云数据库 Memcache 版

快速入门

快速入门

文档目的

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

目标读者

首次购买 Memcache 实例的用户

想要了解如何连接 Memcache 实例的用户

快速入门流程图

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

从新购实例到可以开始使用实例，您需要完成如下操作：

Memcache 管理控制台是用于管理 Memcache 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。

Memcache 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 Memcache 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 Memcache 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 Memcache 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 Memcache 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**、**实例规格**、**创建时间**、**付费方式**、**网络类型**等信息。

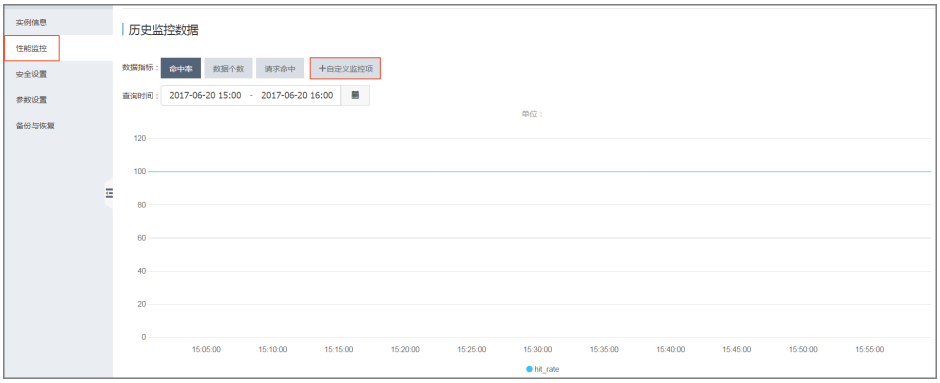
> **注意**：**已用内存及配额**信息是由底层系统根据采集信息进行的一个离线汇总，所以有一个时间延时，这个延时会在10分钟左右。

可维护时间段

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 Memcache 的历史性能数据，可以查看到不同的监控项。



单击**自定义监控项**之后可以添加到不同的监控项。详细信息请参见**性能监控**。

安全设置

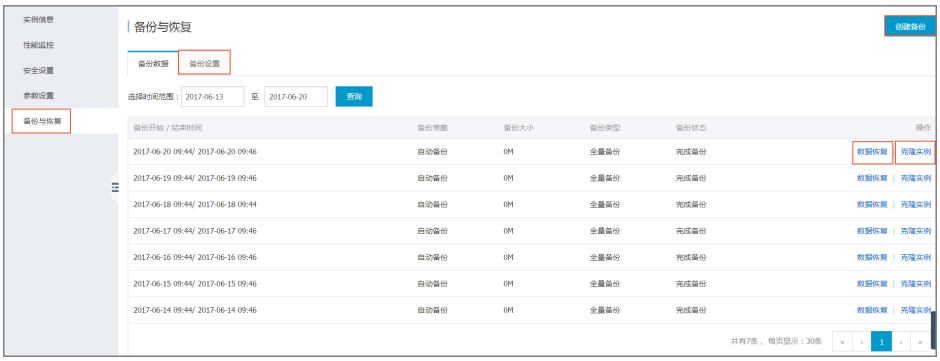
选择左侧导航栏的**安全设置**，您可以添加允许访问 Memcache 实例的 IP 地址。详细操作请参见**安全设置**。

参数设置

您可以在**参数设置**页面设置 Memcache 的数据逐出策略。详细操作请参见**参数设置**。

备份与恢复

您可以在**备份与恢复**页面进行创建备份、数据恢复、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**。



项目	限制说明
数据类型	云数据库 Memcache 版仅支持 Key-Value 格式的数据，不支持 array、map、list 等复杂类型的数据。
数据可靠性	云数据库 Memcache 版的数据存储在内存中，服务并不保证缓存数据不会丢失，有强一致性要求的数据不适合存储。
数据大小	云数据库 Memcache 版支持的单条缓存数据的 Key 最大不超过 1 KB，Value 最大不超过 1 MB，过大的数据不适合存储。
事务支持	云数据库 Memcache 版不支持事务，有事务性要求的数据不适合写入，而应该直接写入数据库。
使用场景	当数据访问分布比较均匀，数据没有明显的冷热分别时，大量的访问请求在云数据库 Memcache 版无法命中，使用云数据库 Memcache 版作为数据库缓存的效果不明显。在选择缓存时，需要充分考虑到业务模式对数据访问的要求。
数据删除策略	云数据库 Memcache 版的过期机制是：每个 Key 的过期时间是按照用户设定的过期时间过期的，过期之后用户就无法再访问到该 Key。但是在过期后并不会对这些 Key 所占的空间进行马上回收，而是统一在凌晨2点多的时候做回收。
数据过期策略	和开源 Memcached 策略一致，采用 LRU 算法过期数据，但已过期数据不会被立即删除回收空间，回收空间操作由后台程序定期执行。
连接处理	云数据库 Memcache 版服务端不会主动关闭空闲的客户端连接。
数据过期	Key 过期时间建议由用户主动控制和管理。

任何兼容 Memcached 协议的客户端都可以访问云数据库 Memcache 版服务，您可以根据自身应用特点选用支持 SASL 和 Memcached Binary Protocol 的任何 Memcached 客户端。

协议

- Memcached Binary Protocol（二进制协议）
- SASL 认证协议

操作

云数据库 Memcache 版支持如下命令操作。

操作码	操作命令	备注

0x00	Get	
0x01	Set	
0x02	Add	
0x03	Replace	
0x04	Delete	
0x05	Increment	
0x06	Decrement	
0x07	Quit	
0x08	Flush	Memcache 在时间精度上是秒级
0x09	GetQ	
0x0a	No-op	
0x0b	Version	
0x0c	GetK	
0x0d	GetKQ	
0x0e	Append	
0x0f	Prepend	
0x10	Stat	不支持
0x11	SetQ	
0x12	AddQ	
0x13	ReplaceQ	
0x14	DeleteQ	
0x15	IncrementQ	
0x16	DecrementQ	
0x17	QuitQ	
0x18	FlushQ	
0x19	AppendQ	
0x1a	PrependQ	
0x1b	Verbosity	不支持
0x1c	Touch	
0x1d	GAT	
0x1e	GATQ	
0x20	SASL list mechs	

0x21	SASL Auth	
0x22	SASL Step	

云数据库 Memcache 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。您可根据需求自主选择，以下对购买流程做介绍。

注意事项

实例开通后即开始收费，不管实例是否有使用，因此建议在使用时再申请，避免不必要的费用。

开通云数据库 Memcache 版需要至少有一台 ECS，请参见购买 ECS。

操作步骤

登录**云数据库 Memcache 版控制台**，单击右上角的**创建实例**按钮。

默认进入包年包月的购买模式，选择**地域**、**可用区**、**实例规格**、**购买时长**，设置实例**登录密码**，**实例名称**、**数量**。

注意：

云数据库 Memcache 版仅限于内网访问，建议和 ECS 选择在同一地域同一可用区。

如果需要按量付费实例，则选择按量付费，填写项和包年包月基本一致。

购买时建议填写密码用于云实例访问，若未设置，则需要在**实例列表**>**管理**>**修改密码**中重置密码。

选好后单击**立即购买**，进入**确认订单**页面，确认订单信息无误后，单击**去支付**按钮。

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。

客户端连接实例

任何兼容 Memcached 协议的客户端都可以访问云数据库 Memcache 版服务。每一个 Memcached 客户端都有自己的特点，您可以根据应用特点选用支持 SASL 和 Memcached Binary Protocol 的任何一款 Memcached 客户端。

以下的 Memcached 客户端与云数据库 Memcache 版交互时工作顺畅，推荐您使用。

注意：

可以通过远程登录 ECS 服务器访问云数据库 Memcache 实例。请参考公网连接。

以下所有第三方开源客户端，均非阿里云官方提供，可能存在潜在 bug。开发者须自行保证客户端的质量。因客户端直接或间接导致的故障或损失，阿里云概不负责。

客户端下载

客户端下载地址

客户端介绍

客户端版本介绍

Java 代码示例

准备 Java 开发环境。登录已有的阿里云 ECS 服务器，在上面安装 Java JDK 和常用的 IDE（比如 Eclipse）。

Java JDK 下载地址

Eclipse（下载地址1，下载地址2）

第一个代码示例如下，把里面的 Java 代码复制到 Eclipse Project 里面去。

注意：此时的代码是编译不成功的，因为要想调用 Memcache 缓存服务还需要一个第三方提供的 jar 包下载地址。添加这个 jar 包之后，代码就能编译通过了。

OcsSample1.java 代码示例（需要用户名和密码）

```
 import java.io.IOException;
import java.util.concurrent.ExecutionException;
import net.spy.memcached.AddrUtil;
import net.spy.memcached.ConnectionFactoryBuilder;
import net.spy.memcached.ConnectionFactoryBuilder.Protocol;
import net.spy.memcached.MemcachedClient;
import net.spy.memcached.auth.AuthDescriptor;
import net.spy.memcached.auth.PlainCallbackHandler;
import net.spy.memcached.internal.OperationFuture;

public class OcsSample1 {

public static void main(String[] args) {

final String host = "xxxxxxxx.m.yyyyyyyyyy.ocs.aliyuncs.com";//控制台上的“内网地址”
final String port ="11211"; //默认端口 11211，不用改
final String username = "xxxxxxxxx";//控制台上的“实例ID”，新版ocs的username可以置空
final String password = "my_password";//邮件中提供的“密码”
MemcachedClient cache = null;
try {
AuthDescriptor ad = new AuthDescriptor(new String[]{"PLAIN"}, new PlainCallbackHandler(username,
password));

cache = new MemcachedClient(
new ConnectionFactoryBuilder().setProtocol(Protocol.BINARY)
.setAuthDescriptor(ad)
.build(),
AddrUtil.getAddresses(host + ":" + port));

System.out.println("OCS Sample Code");

//向OCS中存一个key为"ocs"的数据，便于后面验证读取数据
String key = "ocs";
String value = "Open Cache Service, from www.Aliyun.com";
int expireTime = 1000; // 过期时间，单位s; 从写入时刻开始计时，超过expireTime s后，该数据过期失效，无
法再读出；
OperationFuture<Boolean> future = cache.set(key, expireTime, value);
future.get(); // spymemcached set()是异步的，future.get() 等待cache.set()操作结束，也可以不等待，用户根
据自己需求选择
//向OCS中存若干个数据，随后可以在OCS控制台监控上看到统计信息
for(int i=0;i<100;i++){
key="key-"+i;
value="value-"+i;

//执行set操作，向缓存中存数据
expireTime = 1000; // 过期时间，单位s
future = cache.set(key, expireTime, value);
future.get(); // 确保之前(cache.set())操作已经结束
}
System.out.println("Set操作完成!");
//执行get操作，从缓存中读数据,读取key为"ocs"的数据

System.out.println("Get操作:"+cache.get(key));

} catch (IOException e) {
```

```
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
} catch (ExecutionException e) {
e.printStackTrace();
}
if (cache != null) {
cache.shutdown();
}

}//eof
}
```

OcsSample2.java 代码示例（不需要用户名和密码）

```
 import java.io.IOException;
import java.util.concurrent.ExecutionException;

import net.spy.memcached.AddrUtil;
import net.spy.memcached.BinaryConnectionFactory;
import net.spy.memcached.MemcachedClient;
import net.spy.memcached.internal.OperationFuture;

public class OcsSample2 {

public static void main(String[] args) {

final String host = "xxxxxxxx.m.yyyyyyyyyy.ocs.aliyuncs.com"; //控制台上的“内网地址”
final String port = "11211"; //默认端口 11211，不用改

MemcachedClient cache = null;
try {

cache = new MemcachedClient(new BinaryConnectionFactory(), AddrUtil.getAddresses(host + ":" + port));

System.out.println("OCS Sample Code");

//向OCS中存一个key为"ocs"的数据，便于后面验证读取数据
String key = "ocs";
String value = "Open Cache Service, from www.Aliyun.com";
int expireTime = 1000; // 过期时间，单位s; 从写入时刻开始计时，超过 expireTime s后，该数据过期失效，无法再读出；
OperationFuture<Boolean> future = cache.set(key, expireTime, value);
future.get();
//向OCS中存若干个数据，随后可以在OCS控制台监控上看到统计信息
for (int i = 0; i < 100; i++) {
key = "key-" + i;
value = "value-" + i;

//执行set操作，向缓存中存数据
expireTime = 1000; // 过期时间，单位s
future = cache.set(key, expireTime, value);
future.get();
}
```

```
System.out.println("Set操作完成!");

//执行get操作，从缓存中读数据,读取key为"ocs"的数据
System.out.println("Get操作:" + cache.get(key));

} catch (IOException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
} catch (ExecutionException e) {
e.printStackTrace();
}
if (cache != null) {
cache.shutdown();
}

}//eof
}
```

在 Eclipse 里面打开的 OcsSample1.java，根据自己的实例信息修改几个地方。

每个人买到的云数据库 Memcache 实例的 ID 都是不重复的，其对应的阿里云内网地址也是独一无二的，这些信息都在云数据库 Memcache 控制台上显示出来。在同自己的云数据库 Memcache 实例建立连接的时候，需要根据这些信息修改 OcsSample1.java 中的对应地方。

信息修改完毕，可以运行自己的程序了。运行 main 函数，会在 Eclipse 下面的 console 窗口看到下面这样的结果（请忽略可能出现的红色 INFO 调试信息）。

```
 OCS Sample Code
Set操作完成!
Get操作: Open Cache Service, from www.Aliyun.com
```

客户端下载

客户端下载

客户端介绍

客户端版本介绍

系统要求及环境配置

注意：您已经有 php memcache 等环境，请注意教程中的一些提示，以免生产环境被覆盖，导致业务不可用，在升级及再编译环境前请做好环境备份。

windows 系列版本

如果采用标准的 php memcached 扩展不能成功搭建，可以考虑换成手工拼包的形式来访问云数据库 Memcache，连接方式请参考如下链接，示例代码非常简单，与 php memcached 的区别就是仅支持主流接口，需自己补充某些特定接口，安装及使用方法请参见这里。

Centos 及 Aliyun Linux 6系列版本

注意：Memcached 2.2.0 扩展必须使用 libmemcached 1.0.x 的库，低于1.0的库不再能够成功编译。编译 libmemcached 时 GCC 要求在4.2及以上。

确认是否安装了gcc-c++ 等组件（使用 gcc –v 查看版本是否为4.2及以上）。如没有请执行 yum install gcc+ gcc-c++。

执行 rpm –qa | grep php 查看系统中是否有 PHP 环境，如果没有则执行 yum install php-devel,php-common,php-cli 安装包含源码编译的 PHP。

建议使用 php 5.3及以上版本。php 5.2部分版本系列源代码会有 zend_parse_parameters_none 函数会出错，如需使用请参照 php 官方相关文档。如是源代码编译，请按照官方 php 编译升级的办法进行。

检测是否有已安装了 SASL 相关环境包，如没有，则执行 yum install cyrus-sasl-plain cyrus-sasl cyrus-sasl-devel cyrus-sasl-lib 安装 SASL 相关环境。

检测下是否有已安装了 libmemcached 源码包，若没有，则执行以下命令安装 libmemcached 源码包（推荐版本 libmemcached-1.0.18）。

```
 wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz
tar zxvf libmemcached-1.0.18.tar.gz
cd libmemcached-1.0.18
./configure --prefix=/usr/local/libmemcached --enable-sasl
make
make install
cd ..
```

执行 yum install zlib-devel 安装 memcached 源码包（推荐版本为 memcached-2.2.0）。

注意：

安装 memcached 前需要确认是否有 zlib-devel 包需要执行。

请先检测下是否已安装了 memcached 客户端包（包含源码包）。如有则不需要安装，但需要重新编译增加 -enable-memcached-sasl 这个扩展。

```
wget http://pecl.php.net/get/memcached-2.2.0.tgz
tar zxvf memcached-2.2.0.tgz
cd memcached-2.2.0
phpize（如果系统中有两套PHP环境，需绝对路径调用该命令/usr/bin/phpize，该路径为使用云数据库Memcache的PHP环境路径）
./configure --with-libmemcached-dir=/usr/local/libmemcached --enable-memcached-sasl（注意这个参数）
make
make install
```

修改 php.ini 文件（locate 找该文件，如果系统中有两套 PHP 环境，需找到使用云数据库 Memcache 的 PHP 环境路径，对应修改之），增加 extension=memcached.so memcached.use_sasl = 1。

使用该页面最后的测试代码测试下是否环境部署成功，请修改代码中相应的地址、端口、用户名及密码。

Centos及 Aliyun Linux 5系列版本 【64位版本】

确认是否安装了 gcc-c++ 等组件。如没有请执行 yum install gcc+ gcc-c++ 。

执行rpm –qa | grep php 查看系统中是否有 php 环境，如果没有则执行 yum install php53 php53-devel 安装包含源码编译的 php；如有 php 则不要安装。建议使用 php 5.3（含）以上版。

php 5.2部分版本系列源代码会有 zend_parse_parameters_none 函数会出错，如需使用请参照 php 官方相关文档。

执行 yum install cyrus-sasl-plain cyrus-sasl cyrus-sasl-devel cyrus-sasl-lib 安装 SASL 相关环境。

检测下是否已安装了 libmemcached（包含源码包），如有则不需要安装，如没有则执行以下命令安装（推荐版本 libmemcached 1.0.2）。

```
 wget http://launchpad.net/libmemcached/1.0/1.0.2/+download/libmemcached-1.0.2.tar.gz
tar -zxvf libmemcached-1.0.2.tar.gz
cd libmemcached-1.0.2
./configure --prefix=/usr/local/libmemcached --enable-sasl
make
make install
cd ..
```

执行 yum install zlib-devel 安装源码包 memcached（推荐版本 memcached 2.0）。

注意：

安装 memcached 前需要确认是否有 zlib-devel 包需要执行。

请先检测下是否有已安装了 memcached 客户端包（包含源码包）。如有则不需要安装，但需要重新编译增加 -enable-memcached-sasl 这个扩展。

```
wget http://pecl.php.net/get/memcached-2.0.0.tgz tar -zxvf memcached-2.0.0.tgz
cd memcached-2.0.0 phpize（如果系统中有两套PHP环境，需绝对路径调用该命令/usr/bin/phpize，该路径为使用云数据库Memcache的PHP环境路径，请在memcached源码目录内执行phpize）
./configure --with-libmemcached-dir=/usr/local/libmemcached --enable-memcached-sasl（注意这个参数）
make
make install
```

修改 php.ini 文件（locate 找该文件，yum 安装的一般在 /etc/php.ini。 如果系统中有两套 PHP 环境，需找到使用云数据库 Memcache 的 PHP 环境路径，对应修改之），增加 extension=memcached.so memcached.use_sasl = 1 。

执行 php –m |grep ，memcached ，若显结果有 memcache 表示环境已支持 memcache。

使用该页面最后的测试代码测试下是否环境部署成功，请修改代码中相应的地址、端口、用户名及密码。

Ubuntu Debian 等系列版本

变更 ubuntu 源。

方案一：执行 vim /etc/apt/source.list，在最前面添加以下内容。

```
 deb http://mirrors.aliyun.com/ubuntu/ precise main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ precise-security main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ precise-updates main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ precise-proposed main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ precise-backports main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ precise main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ precise-security main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ precise-updates main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ precise-proposed main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ precise-backports main restricted universe multiverse
apt-get update //更新一下列表
```

方案二： 通过 wget http://oss.aliyuncs.com/aliyunecs/update_source.zip 下载 update_source 的压缩包，解压后予执行权限 chmod 777 文件名 ，然后执行该脚本进行自动变更源操作。

通过 ape-get 配置 GCC,G++ 。

首先需要使用 dpkg –s 安装包名，例如 dpkg –s gcc，确认是否安装了 gcc-c++ 等组件。如没有请执行 apt-get build-dep gcc apt-get install build-essential。

安装 php5, php5-dev。

首先需要使用 dpkg –s 安装包名，例如 dpkg –s php，确认是否安装了 php 等组件。如没有请执行 apt-get install php5 php5-dev （同时会自动安装php5-cli和php5-common）。

安装配置 sasl 支持。

首先需要使用 dpkg –s 安装包名， 例如 dpkg –s libsasl2，确认是否安装了 libsasl2 cloog-ppl 等组件，如没有请执行以下命令。

```
apt-get install libsasl2-dev cloog-ppl
cd /usr/local/src
```

执行以下命令安装指定版本的 libmemcache。

注意：请先检测下是否有已安装了这些包（包含源码包），如有则不需要安装。

```
wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz
tar -zxvf libmemcached-1.0.18.tar.gz
cd libmemcached-1.0.18
./configure --prefix=/usr/local/libmemcached
make
make install
cd ..
```

执行以下命令安装指定版本的 memcached。

注意：请先检测下是否有已安装了 memcached 客户端包（包含源码包），如有则不需要安装，但需要重新编译增加 -enable-memcached-sasl 这个扩展。

```
wget
http://pecl.php.net/get/memcached-2.2.0.tgz tar zxvf memcached-2.2.0.tgz
cd memcached-2.2.0 phpize5
./configure --with-libmemcached-dir=/usr/local/libmemcached --enable-memcached-sasl
make
make install
```

配置 php 支持 memcached，然后测试。

```
echo "extension=memcached.so" >>/etc/php5/conf.d/pdo.ini echo "memcached.use_sasl = 1"
>>/etc/php5/conf.d/pdo.ini
```

```
php -m |grep mem memcached
```

如果显示出该组件代表安装完成，配置完毕。

PHP 代码示例

示例1：基本的连接云数据库 Memcache 及 set/get 操作

```
<?php
$connect = new Memcached; //声明一个新的memcached链接
$connect->setOption(Memcached::OPT_COMPRESSION, false); //关闭压缩功能
$connect->setOption(Memcached::OPT_BINARY_PROTOCOL, true); //使用binary二进制协议
$connect->setOption(Memcached::OPT_TCP_NODELAY, true); //重要，php memcached有个bug，当get的值不存在，有固定40ms延迟，开启这个参数，可以避免这个bug
$connect->addServer('aaaaaaaaaa.m.yyyyyyyyyy.ocs.aliyuncs.com', 11211); //添加OCS实例地址及端口号
$connect->setSaslAuthData('aaaaaaaaaa', 'password'); //设置OCS帐号密码进行鉴权，如已开启免密码功能，则无需此步骤；新版OCS的username可以置空
$connect->set("hello", "world");
echo 'hello: ',$connect->get("hello");
$connect->quit();
?>
```

示例2：在云数据库 Memcache 中缓存一个数组

```
<?php
$connect= new Memcached; //声明一个新的memcached链接
$connect->setOption(Memcached::OPT_COMPRESSION, false); //关闭压缩功能
$connect->setOption(Memcached::OPT_BINARY_PROTOCOL, true);//使用binary二进制协议
$connect->setOption(Memcached::OPT_TCP_NODELAY, true); //重要，php memcached有个bug，当get的值不存在，有固定40ms延迟，开启这个参数，可以避免这个bug
$connect->addServer('xxxxxxxx.m.yyyyyyyy.ocs.aliyuncs.com', 11211);//添加OCS实例地址及端口号
$connect->setSaslAuthData('xxxxxxxx', 'bbbbbbbb');//设置OCS帐号密码进行鉴权，如已开启免密码功能，则无需此步骤
$user = array(
"name" => "ocs",
"age" => 1,
"sex" => "male"
); //声明一组数组
$expire = 60; //设置过期时间
test($connect->set('your_name',$user,$expire), true, 'Set cache failed');
if($connect->get('your_name')){
$result =$connect->get('your_name');
}else{
echo "Return code:", $connect->getResultCode();
echo "Retucn Message:", $connect->getResultMessage (); //如出现错误，解析出返回码
$result=" ";
}
print_r($result);
$connect->quit();

function test($val, $expect, $msg)
{
```

```
if($val!= $expect) throw new Exception($msg);
}
?>
```

示例3：云数据库 Memcache 与 MySQL 数据库结合使用

```
<?php
$connect = new Memcached; //声明一个新的memcached链接
$connect->setOption(Memcached::OPT_COMPRESSION, false);//关闭压缩功能
$connect->setOption(Memcached::OPT_BINARY_PROTOCOL, true);//使用binary二进制协议
$connect->setOption(Memcached::OPT_TCP_NODELAY, true); //重要，php memcached有个bug，当get的值不存在，有固定40ms延迟，开启这个参数，可以避免这个bug
$connect->addServer('xxxxxx.m.yyyyyyyy.ocs.aliyuncs.com', 11211);//添加实例地址 端口号
$connect->setSaslAuthData('xxxxxx', 'my_passwd');//设置OCS帐号密码进行鉴权，如已开启免密码功能，则无需此步骤
$user = array(
"name" => "ocs",
"age" => 1,
"sex" => "male"
); //定义一组数组

if($connect->get('your_name'))
{
$result =$connect->get('your_name');
print_r($result);
echo "Found in OCS, get data from OCS"; //如果获取到数据，则打印此数据来源于OCS
exit;
}
else
{
echo "Return code:", $connect->getResultCode();
echo "Retucn Message:", $connect->getResultMessage ();//抛出code返回码
$db_host='zzzzzz.mysql.rds.aliyuncs.com'; //数据库地址
$db_name='my_db'; //database name
$db_username='db_user'; //数据库用户名
$db_password='db_passwd';//数据库用户密码
$connection=mysql_connect($db_host,$db_username,$db_password);
if (!mysql_select_db($db_name, $connection))
{
echo 'Could not select database'; //数据库连接不成功则抛出错误信息
exit;
}
$sql = "SELECT name,age,sex FROM test1 WHERE name = 'ocs'";
$result = mysql_query($sql, $connection);
while ($row = mysql_fetch_assoc($result))
{
$user = array(
"name" => $row["name"],
"age" => $row["age"],
"sex" => $row["sex"],
);
$expire = 5; //设置数据在缓存中的过期时间
test($connect->set('your_name',$user,$expire), true, 'Set cache failed'); //写入OCS缓存
}
mysql_free_result($result);
```

```
mysql_close($connection);
}
print_r($connect->get('your_name')); //打印出 获取到的数据
echo "Not Found in OCS,get data from MySQL"; //确认从数据库获取的数据
$connect->quit();

function test($val, $expect, $msg)
{
if($val!= $expect) throw new Exception($msg);
}
?>
```

客户端下载

客户端下载地址

客户端介绍

客户端版本介绍

环境配置

依赖于 bmemcached（支持 SASL 扩展），下载链接请见这里。

Python 代码示例

```
#!/usr/bin/env python
import bmemcached
client = bmemcached.Client(('ip:port'), 'user', 'passwd')
print client.set('key', 'value11111111111')
print client.get('key')
```

客户端下载

客户端下载地址

客户端介绍

客户端版本介绍

C#/.NET 代码示例

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

```
using Enyim.Caching.Memcached;
namespace OCS.Memcached
{
public sealed class MemCached
{
private static MemcachedClient MemClient;
static readonly object padlock = new object();
//线程安全的单例模式
public static MemcachedClient getInstance()
{
if (MemClient == null)
{
lock (padlock)
{
if (MemClient == null)
{
MemClientInit();
}
}
}
return MemClient;
}

static void MemClientInit()
{
//初始化缓存
MemcachedClientConfiguration memConfig = new MemcachedClientConfiguration();
IPAddress newaddress =
IPAddress.Parse(Dns.GetHostEntry
("your_ocs_host").AddressList[0].ToString())；//your_ocs_host替换为OCS内网地址
IPEndPoint ipEndPoint = new IPEndPoint(newaddress, 11211);

// 配置文件 - ip
memConfig.Servers.Add(ipEndPoint);
// 配置文件 - 协议
memConfig.Protocol = MemcachedProtocol.Binary;
// 配置文件-权限
memConfig.Authentication.Type = typeof(PlainTextAuthenticator);
memConfig.Authentication.Parameters["zone"] = "";
memConfig.Authentication.Parameters["userName"] = "username";
memConfig.Authentication.Parameters["password"] = "password";
//下面请根据实例的最大连接数进行设置
memConfig.SocketPool.MinPoolSize = 5;
memConfig.SocketPool.MaxPoolSize = 200;
MemClient=new MemcachedClient(memConfig);
}
}
}
```

依赖

调用代码：MemcachedClient MemClient = MemCached.getInstance();

客户端下载

客户端下载地址

客户端介绍

客户端版本介绍

环境配置

下载编译安装 C++ 客户端。

https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz

执行以下命令。

```
 tar -xvf libmemcached-1.0.18.tar.gz
cd libmemcached-1.0.18
./configure --enable-sasl
make
make install (可能需要sudo权限)
```

C++代码示例

下载 ocs_test.tar.gz。

执行以下命令。

```
tar -xvf ocs_test.tar.gz
cd ocs_test
vim ocs_test_sample1.cpp
```

修改 TARGET_HOST 为购买到的 Memcache 地址， USERNAME 为购买到的用户名， PASSWORD 为设置好的密码。

执行 build.sh 生成 ocs_test。运行 ./ocs_test 即可看到写入一个 key 到 Memcache 中，再从 Memcache 获取到，最后将这个 key 从 Memcache 中删除。

ocs_test_sample1.cpp 的代码如下：

```
 #include <iostream>
#include <string>
```

```
#include <libmemcached/memcached.h>
using namespace std;

#define TARGET_HOST ""
#define USERNAME ""
#define PASSWORD ""

int main(int argc, char *argv[])
{
memcached_st *memc = NULL;
memcached_return rc;
memcached_server_st *server;
memc = memcached_create(NULL);
server = memcached_server_list_append(NULL, TARGET_HOST, 11211,&rc);

/* SASL */
sasl_client_init(NULL);
rc = memcached_set_sasl_auth_data(memc, USERNAME, PASSWORD);
if(rc != MEMCACHED_SUCCESS) {
cout<<"Set SASL err:"<< endl;
}
rc = memcached_behavior_set(memc,MEMCACHED_BEHAVIOR_BINARY_PROTOCOL,1);
if(rc != MEMCACHED_SUCCESS) {
cout<<"Binary Set err:"<<endl;
}
/* SASL */

rc = memcached_server_push(memc,server);
if(rc != MEMCACHED_SUCCESS) {
cout <<"Connect Mem err:"<< rc << endl;
}
memcached_server_list_free(server);

string key = "TestKey";
string value = "TestValue";
size_t value_length = value.length();
size_t key_length = key.length();
int expiration = 0;
uint32_t flags = 0;

//Save data
rc = memcached_set(memc,key.c_str(),key.length(),value.c_str(),value.length(),expiration,flags);
if (rc != MEMCACHED_SUCCESS){
cout <<"Save data failed: " << rc << endl;
return -1;
}
cout <<"Save data succeed, key: " << key << " value: " << value << endl;

cout << "Start get key:" << key << endl;

char* result = memcached_get(memc,key.c_str(),key_length,&value_length,&flags,&rc);
cout << "Get value:" << result << endl;

//Delete data
cout << "Start delete key:" << key << endl;
rc = memcached_delete(memc,key.c_str(),key_length,expiration);
```

```
if (rc != MEMCACHED_SUCCESS) {
cout << "Delete key failed: " << rc << endl;
}
cout << "Delete key succeed: " << rc << endl;
//free
memcached_free(memc);
return 0;
}
```

下面是另一个 C++程序使用 Memcache 实例，在这里我们可以看见 Memcache 缓存与 MySQL 数据库相结合的场景。编译安装 C++客户端的步骤还是与上一个例子相同。

在 MySQL 数据库中创建示例 database 和 table。

```
mysql -h host -uUSER -pPASSSWORD
create database testdb;
create table user_info (user_id int, user_name char(32) not null, password char(32) not null, is_online int,
primary key(user_id) );
```

下载 ocs_test_2.tar.gz，并执行以下命令。

```
tar -xvf ocs_test_2.tar.gz
cd ocs_test
vim ocs_test_sample2.cpp
```

注意：修改 OCS_TARGET_HOST 为购买到的 Memcache 地址， OCS_USERNAME 为 Memcache 的实例名， OCS_PASSWORD 为设置好的密码；MYSQL_HOST 为 MySQL 地址，MYSQL_USERNAME 为数据库的用户名，MYSQL_PASSWORD 为数据库的密码。

执行 build.sh 生成 ocs_test ，执行 /ocs_test 即可。

ocs_test_sample2.cpp 代码如下：

```
 #include <iostream>
#include <string>
#include <sstream>
#include <libmemcached/memcached.h>
#include <mysql/mysql.h>

using namespace std;

#define OCS_TARGET_HOST "xxxxxxxxxx.m.yyyyyyyyy.ocs.aliyuncs.com"
#define OCS_USERNAME "your_user_name"
#define OCS_PASSWORD "your_password"

#define MYSQL_HOST "zzzzzzzzzz.mysql.rds.aliyuncs.com"
#define MYSQL_USERNAME "db_user"
```

```
#define MYSQL_PASSWORD "db_paswd"
#define MYSQL_DBNAME "testdb"

#define TEST_USER_ID "100"

MYSQL *mysql = NULL;
memcached_st *memc = NULL;
memcached_return rc;

int InitMysql()
{
mysql = mysql_init(0);
if (mysql_real_connect(mysql, MYSQL_HOST, MYSQL_USERNAME, MYSQL_PASSWORD,
MYSQL_DBNAME, MYSQL_PORT, NULL, CLIENT_FOUND_ROWS) == NULL )
{
cout << "connect mysql failure!" << endl;
return EXIT_FAILURE;
}
cout << "connect mysql success!" << endl;
return 0;
}

bool InitMemcached()
{
memcached_server_st *server;
memc = memcached_create(NULL);
server = memcached_server_list_append(NULL, OCS_TARGET_HOST, 11211,&rc);

/* SASL */
sasl_client_init(NULL);
rc = memcached_set_sasl_auth_data(memc, OCS_USERNAME, OCS_PASSWORD);
if (rc != MEMCACHED_SUCCESS)
{
cout<<"Set SASL err:"<< endl;
return false;
}
rc = memcached_behavior_set(memc,MEMCACHED_BEHAVIOR_BINARY_PROTOCOL,1);
if (rc != MEMCACHED_SUCCESS)
{
cout<<"Binary Set err:"<<endl;
return false;
}
/* SASL */
rc = memcached_server_push(memc,server);
if (rc != MEMCACHED_SUCCESS)
{
cout <<"Connect Mem err:"<< rc << endl;
return false;
}
memcached_server_list_free(server);
return true;
}

struct UserInfo
{
int user_id;
```

```
char user_name[32];
char password[32];
int is_online;
};

bool SaveToCache(string &key, string &value, int expiration)
{
size_t value_length = value.length();
size_t key_length = key.length();
uint32_t flags = 0;
//Save data
rc = memcached_set( memc,key.c_str(), key.length(), value.c_str(), value.length(), expiration, flags);
if (rc != MEMCACHED_SUCCESS){
cout <<"Save data to cache failed: " << rc << endl;
return false;
}
cout <<"Save data to cache succeed, key: " << key << " value: " << value << endl;
return true;
}

UserInfo *GetUserInfo(int user_id)
{
UserInfo *user_info = NULL;
//get from cache
string key;
stringstream out;
out << user_id;
key = out.str();
cout << "Start get key:" << key << endl;
size_t value_length;
uint32_t flags;
char* result = memcached_get(memc, key.c_str(), key.size(), &value_length, &flags, &rc);
if (rc != MEMCACHED_SUCCESS)
{
cout << "Get Cache Failed, start get from mysql."<< endl;
int status;
char select_sql[1024];
memset(select_sql, 0x0, sizeof(select_sql));
sprintf(select_sql, "select * from user_info where user_id = %d", user_id);
status = mysql_query(mysql, select_sql);
if (status !=0 )
{
cout << "query from mysql failure!" << endl;
return NULL;
}
cout << "the status is :" << status << endl;
MYSQL_RES *mysql_result = mysql_store_result(mysql);
user_info = new UserInfo;
MYSQL_ROW row;
while (row = mysql_fetch_row(mysql_result))
{
user_info->user_id = atoi(row[0]);
strncpy(user_info->user_name, row[1], strlen(row[1]));
strncpy(user_info->password, row[2], strlen(row[2]));
user_info->is_online = atoi(row[3]);
}
```

```
mysql_free_result(mysql_result);
return user_info;
}
cout << "Get from cache succeed" << endl;
user_info = new UserInfo;
memcpy(user_info, result, value_length);
return user_info;
}

bool DeleteCache(string &key, int expiration)
{
rc = memcached_delete(memc, key.c_str(), key.length(), expiration);
if (rc != MEMCACHED_SUCCESS) {
cout << "Delete key failed: " << rc << endl;
return false;
}
cout << "Delete key succeed: " << rc << endl;
return true;
}

void PrintUserInfo(UserInfo *user_info)
{
cout << "user_id: " << user_info->user_id << " " << " name: " << user_info->user_name << endl;
}

bool SaveMysql(UserInfo *user_info)
{
char insert_sql[1024];
memset(insert_sql, 0x0, sizeof(insert_sql));
sprintf(insert_sql, "insert into user_info(user_id, user_name, password, is_online) values(%d, '%s', '%s',
%d)", user_info->user_id, user_info->user_name, user_info->password, user_info->is_online);
int status = mysql_query(mysql, insert_sql);
if (status != 0)
{
cout << "insert failed" << endl;
return false;
}
cout << "insert user_info" << endl;
//insert mysql
return true;
}

int main(int argc, char *argv[])
{
if (InitMysql() != 0)
{
return -1;
}
if (!InitMemcached())
{
return -1;
}

//generate user_info
UserInfo user_info;
user_info.user_id = atoi(TEST_USER_ID);
```

```
strcpy(user_info.user_name, "James");
strcpy(user_info.password, "12345678");
user_info.is_online = 1;

//save to mysql
if (!SaveMysql(&user_info))
{
//return -1;
}
string user_str;
user_str.assign((char*)&user_info, sizeof(UserInfo));
//save to memcached
string key_str = TEST_USER_ID;
SaveToCache(key_str, user_str, 10);

//start get, exist in memcahced
UserInfo *get_user_info = GetUserInfo(user_info.user_id);
PrintUserInfo(get_user_info);

//wait 10 secons
sleep(2);
//delete memcached or expired
DeleteCache(key_str, 0);

//start get, exist in mysql
delete get_user_info;
get_user_info = GetUserInfo(user_info.user_id);

PrintUserInfo(get_user_info);
delete get_user_info;
//free
memcached_free(memc);
mysql_close(mysql);
return 0;
}
```

客户端介绍

EnyimMemcachedCore 是一个从 EnyimMemcached 迁移至 .NET Core 的 Memcached 客户端，支持 .NET Core。

源代码托管在 GitHub 上的地址：https://github.com/cnblogs/EnyimMemcachedCore

NuGet 包地址：https://www.nuget.org/packages/EnyimMemcachedCore

1. 安装 NuGet 包

```
Install-Package EnyimMemcachedCore
```

2. 配置

2.1 在 appsetting.json 中进行配置

- 不带验证的配置

```
{
"enyimMemcached": {
"Servers": [
{
"Address": "memcached",
"Port": 11211
}
]
}
}
```

- 带验证的配置

```
{
"enyimMemcached": {
"Servers": [
{
"Address": "memcached",
"Port": 11211
}
],
"Authentication": {
"Type": "Enyim.Caching.Memcached.PlainTextAuthenticator",
"Parameters": {
"zone": "",
"userName": "username",
"password": "password"
}
}
}
}
```

- Startup.cs 中的配置代码

```
public class Startup
{
public void ConfigureServices(IServiceCollection services)
{
services.AddEnyimMemcached(options => Configuration.GetSection("enyimMemcached").Bind(options));
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
```

```
app.UseEnyimMemcached();
}
}
```

2.2 直接硬编码配置（无需配置文件）

Startup.cs 中的硬编码配置代码

```
public class Startup
{
public void ConfigureServices(IServiceCollection services)
{
services.AddEnyimMemcached(options =>
{
options.AddServer("memcached", 11211);
//options.AddPlainTextAuthenticator("", "usename", "password");
});
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
app.UseEnyimMemcached();
}
}
```

3. 调用

3.1 使用 IMemcachedClient 接口

```
public class TabNavService
{
private ITabNavRepository _tabNavRepository;
private IMemcachedClient _memcachedClient;

public TabNavService(
ITabNavRepository tabNavRepository,
IMemcachedClient memcachedClient)
{
_tabNavRepository = tabNavRepository;
_memcachedClient = memcachedClient;
}

public async Task<IEnumerable<TabNav>> GetAll()
{
var cacheKey = "aboutus_tabnavs_all";
var result = await _memcachedClient.GetAsync<IEnumerable<TabNav>>(cacheKey);
if (!result.Success)
{
var tabNavs = await _tabNavRepository.GetAll();
await _memcachedClient.AddAsync(cacheKey, tabNavs, 300);
```

```
return tabNavs;
}
else
{
return result.Value;
}
}
}
```

3.2 使用 IDistributedCache 接口（来自 Microsoft.Extensions.Caching.Abstractions ）

```
public class CreativeService
{
private ICreativeRepository _creativeRepository;
private IDistributedCache _cache;

public CreativeService(
ICreativeRepository creativeRepository,
IDistributedCache cache)
{
_creativeRepository = creativeRepository;
_cache = cache;
}

public async Task<IList<CreativeDTO>> GetCreatives(string unitName)
{
var cacheKey = $"creatives_{unitName}";
IList<CreativeDTO> creatives = null;

var creativesJson = await _cache.GetStringAsync(cacheKey);
if (creativesJson == null)
{
creatives = await _creativeRepository.GetCreatives(unitName)
.ProjectTo<CreativeDTO>().ToListAsync();

var json = string.Empty;
if (creatives != null && creatives.Count() > 0)
{
json = JsonConvert.SerializeObject(creatives);
}
await _cache.SetStringAsync(
cacheKey,
json,
new DistributedCacheEntryOptions().SetSlidingExpiration(TimeSpan.FromMinutes(5)));
}
else
{
creatives = JsonConvert.DeserializeObject<List<CreativeDTO>>(creativesJson);
}

return creatives;
}
```

```
}
```

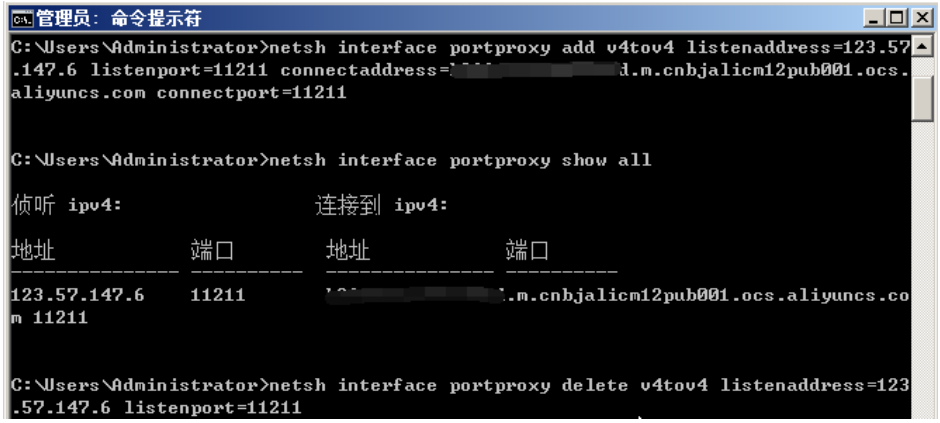
ECS Windows篇

目前云数据库 Memcache 版是需要通过 ECS 的内网进行连接访问，如果您本地需要通过公网访问云数据库 Memcache 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 服务器中，在 CMD 执行以下命令。

```
 netsh interface portproxy add v4tov4 listenaddress=ECS服务器的公网IP地址 listenport=11211 connectaddress=云数据库Memcache的连接地址 connectport=11211
```

如下图：



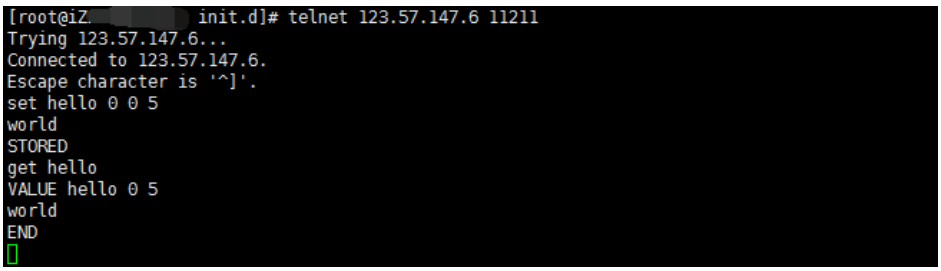
注意：

netsh interface portproxy delete v4tov4 listenaddress=ECS公网服务器的公网IP地址 listenport=11211 //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。

设置完成后进行验证测试。

在本地通过 telnet 连接 ECS Windows 服务器后进行数据写入和查询验证，如果 ECS Windows 服务器的 IP 是1.1.1.1，即 telnet 1.1.1.1 11211。



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 Windows 11211端口，进而访问云数据库 Memcache 版。

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 Memcache 版是需要通过 ECS 的内网进行连接访问，如果您本地需要通过公网访问云数据库 Memcache，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，否则会报错)
mkdir /usr/man&&make&&make install
```

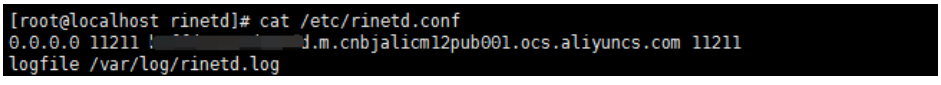
注意：rinetd 安装包下载地址不确保下载可用性，您可以自行搜索安装包进行下载使用。

创建配置文件。

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

输入如下内容。

```
0.0.0.0 11211 Memcache的链接地址 11211
logfile /var/log/rinetd.log
```

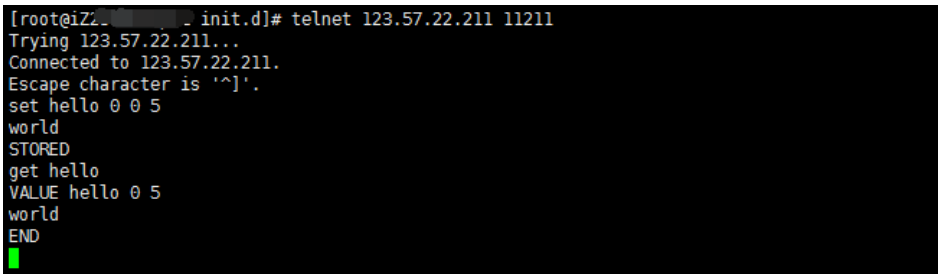


执行 rinetd 命令启动 rinetd。

注意：通过echo rinetd >>/etc/rc.local可以设置为自启动。可以使用 pkill rinetd 结束该进程。

验证测试。

在本地通过 telnet 连接 ECS Linux 服务器后进行数据写入和查询验证，比如安装了 rinetd 的服务器的 IP 是 1.1.1.1，即 telnet 1.1.1.1 11211。



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 11211 端口，进而访问云数据库 Memcache 版。

注意：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。