

云数据库 Memcached

快速入门



腾讯云

【 版权声明 】

©2013–2026 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

快速入门

操作说明及示例

Java 教程及示例代码

PHP 教程及示例代码

Python 教程及示例代码

C++ 教程及示例代码

C Sharp .NET教程及示例代码

实现缓存 PHP session 全局变量

使用限制类

限制说明

兼容的协议说明

标准协议缺陷解决方案说明

访问说明

云数据库 Memcached 管理

数据导出导入

快速入门

操作说明及示例

Java 教程及示例代码

最近更新时间：2024-11-17 16:43:32

环境及依赖

环境：在腾讯云 CVM 上安装对应的 Java JDK，[下载地址](#)。

依赖：本例使用 Memcached-Java-Client.2.5.1 版本，[下载地址](#)，暂不支持 SpyMemcached 客户端。

使用步骤

在本地电脑新建 Java 工程，并导入下载好的 Memcached-Java-Client.2.5.1 源码。

编写源码并导出为 Jar 包。

将导出的 Jar 包上传到 CVM 服务器上并运行 `java -jar *.jar`，只有在 CVM 服务器上才能访问内网的 NoSQL 服务器。

代码示例 MemcachedDemo.java

```
/**
 * qcloud cmem java-memcached2.5.1-client demo
 * 使用方法：
 * 1. memcached2.5.1源码地址：
 * http://qzonestyle.gtimg.cn/qzone/vas/opensns/res/doc/memcached-java-
 * 2.5.1.zip
 * 2. 下载源码并引入到项目中
 * 3. 参照demo代码实现相应功能
 */

package com.qcloud.memcached.demo;

import com.danga.MemCached.MemCachedClient;
import com.danga.MemCached.SockIOPool;

import java.util.Date;

public class MemcachedDemo {
    public static void main(String[] args){
```

```
//管理中心，单击“NoSQL高速存储”，在NoSQL高速存储“管理视图”，可以看到系统分配的IP:Port
//需要在内网IP上访问，不需要账号密码
final String ip = "**.**.*.*.*.*";
final String port = "****";

System.out.println("start test qcloud cmem - " + ip + ":" + port);

MemCachedClient memcachedClient = null;

try{
    //设置缓存服务器列表，当使用分布式缓存的时，可以指定多个缓存服务器
    String[] servers = {ip + ":" + port};

    //创建Socket连接池实例
    SockIOPool pool = SockIOPool.getInstance();
    pool.setServers(servers); //设置连接池可用的cache服务器列表
    pool.setFailover(true); //设置容错开关
    pool.setInitConn(10); //设置开始时每个cache服务器的可用连接数
    pool.setMinConn(5); //设置每个服务器最少可用连接数
    pool.setMaxConn(250); //设置每个服务器最大可用连接数
    pool.setMaintSleep(30); //设置连接池维护线程的睡眠时间
    pool.setNagle(false); //设置是否使用Nagle算法，因为我们的通讯数据量通常都比较大（相对TCP控制数据）而且要求响应及时，因此该值需要设置为false（默认是true）
    pool.setSocketTO(3000); //设置socket的读取等待超时值
    pool.setAliveCheck(true); //设置连接心跳监测开关
    pool.initialize();

    memcachedClient = new MemCachedClient();

    //将数据存入缓存
    memcachedClient.set("cmem", "qcloud cmem service");

    //将数据存入缓存，并设置失效时间
    Date date = new Date(1000);
    memcachedClient.set("test_expire", "test_expire_value", date);
}
```

```
//获取缓存数据
System.out.println("get: cmem = " +
memcachedClient.get("cmem"));
System.out.println("get: test_expire = " +
memcachedClient.get("test_expire"));

//向cmem存入多个数据
for(int i = 0; i < 100; i++){
    String key = "key-" + i;
    String value = "value-" + i;
    memcachedClient.set(key, value);
}

System.out.println("set 操作完成");
} catch (Exception e) {
    e.printStackTrace();
}
if (memcachedClient != null) {
    memcachedClient = null;
}
}
}
```

PHP 教程及示例代码

最近更新时间：2022-11-09 11:39:52

环境及依赖

环境：在腾讯云 CVM 上安装对应的 [Apache](#)、[PHP](#)，建议使用较新版本 Apache2.0+、PHP Version 5.3+。

依赖：安装 [PHP-Memcache-3.0.6+](#) 或者 [PHP-Memcached-1.0.2+](#) 扩展。

[PHP-memcache GitHub 源码](#) [PHP-memcached GitHub 源码](#)

使用步骤

在腾讯云 CVM 上部署好 Apache+PHP 环境并安装好 PHP-Memcache 或者 PHP-Memcached 扩展。编写测试代码并运行。

代码示例 PHP-Memcache

[GitHub 代码参考](#)

```
<?php
$cache = new Memcache; //新建一个memcache连接实例
$cache->connect ('***.***.***.***', ****); //连接到指定的cmem服务器IP和端口
$key = "php-key";
$cache->set ($key, "value1". time()); //向cmem写入一个key
$val = $cache->get ($key); //向cmem获取一个key
var_dump ($val);
$cache->close();
?>
```

代码示例 PHP-Memcached

[GitHub 代码参考](#)

```
<?php
$memcached = new Memcached; //新建一个memcache连接实例
$memcached->setOption (Memcached::OPT_COMPRESSION, false); //关闭压缩功能
$memcached->setOption (Memcached::OPT_BINARY_PROTOCOL, false); //关闭二进制
协议
$memcached->addServer ("***.***.***.***", ****); //添加cmem服务器，指定cmem服
务器IP和端口
```

```
$key = "php-key";  
$memcached->set($key, "value-1-".time()); //向cmem写入一个key  
$val = $memcached->get($key); //向cmem获取一个key  
var_dump ($val);  
$memcached->quit();  
?>
```

Python 教程及示例代码

最近更新时间：2022-11-09 11:39:52

环境及依赖

环境：在腾讯云 CVM 上安装对应的 Python，[下载地址](#)。

依赖：本例使用 Python-memcached 1.5.4 版本，在 CVM 上安装此客户端，[下载地址](#)。

使用步骤

在 CVM 上部署好 Python 环境及 Python-memcached 客户端。

编写测试代码并运行。

代码示例 python-memcached-demo.py

将代码中 *** 号替换为您的 IP:Port，在管理中心，单击 **NoSQL高速存储**，在 **NoSQL 高速存储管理视图**，可以看到系统分配的 IP:Port。

```
#!/usr/bin/env python
import memcache
mc = memcache.Client(['***.***.***.***:***'], debug=0)
mc.set("python-key", "qcloud NoSQL Server")
value = mc.get("python-key")
print value
```

C++ 教程及示例代码

最近更新时间：2022-11-09 11:39:52

环境及依赖

[下载 libmemcached](#)。

安装 libmemcached 客户端。

将 libmemcached.so 文件所在目录加入到变量 LD_LIBRARY_PATH 中，不同系统路径可能不一样，请查看自己的安装目录并替换。

```
tar -xvf libmemcached-1.0.18.tar.gz
cd libmemcached-1.0.18
./configure
sudo make
sudo make install
#配置path
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

使用步骤

编写测试代码 memcachedDemo.cpp。

```
编写测试代码 memcachedDemo.cpp
#include<iostream>
#include <string.h>
#include <libmemcached/memcached.h>

using namespace std;

int main(int argc, char *argv[])
{
    memcached_st *client = NULL;
    memcached_return cache_return;
    memcached_server_st *server = NULL;

    client = memcached_create(NULL);
    server = memcached_server_list_append(server, "***.***.***.***", ***,
    &cache_return); //管理中心，单击“NoSQL高速存储”，在NoSQL高速存储“管理视图”，可以看
    到系统分配的IP:Port
```

```
cache_return = memcached_server_push(client, server);

if(MEMCACHED_SUCCESS != cache_return){
cout<<"memcached server push failed! cache return:"<<cache_return<<endl;
return -1;
}

string key = "cpp_key";
string val = "cpp_value";
size_t key_len = key.length();
size_t val_len = val.length();
int expiration = 0;
uint32_t flags = 0;
cache_return = memcached_set(client, key.c_str(), key_len, val.c_str(),
val_len, expiration, flags);
if(MEMCACHED_SUCCESS == cache_return){
cout<<"set success"<<endl;
}else{
cout<<"set failed! cache return:"<<cache_return<<endl;
}

size_t value_length;
char* getVal = memcached_get(client, key.c_str(), key_len,
&value_length, &flags, &cache_return);
if(MEMCACHED_SUCCESS == cache_return){
cout<<"get success, value = "<<getVal<<endl;
}else{
cout<<"get failed, cache return:"<<cache_return<<endl;
}
return 0;
}
```

编译

```
g++ -g -Wall -std=c++0x memcachedDemo.cpp -lmemcached -lpthread -o
memcached
```

运行

```
./memcached
set success
get success, value = cpp_value
```

C Sharp .NET教程及示例代码

最近更新时间：2024-11-17 16:43:32

本文介绍两种 C# 客户端的使用方法 [.NET memcached client library](#) 和 [EnyimMemcached](#)。

使用 .NET memcached client library

环境和依赖

下载并解压 memcacheddotnet_clientlib-1.1.5.zip。

拷贝 memcached\trunk\clientlib\src\clientlib\bin\2.0\Release 目录下的四个 dll 文件到 .NET 工程，并保持其在同一目录下。

在 .NET 工程中引用 Memcached.ClientLibrary.dll。

代码示例 .NET memcached client library

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Memcached.ClientLibrary;

namespace TestMemcachedApp
{
    class Program
    {
        [STAThread]
        public static void Main(String[] args)
        {
            string[] serverlist = { "***.***.***.***:*" }; //服务器可以是多个，server的构成形式是IP:PORT（如：127.0.0.1:11211）

            //初始化池
            SockIOPool pool = SockIOPool.GetInstance();
            pool.SetServers(serverlist); //设置连接池可用的cache服务器列表
            pool.InitConnections = 3; //初始连接数
            pool.MinConnections = 3; //最小连接数
            pool.MaxConnections = 5; //最大连接数
            pool.SocketConnectTimeout = 1000; //设置连接的套接字超时
            pool.SocketTimeout = 3000; //设置套接字超时读取
        }
    }
}
```

```
pool.MaintenanceSleep = 30; //设置维护线程运行的睡眠时间。如果设置为0, 那么维护线程将不会启动, 30就是每隔30秒醒来一次
```

```
//获取或设置池的故障标志。
```

```
//如果这个标志被设置为true则socket连接失败, 将试图从另一台服务器返回一个套接字 (如果存在的话)
```

```
//如果设置为false, 则得到一个套接字 (如果存在的话)。否则返回NULL, 如果它无法连接到请求的服务器
```

```
pool.Failover = true;
```

```
pool.Nagle = false; //如果为false, 对所有创建的套接字关闭Nagle的算法
```

```
pool.Initialize();
```

```
// 获得客户端实例
```

```
MemcachedClient mc = new MemcachedClient();
```

```
mc.EnableCompression = false;
```

```
mc.Set("cSharp_key", "cSharp_value"); //存储数据到缓存服务器, 这里将字符串"cSharp_value"缓存, key是"cSharp_key"
```

```
if (mc.KeyExists("cSharp_key")) //测试缓存存在key为test的项目
```

```
{
```

```
    Console.WriteLine("cSharp_key is Exists");
```

```
    Console.WriteLine(mc.Get("test").ToString()); //在缓存中
```

```
获取key为cSharp_key的项目
```

```
}
```

```
else
```

```
{
```

```
    Console.WriteLine("cSharp_key not Exists");
```

```
}
```

```
mc.Delete("cSharp_key"); //移除缓存中key为test的项目
```

```
if (mc.KeyExists("cSharp_key"))
```

```
{
```

```
    Console.WriteLine("cSharp_key is Exists");
```

```
    Console.WriteLine(mc.Get("cSharp_key").ToString());
```

```
}
```

```
else
{
    Console.WriteLine("cSharp_key not Exists");
}
Console.ReadLine();

SockIOPool.GetInstance().Shutdown(); //关闭池, 关闭sockets
}
}
}
```

使用 EnyimMemcached

环境和依赖

下载后先修改 build\CommonProperties.targets 文件, 注释掉其中的程序集签名选项, 避免调用 dll 时编译不通过的问题。

使用 Visual Studio 直接打开解决方案, 总共六个项目, 只需右键选中 Enyim.Caching 项目单独生成 dll 即可 (生成的 dll 文件名为 Enyim.Caching.dll, 最好为 Release 版)。

新建 .Net 客户端项目, 并在项目中引用之前生成的dll文件。

编写测试代码并运行。

代码示例 EnyimMemcached

```
using System;
using System.Net;
using Enyim.Caching;
using Enyim.Caching.Configuration;
using Enyim.Caching.Memcached;

namespace DemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            MemcachedClientConfiguration config = new
MemcachedClientConfiguration();
            IPAddress ip =
IPAddress.Parse(Dns.GetHostEntry("***.***.***.***").AddressList[0].ToStr
ing()); //***.***.***.***为cmem控制台列表中的ip地址
```

```
config.Servers.Add(new IPEndPoint(ip, ****));//****为cmem控制  
台列表中的端口  
config.Protocol = MemcachedProtocol.Binary;//使用二进制协议  
memConfig.SocketPool.MinPoolSize = 5;//连接池最小连接数  
memConfig.SocketPool.MaxPoolSize = 200;//连接池最大连接数  
  
var mc = new MemcachedClient(config);  
  
//向cmem写入数据  
mc.Store(StoreMode.Set, "csharp_key", "csharp_value");  
  
//从cmem读取一条数据  
Console.WriteLine(mc.Get("csharp_key"));  
  
//从cmem读取多条数据 only 1.2.4 supports it (Windows version is  
at 1.2.1)  
List<string> keys = new List<string>();  
  
for (int i = 1; i < 100; i++)  
{  
    string k = "aaaa" + i + "--" + (i * 2);  
    keys.Add(k);  
  
    mc.Store(StoreMode.Set, k, i);  
}  
  
IDictionary<string, ulong> cas;  
IDictionary<string, object> retvals = mc.Get(keys, out cas);  
}  
}  
}
```

实现缓存 PHP session 全局变量

最近更新时间：2022-11-09 11:39:52

使用场景

Session 是 Web 程序中常用的功能，默认情况下其数据是以文件方式存储，对大访问量的场景，其处理能力较低。Memcache 是高性能的基于内存的 Key=>Value 存储系统，能大大改善处理 session 的能力。

使用 PHP-Memcache 扩展的实现方法

可以使用修改配置文件或者在程序中实现两种方式。

1. 修改 php.ini 配置文件实现。

修改 session 存储方式。

```
session.save_handler = memcache
```

修改 session 存储地址，*** 号替换为您的 IP:Port，在 [Memcached 控制台](#) 可以查看系统分配的 IP:Port。

```
session.save_path = "tcp://***.***.***.***:****"
```

设置一个合理时间，只缓存热点数据。

```
session.gc_maxlifetime = 1500
```

2. 代码中直接设置，可参考 [memcache](#)。

```
ini_set("session.save_handler", "memcache");
ini_set("session.save_path", "tcp://***.***.***.***:****");
ini_set("session.gc_maxlifetime", 1500);
```

使用 PHP-Memcached 扩展的实现方法

可以使用修改配置文件或者在程序中实现两种方式，与 memcache 不同在于其 IP 前没有 tcp://。

1. 修改 php.ini 配置文件实现。

修改 session 存储方式。

```
session.save_handler = memcached
```

修改 session 存储地址，*** 号替换为您的 IP:Port，在 [Memcached 控制台](#) 可以查看系统分配的 IP:Port。

```
session.save_path = "***.***.***.***:****"
```

设置一个合理时间，只缓存热点数据。

```
session.gc_maxlifetime = 1500
```

2. 代码中直接设置，可参考 [memcache](#)。

```
ini_set("session.save_handler", "memcached");
```

```
ini_set("session.save_path","**.*.*.*.*:*");  
ini_set("session.gc_maxlifetime",1500);
```

使用限制类 限制说明

最近更新时间：2022-11-09 11:39:53

数据淘汰

云数据库 Memcached 为存储产品，不支持数据淘汰，即数据写满后不会自动删除最老的数据。用户必须自行设置 expire 过期时间，并在云数据库 Memcached 中开启 expire 过期删除功能。

key 和 value 的长度限制

考虑到数据拷贝的延时，云数据库 Memcached 对 key 和 value 的长度做出了限制：key 长度限制为不超过 10K；value 长度限制为不超过 1M（Memcached 开源协议限制为 1M），必要时建议压缩 value。

性能限制

云数据库 Memcached 支持最大访问能力为 10000 次/秒/GB，如果超出服务可能会变慢。这里的容量 GB 是指实际分配的容量，不是最大容量上限。

例如：实际分配的容量为 50GB，这 50GB 的数据支持每秒访问次数是 $50 * 10000 = 500000$ 次。

如果超过 10000 次/秒/GB，系统会丢弃超出的请求。

当前提供的 10000 次/秒/GB 访问量已经能够满足大部分应用的访问需求。如果访问量超过 10000 次/秒/GB，请 [提交工单](#) 进行接口/端口扩容申请。

客户端以及连接数访问限制

1. 云数据库 Memcached 是分布式系统，不能支持异步客户端，例如 Spymemcached。
也就是说同一个连接上连续发送请求 A、B、C 后，其回复顺序是不确定的，任何基于顺序的逻辑都不能正常使用，因此必须使用一问一答的同步模式开源客户端。
2. 云数据库 Memcached 的 socket 连接有超时限制。如果从上一次访问后的 180 秒内，客户端没有访问请求，则连接会自动断开。因此客户端每 180 秒内至少要发送一次访问请求。
3. 到一台 Memcached 的 socket 连接数是有上限的，但此上限远大于客户端能够创建的临时端口数，因此使用时无需关注。

协议限制

1. 云数据库 Memcached 支持 Memcached 协议，详见 [兼容的协议说明](#)，因此支持文本和二进制协议。
2. 理论上二进制解码速度快于文本，但实际优势微乎其微。且目前大部分用户还是使用文本协议，比较简单稳定。使用二进制的比较少。
3. Memcached 的标准协议存在部分缺陷，用户需要特别注意。详见 [标准协议缺陷解决方案说明](#)。

安全和容灾限制

云数据库 Memcached 提供主从热备，通过定期镜像和实时流水同步来备份。如果 Memcached 掉电，在极端情况下会损失短时间未落盘的数据，但是概率极小。

容量增长限制

为防止业务异常使实例容量增长过快导致您的费用损失，我们会巡检出容量增长过快的实例，超过20GB开始纳入巡检范围，当日的数据增量超过20%后，实例停止自动扩容，如果您有实例超过20GB，日增量超过20%的扩容需求，请 [提交工单](#) 联系我们。

兼容的协议说明

最近更新时间：2022-12-09 22:19:42

- 1.云数据库 Memcached 基于标准的 Memcached 协议以及接口，开发者可参见 [Memcached 官网](#) 的介绍了解 Memcached。
- 2.在 Memcached 文本协议清单中，列出了云数据库 Memcached 支持的命令列表，请在下表中下载。

文档名称	文件大小	说明
Memcached 文本协议清单	16.1 K	增加 gets 接口使用说明，最多支持255个键值。
Memcached 文本扩展协议	251 K	增加两个扩展的命令 get_ext, gets_ext, 使客户端可以根据返回码判断数据是否存在，并找到未返回数据的原因。

- 3.Memcached 的标准协议存在部分缺陷，开发者需要特别注意。详见 [标准协议缺陷解决方案说明](#)。

标准协议缺陷解决方案说明

最近更新时间：2022-11-09 11:39:53

背景描述

Memcached 标准协议存在部分缺陷，其 Get 操作没有设计返回码，不能明确说明拉取数据失败的具体情况，所以 Memcached API 返回 NO_DATA 时，有可能是网络原因造成的。

注意

使用如下流程将造成用户数据初始化，须谨慎操作：

```
if ( NO_DATA ) InitData();
```

解决方案

1. 为解决上述问题，开发者在存储数据时，需区分 add 和 set 操作：

add 操作：按照相应的 <key> 存储数据，只有在该数据不存在时才保存该数据。

set 操作：按照相应的 <key> 存储数据，无论数据是否存在。

注意

在新增数据时必须使用 add 接口，不能使用 set 接口，否则会造成用户数据重置，给业务带来损失。

2. 云数据库 Memcached 提供了 Memcached 文本扩展协议,详见 [兼容的协议说明](#)，增加两个扩展的命令 get_ext，gets_ext，使客户端可以根据返回码判断数据是否存在。这样可以避免网络和设备故障时 get 不到数据而导致用户数据被误初始化。需要注意的是，使用扩展协议需要用户自己修改 API 来支持。

案例

1. 某业务 A 没有区分新增和修改，在变更接入层时造成部分连接断连，影响查询，导致初始化了部分用户数据。

2. 某业务 B 没有区分新增和修改，在云数据库 Memcached 设备故障时，查询不到数据，导致初始化了部分用户数据。

3. 某业务 C 经常反映部分数据重置的情况，经查证是使用 set 增加数据，在网络闪断时导致重置数据。

以上案例均给业务带来极大的伤害和损失，均停止服务数小时进行回滚。因此请开发者务必注意在新增数据时必须使用 add 接口。

访问说明

最近更新时间：2022-11-09 11:39:53

关于客户端

- 使用同步模式的开源客户端来访问云数据库 Memcached。

云数据库 Memcached 是分布式系统，不能支持异步客户端，请使用同步模式的开源客户端。

(1) 如果需要使用 PHP 语言客户端，则推荐使用 memcache 扩展2.2.6: [memcache-2.2.6](#)，已证实这个扩展是使用同步 I/O 模型实现。

(2) 如果需要使用 Java 语言客户端，则推荐使用如下客户端: [memcached-java-2.5.1](#)。

- 云数据库 Memcached 的 socket 连接超时限制。

如果从上一次访问后的180秒内，客户端没有访问请求，则连接会自动断开。因此客户端每180秒内至少要发送一次访问请求。目前开源的客户端均不检查连接活跃性，需要用户自行处理。

访问云数据库 Memcached 服务

1. 获取要访问的云数据库 Memcached 的IP:Port。

登录 [Memcached 控制台](#)，在实例列表，可以看到系统分配的 IP:Port。该 IP:Port 在访问 Memcached 服务时要用到。

2. 进行代码开发，实现 Memcached 服务的访问。

代码开发请参考 [开发手册](#)。

云数据库 Memcached 管理

最近更新时间：2026-01-16 16:31:01

说明：

- 云数据库 Memcached 已于2025年7月31日正式停止售卖，用户无法再购买该产品。对于已购买的现有实例，其使用不受任何影响，可继续正常运行和提供服务。
- 为满足用户需求，[云数据库 Redis® 的 Memcached 引擎](#) 将作为其替代方案。该引擎兼容 Memcached 协议并提供类似的功能与服务体验。建议新用户及有扩容需求的现有用户使用该产品进行部署。

实例扩容

云数据库 Memcached 的扩容包括存储扩容、接口扩容和端口扩容。

存储扩容

云数据库 Memcached 会自动为每个实例每日预留约20%的空间作为数据增长 buffer。例如，实例的使用空间为80GB，则会分配96GB作为实例的占用空间。如果实例的数据日增长量超过20%，请 [提交工单](#) 进行存储扩容申请。云数据库 Memcached 扩容过程是数据搬迁过程，不会影响业务访问。

接口/端口扩容

请 [提交工单](#)，填写接口/端口扩容进行申请。

实例缩容

实例缩容指的是减少实例的占用空间，即存储缩容。因为需要预留缓冲空间，缩容后实例使用率不会超过80%。实例缩容的最小粒度是1GB，如果缩容会造成使用率超过80%，则不能进行缩容。

目前云数据库 Memcached 的实例暂不支持自动缩容，如实例需要缩容则可提交工单申请，之后需运维人员操作缩容。

在申请缩容之前，计费时仍然会按照原占用空间（包括在原使用空间的基础上自动扩容的缓冲空间）的峰值进行计算。

数据清理

云数据库 Memcached 支持通过控制台手动清理实例数据，单个实例每天只能清理累计50GB的占用空间。

注意：

数据被清空后，不可以再恢复，请在清空前确认实例中的数据已经备份或不再使用。

登录 [云数据库 Memcached 控制台](#)，选择需要清空的实例，在操作列选择更多 > 清空，确认清空后，后台开始清空操作。清空完成后，页面会提示清空成功。

ID/实例名	监控/状态	可用区	网络信息	创建时间 ↓	内网地址	计费模式	所属项目	操作
	运行中	广州三区	基础网络	2018-10-23 ...		按量计费	默认项目	开启淘汰 更多
	运行中	广州三区	基础网络	2018-10-23 ...		按量计费	默认项目	清空 销毁 编辑标签

实例销毁

注意：

实例销毁后，不可再恢复，请在销毁前确认实例中的数据已经备份或不再使用。

在 Memcached 实例列表，选择需要销毁的实例，在操作列选择**更多 > 销毁**，后台开始清理实例数据并删除表。

ID/实例名	监控/状态	可用区	网络信息	创建时间 ↓	内网地址	计费模式	所属项目	操作
	运行中	广州三区	基础网络	2018-10-23 ...		按量计费	默认项目	开启淘汰 更多
	运行中	广州三区	基础网络	2018-10-23 ...		按量计费	默认项目	清空 销毁 编辑标签

实例淘汰

云数据库 Memcached 支持通过开启控制台的淘汰功能来每天定期自动清理实例数据，清理后的数据将无法恢复。

1. 在 Memcached 实例列表，选择所需实例，单击**开启淘汰**打开淘汰开关。

ID/实例名	监控/状态	可用区	网络信息	创建时间 ↓	内网地址	计费模式	所属项目	操作
	运行中	广州三区	基础网络	2018-10-23 ...		按量计费	默认项目	开启淘汰 更多

2. 开启淘汰功能后，需要在代码里设置 key 的有效期，具体请参考各语言 Memcached 设置方法。

说明：

- 开启淘汰功能前设置的 key 是不会自动过期的。
- expire 值与传统 Memcached 用法略有不同，范围为(0,2592000)，超出则无效。

实例监控

在 Memcached 实例列表，单击如下监控图标，或单击实例名进入**实例监控**页面可查看实例监控信息。

ID/实例名	监控/状态	可用区	网络信息	创建时间 ↓	内网地址	计费模式	所属项目	操作
	运行中	广州三区	基础网络	2018-10-23 ...		按量计费	默认项目	开启淘汰 更多

数据回档

请 [提交工单](#) 联系我们。

连接诊断

请参见 [Memcached 连接诊断](#)。

数据导出导入

最近更新时间：2022-11-09 11:39:53

由于系统调整的原因，云数据库 Memcached 的数据导入工具目前无法使用。如有需要，请通过 [提交工单](#) 联系我们。