云数据库 Redis 版

快速入门

快速入门

开始使用 Redis

文档目的

快速入门旨在介绍如何创建 Redis 实例以及连接实例数据库，使用户能够了解从购买 Redis 实例到开始使用实例的流程。

目标读者

首次购买 Redis 实例的用户

想要了解如何连接 Redis 实例的用户

快速入门流程图

若您初次使用云数据库 Redis 版，请先了解使用限制以及关于 Redis 管理控制台。

通常，从新购实例到可以开始使用实例，您需要完成如下操作：

关于 Redis 管理控制台

Redis 管理控制台是用于管理 Redis 实例的 Web 应用程序，您可以通过该控制台上直观的用户界面进行实例创建、网络设置、实例管理、密码设置等操作。

Redis 管理控制台是阿里云管理控制台的一部分，关于控制台的通用设置和基本操作请参见使用阿里云管理控制台。本文将介绍 Redis 控制台的通用界面，若有差异，请以控制台实际界面为准。

前提条件

使用阿里云账号登录 Redis 管理控制台。若没有阿里云账号，请单击注册。

控制台简介

控制台首页

对于 Redis 所有类型的实例而言，控制台首页的界面信息都是相同的。

登录 Redis 管理控制台，进入**实例列表**页面，如下图所示（仅为示例，请以实际界面为准）。

实例列表页面中会展示**实例 ID**、**状态**、**已用内存及配额**、**可用区**、**创建时间**、**付费方式**、**网络类型**等信息。

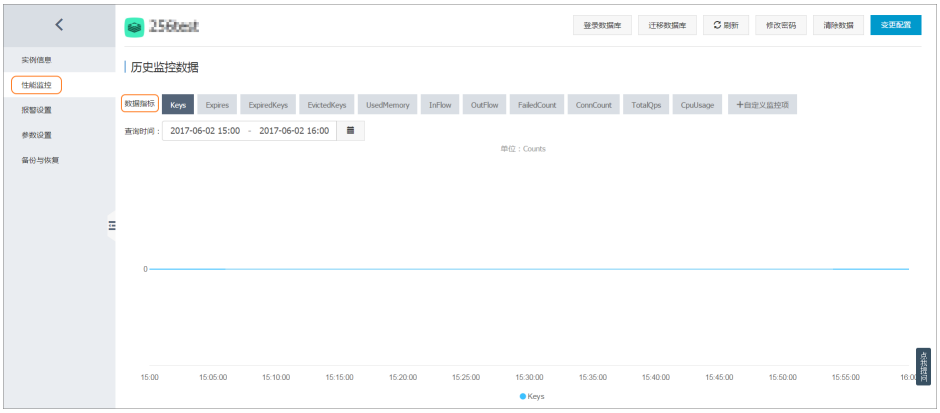
> **注意**：**已用内存及配额**信息是由底层系统根据采集信息进行的一个离线汇总，所以有一个时间延时，这个延时会在10分钟左右。如果需要查看实时信息，建议登录 DMS 进行查看，详细步骤请参见DMS 登录云数据库。

可运维时间段

您可以在**实例信息**页面对可运维时间进行修改，阿里云会在可运维时间对实例进行生产维护，维护期间可能会发生闪断，建议您尽量选择业务低峰期为运维时间段。

性能监控

单击**实例 ID** 即可进入**实例信息**页面，在左侧导航栏中选择**性能监控**查看 Redis 的历史性能数据，可以查看到不同的监控项。



单击**性能监控**之后可以查看到不同的监控项，以下对基础监控组的监控项进行说明。

基础监控项	**说明**
Keys	后端 Redis 所有 db 的 key 个数的总和，对于集群实例会汇聚后端所有的节点的数据。
Expires	当前设置了过期数据的 key 的个数的总和。
ExpiredKeys	历史过期掉的 key 的个数。 这个值是历史过期掉的 key 的个数的总和，所以是不包含当前设置了过期 key 同时没有过期掉的值。同时，它是一个历史累加值，不是当前已经过期的 key 的个数。 **注意**：如果发生主备切换，该值会以新的主库为准。
EvictedKeys	历史淘汰掉的 key 的个数。 这个值是因内存满被淘汰掉的 key 的历史个数的总和，所以它不是当前每秒淘汰的 key 的个数。 **注意**：如果发生主备切换，该值会以新的主库为准。

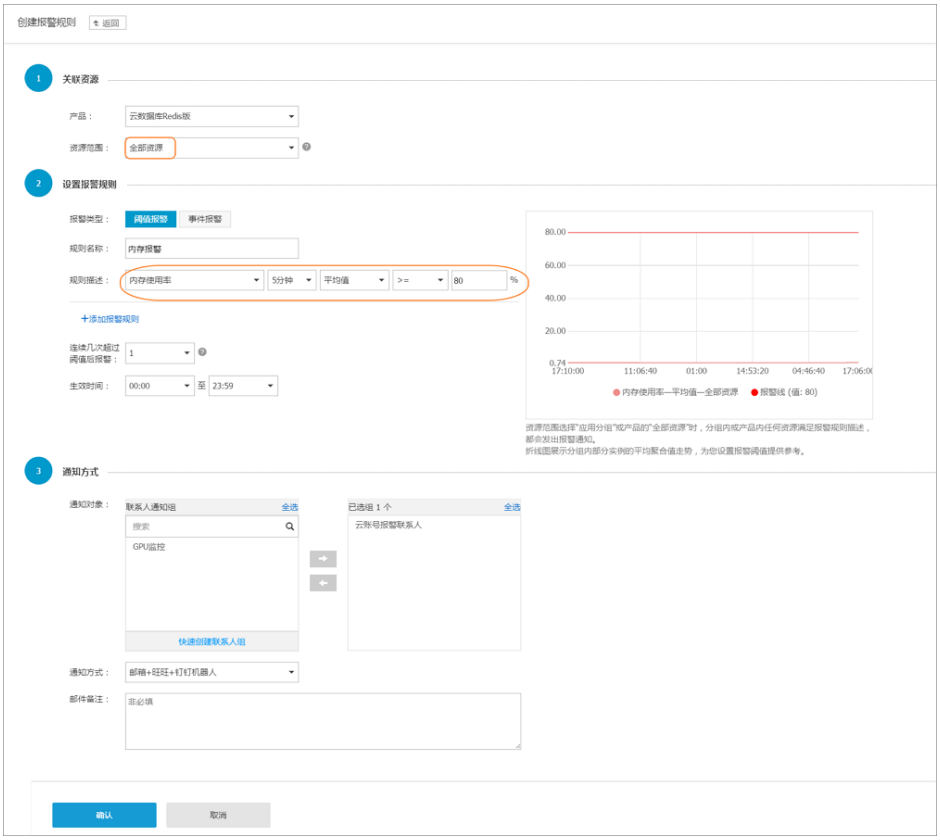
UsedMemory	当前内存的使用值。 由于新建实例时会产生一定的元信息，所以对于主从实例这个值最小是 30 MB，对于集群实例这个数据的初始值为 30 MB乘以节点数，最小为 200 MB。
InFlow	后端 Redis 入口当前每秒的流量值，单位为 KBytes/s。
OutFlow	后端 Redis 出口当前每秒的流量值，单位为 KBytes/s。
ConnCount	当前 Redis 的客户端连接个数。
FailedCount	对于主从版本，目前这个值没有意义，因为客户端直接连接到后端 DB。对于集群版实例，该统计项标识 Proxy 到 Redis 的操作失败数目，包括超时、连接断开等异常引起的操作异常的数目。 对于部分旧版本的 Redis，该值为一个历史值，对于这种情况如果 FaileCount 没有增加则没有问题。对于新版本，该值为每秒的一个统计均值。后续会都升级成每秒的统计均值。
TotalQps	当前 Redis 的每秒操作次数。
CpuUsage	当前 Redis 后端的 CPU 使用率。

说明：您可以单击**自定义监控项**添加不同操作命令的访问次数的监控，比如查看 set 命令每秒的次数。详细信息请参见**性能监控**。

报警设置

选择左侧导航栏的**报警设置**，单击**报警设置**按钮跳转到云监控的设置页面。

您可以根据指引创建 Redis 的监控。对于集群实例建议添加所有实例的内存监控，这样可以对集群实例的子节点的内存进行监控，告警设置如下：



参数设置

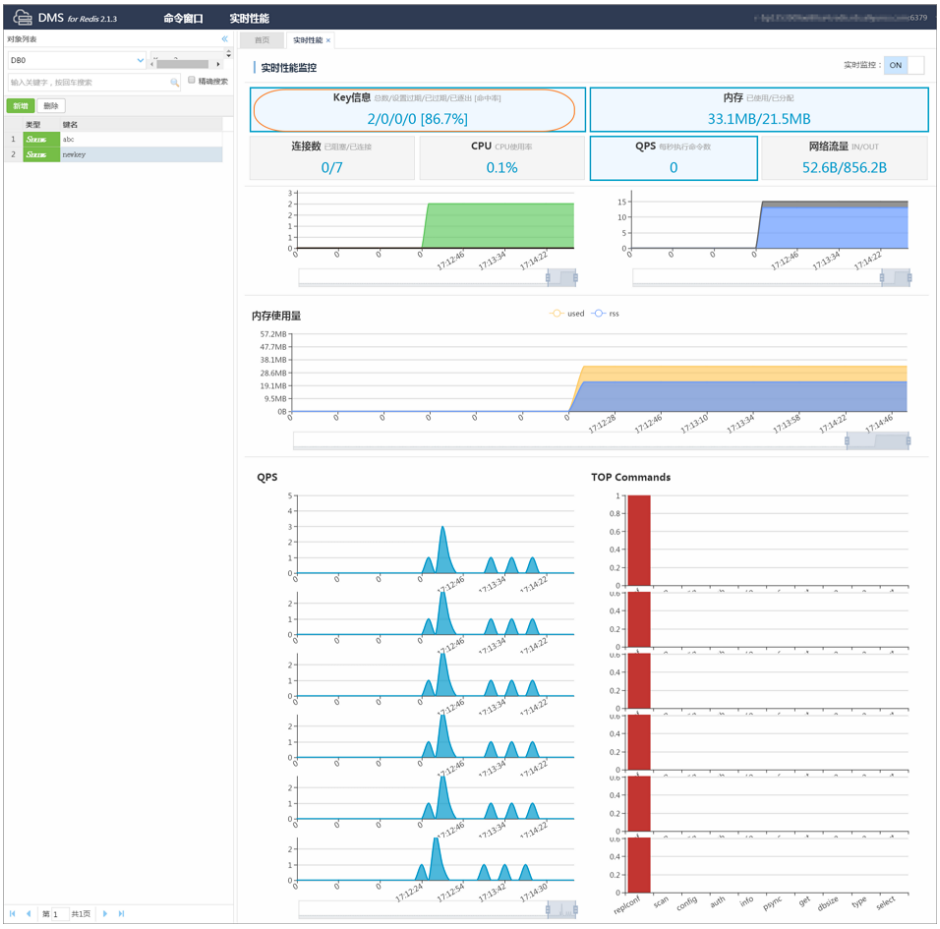
您可以在**参数设置**页面对 Redis 的常见参数进行设置，比如淘汰策略及 notify-keypsace-events 等。详细操作请参见**参数设置**。

备份恢复

您可以在备份恢复页面进行备份的设置和克隆实例，另外可以设置自动备份的时间。详细操作请参见**备份与恢复**。

DMS 页面

DMS 实例信息页面展示如下，其中 Key 信息这一栏中设置过期和已逐出这两个项目为历史上的值，也就是前面提到的 ExpiredKeys 和 EvictedKeys 值，不是当前每秒的值。



使用限制

项目	说明
List 数据类型	没有 List 个数限制，单个元素最大值为 512 MB，推荐 list的元素个数小于 8192, value 最大长度不超过 1 MB。
Set 数据类型	没有 set 个数限制，单个元素最大值为 512 MB，推荐 set 的元素个数小于 8192， value 最大长度不超过 1 MB。
Sorted set 数据类型	没有 sorted set 个数限制，单个元素最大值为 512 MB，推荐 sorted set 的元素个数小于 8192， value 最大长度不超过 1 MB。
Hash 数据类型	没有 field 个数限制，单个元素最大值为 512 MB，推荐元素个数小于 8192， value 最大长度不超过 1 MB。

DB 数限制	每个实例支持 256 个 DB。
Redis 命令支持	详情请参见文档。
监控报警	云数据库 Redis 版未提供容量告警，需要用户到云监控中进行配置。配置方法请参见文档。 建议设置好以下监控的报警：实例故障、实例主备切换、已使用连接百分比、操作失败数、已用容量百分比、写入带宽使用率、读取带宽使用率。
数据过期删除策略	- 主动过期，系统后台会周期性的检测，发现已过期的key时，会将其删除。 - 被动过期，当用户访问某个key时，如果该key已经过期，则将其删除。
空闲连接回收机制	服务端不主动回收 Redis 空闲连接，由用户管理。
数据持久化策略	采用 AOF_FSYNC_EVERYSEC 方式，每秒 fysnc。

支持的 Redis 命令

云数据库 Redis 版兼容 Redis 3.0 版本，支持 Redis 3.0 的 Geo 命令。目前还有小部分暂未开放的命令和受限制的命令。

支持的命令操作

Keys（键）	String（字符串）	Hash（哈希表）	List（列表）	Set（集合）	SortedSet（有序集合）
DEL	APPEND	HDEL	BLPOP	SADD	ZADD
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD
EXISTS	BITOP	HGET	BRPOPLPUSH	SDIFF	ZCOUNT
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY
EXPIREAT	DECR	HINCRBY	LINSERT	SINTER	ZRANGE
MOVE	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE
PERSIST	GET	HKEYS	LPOP	SISMEMBER	ZRANK
PEXPIRE	GETBIT	HLEN	LPUSH	SMEMBERS	ZREM
PEXPTREAT	GETRANGE	HMGET	LPUSHX	SMOVE	ZREMRANG

					EBYRANK
PTTL	GETSET	HMSET	LRANGE	SPOP	ZREMRANGEBYSCORE
RANDOMKEY	INCR	HSET	LREM	SRANDMEMBER	ZREVRANGE
RENAME	INCRBY	HSETNX	LSET	SREM	ZREVRANGEBYSCORE
RENAMENX	INCRBYFLOAT	HVALS	LTRIM	SUNION	ZREVRANK
RESTORE	MGET	HSCAN	RPOP	SUNIONSTORE	ZSCORE
SORT	MSET		RPOPLPUSH	SSCAN	ZUNIONSTORE
TTL	MSETNX		RPUSH		ZINTERSTORE
TYPE	PSETEX		RPUSHX		ZSCAN
SCAN	SET				ZRANGEBYLEX
OBJECT	SETBIT				ZLEXCOUNT
	SETEX				ZREMRANGEBYLEX
	SETNX				
	SETRANGE				
	STRLEN				

以及

HyperLogLog	Pub/Sub（发布/订阅）	Transaction（事务）	Connection（连接）	Server（服务器）	Scripting(脚本)	Geo(地理位置)
PFADD	PSUBSCRIBE	DISCARD	AUTH	FLUSHALL	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	FLUSHDB	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	DBSIZE	SCRIPT EXISTS	GEOPOS
	PUNSUBSCRIBE	UNWATCH	QUIT	TIME	SCRIPT FLUSH	GEODIST
	SUBSCRIBE	WATCH	SELECT	INFO	SCRIPT KILL	GEORADIUS
	UNSUBSCRIBE			KEYS	SCRIPT LOAD	GEORADIUSBYME

						MBER
				CLIENT KILL		
				CLIENT LIST		
				CLIENT GETNAME		
				CLIENT SETNAME		
				CONFIG GET		
				MONITOR		
				SLOWLOG		

说明：

集群实例下，client list 命令列出所有连接到该 proxy 的 user connection。其中，id、age、idle、addr、fd、name、db、multi、omem、cmd 字段和redis内核表达的意思一样。sub、psub 在 proxy 层没作区分，要么都为1，要么都为0。qbuf、qbuf-free、obl、oll 字段目前没有意义。

集群实例下，client kill 命令目前支持两种形式：client kill ip:port和client kill addr ip:port。

暂未开放的命令

Keys（键）	Server（服务器）
MIGRATE	BGREWRITEAOF
	BGSAVE
	CONFIG REWRITE
	CONFIG SET
	CONFIG RESETSTAT
	COMMAND
	COMMAND COUNT
	COMMAND GETKEYS
	COMMAND INFO

	DEBUG OBJECT
	DEBUG SEGFAULT
	LASTSAVE
	ROLE
	SAVE
	SHUTDOWN
	SLAVEOF
	SYNC

集群实例受限制的命令

Keys	Strings	Lists	HyperLogLog	Transaction	Scripting
RENAME	MSETNX	RPOPLPUSH	PFMERGE	DISCARD	EVAL
RENAMENX			PFCOUNT	EXEC	EVALSHA
SORT				MULTI	SCRIPT EXISTS
				UNWATCH	SCRIPT FLUSH
				WATCH	SCRIPT KILL
				WATCH	SCRIPT LOAD

说明：

集群实例受限命令只支持所操作 key 均分布在单个 hash slot 中的场景，没有实现多个 hash slot 数据的合并功能，因此需要用 hash tag 的方式确保要操作的 key 均分布在一个 hash slot 中。

比如有 key1，aakey，abkey3，那么我们在存储的时候需要用 {key}1，aa{key}，ab{key}3 的方式存储，这样调用受限命令时才能生效。具体关于 hash tag 的用法请参见 Redis 官方文档：http://redis.io/topics/cluster-spec。

事务之前没有使用 watch 命令且事务中都是单 key 的命令场景，不再要求所有 key 必须在同一个 slot 中，使用方式和直连 redis 完全一致。其他场景要求事务中所有命令的所有 key 必须在同一个 slot 中。

多 key 命令包括：DEL、SORT、MGET、MSET、BITOP、EXISTS、MSETNX、

RENAME、 RENAMENX、BLPOP、BRPOP、RPOPLPUSH、BRPOPLPUSH、SMOVE、SUNION、SINTER、SDIFF、SUNIONSTORE、SINTERSTORE、SDIFFSTORE、ZUNIONSTORE、ZINTERSTORE、 PFMERGE、PFCOUNT。

不允许在事务中使用的命令包括：WATCH、UNWATCH、RANDOMKEY、KEYS、SUBSCRIBE、 UNSUBSCRIBE、PSUBSCRIBE、PUNSUBSCRIBE、PUBLISH、PUBSUB、SCRIPT、EVAL、 EVALSHA、SCAN、ISCAN、DBSIZE、ADMINAUTH、AUTH、PING、ECHO、FLUSHDB、 FLUSHALL、MONITOR、IMONITOR、RIMONITOR、INFO、IINFO、RIINFO、CONFIG、 SLOWLOG、TIME、CLIENT。

自研的集群实例命令

info key 命令：查询 key 所属的 slot 和 db。Redis 原生的 info 命令中最多可以带一个可选的 section (info [section])。目前云数据库 Redis 版的集群实例，部分命令限制所有 key 必须在同一个 slot 中，info key 命令方便用户查询某些 key 是否在同一个 slot 中。用法如下：

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

iinfo 命令：用法类似于 info，用于在指定的 Redis 节点上执行 info 命令。用法如下：

iinfo db_idx [section]

其中，db_idx 的范围是[0, nodecount)，nodecount 可以通过 info 命令获取， section 为 info 官方一致的值。要了解某个 Redis 节点的 info 可以使用 iinfo 命令或者从控制台上查看实例拓扑图，详情请参见 如何查看 Redis 集群子实例内存。

riinfo 命令：和 iinfo 命令类似，但只能在读写分离的模式下使用。用法中增加了一个 readonly slave 的 idx，用于指定在第几个 readonly slave 上执行 info 命令。在读写分离集群中可以用来在指定 readonly slave 上执行 info 命令。如果在非读写分离集群中使用，会返回错误。用法如下：

riinfo db_idx ro_slave_idx [section]

iscan 命令：在集群模式下可以在指定的 db 节点上执行 scan 命令。在 scan 命令的基础上扩展了一个参数用于指定 db_idx， db_idx 的范围是[0, nodecount), nodecount 可以通过 info 命令获取或者从控制台上查看实例拓扑图。用法如下：

iscan db_idx cursor [MATCH pattern] [COUNT count]

imonitor 命令：和 iinfo, iscan 类似，在 monitor 的基础上新增一个参数指定 monitor 执行的 db_idx，db_idx 的范围是[0, nodecount), nodecount 可以通过 info 命令获取或者从控制台上查看实例拓扑图。用法如下：

imonitor db_idx

rimonitor 命令：和 riinfo 类似，用于读写分离场景下，在指定的 shard 里的指定只读从库上执行 monitor 命令。用法如下：

rimonitor db_idx ro_slave_idx

说明：

关于 Redis 命令的详细信息，请参见 官方文档。

云数据库 Redis 版集群实例最新的命令支持详情，请参见 云栖社区说明。

创建实例

云数据库 Redis 版支持按量付费和包年包月两种模式，按量付费可转为包年包月模式，反之则不可以。您可根据自己的需求自主选择，以下对购买流程做介绍。

前提条件

开通云数据库 Redis 版需要至少有一台 ECS，具体操作参考 购买 ECS。

操作步骤

进入 云数据库 Redis 版产品首页，单击**立即购买**。或者进入 Redis 管理控制台，单击右上角的**创建实例**。

选择按量付费或预付费模式，完成**网络类型**、**地域**、**可用区**、**架构类型**等实例配置。两种付费模式的填写项基本一致。

注意事项：

主从版本支持变配至集群版，集群版实例在功能上和主从版实例有所不同，具体请参

见 Redis 支持命令。

如何选择网络类型，请参考设置网络类型。

云数据库 Redis 版仅限于内网访问，同一地域下的 ECS （不分可用区）都可以进行内网访问。建议和 ECS 选择在同一地域同一可用区。

单击**立即购买**，进入**订单确认**页面。阅读接受《云数据库 Redis 版服务条款》并确认订单信息无误后，单击**去支付**进行付款。

进入支付页面，选择支付方式，单击**确认支付**按钮。支付成功后会提示支付成功。等1-5分钟后进入控制台即可看见刚才购买的实例。

说明：云数据库 Redis 版在产品行为上与 Redis 一致，当新建一个实例后它会自动生成一些数据库元信息，因此在 Redis 控制台上会看到该实例已经有少量的存储空间被占用，这是正常现象。

对于主从版和单节点实例，占用空间约为 32 MB。

对于集群版实例，占用空间约为：节点数目*32 MB。

连接实例

DMS 登录云数据库

DMS 是一款访问管理云端数据的 Web 服务，支持 Redis、 MySQL、SQL Server、PostgreSQL 和 MongoDB 等数据源。DMS 提供了数据管理、对象管理、数据流转和实例管理四部分功能。您可以通过以下两种方式登录 DMS。

通过 Redis 管理控制台，选择要登录的实例，单击右上角的**登录数据库**打开 DMS。通过该种方式打开 DMS，系统会自动填写登录页面中的数据库连接地址，您只要输入密码即可。

打开 DMS，手工输入要登录的数据库连接地址和密码，如下图所示。

注意：

连接信息可以在 Redis 管理控制台 的**实例信息**页获取。

经典网络及 VPC 网络的实例都支持 DMS。由于 VPC 网络需要申请一个特殊通道，对于第一次登录的实例需要一定的缓冲时间。

更多的 DMS 相关信息请参见**数据管理**。

Redis 客户端连接

由于云数据库 Redis 提供的数据库服务与原生的数据库服务完全兼容，连接数据库的方式也基本类似。任何兼容 Redis 协议的客户端都可以访问云数据库 Redis 版服务，您可以根据自身应用特点选用任何 Redis 客户端。

注意：云数据库 Redis 版仅支持阿里云内网访问，不支持外网访问，即只有在同节点的 ECS上安装 Redis 客户端才能与云数据库建立连接并进行数据操作。

Redis 的客户端请参考 http://redis.io/clients 。

- Jedis 客户端
- phpredis 客户端
- redis-py 客户端
- C/C++ 客户端
- .net 客户端
- node-redis 客户端
- C# 客户端 StackExchange.Redis

Jedis 客户端

Jedis 客户端访问云数据库 Redis 版服务，有以下两种方法：

- Jedis单链接
- JedisPool连接池连接

操作步骤如下：

下载并安装Jedis客户端：单击下载地址。

Jedis 单连接示例

打开 Eclipse 客户端，创建一个 Project，输入如下代码段：

```
import redis.clients.jedis.Jedis;

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

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

```
Set Key redis Value aliyun-redis
```

```
Get Key redis ReturnValue aliyun-redis
```

接下来您就可以通过自己的本地客户端 Jedis 操作您的云数据库 Redis。您也可以通过 JedisPool 连接池来连接您的云数据库 Redis。

JedisPool 连接池示例

打开 Eclipse 客户端，创建一个 Project，配置 pom 文件，具体配置如下所示：

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

在 project 中添加如下应用：

```
import org.apache.commons.pool2.PooledObject;
import org.apache.commons.pool2.PooledObjectFactory;
import org.apache.commons.pool2.impl.DefaultPooledObject;
import org.apache.commons.pool2.impl.GenericObjectPoolConfig;

import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;
```

如果您的 Jedis 客户端版本是 Jedis-2.7.2，在 Project 中输入如下代码：

```
JedisPoolConfig config = new JedisPoolConfig();
//最大空闲连接数, 应用自己评估，不要超过ApsaraDB for Redis每个实例最大的连接数
config.setMaxIdle(200);
//最大连接数, 应用自己评估，不要超过ApsaraDB for Redis每个实例最大的连接数
config.setMaxTotal(300);
config.setTestOnBorrow(false);
config.setTestOnReturn(false);

String host = "*.aliyuncs.com";
String password = "密码";
JedisPool pool = new JedisPool(config, host, 6379, 3000, password);
Jedis jedis = null;
try {
jedis = pool.getResource();
/// ... do stuff here ... for example
jedis.set("foo", "bar");
String foobar = jedis.get("foo");
```

```
jedis.zadd("sose", 0, "car");
jedis.zadd("sose", 0, "bike");
Set<String> sose = jedis.zrange("sose", 0, -1);
} finally {
if (jedis != null) {
jedis.close();
}
}
/// ... when closing your application:
pool.destroy();
```

如果您的 Jedis 客户端版本是 Jedis-2.6、Jedis-2.5，在 Project 中输入如下代码：

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

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

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

接下来您就可以通过自己的本地客户端Jedis操作您的云数据库 Redis。

phpredis 客户端

操作步骤如下所示：

下载并安装phpredis客户端：单击 下载地址。

在任何一款可以编辑 php 的编辑器中输入如下代码：

```
<?php
/* 这里替换为连接的实例host和port */
$host = "localhost";
$port = 6379;

/* 这里替换为实例id和实例password */
$user = "test_username";
$pwd = "test_password";
$redis = new Redis();
if ($redis->connect($host, $port) == false) {
die($redis->getLastError());
}

if ($redis->auth($pwd) == false) {
die($redis->getLastError());
}
/* 认证后就可以进行数据库操作，详情文档参考https://github.com/phpredis/phpredis */
if ($redis->set("foo", "bar") == false) {
die($redis->getLastError());
}
$value = $redis->get("foo");
echo $value;
?>
```

3. 执行上述代码，您就可以通过自己的本地客户端 phpredis 访问您的云数据库 Redis，详情文档参考 https://github.com/phpredis/phpredis 。

redis-py 客户端

操作步骤如下：

下载并安装 redis-py 客户端：单击 下载地址。

在任何一款可以编辑 Python 的编辑器中输入如下代码，即可建立连接通过本地客户端 redis-py 进行数据库操作。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
import redis

#这里替换为连接的实例host和port
host = 'localhost'
port = 6379

#这里替换为实例password
pwd = 'test_password'
r = redis.StrictRedis(host=host, port=port, password=pwd)

#连接建立后就可以进行数据库操作，详情文档参考https://github.com/andymccurdy/redis-py
r.set('foo', 'bar');
print r.get('foo')
```

C/C++ 客户端

操作步骤如下所示：

下载并编译安装C客户端，编译安装代码如下所示：

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

在 C/C++编辑器中编写如下代码：

```
    #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hiredis.h>
int main(int argc, char **argv) {
unsigned int j;
redisContext *c;
redisReply *reply;
if (argc < 4) {
printf("Usage: example xxx.kvstore.aliyuncs.com 6379 instance_id password\n");
exit(0);
}
const char *hostname = argv[1];
const int port = atoi(argv[2]);
const char *instance_id = argv[3];
const char *password = argv[4];
struct timeval timeout = { 1, 500000 }; // 1.5 seconds
c = redisConnectWithTimeout(hostname, port, timeout);
if (c == NULL || c->err) {
if (c) {
```

```
printf("Connection error: %s\n", c->errstr);
redisFree(c);
} else {
printf("Connection error: can't allocate redis context\n");
}
exit(1);
}
/* AUTH */
reply = redisCommand(c, "AUTH %s", password);
printf("AUTH: %s\n", reply->str);
freeReplyObject(reply);
/* PING server */
reply = redisCommand(c,"PING");
printf("PING: %s\n", reply->str);
freeReplyObject(reply);
/* Set a key */
reply = redisCommand(c,"SET %s %s", "foo", "hello world");
printf("SET: %s\n", reply->str);
freeReplyObject(reply);
/* Set a key using binary safe API */
reply = redisCommand(c,"SET %b %b", "bar", (size_t) 3, "hello", (size_t) 5);
printf("SET (binary API): %s\n", reply->str);
freeReplyObject(reply);
/* Try a GET and two INCR */
reply = redisCommand(c,"GET foo");
printf("GET foo: %s\n", reply->str);
freeReplyObject(reply);
reply = redisCommand(c,"INCR counter");
printf("INCR counter: %lld\n", reply->integer);
freeReplyObject(reply);
/* again ... */
reply = redisCommand(c,"INCR counter");
printf("INCR counter: %lld\n", reply->integer);
freeReplyObject(reply);
/* Create a list of numbers, from 0 to 9 */
reply = redisCommand(c,"DEL mylist");
freeReplyObject(reply);
for (j = 0; j < 10; j++) {
char buf[64];
snprintf(buf,64,"%d",j);
reply = redisCommand(c,"LPUSH mylist element-%s", buf);
freeReplyObject(reply);
}
/* Let's check what we have inside the list */
reply = redisCommand(c,"LRANGE mylist 0 -1");
if (reply->type == REDIS_REPLY_ARRAY) {
for (j = 0; j < reply->elements; j++) {
printf("%u) %s\n", j, reply->element[j]->str);
}
}
freeReplyObject(reply);
/* Disconnects and frees the context */
redisFree(c);
return 0;
}
```

编译上述代码。

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

测试运行。

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

至此完成通过 C/C++ 客户端连接云数据库 Redis。

.net 客户端

操作步骤如下所示：

下载并使用.net 客户端。

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

在.net 客户端中新建 .net 项目。

添加客户端引用，引用文件在库文件的 ServiceStack.Redis/lib/tests 中。

在新建的.net 项目中输入如下代码来连接云数据库 Redis。详细的接口用法请参见 https://github.com/ServiceStack/ServiceStack.Redis 。

```
 using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ServiceStack.Redis;
namespace ServiceStack.Redis.Tests
{
class Program
{
public static void RedisClientTest()
{
string host = "127.0.0.1";/*访问host地址*/
string password = "password";/*密码*/
RedisClient redisClient = new RedisClient(host, 6379, password);
string key = "test-aliyun";
string value = "test-aliyun-value";
redisClient.Set(key, value);
```

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

node-redis 客户端

操作步骤如下所示：

下载并安装 node-redis。

```
npm install hiredis redis
```

在 node-redis 客户端中输入如下代码并执行以此连接云数据 Redis 版。

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

使用云数据 Redis 版。

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

C# 客户端 StackExchange.Redis

操作步骤如下所示：

下载并安装 StackExchange.Redis。

添加引用。

using StackExchange.Redis;

初始化 ConnectionMultiplexer。

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

```
 // redis config
private static ConfigurationOptions configurationOptions =
ConfigurationOptions.Parse("127.0.0.1:6379,password=xxx,connectTimeout=2000");

//the lock for singleton
private static readonly object Locker = new object();
```

```
//singleton
private static ConnectionMultiplexer redisConn;

//singleton
public static ConnectionMultiplexer getRedisConn()
{

if (redisConn == null)
{
lock (Locker)
{
if (redisConn == null || !redisConn.IsConnected)
{
redisConn = ConnectionMultiplexer.Connect(configurationOptions);
}
}
}
return redisConn;
}
```

说明：ConfigurationOptions 包含很多选项，例如 keepAlive、connectRetry、name，具体可以参考StackExchange.Redis.ConfigurationOptions。

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

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

下面给出5种数据结构的 demo，它们的 API 和原生略有不同，分别用 String、Hash、List、Set、SortedSet 开头代表5种数据结构。

string：

```
//set get
string strKey = "hello";
string strValue = "world";
bool setResult = db.StringSet(strKey, strValue);
Console.WriteLine("set " + strKey + " " + strValue + ", result is " + setResult);

//incr
string counterKey = "counter";
long counterValue = db.StringIncrement(counterKey);
Console.WriteLine("incr " + counterKey + ", result is " + counterValue);

//expire
db.KeyExpire(strKey, new TimeSpan(0, 0, 5));
Thread.Sleep(5 * 1000);
Console.WriteLine("expire " + strKey + ", after 5 seconds, value is " + db.StringGet(strKey));
```

```
//mset mget
KeyValuePair<RedisKey, RedisValue> kv1 = new KeyValuePair<RedisKey, RedisValue>("key1",
"value1");
KeyValuePair<RedisKey, RedisValue> kv2 = new KeyValuePair<RedisKey, RedisValue>("key2",
"value2");
db.StringSet(new KeyValuePair<RedisKey, RedisValue>[] {kv1,kv2});
RedisValue[] values = db.StringGet(new RedisKey[] {kv1.Key, kv2.Key});
Console.WriteLine("mget " + kv1.Key.ToString() + " " + kv2.Key.ToString() + ", result is " +
values[0] + "&&" + values[1]);
```

hash

```
string hashKey = "myhash";

//hset
db.HashSet(hashKey,"f1","v1");
db.HashSet(hashKey,"f2", "v2");
HashEntry[] values = db.HashGetAll(hashKey);

//hgetall
Console.Write("hgetall " + hashKey + ", result is");
for (int i = 0; i < values.Length;i++)
{
HashEntry hashEntry = values[i];
Console.Write(" " + hashEntry.Name.ToString() + " " + hashEntry.Value.ToString());
}
Console.WriteLine();
```

list

```
//list key
string listKey = "myList";

//rpush
db.ListRightPush(listKey, "a");
db.ListRightPush(listKey, "b");
db.ListRightPush(listKey, "c");

//lrange
RedisValue[] values = db.ListRange(listKey, 0, -1);

Console.Write("lrange " + listKey + " 0 -1, result is ");
for (int i = 0; i < values.Length; i++)
{
Console.Write(values[i] + " ");
}
Console.WriteLine();
```

set

```
//set key
string setKey = "mySet";

//sadd
db.SetAdd(setKey, "a");
db.SetAdd(setKey, "b");
db.SetAdd(setKey, "c");

//sismember
bool isContains = db.SetContains(setKey, "a");
Console.WriteLine("set " + setKey + " contains a is " + isContains );
```

sortedset

```
string sortedSetKey = "myZset";

//sadd
db.SortedSetAdd(sortedSetKey, "xiaoming", 85);
db.SortedSetAdd(sortedSetKey, "xiaohong", 100);
db.SortedSetAdd(sortedSetKey, "xiaofei", 62);
db.SortedSetAdd(sortedSetKey, "xiaotang", 73);

//zrevrangebyscore
RedisValue[] names = db.SortedSetRangeByRank(sortedSetKey, 0, 2, Order.Ascending);
Console.Write("zrevrangebyscore " + sortedSetKey + " 0 2, result is ");
for (int i = 0; i < names.Length; i++)
{
Console.Write(names[i] + " ");
}
Console.WriteLine();
```

Redis-cli 连接

云数据库 Redis 版仅支持阿里云内网访问，不支持外网访问，即只有在同节点的 ECS 上安装 Redis-cli 才能与云数据库建立连接并进行数据操作。

> **说明**：Redis-cli 是 Redis 原生的命令行工具，可以先下载安装 Redis 即可使用 Redis-cli。在 ECS 上安装 Redis 的命令请参考 Redis 官方网页。

Redis-cli 连接云数据库 Redis 版的命令如下：

```
redis-cli -h 实例连接地址 -a 密码
```

公网连接

前提条件

如果您需要从本地 PC 端访问 Redis 实例进行数据操作，可以通过在 ECS 上配置端口映射或者端口转发实现。但必须符合以下前提条件：

若 Redis 实例属于专有网络（VPC），ECS 必须与 Redis 实例属于同一个 VPC。

若 Redis 实例属于经典网络，ECS 必须与 Redis 实例属于同一节点（地域）。

若 Redis 实例开启了 IP 白名单，必须将 ECS 的内网地址加入白名单列表内。

ECS Windows 篇

目前云数据库 Redis 版需要通过 ECS 的内网进行连接访问，如果您需要本地通过公网访问云数据库 Redis 版，可以在 ECS Windows 云服务器中通过 netsh 进行端口映射实现。

登录 ECS Windows 服务器，在 cmd 执行以下命令：

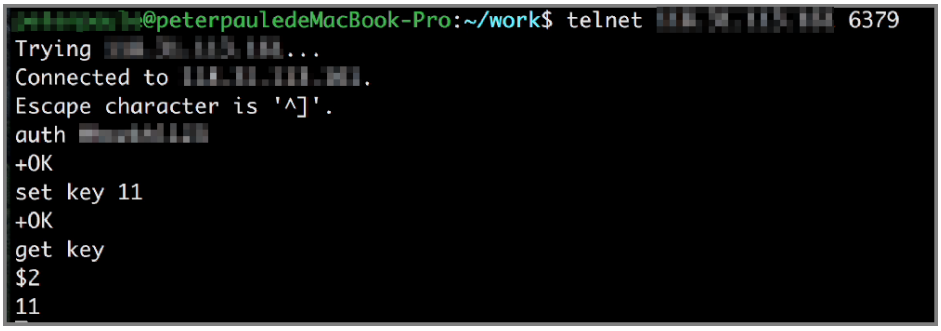
```
netsh interface portproxy add v4tov4 listenaddress=ECS服务器的私有IP地址 listenport=6379
connectaddress=云数据库Redis的连接地址 connectport=6379
```

其中：

netsh interface portproxy delete v4tov4 listenaddress=ECS公网服务器的公网IP地址 listenport=6379 可以删除不需要的映射。

netsh interface portproxy show all 可以查看当前服务器中存在的映射。

设置完成后在本地进行验证测试。



在本地通过 redis-cli 连接 ECS Windows 服务器。假设 ECS Windows 服务器的 IP 是 1.1.1.1，即 telnet 1.1.1.1 6379。

连接上 ECS windows 服务器后，输入连接 Redis 的密码：auth Redis的连接密码。

进行数据写入及查询验证。

通过上述步骤即可实现：您本地 PC 或服务器通过公网连接 ECS Windows 6379端口，对云数据库 Redis 进行访问。

注意：因 portproxy 由微软官方提供，未开源使用，您如果配置使用过程中遇到疑问，可参看 netsh 的 portproxy 使用说明或向微软官方咨询确认。或者您也可以考虑通过其他的方案实现，比如通过 portmap 配置代理映射。

ECS Linux 篇

目前云数据库 Redis 版需要通过 ECS 进行内网连接访问。如果您本地需要通过公网访问云数据库 Redis，可以在 ECS Linux 云服务器中安装 rinetd 进行转发实现。

在云服务器 ECS Linux 中安装 rinetd。

```
wget http://www.boutell.com/rinetd/http/rinetd.tar.gz&&tar -xvf rinetd.tar.gz&&cd rinetd
sed -i 's/65536/65535/g' rinetd.c (修改端口范围)
mkdir /usr/man&&make&&make install
```

注意：rinetd 安装包下载地址不确保下载可用性，您可以自行搜索安装包进行下载使用。

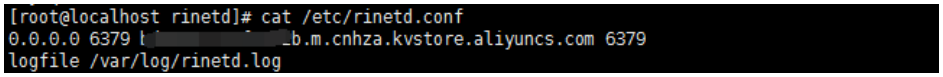
打开配置文件 rinetd.conf。

```
vi /etc/rinetd.conf
```

在配置文件中输入如下内容：

```
 0.0.0.0 6379 Redis 的链接地址 6379
logfile /var/log/rinetd.log
```

说明：您可以使用 cat /etc/rinetd.conf命令来检验配置文件是否修改正确。



执行如下命令启动 rinetd。

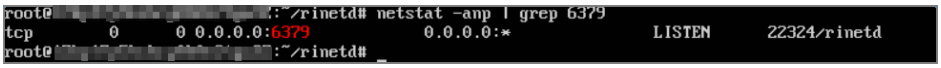
```
rinetd
```

注意

您可以通过 echo rinetd >>/etc/rc.local 将 rinetd 设置为自启动。

若遇到绑定报错，可以执行 pkill rinetd 结束进程，再执行 rinetd启动进程 rinetd。

rinetd 正常启动后， 执行netstat -anp | grep 6379 确认服务是否正常运行。

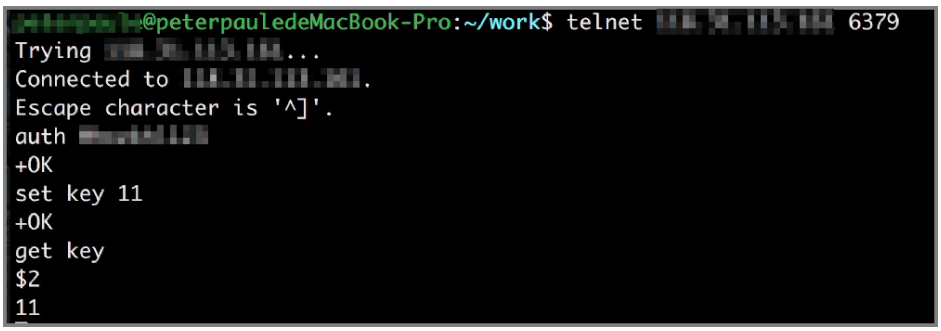


在本地进行验证测试。

您可以在本地通过 redis-cli 连接 ECS Linux 服务器后进行登录验证，比如安装了 rinetd 的服务器的 IP 是 1.1.1.1，即redis-cli -h 1.1.1.1 -a Redis的实例ID:Redis密码。或者通过 telent 连接 ECS Linux 服务器后进行操作验证。假设 ECS Linux 服务器的 IP 是 1.1.1.1，即 telnet 1.1.1.1 6379。

连接上 ECS Linux 服务器后，输入连接 Redis 的密码：auth Redis的连接密码。

进行数据写入及查询验证。



通过上述步骤即可实现：您本地的 PC 或服务器通过公网连接 ECS Linux 6379 端口，对云数据库 Redis 进行访问。

> **注意**：您可以通过该方案进行测试使用，因 rinetd 为开源软件，如在使用过程中存在疑问，您可以参看其官方文档或与 rinetd 官方进行联系确认。