

Alibaba Cloud ApsaraDB for Redis

FAQs

Issue: 20190910

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, consequential, 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 content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of the Alibaba Cloud website, product programs, or content 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.

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information 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.
	This warning information 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 restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	 Notice: Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	 Note: You can use Ctrl + A to select all files.
>	Multi-level menu cascade.	Settings > Network > Set network type
Bold	It is used for buttons, menus, page names, and other UI elements.	Click OK .
Courier font	It is used for commands.	Run the <code>cd / d C :/ windows</code> command to enter the Windows system folder.
<i>Italics</i>	It is used for parameters and variables.	<code>bae log list --instanceid <i>Instance_ID</i></code>
[] or [a b]	It indicates that it is an optional value, and only one item can be selected.	<code>ipconfig [-all -t]</code>

Style	Description	Example
<code>{}</code> or <code>{a b}</code>	It indicates that it is a required value, and only one item can be selected.	<code>swich {stand slave}</code>

Contents

Legal disclaimer.....	I
Generic conventions.....	I
1 Troubleshooting for connection issues in ApsaraDB for Redis.....	1
2 Common Jedis exceptions in ApsaraDB for Redis.....	5
3 Comparison among Redis 4.0, Codis, and ApsaraDB for Redis clusters.....	27
4 View memory usage of all instances under the current account.....	34
5 Why cannot an ECS instance connect to an ApsaraDB for Redis instance?.....	35
6 How do I search for large keys?.....	36
7 How do I view the memory of child instances of an ApsaraDB for Redis cluster instance?.....	39
8 Troubleshooting for the JedisPool error.....	42
9 Analysis of the Redis CSRF vulnerability and the corresponding security measure in ApsaraDB for Redis..	45
10 Can an ECS instance in a VPC connect to an ApsaraDB for Redis instance in a classic network?.....	49
11 How can I obtain the requestId?.....	50
12 Connect ECS and ApsaraDB for Redis instances in different networks.....	51
13 Connect an ApsaraDB for Redis instance to an ECS instance across different accounts.....	53
14 Delete multiple keys in the Linux operating system.....	56
15 View the architecture and monitoring data of an ApsaraDB for Redis cluster instance.....	58

1 Troubleshooting for connection issues in ApsaraDB for Redis

ApsaraDB for Redis supports multiple connection methods. This topic describes the troubleshooting for different connection issues.

Connection issues between an ApsaraDB for Redis instance and an ECS instance

The ApsaraDB for Redis instance only supports connections over an internal network. Therefore, make sure that the ECS instance be connected with the ApsaraDB for Redis instance. If these instances cannot interconnect with each other, possible causes are listed as follows:

- The ApsaraDB for Redis instance and the ECS instance do not belong to the same account. Solution:
 - Migrate the ApsaraDB for Redis instance to the virtual private cloud (VPC) of the account that the ECS instance belongs to. For more information about how to migrate the ApsaraDB for Redis instance across accounts, see [#unique_4](#).
 - [#unique_5](#).
- The ECS instance and the ApsaraDB for Redis instance run in different regions. Solution:

Migrate one of the instances across regions, see [#unique_6](#).
- The ECS instance and the ApsaraDB for Redis instance run in different types of networks. one runs in a classic network and the other runs in a VPC. Solution:
 - Switch the network type of the Redis instance from classic network to VPC. For more information, see [#unique_7](#).
 - [#unique_8](#).
- The ECS security group rules restrict the connections to the connection address and port of the ApsaraDB for Redis instance. Solution:

[#unique_9](#) to allow the connections to the connection address and port of the ApsaraDB for Redis instance.

- The whitelists of the ApsaraDB for Redis instance do not include the internal IP address of the ECS instance. Solution:

[set the IP whitelists](#) to include the internal IP address of the ECS instance.



Note:

In the case of the following error: `Caused by : redis . clients . jedis . exceptions . JedisConnectionExcep tion : java . net . ConnectExc eption : Connection refused` , check the whitelists of the ApsaraDB for Redis instance. If the whitelist settings are correct and the ECS instance can reach the ApsaraDB for Redis instance by using [ping](#) messages, check the connection configuration in your application.

- An abnormal behavior on the ECS instance can trigger a security policy to disable the service. Multiple ECS instances normally connect to the ApsaraDB for Redis instance. Then, one of the ECS instance has a sudden connection issue. For example, the ECS instance can reach the ApsaraDB for Redis instance by using [ping](#) messages, but the ECS instance fails to connect to port 6379 by using the [telnet](#) command. In this case, the abnormal behavior such as outbound attacks on the ECS instance may cause the service to be disabled. Solution:

Check the ECS instance and set precise outbound rules in a security group. For example, you can define that the ECS instance can only connect to the required IP address and port such as port 6379 of the ApsaraDB for Redis instance. If the issue persists, submit a ticket to request technical support.

- During DNS resolution, the client has these errors: `UnknownHos tException or failed to connect : r -*****. redis . rds . aliyuncs . com could not be resolved` . Solution:

Use the [ping](#) message or the [telnet](#) command to test the result of DNS resolution for the ApsaraDB for Redis instance. If the resolution failed, check the DNS configuration.



Note:

If these solutions are not available due to limited conditions, submit a ticket to create an ECS instance or ApsaraDB for Redis instance again and run both instances in the same VPC.

Connect to an ApsaraDB for Redis instance from a public network

To connect to an ApsaraDB for Redis instance from an on-premises server, see [#unique_13](#).



Note:

We recommend that you connect the ECS instance to the ApsaraDB for Redis instance over the Alibaba Cloud intranet to improve security.

Reset the password

If you forgot the password for connecting to the service, [reset the password](#) in the console.

Connect clients to an ApsaraDB for Redis instance

You can connect Jedis, phpredis, redis-py, C/C++, .net, node-redis, and C# clients to an ApsaraDB for Redis instance. For more information, see [#unique_15](#).

You can also use the Redis command line interface (redis-cli) program to connect to and manage the ApsaraDB for Redis instance. For more information, see [#unique_16](#).



Note:

If you failed to connect to the ApsaraDB for Redis instance by using the clients or redis-cli, check [Connection issues between the ApsaraDB for Redis instance and the ECS instance](#).

Client connection issues

- [Common Jedis exceptions](#).
- [Common errors for cluster instances of ApsaraDB for Redis](#).

Bandwidth restrictions cause connection failures

Each type of ApsaraDB for Redis instance has a maximum bandwidth configured. For more information, see [#unique_20](#). If network resources are sufficient, the bandwidth is unlimited for ApsaraDB for Redis instances. However, if network resources are insufficient, the specified maximum bandwidth takes effect for the instances. In the case of heavy network traffic, the maximum bandwidth restricts service requests.

Solution:

- [Analyze the memory distribution on the instance](#) or [use the SCAN command](#) to find large keys and optimize the ApsaraDB for Redis service.

- Upgrade the instance type to increase the bandwidth efficiency.
- Use the same type of cluster instance to increase the bandwidth efficiency.

Poor or failed connections due to performance issues

The `KEYS *` and `HGETALL` commands may affect the performance of ApsaraDB for Redis, and cause thread blocking and connection issues. Solution:

- Check [monitoring data](#) to identify the cause of the issue and take the corresponding measures.
- Query slow logs, and optimize the ApsaraDB for Redis service based on slow log details. You can [view slow logs](#) in the console or by running the `SHOW LOG` command.
- Do not run the `KEYS`, `FLUSHALL`, and `FLUSHDB` commands when the ApsaraDB for Redis instance is running in your business environment. You can disable these commands by using the `RENAME` mechanism of ApsaraDB for Redis, or run the `SCAN` command to process data gradually.
- If your ApsaraDB for Redis instance uses engine version 4.0, you can use the `UNLINK`, `FLUSHALL ASYNC`, and `FLUSHDB ASYNC` commands and related parameters in the Lazyfree feature to optimize service code.
- [Analyze the memory distribution on the instance](#) or [use the SCAN command](#) to find large keys and optimize the ApsaraDB for Redis service.
- Optimize hotkeys. If you use a cluster instance, you can [analyze hotkeys on a specified child node](#).

Summary

When the connection to the ApsaraDB for Redis instance is faulty, you can follow the preceding methods and locate issues related to [environment prerequisites](#), [connection methods](#), [error messages](#), [bandwidth restrictions](#), and [performance conditions](#).

2 Common Jedis exceptions in ApsaraDB for Redis

Jedis is simple to use. However, if you cannot set valid parameters such as JedisPool parameters that are suitable for your scenarios and if you use some features such as Lua and transactions in an incorrect way, some issues may occur. This topic describes how to solve these issues.

List of exceptions

- 1. `redis.clients.jedis.exceptions.JedisConnectionException`: Could not get a resource from the pool
- 2. `redis.clients.jedis.exceptions.JedisConnectionException`: Unexpected end of stream
- 3. `redis.clients.jedis.exceptions.JedisDataException`: ERR illegal address
- 4. `redis.clients.jedis.exceptions.JedisDataException`: ERR max number of clients reached
- 5. `redis.clients.jedis.exceptions.JedisConnectionException`: `java.net.SocketTimeoutException`: Read timed out
- 6. `redis.clients.jedis.exceptions.JedisDataException`: NOAUTH Authentication required
- 7. `redis.clients.jedis.exceptions.JedisDataException`: EXECABORT Transaction discarded because of previous errors
- 8. `java.lang.ClassCastException`: `java.lang.Long` cannot be cast to `java.util.List`
- 9. `redis.clients.jedis.exceptions.JedisDataException`: WRONGTYPE Operation against a key holding the wrong kind of value
- 10. `redis.clients.jedis.exceptions.JedisDataException`: OOM command not allowed when used memory > 'maxmemory'
- 11. `redis.clients.jedis.exceptions.JedisDataException`: LOADING Redis is loading the dataset in memory
- 12. `redis.clients.jedis.exceptions.JedisDataException`: BUSY Redis is busy running a script. You can only call SCRIPT KILL or SHUTDOWN NOSAVE
- 13. `redis.clients.jedis.exceptions.JedisConnectionException`: `java.net.SocketTimeoutException`: connect timed out

- **14. UNKILLABLE Sorry the script already executed write commands against the dataset. You can either wait the script termination or kill the server in a hard way using the SHUTDOWN NOSAVE command**
- **15. java.lang.NoClassDefFoundError**
- **16. redis.clients.jedis.exceptions.JedisDataException: ERR unknown command**
- **17. redis.clients.jedis.exceptions.JedisDataException: Please close pipeline or multi block before calling this method**
- **18. redis.clients.jedis.exceptions.JedisDataException: ERR command role not support for normal user**

1. Failure to obtain Jedis connections from JedisPool

1.1. Exception stack

(1) The JedisPool parameter `blockWhenExhausted` is set to true by default.

If no Jedis connection is available in JedisPool, the Jedis client waits for a period in milliseconds as specified in the `maxWaitMillis` parameter. Afterward, if the Jedis connection is still not available, the Jedis client throws the following exception:

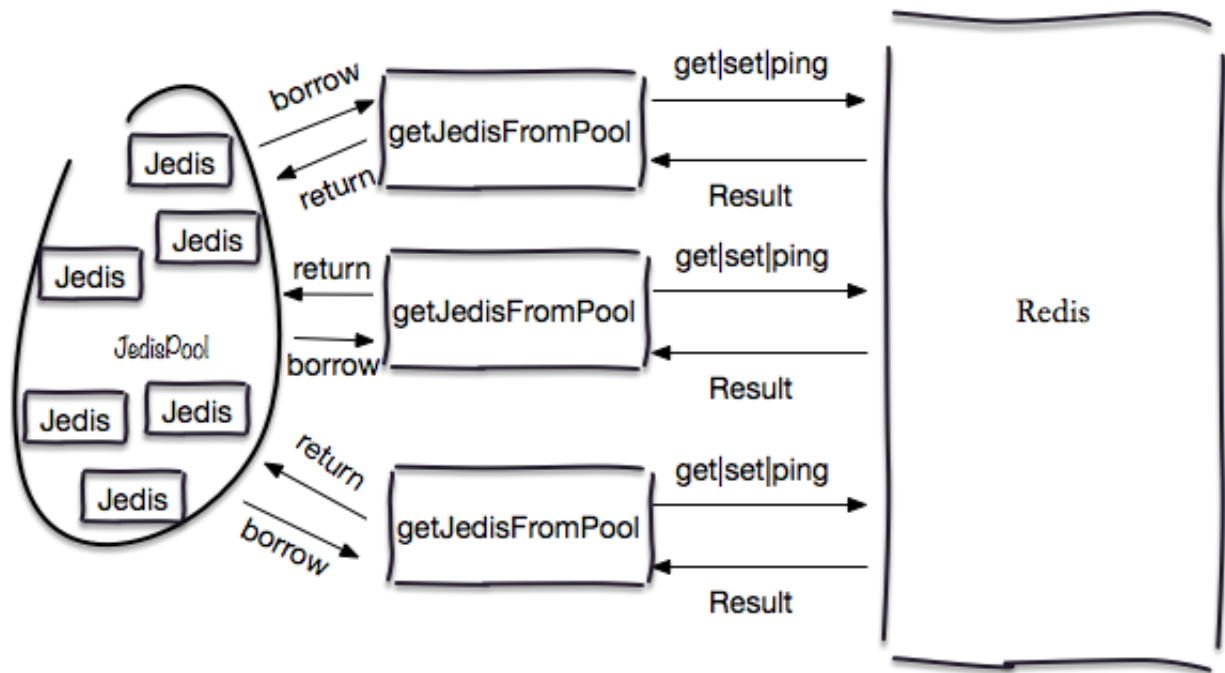
```
redis . clients . jedis . exceptions . JedisConne ctionExcep tion
: Could not get a resource from the pool
...
Caused by : java . util . NoSuchElem entExcepti on : Timeout
waiting for idle object
at org . apache . commons . pool2 . impl . GenericObj ectPool
. borrowObj e ct ( GenericObj ectPool . java : 449 )
```

(2) The JedisPool parameter `blockWhenExhausted` is set to false.

When you set this parameter in this way, if no Jedis connection is available in JedisPool, the Jedis client directly throws the following exception:

```
redis . clients . jedis . exceptions . JedisConne ctionExcep tion
: Could not get a resource from the pool
...
Caused by : java . util . NoSuchElem entExcepti on : Pool
exhausted
at org . apache . commons . pool2 . impl . GenericObj ectPool
. borrowObj e ct ( GenericObj ectPool . java : 464 )
```

1.2. Exception description



The preceding exception occurs when the client fails to obtain any Jedis connection from JedisPool. You can set the `maxTotal` parameter to specify the maximum number of available connections in JedisPool. Possible causes are listed as follows:

(1) Connection exposure (a common cause)

The value of `maxTotal` is set to 8 by default. In the following code, the client has obtained eight Jedis connections from JedisPool, and not returned these connections. When the client attempts to obtain the ninth Jedis connection by using `jedisPool.getResource().ping()`, the attempt fails.

```
GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
JedisPool jedisPool = new JedisPool(poolConfig, "127.0.0.1", 6379);
// The client borrows connections from JedisPool for eight times, but does not return these connections.
for (int i = 0; i < 8; i++) {
    Jedis jedis = null;
    try {
        jedis = jedisPool.getResource();
        jedis.ping();
    } catch (Exception e) {
        logger.error(e.getMessage(), e);
    }
}
jedisPool.getResource().ping();
```

We recommend that you follow these code rules:

```
Run the following command :
Jedis jedis = null;
try {
```

```

    jedis = jedisPool . getResource ();
    // The specific service request that the client
    processes after obtaining a Jedis connection .
    jedis . executeCom mand ()
} catch ( Exception e ) {
    // If the command contains a key , we recommend
    that you also print the key in the error log . You
    can use the key to locate the node where the
    exception occurs in a cluster .
    logger . error ( e . getMessage (), e );
} finally {
    // The client does not close a borrowed Jedis
    connection . Instead , the client returns the connection
    to JedisPool when you use this resource pool .
    if ( jedis != null )
        jedis . close ();
}

```

(2) For a high concurrency level, the `maxTotal` parameter is set to a low value.

The following example describes this issue:

- 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. The time is approximately 1 ms on average. The QPS for each connection is approximately 1,000.
- The required QPS is 50,000.

Therefore, the total number of available connections in JedisPool is theoretically 50 ($50,000/1,000 = 50$), and you can fine-tune the actual value of `maxTotal` according to the theoretical value.

(3) Jedis connection blocking

For example, connections to Redis are blocked in the conditions such as slow query . All connections wait for a period less than the timeout value. In this case, a high concurrency level can result in insufficient resources in JedisPool.

(4) Jedis connection denied

When the client attempts to obtain a connection from JedisPool, JedisPool has to generate a connection due to the lack of idle connections. However, the new connection is denied, as shown in the following code:

```

redis . clients . jedis . exceptions . JedisConne ctionExcep tion
: Could not get a resource from the pool
  at redis . clients . util . Pool . getResource ( Pool . java
: 50 )
  at redis . clients . jedis . JedisPool . getResource (
JedisPool . java : 99 )
  At testadmin . Main ( testadmin . Java : 14 )

```



```

Caused by : redis . clients . jedis . exceptions . JedisConnec
tionExcept ion : java . net . ConnectExc eption : Connection
refused
    at redis . clients . jedis . Connection . connect ( Connection
. java : 164 )
    at redis . clients . jedis . BinaryClie nt . connect (
BinaryClie nt . java : 80 )
    at redis . clients . jedis . BinaryJedi s . connect (
BinaryJedi s . java : 1676 )
    at redis . clients . jedis . JedisFacto ry . makeObject (
JedisFacto ry . java : 87 )
    at org . apache . commons . pool2 . impl . GenericObj ectPool
. create ( GenericObj ectPool . java : 861 )
    at org . apache . commons . pool2 . impl . GenericObj ectPool
. borrowObj ect ( GenericObj ectPool . java : 435 )
    at org . apache . commons . pool2 . impl . GenericObj ectPool
. borrowObj ect ( GenericObj ectPool . java : 363 )
    at redis . clients . util . Pool . getResourc e ( Pool . java
: 48 )
... 2 more
Caused by : java . net . ConnectExc eption : Connection
refused
    at java . net . PlainSocke tImpl . socketConn ect ( Native
Method )
    at java . net . AbstractPl ainSocketI mpl . doConnect (
AbstractPl ainSocketI mpl . java : 339 )
    at java . net . AbstractPl ainSocketI mpl . connectToA
ddress ( AbstractPl ainSocketI mpl . java : 200 )
    at java . net . AbstractPl ainSocketI mpl . connect (
AbstractPl ainSocketI mpl . java : 182 )
    at java . net . SocksSocke tImpl . connect ( SocksSocke tImpl
. java : 392 )
    at java . net . Socket . connect ( Socket . java : 579 )
    at redis . clients . jedis . Connection . connect ( Connection
. java : 158 )
... 9 more

```

The code at `Redis.clients.jedis.connection.connect(connection.java:158)` indicates that the connection is a socket connection as follows:

```

socket . setSoLinge r ( true , 0 ); // Control calls close
() method ,
    // the underlying socket is closed
    // immediatel y
    // <-@ wjw_add
158 : socket . connect ( new InetSocket Address ( host , port
), connection Timeout );

```



Note:

In this case, you must check whether the domain name configuration for ApsaraDB for Redis is correct and whether network conditions are normal during this period less than the timeout value.

(4) Other issues

1.3. Solutions

Based on the preceding analysis, the causes of the failure to obtain connections from JedisPool are complex. If you want to provide sufficient resources in JedisPool, increasing the value of `maxTotal` is not the only solution. The solution varies according to actual conditions.

1.4. Approaches

You can locate the issue and perform troubleshooting according to the preceding description. If the issue persists, submit a ticket.

2. Client buffer exception

2.1. Exception stack

```
redis.clients.jedis.exceptions.JedisConnectionException: Unexpected end of stream.
    at redis.clients.util.RedisInputStream.ensureFill(RedisInputStream.java:199)
    at redis.clients.util.RedisInputStream.readByte(RedisInputStream.java:40)
    at redis.clients.jedis.Protocol.process(Protocol.java:151)
    .....
```

2.2. Exception description

This client buffer exception may occur due to the following causes:

(1) **Common cause: multiple threads use the same Jedis connection.** Normally, one thread uses one Jedis connection. You can use JedisPool to manage Jedis connections to secure threads and avoid this issue. The following example shows that two threads share one Jedis connection:

```
new Thread ( new Runnable () {
    public void run () {
        for ( int i = 0 ; i < 100 ; i ++ ) {
            jedis . get ( " hello " );
        }
    }
}). start ();
new Thread ( new Runnable () {
    public void run () {
        for ( int i = 0 ; i < 100 ; i ++ ) {
            jedis . hget ( " haskey ", " f " );
        }
    }
}). start ();
```

(2) **The client buffer is full.**

ApsaraDB for Redis provides three types of client buffers:

- **Normal client buffer:** receives normal commands, such as GET, SET, MSET, HGETALL, and ZRANGE.
- **Replica client buffer:** synchronizes write commands from a master node during replications.
- **PUBSUB buffer:** the PUBSUB command is not a normal command, so the command has an independent buffer.

You can configure the client buffer for ApsaraDB for Redis in the following way:

```
client - output - buffer - limit < class > < hard limit > < soft limit > < soft seconds >
```

- **class:** specifies the type of the client. Valid values: normal, slave, and pubsub.
- **hard limit:** if a client uses the output buffer that is more than the value of hard limit, the system terminates the client immediately. Unit: bytes.
- **soft limit and soft seconds:** if a client uses the output buffer more than the value of soft limit (unit: bytes) and if the client uses the output buffer for a period of time equal to the value of soft seconds (unit: seconds), the system terminates the client immediately.

The following example shows the buffer configuration for ApsaraDB for Redis. In the specified condition, the system terminates the client immediately and shows the exception `Unexpected end of stream`.

```
redis > config get client - output - buffer - limit
1 ) " client - output - buffer - limit "
2 ) " normal 524288000 0 0 slave 2147483648 536870912
480 pubsub 33554432 8388608 60 "
```

(3) The Redis service automatically disconnects a long idle connection. You can check the timeout setting and JedisPool configuration to determine whether the idle connection detection is required.

3. Solutions and approaches

Customer: you can check the code of your service to make sure that you use JedisPool to manage Jedis connections and check whether multiple threads share one Jedis connection.

Ticket: you can submit a ticket to check whether the preceding condition (2) or (3) causes the issue. The value of `timeout` is 0 in ApsaraDB for Redis by default. You cannot modify this value. The value of `client - output - buffer - limit` is 500

MB by default. This is a value that Alibaba Cloud has optimized. If the value of the parameter is more than 500 MB, the client returns too many values. In this case, to ensure performance and stability of the service, we recommend that you optimize the application.

3. Invalid client address (ApsaraDB for Redis provides a whitelist of clients)

3.1. Exception stack

```
Caused by : redis . clients . jedis . exceptions . JedisDataE
xception : ERR illegal address
    at redis . clients . jedis . Protocol . processError (
Protocol . java : 117 )
    at redis . clients . jedis . Protocol . process ( Protocol .
java : 151 )
    at redis . clients . jedis . Protocol . read ( Protocol . java
: 205 )
    . . . . .
```

3.2. Exception description

The ApsaraDB for Redis instance has a whitelist of clients configured. The IP address of the current client that connects to the instance does not exist in the whitelist.

3.3. Solutions

Add the IP address of the current client to the whitelist.

3.4. Approaches

You can perform troubleshooting or submit a ticket.

4. The maximum number of client connections reached

4.1. Exception stack

```
redis . clients . jedis . exceptions . JedisDataE xception : ERR
max number of clients reached
```

4.2. Exception description

The number of client connections is more than the value of maxclients configured on an ApsaraDB for Redis instance.

4.3. Solutions

You can submit a ticket to temporarily increase the value of maxclients and locate the cause of the connection burst.

4.4. Approaches

- **Ticket:** you can submit a ticket to temporarily adjust the value of maxclients and locate the cause of the connection burst.
- **Customer:** you can locate the client that most frequently connects to the ApsaraDB for Redis instance, and analyze the cause, such as JedisPool configuration.

5. Client read and write timeout

5.1. Exception stack

```
redis . clients . jedis . exceptions . JedisConnectionException : java . net . SocketTimeoutException : Read timed out
```

5.2. Exception description

The possible causes of the exception are listed as follows: (1) The read and write timeout values are too low. (2) Slow query or connection blocking occurs. (3) The network is unstable.

5.3. Solutions

You can submit a ticket and provide read and write timeout values for troubleshooting purposes.

5.4. Approaches

You can submit a ticket for troubleshooting purposes.

6. Exceptions related to passwords

6.1. Exception stack

ApsaraDB for Redis has password verification configured. But a client does not provide any password in a request as shown in the following code:

```
Exception in thread "main" redis . clients . jedis . exceptions . JedisDataException : NOAUTH Authentication required .
    at redis . clients . jedis . Protocol . processError ( Protocol . java : 127 )
    at redis . clients . jedis . Protocol . process ( Protocol . java : 161 )
    at redis . clients . jedis . Protocol . read ( Protocol . java : 215 )
```

ApsaraDB for Redis does not have password verification configured. But a client provides a password in a request as shown in the following code:

```
Exception in thread "main" redis . clients . jedis . exceptions . JedisDataException : ERR Client sent AUTH , but no password is set
```

```

    at redis.clients.jedis.Protocol.processError (
Protocol.java:127)
    at redis.clients.jedis.Protocol.process (Protocol.
java:161)
    at redis.clients.jedis.Protocol.read (Protocol.java
:215)

```

A client provides an incorrect password in a request as shown in the following code:

```

redis.clients.jedis.exceptions.JedisDataException: ERR
invalid password
    at redis.clients.jedis.Protocol.processError (
Protocol.java:117)
    at redis.clients.jedis.Protocol.process (Protocol.
java:151)
    at redis.clients.jedis.Protocol.read (Protocol.java
:205)

```

6.2. Solutions: check whether the service has password authentication configured and whether the client provides a correct password.

7. Transaction exception

7.1. Exception stack

```

redis.clients.jedis.exceptions.JedisDataException:
EXECABORT Transaction discarded because of previous
errors

```

7.2. Exception description

This is a transaction exception in ApsaraDB for Redis. The transaction contains an incorrect command, such as the unknown `set` command in the following code:

```

127.0.0.1:6379 > multi
OK
127.0.0.1:6379 > set key world
(error) ERR unknown command 'set'
127.0.0.1:6379 > incr counter
QUEUED
127.0.0.1:6379 > exec
(error) EXECABORT Transaction discarded because of
previous errors.

```

7.3. Solutions and approaches

Customer: you can check the code of your service for troubleshooting purposes.

8. Class conversion error

8.1. Exception stack

```

java.lang.ClassCastException: java.lang.Long cannot
be cast to java.util.List
    at redis.clients.jedis.Connection.getBinaryM
ultiBulkReply (Connection.java:199)

```

```

        at redis.clients.jedis.Jedis.hgetAll (Jedis.java : 851)
        at redis.clients.jedis.ShardedJedis.hgetAll (ShardedJedis.java : 198)

java.lang.ClassCastException: java.util.ArrayList
cannot be cast to [B
        at redis.clients.jedis.Connection.getBulkReply (Connection.java : 182)
        at redis.clients.jedis.Connection.getBulkReply (Connection.java : 171)
        at redis.clients.jedis.Jedis.rpop (Jedis.java : 1109)
        at redis.clients.jedis.ShardedJedis.rpop (ShardedJedis.java : 258)
        .....

```

8.2. Exception description

Normally, one thread uses one Jedis connection. If multiple threads share the same Jedis connection, this exception occurs. You can use JedisPool to avoid this exception. The following example shows that two threads share the same Jedis connection. The GET and HGETALL commands return different types of data.

```

new Thread ( new Runnable () {
    public void run () {
        for ( int i = 0 ; i < 100 ; i ++ ) {
            jedis . set ( " hello ", " world " );
            jedis . get ( " hello " );
        }
    }
}). start ();
new Thread ( new Runnable () {
    public void run () {
        for ( int i = 0 ; i < 100 ; i ++ ) {
            jedis . hset ( " hashkey ", " f ", " v " );
            jedis . hgetAll ( " hashkey " );
        }
    }
}). start ();

```

8.3. Solutions and approaches

You can check the code of your service for troubleshooting purposes.

9. Command error

9.1. Exception stack

```

Exception in thread "main" redis.clients.jedis.exceptions.JedisDataException: WRONGTYPE Operation against a key holding the wrong type of value
        at redis.clients.jedis.Protocol.processError (Protocol.java : 127)
        at redis.clients.jedis.Protocol.process (Protocol.java : 161)

```

```

        at redis.clients.jedis.Protocol.read(Protocol.java
: 215)
.....

```

9.2. Exception description

For example, key=" hello" indicates a key of String type, but the HGETALL command returns a key of hash type. Therefore, the exception occurs.

```

jedis.set("hello","world");
jedis.hgetAll("hello");

```

9.3. Solutions and approaches

You can check the code of your service for troubleshooting purposes.

10. Memory usage of ApsaraDB for Redis is more than the value of maxmemory

10.1. Exception stack

```

redis.clients.jedis.exceptions.JedisDataException: OOM
command not allowed when used memory > 'maxmemory'.

```

10.2. Exception description

The memory usage of a node of ApsaraDB for Redis is more than the value of maxmemory on the instance.

10.3. Solutions

Possible causes are listed as follows:

- Service data increases normally.
- Client buffer exceptions occur due to reasons such as the improper use of the MONITOR and PUBSUB commands.
- In cache-only scenarios, ApsaraDB for Redis has maxmemory-policy configured in an incorrect way. For example, the volatile-lru eviction policy is not configured for expired keys.

In emergency conditions, you can submit a ticket to temporarily adjust the value of maxmemory. Afterward, you can require upgrading or downgrading of the configuration.

10.4. Approaches

- Customer: you can locate the cause of increased memory usage.
- Ticket: you can submit a ticket to temporarily adjust the value of maxmemory. Afterward, you can require upgrading or downgrading the configuration.

11. ApsaraDB for Redis is loading persistence files

11.1. Exception stack

```
redis.clients.jedis.exceptions.JedisDataException: LOADING Redis is loading the dataset in memory
```

11.2. Exception description

When a Jedis client tries to connect to an ApsaraDB for Redis instance, the instance is loading persistence files and cannot normally process read and write requests.

11.3. Solutions

Normally, this exception does not occur in ApsaraDB for Redis. If this exception occurs, submit a ticket.

4. Approaches

You can submit a ticket for troubleshooting purposes.

12. Lua script timeout

12.1. Exception stack

```
redis.clients.jedis.exceptions.JedisDataException: BUSY Redis is busy running a script. You can only call SCRIPT KILL or SHUTDOWN NOSAVE .
```

12.2. Exception description

An ApsaraDB for Redis instance is running a Lua script for a period of time that is more than the value of `LUA-time-limit`. In this case, if a Jedis client tries to connect to the instance, the preceding exception occurs.

12.3. Solutions

The system displays the exception message: `You can only call SCRIPT KILL or SHUTDOWN NOSAVE`. Therefore, you can run the `SCRIPT KILL` command to terminate the Lua script.

12.4. Approaches

You can handle the exception by yourself. If the exception persists, submit a ticket to request technical support.

13. Connection timeout

13.1. Exception stack

```
redis . clients . jedis . exceptions . JedisConnectionException  
: java . net . SocketTimeoutException : connect timed out
```

13.2. Exception description

Possible causes are listed as follows:

- The connection timeout value is too low.
- The value of tcp-backlog reaches the upper limit. This fails the new connection.
- Network failures occur between the client and the service.

13.3. Solutions

You can submit a ticket and provide the connection timeout value for troubleshooting purposes.

13.4. Approaches

You can submit a ticket for troubleshooting purposes.

14. Lua script write timeout

14.1. Exception stack

```
( error ) UNKILLABLE Sorry the script already executed  
write commands against the dataset . You can either  
wait the script termination or kill the server in  
a hard way using the SHUTDOWN NOSAVE command .
```

14.2. Exception description

An ApsaraDB for Redis instance is running a Lua script for a period of time more than the value of lua-time-limit, and has run a write command. In this case, if a Jedis client tries to connect to the instance, the preceding exception occurs.

14.3. Solutions

You can submit a ticket to require emergency troubleshooting. Afterward, the administrator restarts the service or performs the failover operation on the ApsaraDB for Redis instance.

14.4. Approaches

You can submit a ticket for troubleshooting purposes.

15. Class loading error

15.1. Exception stack

The following example shows that required classes and methods are not available:

```
Exception in thread "commons - pool - EvictionTi mer " java .
lang . NoClassDef FoundError : redis / clients / util / IOUtils
    at redis . clients . jedis . Connection . disconnect (
Connection . java : 226 )
    at redis . clients . jedis . BinaryClie nt . disconnect (
BinaryClie nt . java : 941 )
    at redis . clients . jedis . BinaryJedi s . disconnect (
BinaryJedi s . java : 1771 )
    at redis . clients . jedis . JedisFacto ry . destroyObj ect (
JedisFacto ry . java : 91 )
    at org . apache . commons . pool2 . impl . GenericObj
ectPool . destroy ( GenericObj ectPool . java : 897 )
    at org . apache . commons . pool2 . impl . GenericObj ectPool
. evict ( GenericObj ectPool . java : 793 )
    at org . apache . commons . pool2 . impl . BaseGeneri
cObjectPoo l $ Evictor . run ( BaseGeneri cObjectPoo l . java :
1036 )
    at java . util . TimerThrea d . mainLoop ( Timer . java : 555
)
    at java . util . TimerThrea d . run ( Timer . java : 505 )
Caused by : java . lang . ClassNotFo undExcepti on : redis .
clients . util . IOUtils
.....
```

15.2. Exception description

A Jedis client throws an exception to indicate that the target class is not available when running the required command. This exception is probably caused by loading multiple Jedis versions such as Jedis 2.9.0 and Jedis 2.6. The code runs well during compilation. However, the class loader loads an earlier Jedis version and leads to the failure to load the target class at runtime.

15.3. Solutions

You can exclude repeated Jedis code to solve this issue. For example, use the Maven dependency tree to remove or exclude useless dependencies.

15.4. Approaches

You can check the code of your service for troubleshooting purposes.

16. Unknown commands for the service

16.1. Exception stack

The following example shows that a client runs the GEOADD command, but the ApsaraDB for Redis service indicates that the command is unknown in the response.

```
redis.clients.jedis.exceptions.JedisDataException: ERR
unknown command 'GEOADD'
```

16.2. Exception description

Possible causes of the exception are listed as follows:

- ApsaraDB for Redis does not support some Redis commands, or only some minor versions of ApsaraDB for Redis support these Redis commands. For example, the GEOADD command is included in the GEO API for Redis 3.2.
- The command is incorrect. The Jedis client does not support the commands that you combine directly. Instead, each API calls a required function.

16.3. Solutions

You can ask the administrator to check whether any ApsaraDB for Redis edition supports the required command. If so, you can upgrade to the target minor version.

16.4. Approaches

- Administrator: checks whether the ApsaraDB for Redis edition supports the required command.
- Customer: you can upgrade to the target minor version if the ApsaraDB for Redis edition supports the required command.

17. Improper use of a pipeline

17.1. Exception stack

```
redis.clients.jedis.exceptions.JedisDataException: Please close
pipeline or multi block before calling this method .
```

17.2. Exception description

A Jedis client usually calls `response.get()` to obtain target values before calling `pipeline.sync()`. However, the client fails to call `pipeline.set()` before calling `pipeline.sync()`. You can run the MONITOR command to check whether any write command runs successfully. The following example shows the exception.

```
Jedis jedis = new Jedis("127.0.0.1", 6379);
Pipeline pipeline = jedis.pipelined();
pipeline.set("hello", "world");
pipeline.set("java", "jedis");
Response <String> pipeString = pipeline.get("java");
```

```
// The GET command must follow the SYNC command . To
// obtain multiple values , we recommend that you use
List < Object > objectList = pipeline . syncAndRet urnAll ();
System . out . println ( pipeString . get ());
// The command takes effect .
pipeline . sync ();
```

The set field in the response is initialized as false. When the Jedis client obtains the analysis response, if the set field is set to false, the client reports an error.

```
public T get () {
    // if response has dependency response and dependency
    // is not built ,
    // build it first and no more !!
    if ( dependency != null && dependency . set && ! dependency
    . built ) {
        dependency . build ();
    }
    if ( ! set ) {
        throw new JedisDataE xception (
            " Please close pipeline or multi block before
calling this method .");
    }
    if ( ! built ) {
        build ();
    }
    if ( exception != null ) {
        throw exception ;
    }
    return response ;
}
```

The pipeline.sync() method sets the set field to true in each result value.

```
public void sync () {
    if ( getPipelin edResponse Length () > 0 ) {
        List < Object > unformatte d = client . getAll ();
        for ( Object o : unformatte d ) {
            generateRe sponse ( o );
        }
    }
}
```

generateResponse(o):

```
protected Response <? > generateRe sponse ( Object data ) {
    Response <? > response = pipelinedR esponses . poll ();
    If ( response != null ) {
        response . set ( data );
    }
    return response ;
}
```

response.set(data);

```
public void set ( Object data ) {
    this . data = data ;
    set = true ;
}
```

```
}
}
```

17.3. Solutions

To parse multiple result values, we recommend that you use `pipeline.syncAndRet urnAll()`. In the following example, the pipeline simulates running the HGETALL command for multiple times.

```
/**
 * The pipeline simulates running the HGETALL command
 * for multiple times .
 * @ param keyList
 * @ return
 */
public Map < String , Map < String , String >> mHgetAll ( List <
String > keyList ) {
// 1 . Generate the pipeline object .
Pipeline pipeline = jedis . pipelined () ;
// 2 . The pipeline runs the command . The command does
not take effect at the moment .
for ( String key : keyList ) {
pipeline . hgetAll ( key ) ;
}
// 3 . The Jedis client runs the command . The
syncAndRet urnAll () method returns the result .
List < Object > objectList = pipeline . syncAndRet urnAll () ;
if ( objectList == null || objectList . isEmpty () ) {
return Collection s . emptyMap () ;
}
// 4 . Parse the result .
Map < String , Map < String , String >> resultMap = new HashMap
< String , Map < String , String >>() ;
for ( int i = 0 ; i < objectList . size () ; i ++ ) {
Object object = objectList . get ( i ) ;
Map < String , String > map = ( Map < String , String > )
object ;
String key = keyList . get ( i ) ;
resultMap . put ( key , map ) ;
}
return resultMap ;
}
```

17.4. Approaches

You can modify the code of your service.

18. Administrator commands unavailable to common users

18.1. Exception stack

```
redis . clients . jedis . exceptions . JedisDataE xception : ERR
command role not support for normal user
```

18.2. Exception description

The required command is not available.

18.3. Solutions

You cannot use the command directly. To solve this issue, submit a ticket.

18.4. Approaches

You can check whether the command is available by reading related documents.

Other issues:

1. How do I select a Jedis version?

In principle, you can select the latest [release](#). However, we recommend that you select a version that has been released for a certain period. A serious issue occurred in an earlier Jedis version during the release history. Jedis 2.9.0 is a stable version so far.

```
< dependency >
  < groupId > redis . clients </ groupId >
  < artifactId > jedis </ artifactId >
  < version > 2 . 9 . 0 </ version >
  < type > jar </ type >
  < scope > compile </ scope >
</ dependency >
```

2. Is JedisCluster in Jedis is the client of the ApsaraDB for Redis cluster edition?

For more information, see [#unique_30](#)

JedisPool parameters

1. Resource setting and usage

Number	Name	Description	Default value	Recommended value
1	maxTotal	The maximum number of connections in the JedisPool.	8	-
2	maxIdle	The maximum number of idle connections in JedisPool.	8	-
3	minIdle	The minimum number of idle connections in JedisPool.	0	-
4	blockWhenExhausted	Specifies whether the client waits for the connection when no resource is available in JedisPool . The maxWaitMillis parameter takes effect only when the blockWhenExhausted parameter is set to true.	true	We recommend that you use the default value.

Number	Name	Description	Default value	Recommended value
5	maxWaitMillis	Specifies the maximum time for the client to wait for the connection when no resource is available in JedisPool. Unit: milliseconds.	-1: specifies no timeout.	We recommend that you do not use the default value.
6	testOnBorrow	Specifies whether to use the Ping command to check the connection validity when a client borrows a connection from JedisPool. If the connection is invalid, JedisPool removes the connection.	false	We recommend that you set the parameter to false when handling large data volumes. If you set the parameter to true, the system runs the Ping command once more.

Number	Name	Description	Default value	Recommended value
7	testOnReturn	Specifies whether to use the Ping command to check the connection validity when a client returns a borrowed connection to JedisPool . If the connection is invalid, JedisPool removes the connection.	false	We recommend that you set the parameter to false when handling large data volumes . If you set the parameter to true, the system runs the Ping command once more .
8	jmxEnabled	Specifies whether to enable Java Management Extensions (JMX) monitoring to monitor the utilization of Jedis clients.	true	We recommend that you set the parameter to true . The monitored application must be running during the monitoring .

2. Monitoring of idle resources

This is to detect idle Jedis objects. This feature includes four parameters. You can set the testWhileIdle parameter to true to enable the feature. The parameters are described in the following table.

Number	Name	Description	Default value	Recommended value
1	testWhileIdle	Specifies whether to enable the monitoring of idle resources.	false	We recommend that you set the parameter to true.
2	timeBetweenEvictionRunsMillis	Specifies the cycle of monitoring idle resources. Unit: milliseconds.	-1: specifies that the system does not monitor idle resources.	We recommend that you customize a monitoring cycle. You can also use the default value.
3	minEvictableIdleTimeMillis	Specifies the minimum idle time for resources in JedisPool. The system removes the resource that has been idle for the specified period. Unit: milliseconds.	$1,000 \times 60 \times 30$ milliseconds = 30 minutes	You can customize the value as needed, or use the default value in most scenarios.
4	numTestsPerEvictionRun	Specifies the number of connections sampled during the monitoring of idle resources.	3	You can fine-tune the value according to the number of connections in your application. If you set the parameter to -1, JedisPool monitors all connections.

3 Comparison among Redis 4.0, Codis, and ApsaraDB for Redis clusters

Architecture comparison

Redis 4.0 cluster

The Redis 4.0 cluster runs in a decentralized structure. The metadata of the cluster is distributed across nodes. During the master-replica failover operation, multiple nodes negotiate with each other and elect a master node. Redis provides the `redis-trib.rb` tool for cluster deployment and operations and maintenance (O&M).

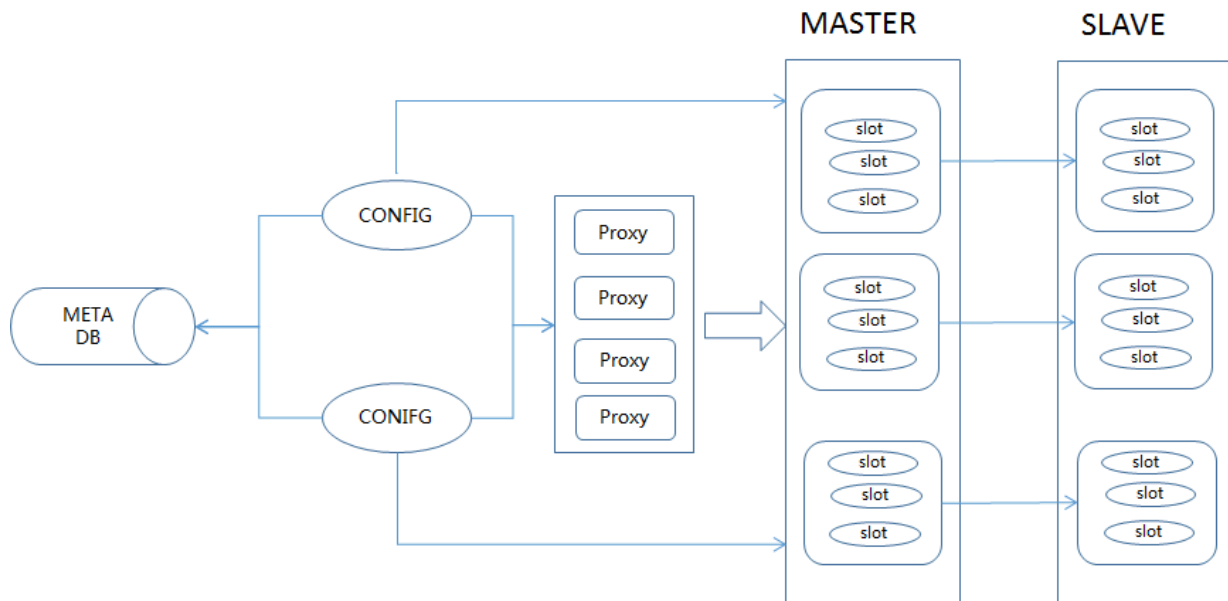
The connection from a client to hashed database nodes is dependent on a smart client. The client evaluates and selects a route based on the node information that Redis returns. For example, if a client initiates a request for a node and the requested key is not located on this node, the client evaluates the returned `MOVE` or `ASK` redirection and redirects the request to the corresponding node.

Codis cluster

- The Codis cluster consists of the following components:
 - **Codis-server:** a Redis database where the source code has been modified and that supports slots, scaling up and out, and migration.
 - **Codis-proxy:** a multi-thread kernel written in Go.
 - **Codis Dashboard:** a cluster management tool.
- Codis provides a Web-based graphical interface for managing the cluster.
- The metadata of the cluster is stored in ZooKeeper or etcd.
- The independent module Codis-HA performs the master-replica failover operation for Redis nodes.
- The client of proxy-based Codis is insensitive to changes of a route table. The client uses the `LIST PROXY` command from Codis Dashboard to retrieve the list of all proxy nodes. Based on the round-robin scheduling policy, the client determines the target proxy node to perform load balancing.

ApsaraDB for Redis

The ApsaraDB for Redis cluster edition uses the architecture as follows:



- The ApsaraDB for Redis cluster edition consists of the following components:
 - Redis-config: a cluster management tool that uses a dual-node structure and that supports disaster recovery.
 - Redis-server: a Redis database where source code has been optimized and that supports slots, scaling up and out, and migration.
 - Redis-proxy: a single-thread stateless kernel written in C++ 14. An ApsaraDB for Redis cluster can contain multiple proxy nodes according to the cluster type.
- The metadata of the cluster is stored on a meta database.
- The independent high-availability (HA) module performs the master-replica failover operation in the cluster.
- In the proxy-based ApsaraDB for Redis cluster, a client accesses the service based on a virtual IP address (VIP) that is resolved into a connection address. You are insensitive to the route information and do not need to handle load balancing for proxy nodes.

Performance comparison

Stress testing environment

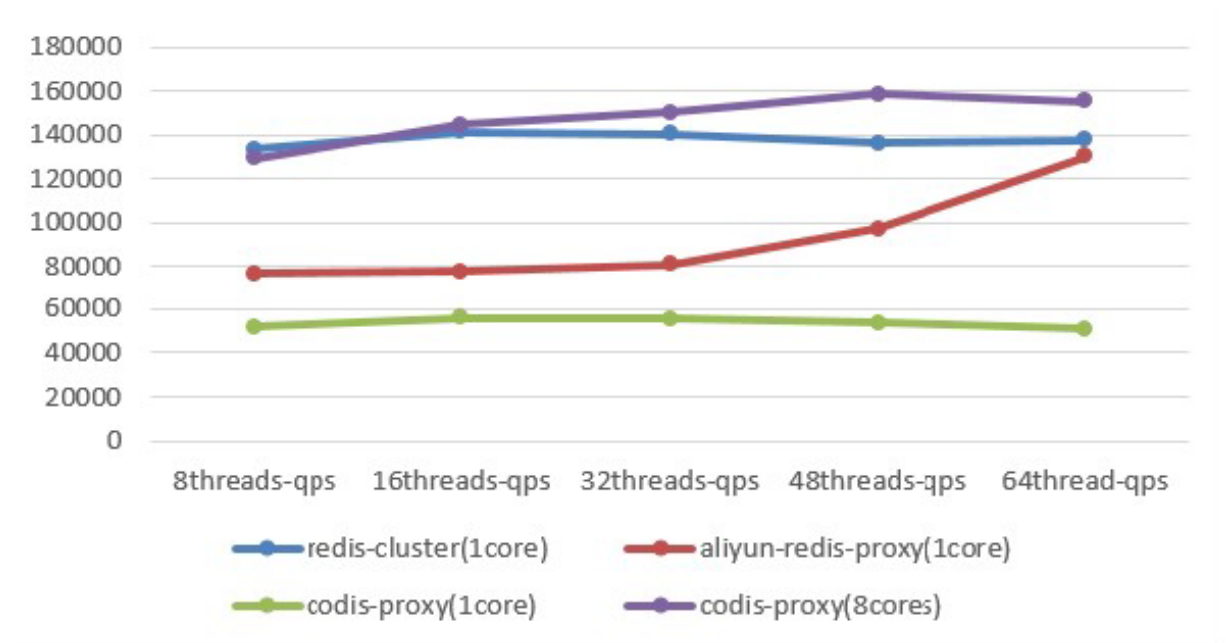
Each of the preceding Redis clusters has been established on three different physical servers. Each physical server supports the gigabit network interface controller (NIC), 24-core CPU, and 189 GB memory. The memtier_benchmark, Codis proxy or Alibaba Cloud proxy, and Redis server stress testing tools run on different physical servers. The Redis server uses the Redis kernel of the corresponding cluster.

The key size is 32 bytes and the ratio of set/get operations is 1:10. Each thread processes 16 clients. The stress testing for each cluster lasts five minutes in the 8-thread, 16-thread, 32-thread, 48-thread and 64-thread scenarios.

[DO NOT TRANSLATE]

Every cluster has eight master databases and eight replica databases. The append-only file (AOF) feature is enabled for these databases. The minimum buffer for an AOF rewrite operation is 64 MB. The stress testing is targeted to a single Redis 4.0 node, a single-core Redis-proxy node of ApsaraDB for Redis, a single-core Codis-proxy node, or an 8-core Codis-proxy node. Codis uses Go 1.7.4.

The stress testing results are as follows.



According to the result, the single-core Codis-proxy delivers the poorest performance. In the stress testing for the 8-core Codis-proxy node, hashtags are not used for keys. This is equivalent to scattering the requests to eight backend database nodes, or equivalent to eight Redis-proxy nodes of ApsaraDB for Redis. Therefore, the result data indicates excellent performance.

The performance of the single-core Redis-proxy node of ApsaraDB for Redis approximates to that of the original Redis database node under sufficiently stressed conditions. In the running environment that your service requires, if you use an original Redis cluster, the client must support cluster protocols, parse the `MOVE` and `ASK` redirections, and redirect requests to the corresponding node. If the client tries to access a key randomly, the access attempts may be repeated twice before the client

can access the key. Therefore, the performance cannot be the same as that of a single-node cluster.

Comparison of features

Comparison of protocols

Feature	Redis 4.0 cluster	Codis cluster	Alibaba Cloud ApsaraDB for Redis cluster
Transactions	Supported in the same slot	Not supported	Supported in the same slot
SUB/PUB	Supported in the same slot	Not supported	Supported
FLUSHALL	Supported	Not supported	Supported
SELECT	Not supported	Not supported	Supported
MSET/MGET	Supported in the same slot	Supported	Supported

Comparison of horizontal scaling

The Redis 4.0 cluster, Codis cluster, and ApsaraDB for Redis distributed cluster support slot-based management. A slot is the basic unit for scaling.

Horizontal scaling of a distributed cluster includes managing routing information and migrating data for cluster nodes. A key is the basic unit for data migration in these clusters.

Principle of horizontal scaling of the Redis 4.0 cluster

In the Redis 4.0 cluster, specified slots can move in a node. The cluster automatically redistributes slots that have been added to an empty node according to the distribution of existing slots in the nodes of the cluster. You can use the `move_slot` method of `redis-trib.rb` to move slots in the way as follows:

1. Call the `SETSLLOT` command to modify status of source and target nodes.
2. Retrieve the list of slots on the source node.
3. Call the `MIGRATE` command to migrate keys. During the migration, the Redis 4.0 cluster stays in blocking status. The system indicates a successful migration only after the migrated keys are restored on the target node.

4. Call the `SETSLOT` command to modify status of source and target nodes.

How can the system ensure data consistency during the migration process?

The Redis 4.0 cluster supports redirections during a migration. The system returns the ASK redirection to a client. After receiving the redirection, the client sends the `ASKING` command and then a request to the target node. Afterward, the client can connect to the target node. The system returns the redirection to the client when the target key stays in all of the following conditions:

- The slot that the key belongs to is located on the node. If not, the system returns the MOVE redirection.
- The slot stays in migration status.
- The key does not exist.

Therefore, the migration in the Redis 4.0 cluster is a synchronous-blocking operation. If the key is not empty, the key is readable and writable even when the corresponding slot stays in migration status to ensure data consistency.

Principle of horizontal scaling of the Codis cluster

Codis redistributes slots in the same way as the Redis 4.0 cluster. The kernel of Codis-server neither stores slot information nor analyzes the slot that the target key belongs to. Codis-server records the key in the slot-key format to a dictionary that contains key-value pairs only during the `DBADD` operation. If the key contains a tag, Codis-server performs the CRC32 calculation for the tag, and adds the key in the CRC value-key format to the skip list that contains key-value pairs.

Codis Dashboard initiates the migration state machine program in the background. The program notifies all proxy nodes to start the migration in the preparation stage. If one or more proxy nodes fail, the migration fails. The migration procedure of the Codis cluster is similar to that of the Redis 4.0 cluster, except the following items:

- The Codis cluster stores slot status in ZooKeeper or etcd.
- The Codis cluster uses the `SLOTSMGRRTT AGSLOT` command instead of the `MIGRATE` command. When running the `SLOTSMGRRTT AGSLOT` command, Codis migrates a key. If the key contains a tag, Codis migrates all keys from the preceding skip list.

How can the system ensure data consistency during the migration process?

The migration in the Codis cluster is also a synchronous-blocking operation. The proxy module works to ensure data consistency, because the Codis-server kernel does not maintain slot status. In response to a request, the Codis-proxy node checks the status of the slot that the target key belongs to. If the slot stays in migration status, the Codis-proxy node sends the command for migrating the specified key to the target Codis-server node. At the end of key migration, the Codis-proxy node routes the request to the target Codis-server node. This migration is simple and requires few modifications to the Redis kernel. But the slow migration may cause client lag for long time.

Principle of horizontal scaling of ApsaraDB for Redis

ApsaraDB for Redis allows you to specify a source, a node, and a slot to distribute the slot. The service also supports dynamic distribution of slots based on parameters such as node capacity and slot size to minimize the interruption of the cluster service. The migration procedure is as follows:

1. The Redis-config node calculates the source and target nodes and slots.
2. The Redis-config node sends the command for migrating slots to the Redis-server node.
3. The Redis-server node starts the state machine program and migrates multiple keys.
4. The Redis-config node regularly checks the Redis-server node and updates the slot status.

How can the system ensure data consistency during the migration process?

Different from Codis, ApsaraDB for Redis maintains slot information in the kernel. Codis supports migrating a complete slot and Redis 4.0 supports migrating a single key. However, Alibaba Cloud has optimized the kernel of ApsaraDB for Redis to support multiple migrations and accelerate the migration speed.

The data migration in ApsaraDB for Redis is an asynchronous operation. The system does not check whether data has been restored on the target node before the response of successful migration. Instead, to verify that the migration is successful, the target node notifies the source node of the successful migration and the source node regularly checks the migration status on the target node. In this way, the migration can minimize the synchronous-blocking impact on the connections to other slots.

During the asynchronous migration, to ensure data consistency, ApsaraDB for Redis processes a normal write operation if the request is a write request and if the key does not exist in the list of keys for migration. Other mechanisms to ensure data consistency in ApsaraDB for Redis are the same as that in the Redis 4.0 cluster.

Other comparisons

Feature	Redis 4.0	Codis	ApsaraDB for Redis
Kernel hot upgrade	Not supported	Not supported	Supported
Proxy hot upgrade	No proxy	Not supported	Supported
Number of slots	16,384	1,024	16,384
Password	Not supported. You need to modify the redis-trib.rb script.	Supported. All components must use the same password.	Supported

The kernel and proxy node of ApsaraDB for Redis support hot upgrades. Connections to the service are maintained and clients still work normally during hot upgrades.

4 View memory usage of all instances under the current account

You can call an API operation based on the RAM user information and the region ID to check memory usage of the instances under the current account. The script is as follows:

[get_used_memory.zip](#)



Note:

The query result is the memory usage during the period 20 minutes before the current time of the instances.

Procedure

1. Install the Alibaba Cloud Python SDK dependency package by running the following command: `pip install aliyun-python-sdk-core`.
2. Run the preceding script file in Python. In the script, the `-r` parameter is `cn-hangzhou` by default. Separate multiple region values with commas. The detailed command is as follows:

```
python get_used_memory.py -i ***** -s ***** -r cn-hangzhou, cn-beijing -o XXX.csv
```



Note:

You can run the `python get_used_memory.py -h` command to check the usage of each parameter.

5 Why cannot an ECS instance connect to an ApsaraDB for Redis instance?

Connect through the Alibaba Cloud intranet

Both instances can interconnect with each other over the Alibaba only when:

- The ECS instance and the ApsaraDB for Redis instance reside in the same region.
- The ECS instance and the ApsaraDB for Redis instance run in the same type of network, a classic network or a virtual private cloud (VPC).
- A whitelist of the ApsaraDB for Redis instance includes the internal IP address of the ECS instance.

The ECS instance cannot directly connect to the ApsaraDB for Redis instance over an internal network in the following conditions:

- The ECS instance and the ApsaraDB for Redis instance run in different VPCs in different regions. To connect the ECS instance and the ApsaraDB for Redis instance from different regions, you have to use the Express Connect service and establish a connection across VPCs over an internal network. For more information, see [#unique_34](#).
- The ECS instance and the ApsaraDB for Redis instance run in different types of networks. You can solve this issue in the following ways:
 - If permitted, switch the network type to the VPC. For more information, see [#unique_7](#) or [#unique_35](#).
 - If your business does not allow switching instances to VPC, you have to purchase the ECS instance and the ApsaraDB for Redis instance that run in a classic network and in the same region.

Connect through the Internet

To connect an ApsaraDB for Redis instance through the Internet, you must apply a public endpoint for the Redis instance first, see [#unique_13](#), then connect to the Redis instance with the public endpoint.

6 How do I search for large keys?

Background

ApsaraDB for Redis provides complex data structure types such as lists, hash tables, and sorted sets. When you use the service, improper key design may result in large keys. Due to Redis single-thread model, the operation to obtain or delete large keys may affect the service. In a cluster, large keys are prone to run out of memory of a certain node. Therefore, you can use a search tool to find large keys.

To scan for large keys of ApsaraDB for Redis, run the `SCAN` command for the master-replica edition or `ISCAN` command for the cluster edition. To obtain the number of nodes, run the `INFO` command. The command rules are as follows:

```
ISCAN idx cursor [ MATCH pattern ] [ COUNT count ]
```

In this command, `idx` specifies the node ID that starts from 0. For an eight-node cluster instance of 16 GB to 64 GB, `idx` ranges from 0 to 7. A 128 GB or 256 GB cluster instance contains 16 nodes.

Procedure

1. Run the following command to download the Python client package:

```
wget "https://pypi.python.org/packages/68/44/5efe9e98ad83ef5b742ce62a15bea609ed5a0d1caf35b79257ddb324031a/redis-2.10.5.tar.gz#md5=3b26c2b9703b4b56b30a1ad508e31083"
```

2. Decompress the package and install the Python client.

```
tar -xvf redis-2.10.5.tar.gz
cd redis-2.10.5
sudo python setup.py install
```

3. Create the following scanning script:

```
import sys
import redis
def check_big_key(r, k):
    bigKey = False
    length = 0
    try:
        type = r.type(k)
        if type == "string":
            length = r.strlen(k)
        elif type == "hash":
            length = r.hlen(k)
        elif type == "list":
```

```

        length = r.llen(k)
    elif type == "set":
        length = r.scard(k)
    elif type == "zset":
        length = r.zcard(k)
except:
    return
if length > 10240:
    bigKey = True
    if bigKey:
        print db, k, type, length
def find_big_key_normal(db_host, db_port, db_password, db_num):
    r = redis.StrictRedis(host=db_host, port=db_port, password=db_password, db=db_num)
    for k in r.scan_iter(count=1000):
        check_big_key(r, k)
def find_big_key_sharding(db_host, db_port, db_password, db_num, nodecount):
    r = redis.StrictRedis(host=db_host, port=db_port, password=db_password, db=db_num)
    cursor = 0
    for node in range(0, nodecount):
        while True:
            iscan = r.execute_command("iscan", str(node), str(cursor), "count", "1000")
            for k in iscan[1]:
                check_big_key(r, k)
            cursor = iscan[0]
            print cursor, db, node, len(iscan[1])
            if cursor == "0":
                break
if __name__ == '__main__':
    if len(sys.argv) != 4:
        print 'Usage: python ', sys.argv[0], ' host port password '
        exit(1)
    db_host = sys.argv[1]
    db_port = sys.argv[2]
    db_password = sys.argv[3]
    r = redis.StrictRedis(host=db_host, port=int(db_port), password=db_password)
    nodecount = r.info()['nodecount']
    keyspaces_info = r.info("keyspace")
    for db in keyspaces_info:
        print 'check ', db, ' ', keyspaces_info[db]
        if nodecount > 1:
            find_big_key_sharding(db_host, db_port, db_password, db.replace("db", ""), nodecount)
        else:
            find_big_key_normal(db_host, db_port, db_password, db.replace("db", ""))

```

4. Run the `python find_bigkey.py <host> 6379 <password>` command to search for large keys.



Note:

- The command returns a list of large keys in the master-replica edition and cluster edition of ApsaraDB for Redis. The default threshold for large keys is 10

,240. Large keys include string-type keys with a value greater than 10,240, list-type keys with a length greater than 10,240, or hash-type keys with more than 10,240 hash fields.

- By default, the script searches 1,000 keys to minimize the adverse impact on service performance. However, we recommend that you run the script during off-peak hours to prevent the adverse impact caused by running the SCAN command.

7 How do I view the memory of child instances of an ApsaraDB for Redis cluster instance?

Background

An ApsaraDB for Redis cluster instance contains multiple nodes. You need to check the memory and the number of keys of each node. Alibaba Cloud provides the `iinfo` command for you to check performance of a specified node. In the future, you will be able to view the data of each node in the console.

```
iinfo db_idx [ section ]
```

In the command, the value range of `db_idx` is `[0, nodecount)`. You can obtain the value of `nodecount` by running the `INFO` command. To set the `section` parameter, you can follow the way to set the `section` parameter for the `INFO` command of Redis.

Procedure

1. Download the Python client package.

```
wget "https://pypi.python.org/packages/68/44/5efe9e98ad83ef5b742ce62a15bea609ed5a0d1caf35b79257ddb324031a/redis-2.10.5.tar.gz#md5=3b26c2b9703b4b56b30a1ad508e31083"
```

2. Decompress the package and install the Python client.

```
tar -xvf redis-2.10.5.tar.gz
cd redis-2.10.5
sudo python setup.py install
```

3. Run the following scanning script:

```
import sys
import redis
from redis._compat import nativestr
def parse_info ( response ):
    " Parse the result of Redis 's INFO command into
    a Python dict "
    info = {}
    response = nativestr ( response )
    def get_value ( value ):
        if ',' not in value or '=' not in value :
            try :
                if '.' in value :
                    return float ( value )
                else :
                    return int ( value )
            except ValueError :
                return value
    else :
```

```

        sub_dict = {}
        for item in value . split (','):
            k , v = item . rsplit ('=', 1 )
            sub_dict [ k ] = get_value ( v )
        return sub_dict
    for line in response . splitlines ():
        if line and not line . startswith ('#'):
            if line . find (':') != - 1 :
                key , value = line . split (':', 1 )
                info [ key ] = get_value ( value )
            else :
                # if the line isn ' t splittable , append
it to the " __raw__ " key
                info . setdefault ( ' __raw__ ' , []). append ( line
)
    return info
if __name__ == ' __main__ ':
    if len ( sys . argv ) != 4 :
        print ' Usage : python ' , sys . argv [ 0 ] , ' host
port password '
        exit ( 1 )
    db_host = sys . argv [ 1 ]
    db_port = sys . argv [ 2 ]
    db_passwor d = sys . argv [ 3 ]
    r = redis . StrictRedi s ( host = db_host , port = int (
db_port ) , password = db_passwor d )
    nodecount = r . info () [' nodecount ' ]
    for node in range ( 0 , nodecount ):
        info = r . execute_co mmand ( " iinfo " , str ( node ))
        info_res = parse_info ( info )
        print "===== node " , str ( node ) , "
=====
        print ' used_memor y_human : ' , info_res [' used_memor
y_human ' ]
        print r . execute_co mmand ( " iinfo " , str ( node ) , "
keyspace " )
        info_res = r . info ()
        print "===== total ====="
        print ' used_memor y_human : ' , info_res [' used_memor
y_human ' ]
    print r . info ( ' keyspace ' )

```

Run the `python check_shard.py host port password` command to output the following content:

```

===== node 0 =====
used_memor y_human : 37 . 56M
# Keyspace
db0 : keys = 9887 , expires = 0 , avg_ttl = 0
===== node 1 =====
used_memor y_human : 37 . 58M
# Keyspace
db0 : keys = 9835 , expires = 0 , avg_ttl = 0
Db1 : Keys = 1 , expires = 0 , avg_ttl = 0
===== node 2 =====
used_memor y_human : 41 . 24M
# Keyspace
db0 : keys = 9956 , expires = 0 , avg_ttl = 0
db1 : keys = 1 , expires = 0 , avg_ttl = 0
===== node 3 =====
used_memor y_human : 37 . 58M
# Keyspace

```



```
db0 : keys = 9863 , expires = 0 , avg_ttl = 0
===== node 4 =====
used_memor y_human : 37 . 61M
# Keyspace
db0 : keys = 10045 , expires = 0 , avg_ttl = 0
===== node 5 =====
used_memor y_human : 37 . 58M
# Keyspace
db0 : keys = 10038 , expires = 0 , avg_ttl = 0
===== node 6 =====
used_memor y_human : 37 . 58M
# Keyspace
db0 : keys = 10055 , expires = 0 , avg_ttl = 0
===== node 7 =====
used_memor y_human : 37 . 57M
# Keyspace
db0 : keys = 9969 , expires = 0 , avg_ttl = 0
===== total =====
used_memor y_human : 304 . 31M
{' db1 ': {' keys ': 2 , ' expires ': 0 , ' avg_ttl ': 0 }, '
db0 ': {' keys ': 79648 , ' expires ': 0 , ' avg_ttl ': 0 }}
```

8 Troubleshooting for the JedisPool error

When you use JedisPool, the following error may occur, indicating the failure to obtain a resource from the pool.

```
redis.clients.jedis.exceptions.JedisConnectionException: Could not get a resource from the pool
```

You can solve this issue in the following ways.

Check the network

First, check the network condition. You can test the network by running the `telnet host 6379` command. After the host is connected, input a combination of `auth , a space and a password` , and then press ENTER. If the system returns `+ OK \r \n` , run the `Ping` command or read and write requests to check whether the system indicates a successful operation. Perform multiple test operations to check the network conditions.

Check the setting of JedisPool connections

You must set JedisPool connections when using the connection pool. A failure to obtain a resource from the pool may occur when the number of connections is more than the value of `MaxTotal`. In this case, you can run `netstat -an | grep 6379 | grep EST | wc -l` on the client to view the number of connections and compare the result with the value of `MaxTotal`. If the two values are close to each other, the setting of the connection pool is correct.

Check JedisPool code

When you use JedisPool, after the `getResource` operation, you must call the `returnResource` or `close` operation to return the resource. You can check whether the operation code is correct. The example code is as follows:

```
JedisPoolConfig config = new JedisPoolConfig();  
// The maximum number of idle connections. You can  
set the parameter for your application as needed.  
The value of this parameter cannot be more than the  
maximum number of clients connected to each ApsaraDB  
for Redis instance.  
config.setMaxIdle(200);  
// The maximum number of connections. You can set  
the parameter for your application as needed. The  
value of this parameter cannot be more than the
```

```

maximum number of clients connected to each ApsaraDB
for Redis instance .
config . setMaxTotal ( 300 );
config . setTestOnBorrow ( false );
config . setTestOnReturn ( false );
String host = "*. aliyuncs . com ";
String password = " Password ";
JedisPool pool = new JedisPool ( config , host , 6379 , 3000
, password );
Jedis jedis = null ;
try {
    jedis = pool . getResource ();
    /// ... do stuff here ... for example
    jedis . set ( " foo " , " bar " );
    String foobar = jedis . get ( " foo " );
    jedis . zadd ( " sose " , 0 , " car " );
    jedis . zadd ( " sose " , 0 , " bike " );
    Set < String > sose = jedis . zrange ( " sose " , 0 , - 1 );
} finally {
    if ( jedis != null ) {
        jedis . close ();
    }
}
/// ... when closing your application :
pool . destroy ();

```

Check for nf_conntrack packet loss

Run the `dmesg` command to check whether the client has any exceptions.

```
nf_conntrack : table full , dropping packet
```

If nf_conntrack packet loss occurs, modify the setting `sysctl -w net .`

```
netfilter . nf_conntrack_max = 120000 .
```

Check TIME_WAIT connections

Run the `ss -s` command to check whether excessive TIME_WAIT connections exist.

```

$ss -s
Total: 2385 (kernel 2872)
TCP: 2292 (estab 1772, closed 67, orphaned 0, synrecv 64, timewait 34/59), ports 1946

Transport Total      IP        IPv6
*          2872      -         -
RAW        0         0         0
UDP        6         6         0
TCP        2161      2161      0
INET       2167      2167      0
FRAG       0         0         0

```

If so, modify the following parameters:

```

sysctl -w net . ipv4 . tcp_max_tw_buckets = 180000
sysctl -w net . ipv4 . tcp_tw_recycle = 1

```

Check DNS resolution

Bind the host address in the `/etc/hosts` file and then check whether the issue persists. If so, DNS resolution is normal.

```
192 . 168 . 1 . 1    *. redis . rds . aliyuncs . com
```

Help

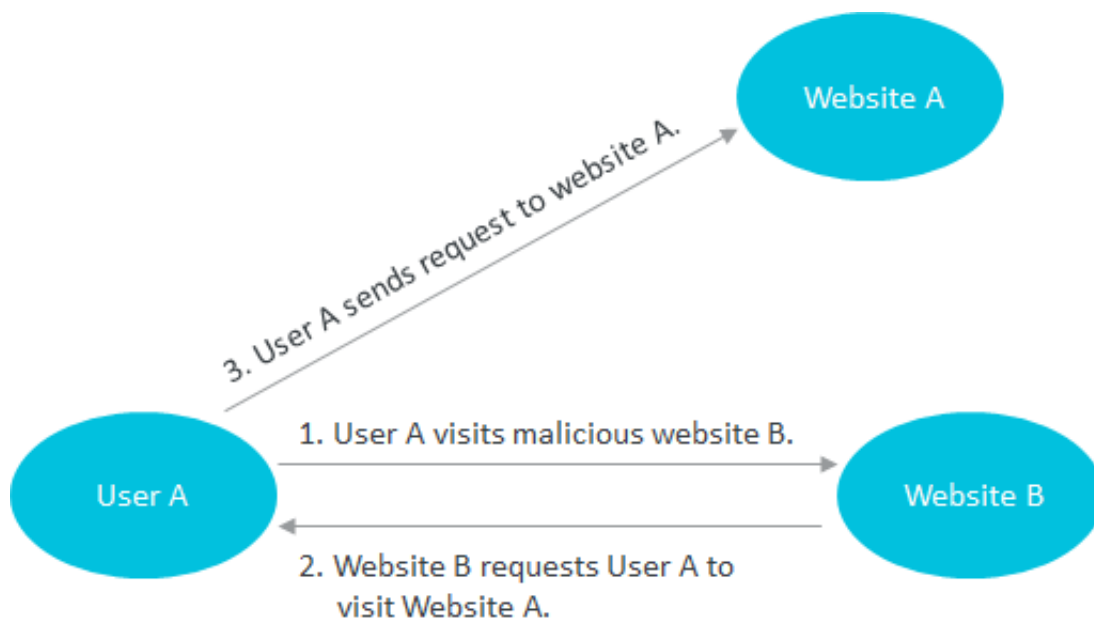
If the issue persists, capture packets, record the error occurrence time and error messages, and then submit a ticket to request technical support. You need to provide the record and the packet capture file in the ticket. The packet capture command is as follows:

```
sudo tcpdump -i eth0 tcp and port 6379 -n -nn -s  
74 -w redis . cap
```

9 Analysis of the Redis CSRF vulnerability and the corresponding security measure in ApsaraDB for Redis

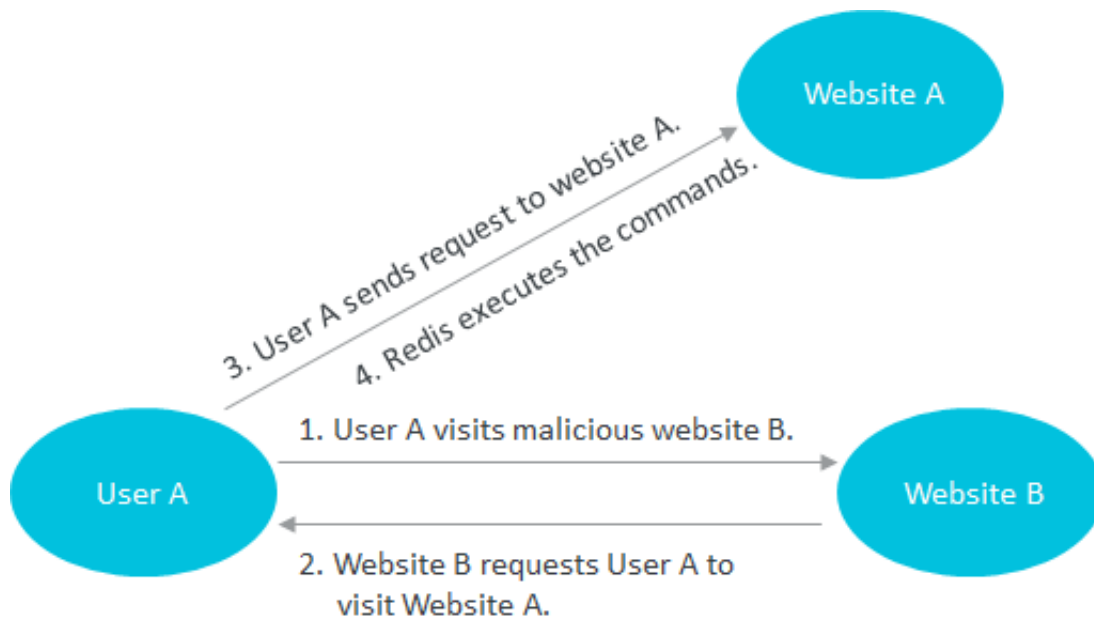
What is CSRF?

Cross-site request forgery (CSRF), also known as XSRF, one-click attack, or session riding, is a type of malicious exploitation of websites.



The preceding figure shows a simple model of a CSRF attack. A user visits the malicious website B, and the malicious website B replies to the user with an HTTP message that requires the user to visit the website A. If the user has maintained a trust relationship with the website A, the system processes the request as if the user personally sent the request to visit the website A.

Redis CSRF attack model



Based on the preceding principle of CSRF, a malicious website can require a user to send an HTTP request to Redis. Redis supports text protocols, and does not break off the connection in the case of illegal protocols during protocol parsing. An attacker can attach a Redis command to a normal HTTP request to run the command in Redis. If the user and Redis do not require password verification, Redis runs the command normally. Consequently, the attacker can encrypt data to extort money, similar to the earlier MongoDB ransom attacks.

Repair the kernel

Redis 3.2.7 provides a fix for this issue. The system processes the POST method and HOST keywords in a special way, keeps a log of events, and disconnects from the service to prevent Redis from running subsequent legitimate requests.

Redis security risks

Earlier Redis versions have exposed a security vulnerability where an attacker can obtain the root permissions of the Redis service in a certain condition. Similar security vulnerabilities occur because some users know less about security mechanisms of Redis and have little experience of operations and maintenance for Redis. In addition, Redis lacks sufficient security protection mechanisms. However, the ApsaraDB for Redis service can provide more security mechanisms. We recommend that you use ApsaraDB for Redis as the Redis service in the cloud.

Security rules of ApsaraDB for Redis

Connections over an internal network instead of a public network

By default, ApsaraDB for Redis only supports trusted connections over the Alibaba Cloud intranet. Without applying for a public endpoint, your instance is not open to the Internet.

Physical network isolation

ApsaraDB for Redis provides a physical isolation between the physical server network and the virtual server network. Your virtual servers cannot directly connect to the backend physical server network.

VPC network isolation

If you use a virtual private cloud (VPC) of Alibaba Cloud, only the services in the same VPC can interconnect with each other.

Whitelist

ApsaraDB for Redis supports whitelists. You can set a whitelist of IP addresses in the console to allow connections based on these IP addresses.

Password verification

ApsaraDB for Redis enforces password verification for instances in a classic network. You can set a complex password to prevent password cracking.

Access permission isolation

ApsaraDB for Redis isolates permissions and accessible directories for each backend instance. The instances can only access resources by using their own path to avoid mutual interference.

Dangerous commands forbidden

ApsaraDB for Redis forbids some dangerous system management commands such as CONFIG and SAVE. If you want to modify parameters, you must pass the secondary authentication in the console. This can also avoid direct operations of the backend configuration files and management commands.

Security monitoring

ApsaraDB for Redis provides comprehensive security monitoring for physical servers

. The system performs regular scans and updates security monitoring policies to locate security risks in advance.

Redis cluster password

The original Redis 3.0 cluster does not support password verification. ApsaraDB for Redis clusters support password verification to improve system security.

10 Can an ECS instance in a VPC connect to an ApsaraDB for Redis instance in a classic network?

A virtual private cloud (VPC) and a classic network are two subnets. They cannot interconnect with each other over an internal network. You can solve this issue in the following ways:

- If you allow changing the network type of the ApsaraDB for Redis instance, you can switch the ApsaraDB for Redis instance to the VPC where the ECS instance is located. For more information, see [#unique_7](#). However, you cannot switch the ApsaraDB for Redis instance from the VPC to a classic network.



Note:

When you change the network type, if no switch is available, you can create a switch in the VPC. The switch stays in the same zone as the ApsaraDB for Redis instance. Afterward, switch to the VPC. For more information about how to create a switch, see [#unique_41](#).

- If you do not allow changing the network type of the ApsaraDB for Redis instance, you have to purchase an ECS instance in a classic network. Because the ECS instance does not support switching from a VPC to a classic network.

11 How can I obtain the requestId?

When you create an instance, upgrade or downgrade an instance, or fail to call an API operation in ApsaraDB for Redis, you may submit a ticket to request technical support. We recommend that you provide the corresponding requestId in these cases for troubleshooting. This topic describes the method for generating the requestId when you use Google Chrome.

Procedure


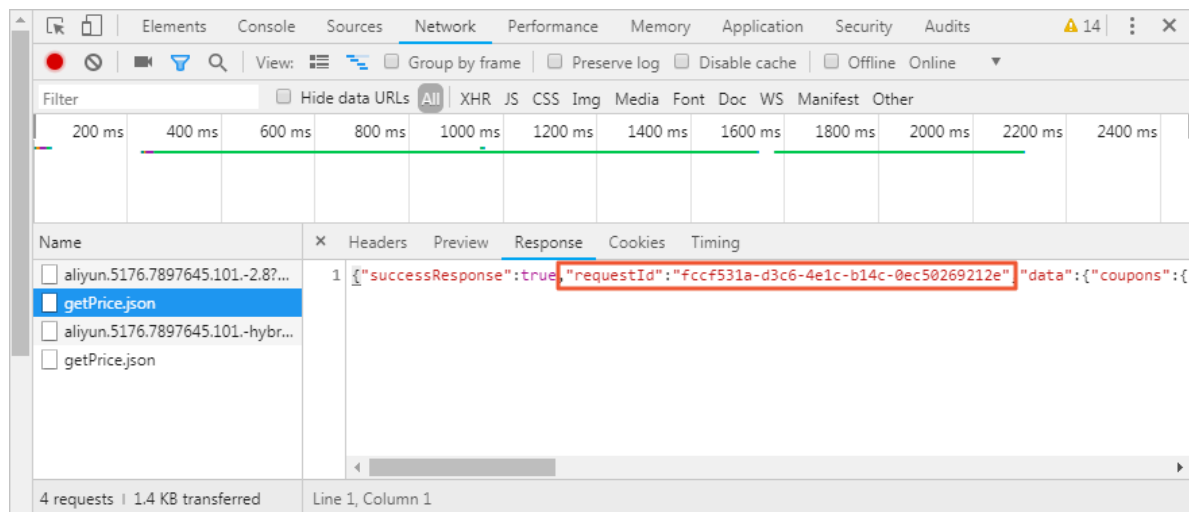
1. In Google Chrome, click  and choose More Tools > Developer Tools, or press F12 on the keyboard.
Afterward, the DevTools window appears.
2. Submit the last failed request.
3. In the Response panel, locate the requestId of the target request.

Figure 11-1: Response details



12 Connect ECS and ApsaraDB for Redis instances in different networks

If your ECS instance runs in a classic network and the ApsaraDB for Redis instance runs in a virtual private cloud (VPC), or the ApsaraDB for Redis instance runs in a classic network and the ECS instance runs in a VPC, you can quickly connect both instances by following the procedures in this topic.

Background

A classic network and a VPC are two different subnets and cannot directly interconnect with each other. ECS and ApsaraDB for Redis instances may run in different types of networks if you have selected the wrong network type or your business architecture is changed. However, your business requires a connection between the ECS instance and the ApsaraDB for Redis instance. Then, you can change the network type of the ApsaraDB for Redis instance, or use ClassicLink to allow both instances to interconnect with each other.



Notice:

The ClassicLink-based interconnection is a temporary solution in special conditions. To achieve efficient caching in the running environment that your service requires, we recommend that you create the ECS and ApsaraDB for Redis instances in the same VPCs.

The ECS instance in a classic network connects to the ApsaraDB for Redis instance in a VPC

You can establish a ClassicLink-based connection to allow the ECS instance in a classic network and the ApsaraDB for Redis instance in a VPC to interconnect with each other.

Prerequisites

- The ECS and ApsaraDB for Redis instances belong to the same account and run in the same region.
- The Classless Inter-Domain Routing (CIDR) block settings in the VPC must follow the rules of using the ClassicLink feature. For more information, see [#unique_44](#).



Note:

To allow the ECS and ApsaraDB for Redis instances in different regions to interconnect with each other quickly, you can migrate the ApsaraDB for Redis instance to the region where the ECS instance is located. For more information, see [#unique_6](#).

Procedure

1. Log on to the [ApsaraDB for Redis console](#).
2. Add the internal IP address of the ECS instance to a whitelist of the ApsaraDB for Redis instance. For more information, see [Set IP whitelists](#).
3. [#unique_45](#).
4. Test the connection on the ECS instance.

```
[root@ ~]# ping r-...redis.rds.aliyuncs.com
PING r-...redis.rds.aliyuncs.com ( ) 56(84) bytes of data:
64 bytes from ( ): icmp_seq=1 ttl=63 time=0.117 ms
64 bytes from ( ): icmp_seq=2 ttl=63 time=0.141 ms
64 bytes from ( ): icmp_seq=3 ttl=63 time=0.142 ms
64 bytes from ( ): icmp_seq=4 ttl=63 time=0.149 ms
64 bytes from ( ): icmp_seq=5 ttl=63 time=0.135 ms
64 bytes from ( ): icmp_seq=6 ttl=63 time=0.158 ms
^C
--- r-...redis.rds.aliyuncs.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.117/0.140/0.158/0.015 ms
[root@ ~]#
```

The ECS instance in a VPC connects to the ApsaraDB for Redis instance in a classic network

In this case, you can switch the ApsaraDB for Redis instance to a VPC at the appropriate time to allow both instances to quickly interconnect with each other. For more information, see [#unique_7](#).



Note:

To allow the ECS and ApsaraDB for Redis instances in different regions to interconnect with each other quickly, you can migrate the ApsaraDB for Redis instance to the region where the ECS instance is located. For more information, see [#unique_6](#).

13 Connect an ApsaraDB for Redis instance to an ECS instance across different accounts

You can use the Cloud Enterprise Network (CEN) or Express Connect service of Alibaba Cloud to connect virtual private clouds (VPCs) under different Alibaba Cloud accounts. Therefore, an ApsaraDB for Redis instance and an ECS instance under different Alibaba Cloud accounts can interconnect with each other over an internal network.

Connect instances based on CEN

CEN allows you to create a global network for rapidly building a distributed business system with a hybrid cloud computing solution. CEN enables you to build a secure, private, and enterprise-class interconnected network between VPCs in different regions and your local data centers. CEN provides enterprise-class scalability that automatically responds to your dynamic computing requirements. You can connect resources under different Alibaba Cloud accounts based on CEN.



Note:

- If you do not deploy resources across regions and have no account requirements, we recommend that you create an ECS instance and an ApsaraDB for Redis instance in the same VPC of the same region under the same account.
- If permitted, we recommend that you migrate the ECS and ApsaraDB for Redis instances under different accounts to the same account. For more information, see [#unique_6](#).

Procedure

1. Based on the running environment that your service requires, select one of the CEN-based connections over the Alibaba Cloud intranet from [#unique_47](#).
2. Allow connections between an ECS instance and an ApsaraDB for Redis instance in the inbound and outbound rules of an ECS security group.



Note:

You can use the `ping < host >` command to view the internal IP address of the ApsaraDB for Redis instance, and add the internal IP address to the security

group rules. In the command, the host parameter is the connection address of the ApsaraDB for Redis instance. For more information about how to configure security group rules, see [#unique_48](#).

3. Add the internal IP address of the ECS instance to a whitelist of the ApsaraDB for Redis instance. For more information, see [Set IP whitelists](#).
4. Run the `ping < host >` command on the ECS instance to confirm that the instances are connected, as shown in the following example.

```
[root@ ~]# ping .redis.rds.aliyuncs.com
PING cluster40.redis.rds.aliyuncs.com (172.16.172.224) 56(84) bytes of data.
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=1 ttl=64 time=0.628 ms
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=2 ttl=64 time=0.628 ms
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=3 ttl=64 time=0.633 ms
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=4 ttl=64 time=0.635 ms
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=5 ttl=64 time=0.626 ms
^C
--- cluster40.redis.rds.aliyuncs.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.626/0.630/0.635/0.003 ms
```



Note:

- If the connection fails for the first time, check the configuration of CEN first. If the configuration is correct but the instances are still disconnected, you can submit a ticket to require troubleshooting for CEN-based connections.
- If a connection issue occurs when the service is running, see [#unique_49](#).

Connect instances based on Express Connect

Alibaba Cloud Express Connect helps you build private network communication channels between VPCs or between a VPC and your local data center. These channels increase network topology flexibility and enhance cross-network communication quality and security. Express Connect can also avoid unstable network quality, and prevent data theft during transmission. You can use Express Connect to connect VPCs under different accounts over an internal network.

Procedure

1. Connect the VPCs under two accounts based on Express Connect. For more information, see [#unique_34](#).
2. Allow connections between an ECS instance and an ApsaraDB for Redis instance in the inbound and outbound rules of an ECS security group.



Note:

You can use the `ping < host >` command to view the internal IP address of the ApsaraDB for Redis instance, and add the internal IP address to the security group rules. In the command, the host parameter is the connection address of the ApsaraDB for Redis instance. For more information about how to configure security group rules, see [#unique_48](#).

3. Add the internal IP address of the ECS instance to a whitelist of the ApsaraDB for Redis instance. For more information, see [Set IP whitelists](#).
4. Run the `ping < host >` command on the ECS instance to confirm that the instances are connected, as shown in the following example.

```
[root@ ~]# ping cluster40.redis.rds.aliyuncs.com
PING cluster40.redis.rds.aliyuncs.com (172.16.172.224) 56(84) bytes of data.
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=1 ttl=64 time=0.628 ms
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=2 ttl=64 time=0.628 ms
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=3 ttl=64 time=0.633 ms
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=4 ttl=64 time=0.635 ms
64 bytes from 172.16.172.224 (172.16.172.224): icmp_seq=5 ttl=64 time=0.626 ms
^C
--- cluster40.redis.rds.aliyuncs.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.626/0.630/0.635/0.003 ms
```



Note:

- If the connection fails for the first time, check the configuration of Express Connect first. If the configuration is correct but the instances are still disconnected, you can submit a ticket to require troubleshooting for Express Connect-based connections.
- If a connection issue occurs when the service is running, see [#unique_49](#).

14 Delete multiple keys in the Linux operating system

You can use the `xargs` command of Linux and the `DEL` command of Redis to delete multiple target keys. This topic describes the operations in details.

The `xargs` command is used in the Linux operating system. The command passes a list of parameters to other commands in segments. Therefore, the system does not need to process a long list of parameters and can avoid related issues. The command can be used independently, or used with pipeline operators, relocation operators and other commands.

Cautions

- The `KEYS` command may cause high CPU usage. Use this command during off-peak hours.
- If you use the `KEYS` command in a large database, the command affects the database performance. We recommend that you use this command in a database that contains small amounts of data.

Procedure

1. Install ApsaraDB for Redis in the Linux operating system.

```
yum install redis
```

2. Run the following command to delete multiple target keys in a database. In the following figure, "test *" contains multiple target keys, such as test1, test2, and test3.

```
redis - cli - h < host > - a < password > keys "< key >" |  
xargs redis - cli - h < host > - a < password > del
```

- `<host>`: the connection address of the ApsaraDB for Redis instance.
- `<password>`: the password of the ApsaraDB for Redis instance.
- `"<key>"`: the target key such as "test" in a database.



Note:

The system matches a target key in the following ways.

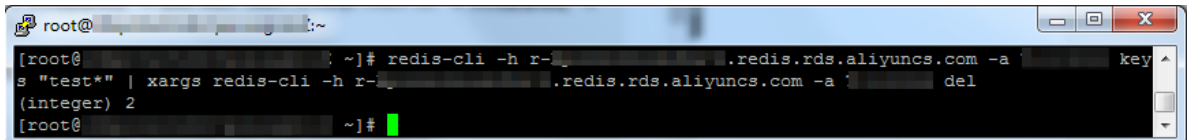
w? rld: matches world, world and wxrld.

w*rld: matches wrld and wooorld.

w[ae]rld: matches world and werld, but does not match world.

w[^e]rld: matches world and warld, but does not match werld.

w[a-b]rld: matches warld and wbrld.

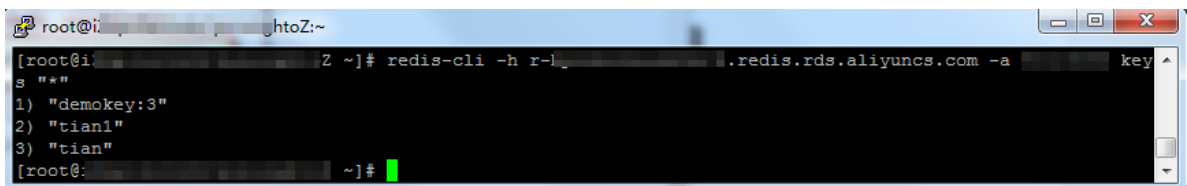
A terminal window showing a Redis CLI session. The user enters the command `redis-cli -h r-... .redis.rds.aliyuncs.com -a ... key`. The output shows `s "test*" | xargs redis-cli -h r-... .redis.rds.aliyuncs.com -a ... del` and `(integer) 2`.

```
[root@ ~]# redis-cli -h r-... .redis.rds.aliyuncs.com -a ... key
s "test*" | xargs redis-cli -h r-... .redis.rds.aliyuncs.com -a ... del
(integer) 2
[root@ ~]#
```

3. Run the following command to check whether the system has deleted the keys that contain test.

```
redis - cli - h < host > - a < password > keys "*" 
```

- <host>: the connection address of the ApsaraDB for Redis instance.
- <password>: the password of the ApsaraDB for Redis instance.

A terminal window showing a Redis CLI session. The user enters the command `redis-cli -h r-... .redis.rds.aliyuncs.com -a ... key`. The output shows a list of keys: `1) "demokey:3"`, `2) "tian1"`, and `3) "tian"`.

```
[root@i: ~]# redis-cli -h r-... .redis.rds.aliyuncs.com -a ... key
s "*"
1) "demokey:3"
2) "tian1"
3) "tian"
[root@i: ~]#
```

15 View the architecture and monitoring data of an ApsaraDB for Redis cluster instance

On the Instance Information page in the ApsaraDB for Redis console, you can view the architecture of an ApsaraDB for Redis cluster instance. The architecture displays information about the proxy servers, master and replica nodes. You can also click a node to view the monitoring data of the node.

Procedure

1. Log on to the [ApsaraDB for Redis console](#).
2. On the menu bar, select the region where the instance is located.
3. On the Instance List page, click the target instance ID or Manage in the Action column next to the target instance.
4. In the Architecture Diagram field on the Instance Information page, view the architecture of the cluster instance.

Figure 15-1: Architecture of the cluster instance



5. Click a node icon to view the monitoring data of the node.



Note:

Place the pointer over the node icon to view the node ID and other information.

