

云数据库 Memcached

快速入门

产品文档



腾讯云

【版权声明】

©2013-2018 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【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。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。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模式的承诺或保证。

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

数据导出导入

云缓存Memcached管理

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：2018-06-07 18:17:57

1 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）

2 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包上传到腾讯云CVM服务器上并运行java -jar *jar. (注意: 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. 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
    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：2017-09-06 16:07:23

1 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](#)]

2 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.

3 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();
?>
```

4 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：2017-06-30 15:06:22

1 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, 在腾讯云CVM上安装此客户端 [[下载地址](#)]

2 使用步骤

在腾讯云CVM上部署好python环境及python-memcached客户端.

编写测试代码并运行.

3 代码示例 python-memcached-demo.py

将代码中 *号替换为你的IP:Port, 在管理中心, 点击“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：2018-06-08 10:26:36

1 环境及依赖

下载libmemcached [[libmemcached-1.0.18.tar.gz](#)].

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.

#安装

```
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
```

2 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 “No
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：2017-06-30 15:05:27

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

1 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.

            //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
```

```
//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，

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
}
}
}
```

2 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].ToString());//***.*?
            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：2018-06-08 09:41:30

1 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。

2 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，在管理中心，单击“云缓存Memcached”，在云缓存Memcached“管理视图”，可以看到系统分配的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，[可参考这里](#)。

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

```
ini_set("session.save_path","tcp://...:...");
```

```
ini_set("session.gc_maxlifetime",1500);
```

3 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，在管理中心，单击“云缓存Memcached”，在云缓存Memcached“管理视图”，可以看到系统分配的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, [可参考这里](#)。

```
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：2018-05-24 16:55:33

1 数据淘汰

云缓存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，并在云缓存Memcached中开启expire过期删除功能。

2 key和value的长度限制

考虑到数据拷贝的延时，云缓存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。

3 性能限制

云缓存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（G）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50G，这50G的数据支持每秒访问次数是 $50 \times 10000 = 500000$ 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，请填写工单接口/端口扩容进行申请。

4 客户端以及连接数访问限制

（1）云缓存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）云缓存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) 到一台云缓存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。

5 协议限制

1. 云缓存Memcached支持memcached协议（详见：[云缓存Memcached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：[Memcached标准协议缺陷解决方案说明](#)。

6 安全和容灾限制

云缓存Memcached提供主从热备，通过定期镜像和实时流水同步来备份。如果云缓存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。

7 容量增长限制

为了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 的扩容需求，请提工单联系我们。

兼容的协议说明

最近更新时间：2017-11-10 11:14:56

1.云缓存Memcached基于标准的Memcached协议以及接口，我们假定开发者在使用云缓存Memcached服务的时候，已经对Memcached有了一定的了解。

开发者可参看Memcached官网中的介绍以了解Memcached：

<http://memcached.org/>

Memcached 开源扩展库载及API说明（C++/PHP/Java/Python/Ruby/Perl...等多语言扩展库）：

<https://code.google.com/archive/p/memcached/downloads>

2.在Memcached文本协议清单中，列出了云缓存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：[Memcached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：2017-06-30 15:10:16

1 背景描述

Memcached的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，不能完全信任。

使用如下流程将是非常危险的，将造成用户数据初始化：

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

2 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：按照相应的存储数据，无论数据是否存在。

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）云缓存Memcached还提供了Memcached文本扩展协议（详见[云缓存Memcached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。

3 真实案例

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，在云缓存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：2018-08-14 10:19:16

注：访问云缓存Memcached前，请务必了解[云缓存Memcached\(原NoSQL高速存储\)使用限制](#)。

1 关于客户端

1. 使用同步模式的开源客户端来访问云缓存Memcached服务

云缓存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模型实现的。

(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](#)。

(3) 如果需要使用其他语言的客户端，请参考[这里](#)。

2 云缓存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。

2 访问云缓存Memcached服务

1. 获取要访问的云缓存Memcached的IP:Port

登录腾讯云，进入控制台，单击“云缓存Memcached”，在云缓存Memcached页面，可以看到系统分配的IP:Port。该IP:Port在访问云缓存Memcached服务时要用到。

云缓存Memcached 全部项目 广州 全部网络 请输入名称或访问地址(换行分隔) + 新建 退还选中表 分配至项目 <table><thead><tr><th>☐ 实例名称</th><th>实例ID</th><th>可用区</th><th>所属网络</th><th>访问地址</th><th>创建时间</th><th>实例类型</th><th>expire过期</th><th colspan="2">操作</th></tr></thead><tbody><tr><td>☒ cmem206984</td><td>104020373</td><td>广州二区</td><td>基础网络</td><td>10.66.95.54:9101</td><td>2018-08-13 18:20:02</td><td>C型</td><td> 未启用 </td><td>管理</td><td>清空</td></tr></tbody></table>										☐ 实例名称	实例ID	可用区	所属网络	访问地址	创建时间	实例类型	expire过期	操作		☒ cmem206984	104020373	广州二区	基础网络	10.66.95.54:9101	2018-08-13 18:20:02	C型	未启用	管理	清空
☐ 实例名称	实例ID	可用区	所属网络	访问地址	创建时间	实例类型	expire过期	操作																					
☒ cmem206984	104020373	广州二区	基础网络	10.66.95.54:9101	2018-08-13 18:20:02	C型	未启用	管理	清空																				

1. 进行代码开发，实现云缓存Memcached服务的访问

代码开发请参考：[开发手册](#)（以php为例子）。

数据导出导入

最近更新时间：2018-06-08 09:42:45

由于系统调整的原因，云缓存Memcached的数据导入工具目前无法使用。如有需要，请通过腾讯云[工单系统](#)联系我们。

云缓存Memcached管理

最近更新时间：2018-08-14 10:31:43

1 表扩容

云缓存Memcached的扩容包括存储扩容、接口扩容和端口扩容。

存储扩容：

云缓存Memcached会自动为每个表每日预留约20%的空间作为数据增长buffer。例如表的使用空间为80G，则会分配96G作为表的占用空间。如果表的数据日增长量超过20%，请填写工单存储扩容进行申请。云缓存Memcached扩容过程是数据搬迁过程，不会影响命中率。

接口/端口扩容：

请填写工单接口/端口扩容进行申请。

2 表缩容

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

目前云缓存Memcached的表暂不支持自动缩容，如表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。

3 自助清理数据

注意：

- （1）数据被清空后，不可以再恢复，请在清空前确认表中的数据已经备份或不再使用。
- （2）单个应用每天只能清理累计50GB的表占用空间。如果超过50GB，请提交工单联系技术支持。

在控制台的云缓存Memcached页面，在需要清空的的表后单击“清空”按钮，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。



云缓存Memcached 全部项目 广州 全部网络

+ 新建 退还选中表 分配至项目

实例名称	实例ID	可用区	所属网络	访问地址	创建时间	实例类型	expire过期	操作
☒ cmem206984	104020373	广州二区	基础网络	10.66.95.54:9101	2018-08-13 18:20:02	C型	☐ 未启用	管理 清空

4 表退还

注意：

表退还后，不可以再恢复，请在退还前确认表中的数据已经备份或不再使用。

进入控制台的云缓存Memcached页面，勾选需要退还的表，然后单击“退还选中表”按钮，单击“提交退还”按钮后，后台开始清理表数据并删除表。



云缓存Memcached 全部项目 广州 全部网络

+ 新建 退还选中表 分配至项目

实例名称	实例ID	可用区	所属网络	访问地址	创建时间	实例类型	expire过期	操作
☒ cmem206984	104020373	广州二区	基础网络	10.66.95.54:9101	2018-08-13 18:20:02	C型	☐ 未启用	管理 清空

5 开启expire过期删除

1、要使用expire功能，首先需要在腾讯云管理控制台打开对应CMEM实例的expire开关



云缓存Memcached 全部项目 华南地区（广州） 全部网络

+ 新建 退还选中表 分配至项目

实例名称	所属网络	访问地址	实例类型	expire过期	操作
☐ cmem127876	基础网络	10.66.158.40:9101	C型	☐ 未启用	管理 清空

2、开启expire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。

3、注意，开启expire功能前设置的key是不会自动过期的。

6 查看监控信息

在NoSQL管理页面单击左侧监控视图按钮进入监控信息页面。

指标说明详见：[云缓存Memcached监控指标说明](#)

7 运营数据查看API

详见：[云缓存Memcached运营数据查看API](#)

8 数据回档

请提交工单联系我们。

9 连接诊断

详见：[云缓存Memcached连接诊断](#)