageListener extends JedisPubSub{
    @Override
    public void onMessage(String channel, String message) {
        System.out.println("  <<< SUBSCRIBE< Channel:" + channel + " >
Message received:" + message );
        System.out.println();
        //When a quit message is received, the subscription is
canceled (passively)
        if(message.equalsIgnoreCase("quit")){
            this.unsubscribe(channel);
        }
    }
    @Override
    public void onPMessage(String pattern, String channel, String
message) {
        // TODO Auto-generated method stub
    }
    @Override
    public void onSubscribe(String channel, int subscribedChannels) {
        // TODO Auto-generated method stub
    }
    @Override
    public void onUnsubscribe(String channel, int subscribedChannels)
 {
        // TODO Auto-generated method stub
    }
    @Override
    public void onPUnsubscribe(String pattern, int subscribedChannels)
 {
        // TODO Auto-generated method stub
    }
    @Override
    public void onPSubscribe(String pattern, int subscribedChannels) {
        // TODO Auto-generated method stub
    }
}
```

Sample main process

```
package message.kvstore.aliyun.com;
import java.util.UUID;
import redis.clients.jedis.JedisPubSub;
public class KVStorePubSubTest {
    //The connection information of ApsaraDB for Redis. This
information can be obtained from the console
    static final String host = "xxxxxxxxxx.m.cnhza.kvstore.aliyuncs.
com";
    static final int port = 6379;
    static final String password="password";//password
    public static void main(String[] args) throws Exception{
            KVStorePubClient pubClient = new KVStorePubClient(host,
port,password);
            final String channel = "KVStore Channel-A";
            //The message sender starts sending messages, but there
are no subscribers, so the messages will not be received
```

```
            pubClient.pub(channel, "Alibaba Cloud message 1: (No
subscribers. This message will not be received)");
            // Message recipient
            KVStoreSubClient subClient = new KVStoreSubClient(host,
port,password);
            JedisPubSub listener = new KVStoreMessageListener();
            subClient.setChannelAndListener(listener, channel);
            // Message recipient starts subscribing
            subClient.start();
            //The message sender continues sending messages
            for (int i = 0; i < 5; i++) {
                String message=UUID.randomUUID().toString();
                pubClient.pub(channel, message);
                Thread.sleep(1000);
            }
            // The message recipient cancels the subscription
            subClient.unsubscribe(channel);
            Thread.sleep(1000);
            pubClient.pub(channel, "Alibaba Cloud message 2:(
Subscription canceled. This message will not be received)");
             //The message publisher stops sending by sending a “quit
” message
            //When other message recipients, if any, receive "quit" in
 listener.onMessage(), the "unsubscribe" operation is performed.
            pubClient.close(channel);
        }
    }
```

Output

After you access the ApsaraDB for Redis instance with the correct address and password and run the preceding Java code, the following output is displayed.

```
  >>> PUBLISH > Channel:KVStore Channel-A > Sends the message Aliyun
Message 1: (No subscribers. This message will not be received)
----------Subscription starts-------
  >>> SUBSCRIBE> Channel: KVStore Channel-A
  >>> PUBLISH> Channel: KVStore Channel-A> sends message: 0f9c2cee-
77c7-4498-89a0-1dc5a2f65889
  <<< SUBSCRIBE< Channel:KVStore Channel-A >receives message: 0f9c2cee
-77c7-4498-89a0-1dc5a2f65889
  >>> PUBLISH> Channel: KVStore Channel-A> sends message: ed5924a9-
016b-469b-8203-7db63d06f812
  <<< SUBSCRIBE< Channel:KVStore Channel-A >receives message: ed5924a9
-016b-469b-8203-7db63d06f812
  >>> PUBLISH> Channel: KVStore Channel-A> sends message: f1f84e0f-
8f35-4362-9567-25716b1531cd
  <<< SUBSCRIBE< Channel:KVStore Channel-A >receives message: f1f84e0f
-8f35-4362-9567-25716b1531cd
  >>> PUBLISH> Channel: KVStore Channel-A> sends message: 746bde54-
af8f-44d7-8a49-37d1a245d21b
  <<< SUBSCRIBE< Channel:KVStore Channel-A >receives message: 746bde54
-af8f-44d7-8a49-37d1a245d21b
  >>> PUBLISH> Channel: KVStore Channel-A> sends message: 8ac3b2b8-
9906-4f61-8cad-84fc1f15a3ef
  <<< SUBSCRIBE< Channel:KVStore Channel-A >receives message: 8ac3b2b8
-9906-4f61-8cad-84fc1f15a3ef
  >>> UNSUBSCRIBE> Channel: KVStore Channel-A
----------Subscription ends-------
  >>> PUBLISH > Channel:KVStore Channel-A > sends the message Aliyun
 Message 2: (The subscription has been canceled, so the message will
not be received)
```

```
>>> PUBLISH ends> Channel:KVStore Channel-A > Message:quit
```

The preceding example demonstrates a situation with one publisher and one subscriber. There can be multiple publishers, subscribers, and even multiple message channels. In such scenarios, you are required to slightly change the code to fit the scenario.

5 Pipeline

Similar to Redis, ApsaraDB for Redis provides the pipeline feature.

Scenario introduction

A client interacts with a server through one-way pipelines, one for sending requests and the other for receiving responses. You can send operation requests consecutiv ely from the client to the server. However, during this period, the server does not send the responses to each operation request. The client receives the response to each request from the server until it sends a quit message to the server.

Pipelines are useful, for example, when several operation commands need to be quickly submitted to the server but the responses and operation results are not required immediately. In this case, pipelines are used as a batch processing tool to optimize the performance. The performance is enhanced because the overhead of the TCP connection is reduced.

However, the client using pipelines in the app connects to the server exclusivel y, and non-pipeline operations are blocked until the pipelines are closed. If you need to perform other operations at the same time, you can establish a dedicated connection for pipeline operations to separate them from conventional operations.

Sample code 1

Performance comparison

```
package pipeline.kvstore.aliyun.com;
import java.util.Date;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Pipeline;
public class RedisPipelinePerformanceTest {
        static final String host = "xxxxxx.m.cnhza.kvstore.aliyuncs.
com";
        static final int port = 6379;
        static final String password = "password";
        public static void main(String[] args) {
            Jedis jedis = new Jedis(host, port);
                //ApsaraDB for Redis instance password
                String authString = jedis.auth(password); // password
                if (! authString.equals("OK")) {
                    System.err.println("AUTH Failed: " + authString);
                    jedis.close();
                    return;
                }
                //Executes several commands consecutively
                final int COUNT=5000;
                String key = "KVStore-Tanghan";
                // 1 ---Without using pipeline operations---
```

```
                jedis.del(key);//Initializes the key
                Date ts1 = new Date();
                for (int i = 0; i < COUNT; i++) {
                     //Sends a request and receives the response
                    jedis.incr(key);
                }
                Date ts2 = new Date();
                System.out.println("Without Pipeline > value is:"+
jedis.get(key)+" > Time elapsed:" + (ts2.getTime() - ts1.getTime())+ "
ms");
                //2 ---Using pipeline operations---
                jedis.del(key);//Initializes the key
                Pipeline p1 = jedis.pipelined();
                Date ts3 = new Date();
                for (int i = 0; i < COUNT; i++) {
                    //Sends the request
                    p1.incr(key);
                }
                // Receives the response
                p1.sync();
                Date ts4 = new Date();
                 System.out.println("Using Pipeline > value is:"+jedis
.get(key)+" > Time elapsed:" + (ts4.getTime() - ts3.getTime())+ "ms");
                jedis.close();
        }
    }
```

Output 1

After you access the ApsaraDB for Redis instance with the correct address and password and run the preceding Java code, the following output is displayed. The output shows that the performance is enhanced with pipelines.

```
Without pipelines > value: 5000 > Time elapsed: 5844 ms
With pipelines > value: 5000 > Time elapsed: 78 ms
```

Sample code 2

With pipelines defined in Jedis, responses are processed in two methods, as shown in the following sample code:

```
package pipeline.kvstore.aliyun.com;
import java.util.List;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Pipeline;
import redis.clients.jedis.Response;
    public class PipelineClientTest {
        static final String host = "xxxxxxxx.m.cnhza.kvstore.aliyuncs.
com";
        static final int port = 6379;
        static final String password = "password";
        public static void main(String[] args) {
            Jedis jedis = new Jedis(host, port);
                //ApsaraDB for Redis instance password
                String authString = jedis.auth(password);// password
                if (! authString.equals("OK")) {
                    System.err.println("AUTH Failed: " + authString);
                    jedis.close();
                    return;
```

```
                }
                String key = "KVStore-Test1";
                jedis.del(key);//Initialization
                // -------- Method 1
                Pipeline p1 = jedis.pipelined();
                System.out.println("-----Method 1-----");
                for (int i = 0; i < 5; i++) {
                    p1.incr(key);
                    System.out.println("Pipeline sends requests");
                }
                 // After sending all requests, the client starts
receiving responses
                System.out.println("Sending requests completed. Start
to receive response");
                List<Object> responses = p1.syncAndReturnAll();
                if (responses == null || responses.isEmpty()) {
                    jedis.close();
                    throw new RuntimeException("Pipeline error: no
responds received");
                }
                for (Object resp : responses) {
                    System.out.println("Pipeline receives response: "
 + resp.toString());
                }
                System.out.println();
                //-------- Method 2
                System.out.println("-----Method 2-----");
                jedis.del(key);//Initialization
                Pipeline p2 = jedis.pipelined();
                //Declare the responses first
                Response<Long> r1 = p2.incr(key);
                System.out.println("Pipeline sends requests");
                Response<Long> r2 = p2.incr(key);
                System.out.println("Pipeline sends requests");
                Response<Long> r3 = p2.incr(key);
                System.out.println("Pipeline sends requests");
                Response<Long> r4 = p2.incr(key);
                System.out.println("Pipeline sends requests");
                Response<Long> r5 = p2.incr(key);
                System.out.println("Pipeline sends requests");
                try{
                     r1.get(); //Errors occur because the client has
not started receiving responses
                }catch(Exception e){
                    System.out.println(" <<< Pipeline error: the
client has not started receiving responses  >>> ");
                }
              // After sending all requests, the client starts
receiving responses
                System.out.println("Sending requests completed. Start
to receive response");
                p2.sync();
                System.out.println("Pipeline receives response: " + r1
.get());
                System. Out. println ("Pipeline receives response:" +
r2.get ());
                System. Out. println ("Pipeline receives response:" +
r3.get ());
                System. Out. println ("Pipeline receives response:" +
r4.get ());
                System. Out. println ("Pipeline receives response:" +
r5.get ());
                jedis.close();
            }
```

```
    }
```

Output 2

After you access the ApsaraDB for Redis instance with the correct address and password and run the preceding Java code, the following output is displayed.

```
----- Method 1 -----
Pipeline sends a request
Pipeline sends a request
Pipeline sends a request
Pipeline sends a request
Pipeline sends a request
After sending all requests, the client starts receiving responses.
Pipeline receives response 1
Pipeline receives response 2
Pipeline receives response 3
Pipeline receives response 4
Pipeline receives response 5
----- Method 2 -----
Pipeline sends a request
Pipeline sends a request
Pipeline sends a request
Pipeline sends a request
Pipeline sends a request
 <Pipeline error: The client has not started receiving responses>
After sending all requests, the client starts receiving responses.
Pipeline receives response 1
Pipeline receives response 2
Pipeline receives response 3
Pipeline receives response 4
Pipeline receives response 5
```

6 Transactions

ApsaraDB for Redis supports a mechanism to define transactions, as in Redis.

Scenario introduction

The transaction feature allows you to use the `MULTI`, `EXEC`, `DISCARD`, `WATCH`, and `UNWATCH` commands to execute atomic transactions.

Note that the definition of *transaction* in Redis is different from that in relational databases. If an operation fails or the transaction is canceled by the `DISCARD` command, Redis does not perform the "transaction rollback".

Sample code 1: Two clients operate on different keys

```
package transcation.kvstore.aliyun.com;
import java.util.List;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Transaction;
public class KVStoreTranscationTest {
    static final String host = "xxxxxx.m.cnhza.kvstore.aliyuncs.com";
    static final int port = 6379;
    static final String password = "password";
    //**Note that these two keys have different content
    static String client1_key = "KVStore-Transcation-1";
    static String client2_key = "KVStore-Transcation-2";
    public static void main(String[] args) {
        Jedis jedis = new Jedis(host, port);
        //ApsaraDB for Redis instance password
        String authString = jedis.auth(password);//password
        if (! authString.equals("OK")) {
            System.err.println("authentication failed: " + authString
);
            jedis.close();
            return;
        }
        jedis.set(client1_key, "0");
        //Starts another thread to simulate another client
        new KVStoreTranscationTest().new OtherKVStoreClient().start();
        Thread.sleep(500);
        Transaction tx = jedis.multi();//Starts the transaction
        //The following operations are collectively submitted to the
server as "atomic operations"
        tx.incr(client1_key);
        tx.incr(client1_key);
        Thread.sleep(400);//The suspension of the thread does not
affect the consecutively executed operations in a transaction. Other
thread operations cannot be executed.
        tx.incr(client1_key);
        Thread.sleep(300);//The suspension of the thread does not
affect the consecutively executed operations in a transaction. Other
thread operations cannot be executed.
        tx.incr(client1_key);
        Thread.sleep(200);//The suspension of the thread does not
affect the consecutively executed operations in a transaction. Other
thread operations cannot be executed.
```

```
        tx.incr(client1_key);
        List<Object> result = tx.exec();//Submits the operation for
execution
        //Parses and prints out the results
        for(Object rt : result){
            System.out.println("Client 1 > in transaction> "+rt.
toString());
        }
        jedis.close();
    }
    class OtherKVStoreClient extends Thread{
        @Override
        public void run() {
            Jedis jedis = new Jedis(host, port);
            //ApsaraDB for Redis instance password
            String authString = jedis.auth(password); //password
            if (! authString.equals("OK")) {
                System.err.println("AUTH Failed: " + authString);
                jedis.close();
                return;
            }
            jedis.set(client2_key, "100");
            for (int i = 0; i < 10; i++) {
                try {
                    Thread.sleep(300);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Client 2 > "+jedis.incr(client2_ke
y));
            }
            jedis.close();
        }
    }
}
```

Output 1

After you access the ApsaraDB for Redis instance with the correct address and password and run the preceding Java code, the following output is displayed. Here , we can see that Client 1 and Client 2 are in different threads. The operations in the transaction submitted by Client 1 are executed sequentially. Client 2 requests for operating on another key during this period, but the operation is blocked and Client 2 has to wait until all the operations in the Client 1 transaction have been completed.

```
Client 2 > 101
Client 2 > 102
Client 2 > 103
Client 2 > 104
Client 1> in transaction> 1
Client 1> in transaction> 2
Client 1> in transaction> 3
Client 1> in transaction> 4
Client 1> in transaction> 5
Client 2> 105
Client 2> 106
Client 2> 107
```

```
Client 2> 108
Client 2> 109
Client 2> 110
```

Sample code 2: Two clients operate on the same key

By slightly modifying the preceding code, we can have the two clients operate on the same key. The other parts of the code remain unchanged.

```
    ......
// *** Note that the content of these two keys is now the same
    static String client1_key = "KVStore-Transcation-1";
    static String client2_key = "KVStore-Transcation-1";
    ......
```

Output 2

After the modified Java code is executed, the output is displayed as follows. We can see that the two clients are in different threads but operate on the same key. However, while Client 1 uses the transaction mechanism to operate on this key, Client 2 is blocked and has to wait until all the operations in the Client 1 transaction are completed.

```
Client 2> 101
Client 2> 102
Client 2> 103
Client 2> 104
Client 1> in transaction> 105
Client 1> in transaction> 106
Client 1> in transaction> 107
Client 1> in transaction> 108
Client 1> in transaction> 109
Client 2> 110
Client 2> 111
Client 2> 112
Client 2> 113
Client 2> 114
Client 2> 115
```

7 ApsaraDB for Redis supports Double 11 Shopping Festival

ApsaraDB for Redis works as an important support for processing surging e-commerce promotions and orders during Double 11 Shopping Festival.

Background

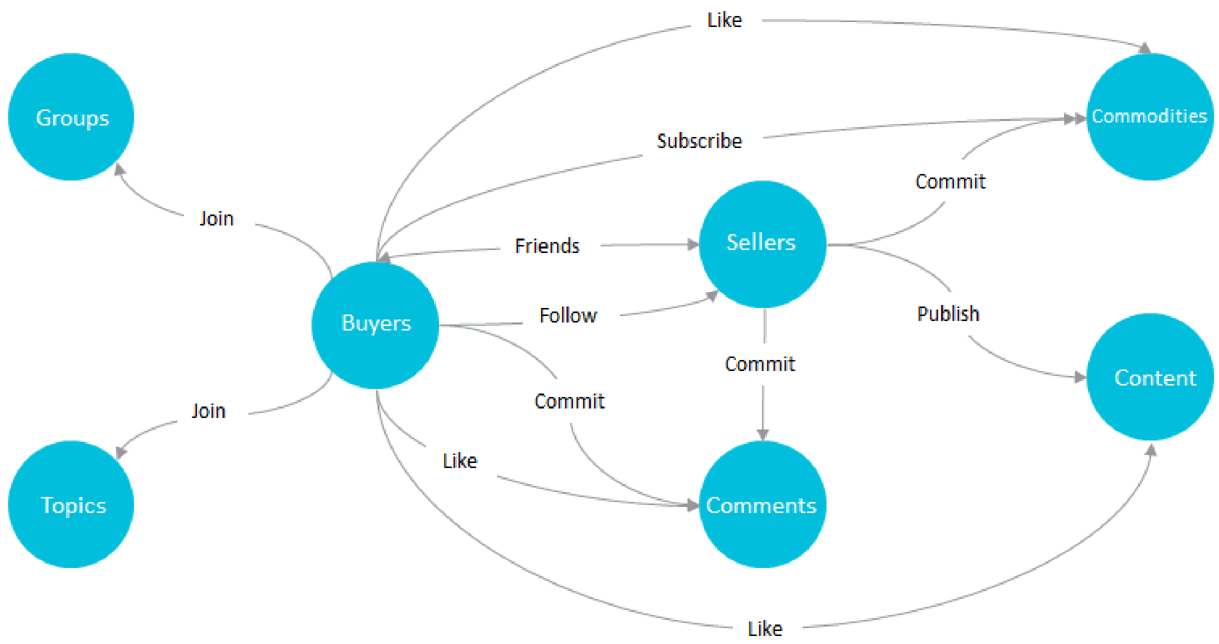
ApsaraDB for Redis provides multiple editions as follows: standard single-replica edition, standard dual-replica edition, and cluster edition.

The standard single-replica edition and standard dual-replica edition feature high compatibility and support Lua scripting and geographical location-based computing. The cluster edition provides large capacities and high performance, and solves the issues caused by single-server performance limits due to Redis single-thread model.

ApsaraDB for Redis works in a two-node hot standby structure by default and supports backup and recovery. Also, the Redis source code team of Alibaba Cloud constantly optimizes and upgrades the ApsaraDB for Redis service, and provides powerful security protections. This topic simplifies some scenarios of Double 11 Shopping Festival and describes the features of ApsaraDB for Redis. Actual scenarios are more complex.

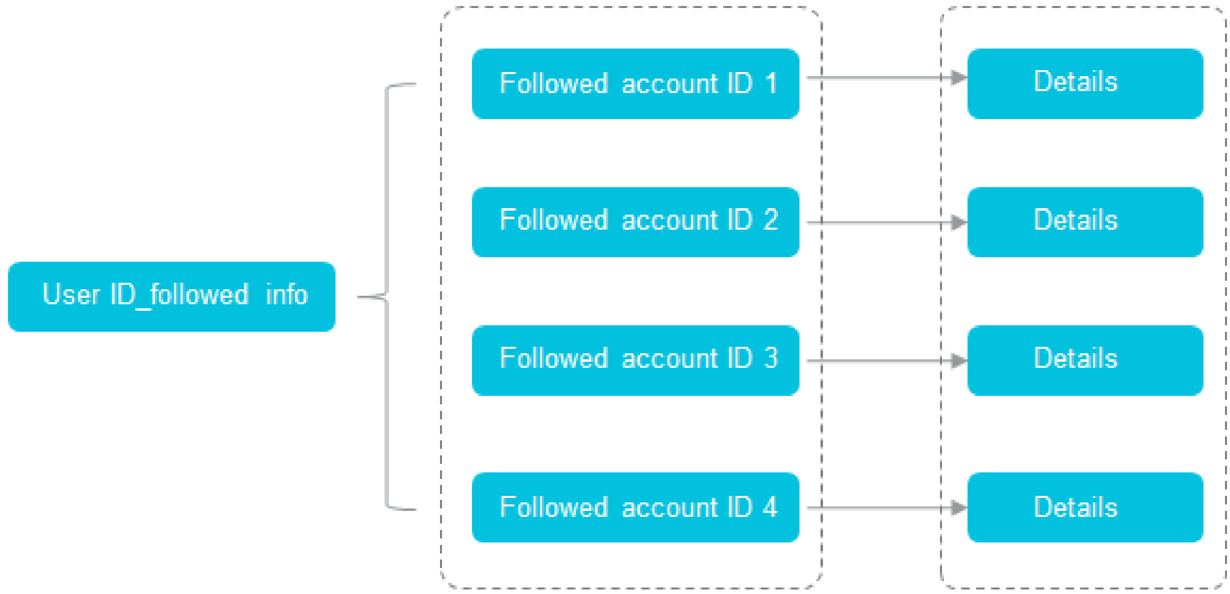
Store social relations for hundreds of millions of users in Weitao community

Weitao community carries social relations for hundreds of millions of Taobao users. Taobao users can specify a list of followers and merchants can maintain the data of regular customers or followers. The following figure shows the overall social relations.



To express these social relations, a traditional relational database model requires complex business design and results in poor user experience. A cluster instance of ApsaraDB for Redis caches followers chains of Weitao community. This simplifies the storage of followers data, and ensures excellent user experience during Double 11 Shopping Festival. Hash tables store followers data of Weitao community. The following figure shows the storage structure. You can call required API operations to query the following data:

- Whether Users A and B are followers of each other
- List of items User A is following

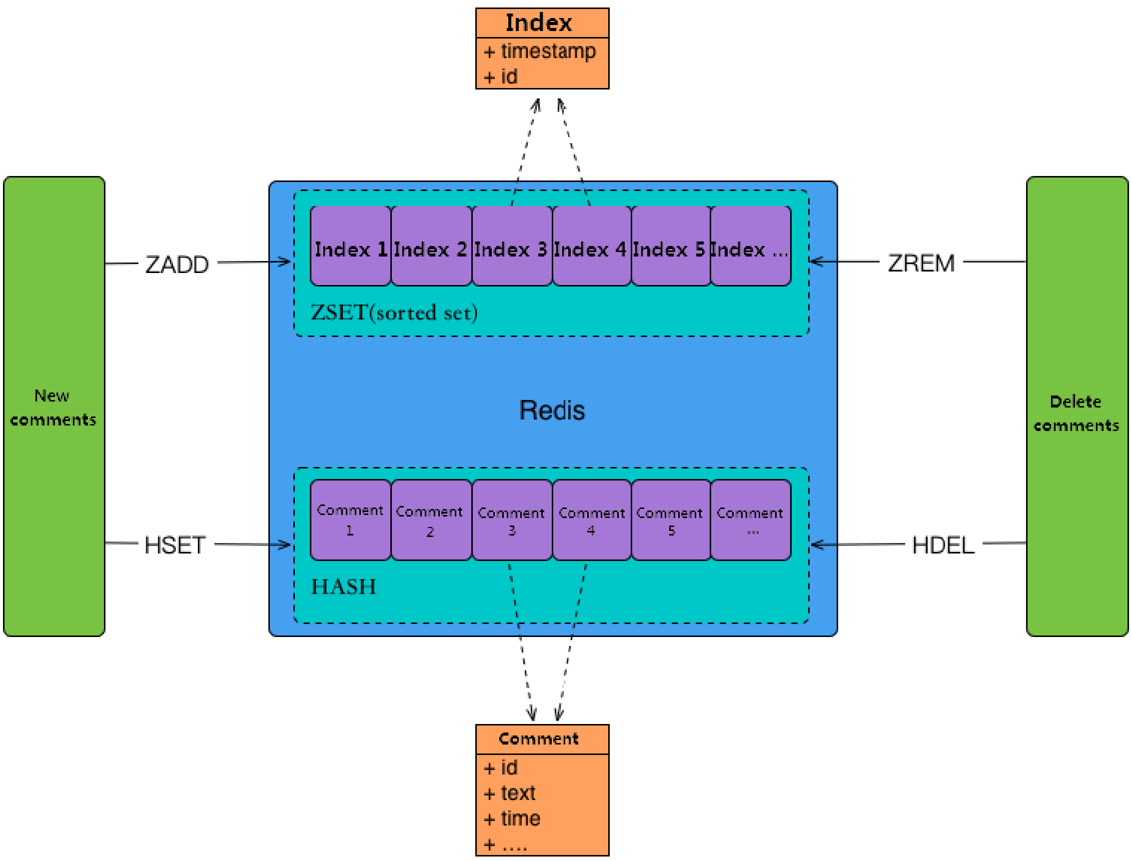


Paginate comments to live videos in Tmall based on a cursor

When mobile users view live videos during Double 11 Shopping Festival, they can obtain more comments to the live videos in three ways:

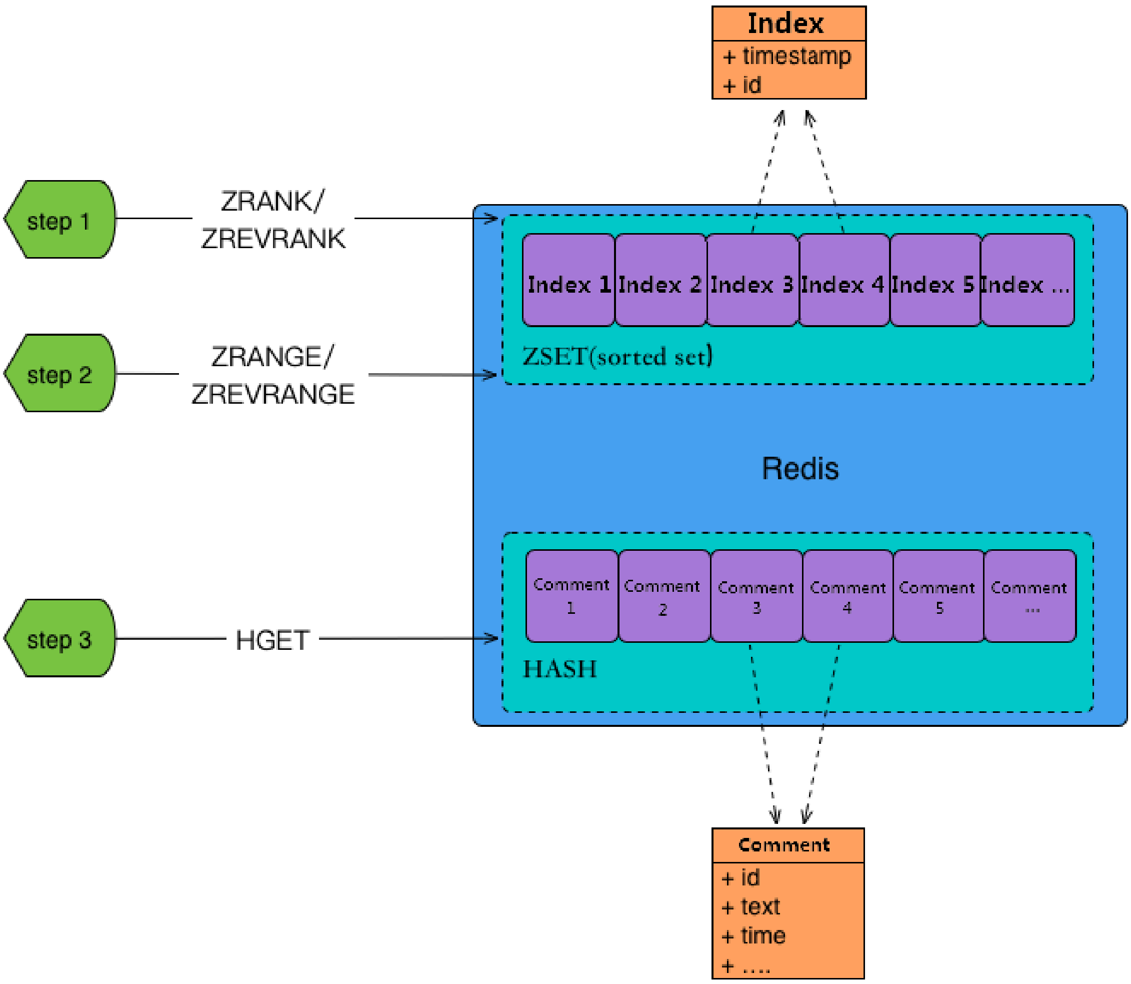
- **Pull down for incremental comments: obtain a specified number of incremental comments from the specified position up.**
- **Pull-down refresh: obtain a specified number of the latest comments.**
- **Pull up for incremental comments: obtain a specified number of incremental comments from the specified position down.**

The mobile live video streaming system uses ApsaraDB for Redis to optimize the business scenario. This ensures the success rate of comments to live videos and supports more than 50,000 transactions per second (TPS) and response time in milliseconds. The live video streaming system writes two types of data for each live video, including indexes and comments. The system writes indexes in sorted sets to sort comments, and stores the comments in hash tables. You can obtain an index ID from the indexes and retrieve a list of comments by reading the hash tables. The following figure shows the process of writing comments.



After a user refreshes the list, the background retrieves the corresponding comments. This process is as follows:

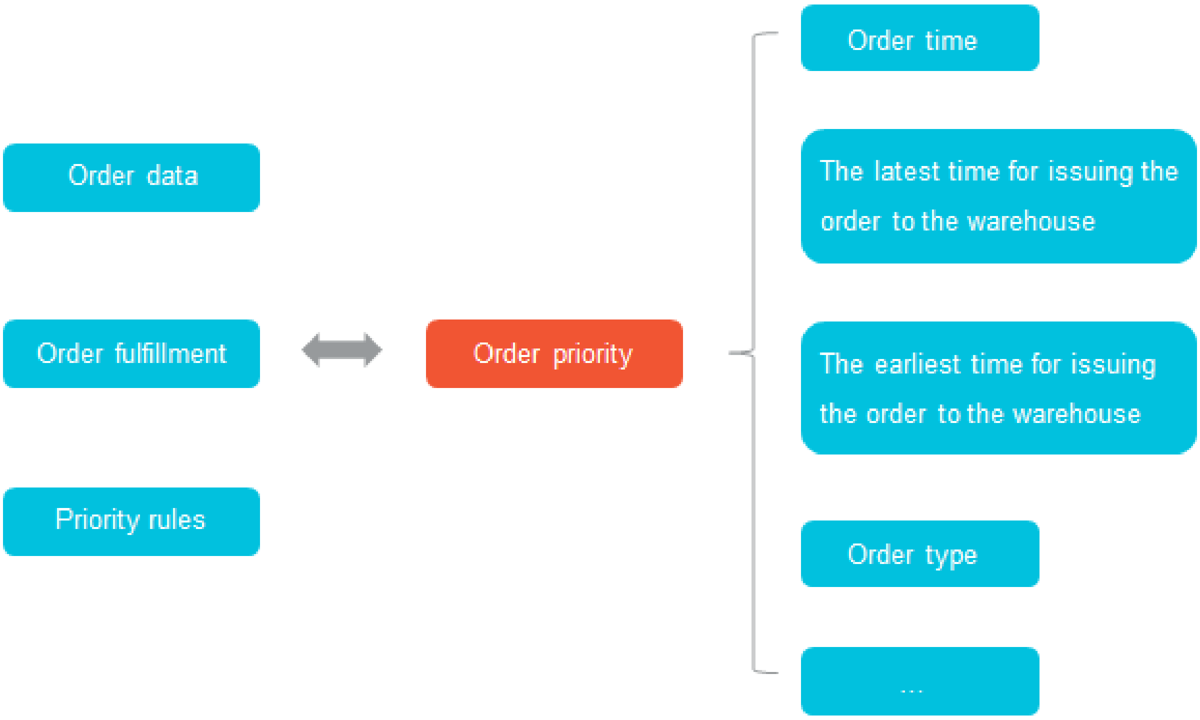
1. Obtain the current index ID.
2. Retrieve the index list.
3. Obtain the comments.



Sort orders in Cainiao order fulfillment center

After a user buys a commodity during Double 11 Shopping Festival, Cainiao warehouse and distribution system generates and processes a corresponding logistics order. The decision-making system generates an order fulfillment plan based on the order data. Therefore, the warehouse and distribution system can provide intelligent and collaborative services across each stage. The plan specifies the time for issuing the order to the warehouse, the time for outbound delivery, the time for item collection, and the time for delivering the item. The order fulfillmen t center provides the logistics service according to the order fulfillment plan. Due to the limited capacities of warehouses and distribution, the system processes the earliest orders in priority. Therefore, ApsaraDB for Redis sorts the orders by priority before the order fulfillment center issues them to the warehouse or for delivery.

The order fulfillment center uses ApsaraDB for Redis to sort logistics orders and determine the priorities of these orders.



8 Build an e-commerce short-term sales promotion system by using ApsaraDB for Redis

The short-term sales promotion is a common method for low-price promotion and brand marketing in the e-commerce industry. This method can bring more traffic and increase the visibility of your platform. An excellent short-term sales promotion system can improve the stability and fairness of the platform, provide a better user experience, and enhance the reputation of the platform. Therefore, this system can maximize the value of short-term sales promotions. This topic describes the short-term sales promotion system that processes high-concurrency requests based on the cache feature of ApsaraDB for Redis.

Characteristics of a short-term sales promotion

A short-term sales promotion is intended to sell scarce or special commodities for specified quantities in a timed manner, and attract a large number of buyers . However, only a few buyers place orders during the promotion. Therefore, the short-term sales promotion brings visits and order requests dozens or hundreds of times that in normal sales on your platform in a short period.

A short-term sales promotion is divided into three phases:

- Before the promotion: buyers constantly refresh the commodity details page, and the number of requests of the page reaches the instantaneous peak.
- The promotion starts: buyers click the promotion button, and the number of order requests reaches the instantaneous peak.
- After the promotion: some buyers that have successfully placed orders continue to refresh their orders or return the orders. Most buyers continue to refresh the commodity details page and wait for returning orders.

A database processes the requests based on row-level rocking when buyers submit orders. The database allows requests that hold the lock to query inventories and place orders. However, if the database cannot process high-concurrency requests, the service may be blocked. This causes server downtime to buyers.

Short-term sales promotion system

The traffic to the short-term sales promotion system is high, but the valid traffic is low. Based on the system hierarchy, if you verify and intercept invalid traffic in advance in each phase, you can reduce large amounts of invalid traffic to the database.

Use the browser cache and Content Delivery Network (CDN) to process traffic that comes to static pages

Before the short-term sales promotion, buyers constantly refresh the commodity details page. This results in a large number of requests to the page. Therefore, you can separate the commodity details page for the short-term sales promotion from that for normal sales. For the commodity details page in the short-term sales promotion, the Web server processes the clicks on the promotion button in real time, and the short-term sales promotion system caches static elements to the browser and CDN. In this way, only a fraction of the traffic caused by refreshing the page before the promotion goes to the Web server.

Use a read/write splitting instance of ApsaraDB for Redis to cache and intercept traffic

You can first use the CDN service to intercept traffic, and then use a read/write splitting instance of ApsaraDB for Redis to intercept more traffic. The read/write splitting instance can support more than 600,000 queries per second (QPS), and process a large number of read requests.

Use the data control module to cache the promotion commodity data to the read/write splitting instance in advance, and specify the flag for starting the promotion as follows:

```
"goodsId_count": 100 //The total number of commodities.
"goodsId_start": 0   //The flag to indicate that the promotion starts.
"goodsId_access": 0  //The number of orders that the promotion system
accepts.
```

1. Before the promotion starts, the server cluster for the short-term sales promotion reads goodsId_start as 0, and indicates in the response that the promotion has not started.
2. When the data control module changes goodsId_start to 1, the promotion starts.
3. The server cluster for the short-term sales promotion caches the promotion start flag and accepts order requests. The cluster records the requests to goodsId_access. The number of remaining commodities is the result of the value of goodsId_count minus the value of goodsId_access.

4. After the number of orders that the short-term sales promotion system accepts reaches the value of goodsId_count, the short-term sales promotion system intercepts all requests. The number of remaining commodities is 0.

In this way, the short-term sales promotion system accepts only a small fraction of the order requests. In the case of high concurrency, you can allow a bit more traffic to the system. Therefore, you can control the percentage of orders that the system accepts.

Use a master-replica instance of ApsaraDB for Redis to accelerate inventory deduction

After accepting an order, the short-term sales promotion system checks the order information and deducts the inventory. To avoid direct access to a database, you can use a master-replica instance of ApsaraDB for Redis to deduct the inventory. The instance supports more than 100,000 QPS. The ApsaraDB for Redis instance optimizes the inventory query, intercepts invalid order requests in advance, and increases the overall throughput of the short-term sales promotion system.

You can use the data control module to cache the inventory to the ApsaraDB for Redis instance in advance. The instance stores the commodity data for promotion in a hash table.

```
"goodsId" : {
    "Total": 100
    "Booked": 100
}
```

To deduct the inventory, the short-term sales promotion server runs the following Lua script and connects to the ApsaraDB for Redis instance to obtain the order permission. Due to Redis single-thread model, the Lua script ensures the atomicity of multiple commands.

```
local n = tonumber(ARGV[1])
if not n  or n == 0 then
    return 0
end
local vals = redis.call("HMGET", KEYS[1], "Total", "Booked");
local total = tonumber(vals[1])
local blocked = tonumber(vals[2])
if not total or not blocked then
    return 0
end
if blocked + n <= total then
    redis.call("HINCRBY", KEYS[1], "Booked", n)

    return n;
end
```

```
return 0
```

Run the `SCRIPT LOAD` command to cache the Lua script to the ApsaraDB for Redis instance in advance, and then run the `EVALSHA` command to call the script. This method requires less network bandwidth than you directly run the `EVAL` command.

```
redis 127.0.0.1:6379>SCRIPT LOAD "lua code"
"438dd755f3fe0d32771753eb57f075b18fed7716"
redis 127.0.0.1:6379>EVAL 438dd755f3fe0d32771753eb57f075b18fed7716 1
goodsId 1
```

If the ApsaraDB for Redis instance returns the value n as the number of commodities that buyers have ordered, the short-term sales promotion system determines that the current inventory deduction is successful.

Use a master-replica instance of ApsaraDB for Redis to asynchronously write order data to the database based on message queues

The short-term sales promotion system writes order data to the database after successful inventory deduction. For a few commodities, the system can directly perform operations in the database. If the number of commodities for promotion is more than 10,000 or 100,000, lock conflicts may occur and cause performance bottlenecks in the database. Therefore, to avoid direct operations in the database, the short-term sales promotion system writes order data to message queues to complete the order process.

1. The ApsaraDB for Redis instance provides message queues in a list structure.

   ```
   orderList {
       [0] = {Order content}
       [1] = {Order content}
       [2] = {Order content}
       ...
   }
   ```

2. The short-term sales promotion system writes order content to the ApsaraDB for Redis instance.

   ```
   LPUSH orderList {Order content}
   ```

3. The asynchronous order module sequentially retrieves order data from the ApsaraDB for Redis instance and writes order data to the database.

   ```
   BRPOP orderList 0
   ```

The ApsaraDB for Redis instance provides message queues and asynchronously writes order data to the database to efficiently complete the order process.

The data control module manages synchronization of promotion data

At the start, the short-term sales promotion system uses the read/write splitting instance of ApsaraDB for Redis to intercept traffic and allows a fraction of valid traffic to continue the order process. Afterward, the short-term sales promotion system has to process more traffic caused by order authentication failures and returning orders. Therefore, the data control module regularly computes data in the database, and synchronizes the data to the master-replica instance and then to the read/write splitting instance.

9 JedisPool optimization

Setting JedisPool parameters helps to improve the performance of Redis. This topic describes how to use JedisPool and configure the resource pool parameters. It also provides recommended parameter configurations to optimize JedisPool.

How to use JedisPool

Take Jedis 2.9.0 as an example. The Maven dependency is as follows:

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.9.0</version>
    <scope>compile</scope>
</dependency>
```

Jedis manages the resource pool by using Apache Commons-pool2. When you define JedisPool, pay attention to the `GenericObjectPoolConfig` **parameter (resource pool). The following example describes how to use this parameter.**

```
GenericObjectPoolConfig jedisPoolConfig = new GenericObjectPoolConfig
();
jedisPoolConfig.setMaxTotal(...);
jedisPoolConfig.setMaxIdle(...);
jedisPoolConfig.setMinIdle(...);
jedisPoolConfig.setMaxWaitMillis(...) ;
...
```

The initialization of JedisPool is as follows:

```
// redisHost indicates the instance IP address. redisPort indicates
 the instance port. redisPassword indicates the password of the
instance. The timeout parameter indicates both the connection timeout
and the read/write timeout.
JedisPool jedisPool = new JedisPool(jedisPoolConfig, redisHost,
redisPort, timeout, redisPasswor//d);
//Run the command as follows:
Jedis jedis = null;
try {
    jedis = jedisPool.getResource();
    // Specific commands
    jedis.executeCommand()
} catch (Exception e) {
    logger.error(e.getMessage(), e);
} finally {
    //In JedisPool mode, the Jedis resource will be returned to the
resource pool.
    if (jedis ! = null)
        jedis.close();
```

```
}
```

Parameter description

The Jedis connection is a resource managed by JedisPool in the connection pool. JedisPool is a thread-safe pool of connections and can keep all resources within a manageable range. Configure the `GenericObjectPoolConfig` parameter properly can improve the performance of Redis and reduce the resource consumption. The following two tables describe some important parameters and provide recommendations for parameter configuration.

Table 9-1: Parameters related to resource settings and resource usage

Parameter	Description	Default value	Recommendation
maxTotal	The maximum number of connections that can be allocated from this pool.	8	See *Recommendations for key parameter configurations*.
maxIdle	The maximum number of connections that can remain idle in the pool, without extra ones being released.	8	See *Recommendations for key parameter configurations*.
minIdle	The minimum number of connections that can remain idle in the pool, without extra ones being created.	0	See *Recommendations for key parameter configurations*.
blockWhenExhausted	Whether the caller has to wait when the resource pool is exhausted. The following `maxWaitMillis` takes effect only when this value is true.	true	We recommend that you use the default value.
maxWaitMillis	The maximum number of milliseconds that the caller needs to wait when no connection is available.	-1 means never timeout.	The default value is not recommended.
testOnBorrow	Whether the connections will be validated by using the ping command before they are borrowed from the pool. If the connection turns out to be invalid, it will be removed from the pool.	false	When the business traffic is high, we recommend that you set it to false to reduce the consumption of a ping test.

Parameter	Description	Default value	Recommendation
testOnReturn	Whether connections will be validated using the ping command before they are returned to the pool. If the connection turns out to be invalid, it will be removed from the pool.	false	When the business traffic is high, we recommend that you set it to false to reduce the consumption of a ping test.
jmxEnabled	Whether to enable JMX monitoring.	true	We recommand that you enable JMX monitoring. Note that your application itself also needs to be enabled.

Idle Jedis object detection consists of the following four parameters. `testWhileIdle` **is the switch of this feature.**

Table 9-2: Parameters related to idle resource detection

Name	Description	Default value	Recommendation
testWhileIdle	Whether to enable the idle resource detection.	false	true
timeBetweenEvictionRunsMillis	The number of milliseconds to sleep between runs of the idle object evictor thread.	-1 means no idle object evictor thread will be run.	We recommend that you set this parameter and make your own cycle selections. Or, you can use the default configuration in `JedisPoolConfig`.
minEvictableIdleTimeMillis	The minimum amount of time an object may sit idle in the pool before it is eligible for eviction by the idle object evictor (if any).	180000 (30 minutes)	The default value is suitable for most cases. You can also use the configuration in `JeidsPoolConfig` based on your business scenario.

Name	Description	Default value	Recommendation
numTestsPerEvictionRun	The number of objects to examine during each run of the idle object evictor thread (if any).	3	It can be changed according to the number of application connections. If this value is set to -1, the idle validation will be performed on all connections.

For your convenience, Jedis provides `JedisPoolConfig` that shares some configurations with `GenericObjectPoolConfig` in validating idle connections.

```
public class JedisPoolConfig extends GenericObjectPoolConfig {
  public JedisPoolConfig() {
    // defaults to make your life with connection pool easier :)
    setTestWhileIdle(true);
    //
    setMinEvictableIdleTimeMillis(60000);
    //
    setTimeBetweenEvictionRunsMillis(30000);
    setNumTestsPerEvictionRun(-1);
    }
}
```

Note:
You can view all the default values in `org.apache.commons.pool2.impl.BaseObjectPoolConfig`.

Recommendations for key parameter configurations

`maxTotal`: The maximum number of connections.

To set `maxTotal` properly, you need to consider the following factors:

- The Redis concurrent connections required by the business.
- How long it takes for the client to execute the command.
- The limit of Redis resources. For example, the product of multiplying maxTotal by the number of nodes (applications) must be smaller than the allowed maximum number of connections in Redis.
- The cost of creating and releasing connections. When the number of connections created and released is high for each request, the creation and release process takes a heavy toll.

The time for running a command to obtain a resource consists of the time for borrowing and returning the resource, the time for JedisPool to run the command, and the time for network connection. If the average time for running a command to obtain a resource is about 1 ms, the QPS of a connection is about 1,.000, and the QPS expected by the business is 50000, then theoretically the required pool size is 50 (50,000 / 1,000 = 50).

But this is only a theoretical value. To reserve some resources, the value of the `maxTotal` parameter can be larger than the theoretical value. However, if this value is too large, the connections will consume too much client and server resources. On the other hand, for servers like Redis that has a high QPS, if there is a blocking of commands, even a large resource pool cannot help to solve this problem.

`maxIdle and minIdle`

`maxIdle` is the actual maximum number of connections required by the business, while `maxTotal` includes the number of idle connections as a surplus. If `maxIdle` is set too low on heavily loaded systems, `new Jedis` (extra connections) will be created to serve the requests. Therefore, `minIdle` is configured to specify the minimum number of established connections that need to be kept in the pool.

The connection pool reaches its best performance when `maxTotal=maxIdle`. In this way, the performance is not affected by the scaling of the connection pool. However, if the amount of concurrent connections is small or the `maxTotal` parameter is set too high, the connection resources will be wasted.

You can evaluate the connection pool size used by each node based on the actual total QPS and the number of clients that call Redis.

Use monitoring tools to obtain the best value

Using monitoring tools to obtain the best parameter value is a more reliable method. You can use JMX monitoring or other monitoring tools to discover the best value.

FAQ

Insufficient resources

In the following two cases, you cannot obtain resources from the resource pool.

- **Timeout:**

```
redis.clients.jedis.exceptions.JedisConnectionException: Could not
get a resource from the pool
…
Caused by: Java. util. nosuchelementexception timeout waiting for
idle object
at org.apache.commons.pool2.impl.GenericObjectPool.borrowObject(
GenericObjectPool.java:449)
```

- **When you set the `blockWhenExhausted` to false, the time specified with borrowMaxWaitMillis will not be used and the borrowObject call will block until there is an idle connection available.**

```
redis.clients.jedis.exceptions.JedisConnectionException: Could not
get a resource from the pool
…
Caused by: java.util.NoSuchElementException: Pool exhausted
at org.apache.commons.pool2.impl.GenericObjectPool.borrowObject(
GenericObjectPool.java:464)
```

This exception occurs not necessarily because the pool size is limited. For more information, see *Recommendations for key parameter configurations*. To solve this problem, we recommend that you check the network, parameter configurations of the resource pool, the resource pool monitoring (JMX monitoring), the code (for example, the reason may be that `jedis.close()` is not executed.), the slow queries, and DNS.

Warm up JedisPool

For some reasons (such as a small timeout setting), the project may time out after it is started. JedisPool does not create a Jedis connection in the connection pool when it defines the maximum number of resources and the minimum number of idle resources. When there is no idle connection in the pool, a `new Jedis` connection will be created. This connection will be released to the pool after it is used. However, creating a new connection and releasing it every time is a time-consuming process. Therefore, we recommend that you warm up JedisPool with the minimum number of idle connections after JedisPool is defined. The example is as follows:

```
List<Jedis> minIdleJedisList = new ArrayList<Jedis>(jedisPoolConfig.
getMinIdle());

for (int i = 0; i < jedisPoolConfig.getMinIdle(); i++) {
    Jedis jedis = null;
    try {
        jedis = pool.getResource();
        minIdleJedisList.add(jedis);
```

```
        jedis.ping();
    } catch (Exception e) {
        logger.error(e.getMessage(), e);
    } finally {
    }
}

for (int i = 0; i < jedisPoolConfig.getMinIdle(); i++) {
    Jedis jedis = null;
    try {
        jedis = minIdleJedisList.get(i);
        jedis.close();
    } catch (Exception e) {
        logger.error(e.getMessage(), e);
    } finally {

    }
}
```

10 Analyze hotkeys in a specific sub-node of a cluster instance

You can use the imonitor command developed by Alibaba Cloud to monitor the request status of a specific node in the Redis cluster, and use redis-faina to discover hotkeys and commands from the monitoring data.

Background information

When you use the ApsaraDB for Redis cluster edition, if the hotkey traffic on a specific node is too large, other services in the server may fail to continue. If the cache of the hotkey exceeds the current cache capacity, the sharding service of the cache will crash.

You can use *Performance monitoring* **and** *#unique_15* **to monitor the cluster status in real time and set alert rules. When you discover an overloaded sub-node, you can use the imonitor command to view the client request of the node, and use redis-faina to analyze the hotkey.**

Prerequisites

- **You have activated an ECS instance that can interconnect with the ApsaraDB for Redis cluster edition.**
- **You have installed Python and Telnet in the ECS instance.**

> **Note:**
> **The sample environment in this topic is CentOS 7.4 and Python 2.7.5.**

Procedure

1. **In the ECS instance, use Telnet to connect to the Redis cluster.**
    a. **Use** `# telnet <host> <port>` **to connect to the Redis cluster.**

    > **Note:**

`host` is the connection address of the Redis cluster. `port` is the connection port (the default port number is 6379).

b. Enter `auth <password>` for verification.

> **Note:**
> `password` is the password for the Redis cluster.

```
Welcome to Alibaba Cloud Elastic Compute Service !

[root@redisTest ~]# telnet r-b[redacted]4.redis.rds.aliyuncs.com 6379
Trying 1[redacted]...
Connected to r-b[redacted]4.redis.rds.aliyuncs.com.
Escape character is '^]'.
auth [redacted]a
+OK
```

> **Note:**
> If `+OK` is returned, the connection is successful.

2. Use `imonitor <db_idx>` to collect the request data of the target node.

```
imonitor 0
+OK
+1543975816.789076 [0 [redacted]] "INFO" "replication"
+1543975833.071774 [0 [redacted]] "INFO" "replication"
+1543975842.251665 [0 127.0.0.1:42442] "INFO" "keyspace"
+1543975842.262597 [0 127.0.0.1:42442] "INFO" "all"
+1543975848.336031 [0 [redacted]] "INFO" "replication"
```

> **Note:**
> The `imonitor` command is similar to the `iinfo`command and the `iscan` command. This command added a parameter to the `monitor` command, and the user can specify the node to run the `monitor` command. In this command, the value range of db_idx is [0, nodecount). You can obtain the value of nodecount by running the `info` command or viewing the instance topology in the console.
>
> In this example, the value of db_idx of the target node is 0.
>
> If `+OK` is returned, the output of of monitored request records continues.

3. Collect the monitoring data based on your business requirements and enter the `QUIT` command. Press Enter to close the Telnet connection.

4. **Store the monitoring data to a `.txt` file, and delete the plus sign (+) at the beginning of the line. You can replace this sign by using the text editing tool. The stored file is as follows:**

```
[root@redisTest ~]# cat imonitorOut.txt
1543995847.659482 [0 [redacted]] "INFO" "replication"
1543995856.057381 [0 127.0.0.1:58802] "INFO" "keyspace"
1543995856.070002 [0 127.0.0.1:58802] "INFO" "all"
1543995861.653458 [0 [redacted]] "INFO" "ALL"
1543995862.782848 [0 [redacted]] "INFO" "ALL"
1543995862.799096 [0 [redacted]] "INFO" "ALL"
1543995862.863230 [0 [redacted]] "INFO" "CLUSTER"
1543995862.876389 [0 [redacted]] "scan" "0" "MATCH" "*" "COUNT" "3000"
1543995862.942649 [0 [redacted] "INFO" "replication"
1543995862.943303 [0 [redacted]] "TYPE" "customer:18016"
1543995862.955943 [0 [redacted]] "TYPE" "customer:17167"
```

5. **Create a Python script for request analysis, and save it as *redis-faina.py*. The code is as follows:**

```
#! /usr/bin/env python
import argparse
import sys
from collections import defaultdict
import re

line_re_24 = re.compile(r"""
    ^(? P<timestamp>[\d\.]+)\s(\(db\s(? P<db>\d+)\)\s)?"(? P<command
>\w+)"(\s"(? P<key>[^(? <! \\)"]+)(? <! \\)")?( \s(? P<args>. +))? $
    """, re.VERBOSE)

line_re_26 = re.compile(r"""
    ^(? P<timestamp>[\d\.]+)\s\[(? P<db>\d+)\s\d+\.\d+\.\d+\.\d+:\d
+]\s"(? P<command>\w+)"(\s"(? P<key>[^(? <! \\)"]+)(? <! \\)")?( \s
(? P<args>. +))? $
    """, re.VERBOSE)

class StatCounter(object):

    def __init__(self, prefix_delim=':', redis_version=2.6):
        self.line_count = 0
        self.skipped_lines = 0
        self.commands = defaultdict(int)
        self.keys = defaultdict(int)
        self.prefixes = defaultdict(int)
        self.times = []
        self._cached_sorts = {}
        self.start_ts = None
        self.last_ts = None
        self.last_entry = None
        self.prefix_delim = prefix_delim
        self.redis_version = redis_version
        self.line_re = line_re_24 if self.redis_version < 2.5 else
line_re_26

    def _record_duration(self, entry):
        ts = float(entry['timestamp']) * 1000 * 1000 # microseconds
        if not self.start_ts:
            self.start_ts = ts
            self.last_ts = ts
        duration = ts - self.last_ts
```

```
        if self.redis_version < 2.5:
            cur_entry = entry
        else:
            cur_entry = self.last_entry
            self.last_entry = entry
        if duration and cur_entry:
            self.times.append((duration, cur_entry))
        self.last_ts = ts

    def _record_command(self, entry):
        self.commands[entry['command']] += 1

    def _record_key(self, key):
        self.keys[key] += 1
        parts = key.split(self.prefix_delim)
        if len(parts) > 1:
            self.prefixes[parts[0]] += 1

    @staticmethod
    def _reformat_entry(entry):
        max_args_to_show = 5
        output = '"%(command)s"' % entry
        if entry['key']:
            output += ' "%(key)s"' % entry
        if entry['args']:
            arg_parts = entry['args'].split(' ')
            ellipses = ' ...' if len(arg_parts) > max_args_to_show
else ''
            output += ' %s%s' % (' '.join(arg_parts[0:max_args_t
o_show]), ellipses)
        return output


    def _get_or_sort_list(self, ls):
        key = id(ls)
        if not key in self._cached_sorts:
            sorted_items = sorted(ls)
            self._cached_sorts[key] = sorted_items
        return self._cached_sorts[key]

    def _time_stats(self, times):
        sorted_times = self._get_or_sort_list(times)
        num_times = len(sorted_times)
        percent_50 = sorted_times[int(num_times / 2)][0]
        percent_75 = sorted_times[int(num_times * . 75)][0]
        percent_90 = sorted_times[int(num_times * . 90)][0]
        percent_99 = sorted_times[int(num_times * . 99)][0]
        return (("Median", percent_50),
                ("75%", percent_75),
                ("90%", percent_90),
                ("99%", percent_99))

    def _heaviest_commands(self, times):
        times_by_command = defaultdict(int)
        for time, entry in times:
            times_by_command[entry['command']] += time
        return self._top_n(times_by_command)

    def _slowest_commands(self, times, n=8):
        sorted_times = self._get_or_sort_list(times)
        slowest_commands = reversed(sorted_times[-n:])
        printable_commands = [(str(time), self._reformat_entry(entry
)) \
                              for time, entry in slowest_commands]
```

```
        return printable_commands

    def _general_stats(self):
        total_time = (self.last_ts - self.start_ts) / (1000*1000)
        return (
            ("Lines Processed", self.line_count),
            ("Commands/Sec", '%. 2f' % (self.line_count / total_time
))
        )

    def process_entry(self, entry):
        self._record_duration(entry)
        self._record_command(entry)
        if entry['key']:
            self._record_key(entry['key'])

    def _top_n(self, stat, n=8):
        sorted_items = sorted(stat.iteritems(), key = lambda x: x[1
], reverse = True)
        return sorted_items[:n]

    def _pretty_print(self, result, title, percentages=False):
        print title
        print '=' * 40
        if not result:
            print 'n/a\n'
            return

        max_key_len = max((len(x[0]) for x in result))
        max_val_len = max((len(str(x[1])) for x in result))
        for key, val in result:
            key_padding = max(max_key_len - len(key), 0) * ' '
            if percentages:
                val_padding = max(max_val_len - len(str(val)), 0) *
 ' '
                val = '%s%s\t(%. 2f%%)' % (val, val_padding, (float(
val) / self.line_count) * 100)
            print key,key_padding,'\t',val
        print


    def print_stats(self):
        self._pretty_print(self._general_stats(), 'Overall Stats')
        self._pretty_print(self._top_n(self.prefixes), 'Top Prefixes
', percentages = True)
        self._pretty_print(self._top_n(self.keys), 'Top Keys',
percentages = True)
        self._pretty_print(self._top_n(self.commands), 'Top Commands
', percentages = True)
        self._pretty_print(self._time_stats(self.times), 'Command
Time (microsecs)')
        self._pretty_print(self._heaviest_commands(self.times), '
Heaviest Commands (microsecs)')
        self._pretty_print(self._slowest_commands(self.times), '
Slowest Calls')

    def process_input(self, input):
        for line in input:
            self.line_count += 1
            line = line.strip()
            match = self.line_re.match(line)
            if not match:
                if line ! = "OK":
                    self.skipped_lines += 1
```

```
                continue
            self.process_entry(match.groupdict())

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        'input',
        type = argparse.FileType('r'),
        default = sys.stdin,
        nargs = '?',
        help = "File to parse; will read from stdin otherwise")
    parser.add_argument(
        '--prefix-delimiter',
        type = str,
        default = ':',
        help = "String to split on for delimiting prefix and rest of
 key",
        required = False)
    parser.add_argument(
        '--redis-version',
        type = float,
        default = 2.6,
        help = "Version of the redis server being monitored",
        required = False)
    args = parser.parse_args()
    counter = StatCounter(prefix_delim = args.prefix_delimiter,
redis_version = args.redis_version)
    counter.process_input(args.input)
    counter.print_stats()
```

Note:

The preceding script is from *redis-faina*.

6. **Run the** `python redis-faina imonitorOut.txt` **command to parse the monitoring data.** *`imonitorOut.txt`* **is the monitoring data stored in the example.**

```
[root@redisTest ~]# python redis-faina.py imonitorOut.txt
Overall Stats
========================================
Lines Processed         311
Commands/Sec            0.88

Top Prefixes
========================================
customer                132     (42.44%)
user_agent              24      (7.72%)
simple_registration     12      (3.86%)
detailed_registration   9       (2.89%)
company                 4       (1.29%)

Top Keys
========================================
customer:1446           122     (39.23%)
ALL                     68      (21.86%)
replication             29      (9.32%)
all                     15      (4.82%)
keyspace                15      (4.82%)
user_agent:17358        8       (2.57%)
user_agent:10722        4       (1.29%)
customer:4968           1       (0.32%)

Top Commands
========================================
INFO    128     (41.16%)
HGET    121     (38.91%)
TYPE    50      (16.08%)
HLEN    3       (0.96%)
TTL     3       (0.96%)
HSCAN   3       (0.96%)
scan    1       (0.32%)
GET     1       (0.32%)

Command Time (microsecs)
========================================
Median          603448.0
75%             1556677.0
90%             5215846.0
99%             8019603.0

Heaviest Commands (microsecs)
========================================
INFO    231775519.75
HGET    103355620.75
GET     7377767.75
HLEN    6155302.75
HSCAN   2166953.0
TYPE    2031287.75
scan    66260.0
TTL     35047.25

Slowest Calls
========================================
8397898.75      "INFO" "replication"
8101143.0       "INFO" "ALL"
8079963.75      "INFO" "ALL"
```

Note:

In the preceding analysis result, Top Keys displays the most requested keys during this time period, and Top Commands displays the most frequently used commands. You can solve the hotkey problem based on the analysis results.

11 Use ApsaraDB for Redis to build a broadcasting channel information system

You can use ApsaraDB for Redis to build a broadcasting channel information system that has a low latency and can handle high traffic volumes.

Background information

The broadcasting channel is one of the key features of the live broadcasting system . Except for the broadcasting window, online users, online gifts, comments, likes , rankings, and other data generated in the live broadcast is time-limited, highly interactive, and delay-sensitive. Redis cache service is a suitable solution to handle such data.

The best practice in this topic introduces how to use ApsaraDB for Redis to build a broadcasting channel information system. This topic describes the construction methods for three information types:

- **Real-time ranking information**
- **Counting information**
- **Timeline information**

Real-time ranking information

The real-time ranking information includes an online user list, a list of online gifts, and live comments. Live comments are similar to a message ranking list that is sorted based on the message dimension. The sorted set structure in Redis is suitable to handle the real-time ranking information.

The Redis set is stored in a hash table. The time complexity of the insert, delete, edit, and search operations is O(1). Each member in the set is associated with a score to facilitate sorting and other operations. The following example describes the added and returned live comments to explain how the sorted set works to build a broadcasting channel information system.

- **Uses "unix timestamp + milliseconds" as the score to record the last five live comments in the user55 broadcasting channel.**

```
redis> ZADD user55:_danmu 1523959031601166 message111111111111
(integer) 1
```

```
11.160.24.14:3003> ZADD user55:_danmu 1523959031601266 message222
222222222
(integer) 1
11.160.24.14:3003> ZADD user55:_danmu 1523959088894232 message33333
(integer) 1
11.160.24.14:3003> ZADD user55:_danmu 1523959090390160 message444444
(integer) 1
11.160.24.14:3003> ZADD user55:_danmu 1523959092951218 message5555
(integer) 1
```

- **Returns the last three live comments:**

```
redis> ZREVRANGEBYSCORE user55:_danmu +inf -inf LIMIT 0 3
1) "message5555"
2) "message444444"
3) "message33333"
```

- **Returns three live comments within the specified time period:**

```
redis> ZREVRANGEBYSCORE user55:_danmu 1523959088894232 -inf LIMIT 0
3
1) "message33333"
2) "message222222222222"
3) "message111111111111"
```

Counting information

In case of the user-related data, the counting information includes the number of unread messages, followers, and fans, and the experience value. The hash structure in Redis is suitable to process this type of data. For example, the number of followers can be processed as follows:

```
redis> HSET user:55 follower 5
(integer) 1
redis> HINCRBY user:55 follower 1 //The number of followers +1
(integer) 6
redis> HGETALL user:55
1) "follow"
2) "6"
```

Timeline information

The timeline information is a list of information sorted in time order. For example, the broadcaster moments and new posts. This type of information is arranged in a fixed chronological order and can be stored using a Redis list or an ordered list. The example is as follows:

```
redis> LPUSH user:55_recent_activitiy  '{datetime:201804112010,type:
publish,title:The show starts, content:Come on}'
(integer) 1
redis> LPUSH user:55_recent_activitiy '{datetime:201804131910,type:
publish,title: Ask for a leave, content: Sorry, I have plans today.}'
(integer) 2
redis> LRANGE user:55_recent_activitiy 0 10
```

```
1) "{datetime:201804131910,type:publish,title:\xe8\xaf\xb7\xe5\x81\x87\x87
\",content:\xe6\x8a\xb1\xe6\xad\x89\xef\xbc\x8c\xe4\xbb\x8a\xe5\xa4\
xa9\xe6\x9c\x89\xe4\xba\x8b\xe9\xb8\xbd\xe4\xb8\x80\xe5\xa4\xa9}"
2) "{datetime:201804112010,type:publish,title:\xe5\xbc\x80\xe6\x92\xad
\xe5\x95\xa6,content:\xe5\x8a\xa0\xe6\xb2\xb9}"
```

Related resources

- **For more information about how to eliminate potential risks and locate business performance bottlenecks, see** *Analyze memory usage of ApsaraDB for Redis***.**
- **For more information about how to handle high concurrency, see** *ApsaraDB for Redis cluster edition***.**

12 Parse AOF files

The append only file (AOF) is the main Redis persistence option. You can parse AOF files to view the history commands and the records of a specific key.

Redis persistence options

- **RDB snapshot option: This option performs point-in-time snapshots of your dataset at specified intervals. It encodes the keys and values as Redis strings and stores the strings in RDB files.**
- **AOF persistence option: Similar to binlog, append-only files keep a record of data changes that occur by writing each change to the end of the file. The entire dataset can be recovered by replaying the append-only log from the beginning to the end.**

Details of the AOF persistence option

Redis clients communicate with the Redis server using a protocol called REdis Serialization Protocol (RESP). RESP can serialize different data types, including:

- **Simple strings:**

  A string that starts with the plus sign (+) and ends with rn. For example, +OKrn.
- **Error messages:**

  A string that starts with the minus sign (-) and ends with rn. For example, -ERR Readonlyrn.
- **Integers**

  A data structure that starts with a colon (:), ends with rn, and has an integer between the beginning and the end. For example, (:1rn).
- **Large strings**

  A string structure that starts with a dollar sign ($), followed by the string length (less than 512 MB) and rn, and ends with the string content and rn. For example, $ 0rnrn.
- **Arrays**

  A data structure that starts with an asterisk symbol (*), followed by array elements that are separated by rn. The above four data types can be used as array elements. For example, *1rn$4rnpingrn.

The Redis client sends an array command to the server. The server responds according to the implementation methods of different commands and records the responses in the AOF file.

Parse append-only files

The following sample parses an append-only file by calling hiredis with Python. The sample code is described as follows:

```
#! /usr/bin/env python

""" A redis appendonly file parser
"""

import logging
import hiredis
import sys

if len(sys.argv) ! = 2:
   print sys.argv[0], 'AOF_file'
   sys.exit()
file = open(sys.argv[1])
line = file.readline()
cur_request = line
while line:
    req_reader = hiredis.Reader()
    req_reader.setmaxbuf(0)
    req_reader.feed(cur_request)
    command = req_reader.gets()
    try:
        if command is not False:
            print command
            cur_request = ''
    except hiredis.ProtocolError:
        print 'protocol error'
    line = file.readline()
    cur_request += line
file.close
```

The result is as follows: After you obtain the following results, you can view the operations related to a certain key at any time.

```
['PEXPIREAT', 'RedisTestLog', '1479541381558']
['SET', 'RedisTestLog', '39124268']
['PEXPIREAT', 'RedisTestLog', '1479973381559']
['HSET', 'RedisTestLogHash', 'RedisHashField', '16']
['PEXPIREAT', 'RedisTestLogHash', '1479973381561']
['SET', 'RedisTestLogString', '79146']
```

13 How to discover hotkeys in Redis 4.0

High performance is the most prominent feature of Redis. A robust Redis performance is crucial to ensure the service availability. A reduced Redis performance can be caused by multiple reasons. The hotkey problem is one of the most common reasons. The discovery of hotkeys is the first step to improve Redis performance. This topic describes how to use the new features of Redis 4.0 to discover the hotkeys.

Background

Redis 4.0 added two data eviction strategies: `allkey-lfu` **and** `volatile-lfu`**. You can also run the** `OBJECT` **command to obtain the access frequency of a specific key, as shown in the following figure.**

```
r-[redacted].redis.rds.aliyuncs.com:6379> OBJECT FREQ mylist
(integer) 220
```

The native Redis client also added the `--hotkeys` **option to help you discover hotkeys in your business.**

Note:
This topic describes how to discover hotkeys to optimize the performance of Redis. This topic is suitable for users who are familiar with the basic features of ApsaraDB for Redis and are seeking advanced skills. If you are not familiar with Redis, we recommend that you read *Product Overview*.

Prerequisites

- **You have activated an ECS instance that can interconnect with the ApsaraDB for Redis instance.**
- **You have installed a Redis version later than Redis 4.0 on the ECS instance.**

Note:
You can use the redis-cli tool based on these prerequisites.

- The `maxmemory-policy` parameter of the ApsaraDB for Redis instance is set to `volatile-lfu` or `allkeys-lfu`.

> **Note:**
> For more information about how to modify the parameters, see *#unique_22*.

Procedure

1. When there is an ongoing business, use the following command to query the hotkey.

```
redis-cli -h r-***************.redis.rds.aliyuncs.com -a <password>
 --hotkeys
```

> **Note:**
> This topic uses *redis-benchmark* to simulate a scenario featuring a high volume of writes.

Table 13-1: Option descriptions

Option	Description
-h	Specifies the server hostname.
-a	Specifies the password for Redis Auth.
--hotkeys	Used to query hotkeys.

Results

The following example shows the result of running this command.

```
[root@yaozhou src]# redis-cli -h r-[redacted].redis.rds.aliyuncs.com --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).

[21.01%] Hot key 'key:__rand_int__' found so far with counter 167
[39.46%] Hot key 'mylist' found so far with counter 167
[67.29%] Hot key 'counter:__rand_int__' found so far with counter 51
[82.73%] Hot key 'myset:__rand_int__' found so far with counter 63

-------- summary -------

Sampled 5008 keys in the keyspace!
hot key found with counter: 167 keyname: key:__rand_int__
hot key found with counter: 167 keyname: mylist
hot key found with counter: 63  keyname: myset:__rand_int__
hot key found with counter: 51  keyname: counter:__rand_int__
```

The `summary` part in the result is the hotkey.

14 Analyze memory usage of ApsaraDB for Redis

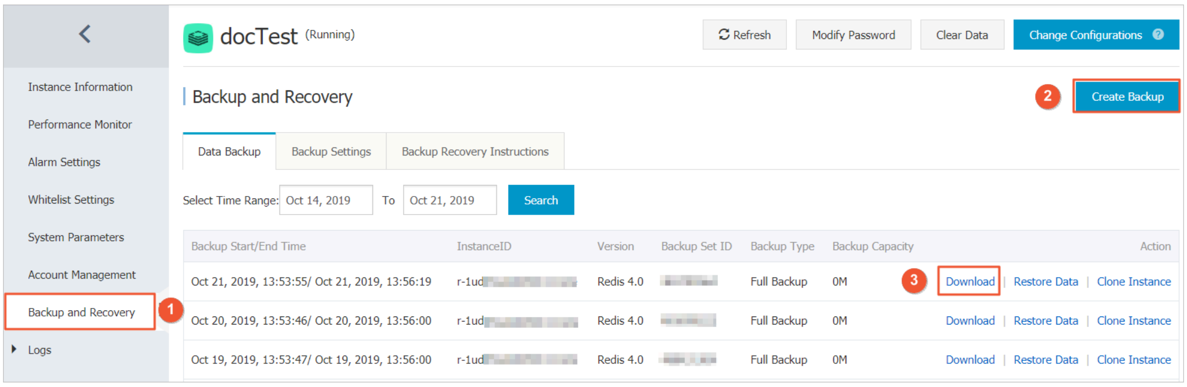
The data structure in ApsaraDB for Redis has a significant impact on service performance. If the number of big keys is large, the service performance or even service stability may deteriorate. Regular memory analysis and optimization can ensure service stability and efficiency. To avoid impacts on online services, you can run the `BGSAVE` command to generate an RDB file and use redis-rdb-tools and SQLite to analyze the file offline.

Prerequisites

- **A Linux-based Elastic Compute Service (ECS) instance is created.**
- **The SQLite database is installed on the ECS instance.**

Create an RDB file

- **For an ApsaraDB for Redis instance, back up the instance and download the backup data as an RDB file in the ApsaraDB for Redis console. For more information, see** *#unique_24/unique_24_Connect_42_section_3pv_4sf_80e*.



- **For an on-premises Redis database, run the `BGSAVE` command on the client to generate an RDB file.**

Introduction to redis-rdb-tools

You need to use redis-rdb-tools to generate a memory snapshot from the RDB file that is obtained. redis-rdb-tools is a Python tool used to parse RDB files. It supports the following features:

- **Generate a memory snapshot.**
- **Convert data in an RDB file to JSON format.**
- **Compare two RDB files to find their differences.**

Install redis-rdb-tools

You can install redis-rdb-tools in either of the following ways:

- **Install it from Python Package Index (PyPI) on the ECS instance.**

```
pip install rdbtools
```

- **Install it from source code on the ECS instance.**

```
git clone https://github.com/sripathikrishnan/redis-rdb-tools
cd redis-rdb-tools
sudo python setup.py install
```

Use redis-rdb-tools to generate a memory snapshot

Run the following command on the ECS instance to generate a memory snapshot in CSV format:

```
rdb -c memory dump.rdb > memory.csv
```

The memory snapshot contains the following data:

- **Database ID**
- **Data type**
- **Key**
- **Memory usage (in bytes), including the memory occupied by the key-value pair and other values**

  Note:
  The memory usage is a theoretical approximation. Generally, it is slightly lower than the actual value.
- **Encoding**

A sample CSV file is as follows:

```
$head memory.csv
database,type,key,size_in_bytes,encoding,num_elements,len_larges
t_element
0,string,"orderAt:377671748",96,string,8,8,
0,string,"orderAt:413052773",96,string,8,8,
0,sortedset,"Artical:Comments:7386",81740,skiplist,479,41,
0,sortedset,"pay:id:18029",2443,ziplist,84,16,
```

```
0,string,"orderAt:452389458",96,string,8,8
```

Import the CSV file to the SQLite database

SQLite is a lightweight relational database. After importing the CSV file to the SQLite database, you can use SQL statements to analyze the data in the CSV file.

Note:

- **The SQLite version must be 3.16.0 or later.**
- **Before importing the CSV file, delete the comma (,) at the end of each line in the CSV file.**

Run the following commands to import the CSV file:

```
sqlite3 memory.db
sqlite> create table memory(database int,type varchar(128),key varchar
(128),size_in_bytes int,encoding varchar(128),num_elements int,
len_largest_element varchar(128));
sqlite>.mode csv memory
sqlite>.import memory.csv memory
```

Analyze the memory snapshot generated by redis-rdb-tools

After importing the CSV file to the SQLite database, you can use SQL statements to analyze the data in the CSV file. For example:

- **Query the number of keys in the memory.**

```
sqlite>select count(*) from memory;
```

- **Query the total memory usage.**

```
sqlite>select sum(size_in_bytes) from memory;
```

- **Query the top 10 keys with the highest memory usage.**

```
sqlite>select * from memory order by size_in_bytes desc limit 10;
```

- **Query lists with over 1,000 elements.**

```
sqlite>select * from memory where type='list' and num_elements >
1000;
```