天翼云分布式缓存

Memcache 用户指南

中国电信股份有限公司云计算分公司

目 录

1. 产品概述

1.1. 产品介绍

天翼云分布式缓存 Memcache 是高性能、高可用的分布式内存缓存服务，用于减轻数据库负载，可有效加快应用速度、提高并发能力、提升应用的可扩展性。以简单易用、高可靠性、可扩展性为设计理念，助力企业和组织对业务应用加速和高读写性能的诉求，保障业务系统高速稳定运行。主要用途如下：

- 能够缓解后端存储服务的压力。
- 能够用于快速响应热点数据。
- 降低了用户部署与管理分布式缓存服务的复杂性。
- 支持原生 Memcached 的 API 接口。

1.2. 规格说明

Memcache 产品规格

单机版

规格名称	内存(GB)	实例可使用内存(GB)	最大连接数	最大内网带宽(Mbyte)
1G	1	0.8	10000	10
2G	2	1.5	10000	16
4G	4	3.2	10000	24
8G	8	6.4	10000	48
16G	16	12.8	20000	96
32G	32	25.6	40000	192
64G	64	51.2	80000	384

主从版

规格名称	内存(GB)	实例可使用内存(GB)	最大连接数	最大内网带宽(Mbyte)
1G	1	0.8	10000	10
2G	2	1.5	10000	16
4G	4	3.2	10000	24
8G	8	6.4	10000	48
16G	16	12.8	20000	96
32G	32	25.6	40000	192
64G	64	51.2	80000	384

1.3. 产品优势

1. 简单易用

部署配置简单快捷，即开即用，提供图形化的管理控制台，便于统一维护，轻松应对集群部署和管理。支持原生 Memcached 协议，兼容更多的开源类型客户端。

2. 支持动态扩展

支持存储容量一键扩容，用户可根据业务需求通过控制台对实例存储容量进行调整。

3. 资源隔离

针对租户级别的网络隔离，可以更好的保障租户的网络安全。

针对实例级别的资源隔离，可以更好地保障单个用户服务的稳定性。

4. 安全可靠

支持密码认证方式以确保访问安全可靠，支持白名单访问。

支持数据备份到磁盘，轻松恢复数据。

5. 分钟级监控

提供基于引擎和资源的分钟级别历史监控。

提供各数据结构和接口的监控信息，访问情况一目了然，便于用户对分布式缓存 Memcache 的使用情况有充分的了解。

6. 高可用

主从版本具有主从双节点，避免单点故障引起的服务中断。

1.4. 产品功能

分布式缓存 Memcache 提供单机、主从实例类型，满足不同用户高读写性能的业务诉求。提供丰富的实例管理功能，为您免去运维的烦恼。从而可以聚焦于业务逻辑本身，无需过多考虑部署、监控、扩容、安全、故障恢复等方面的问题。

- 数据持久化

支持数据持久化，当用户产品实例出现故障时，能够通过持久化数据快速恢复。

- 多重安全认证和管理

1. 支持 IP 白名单配置，最多支持配置 100 个允许连接实例的服务器 IP 地址，从访问源进行直接的风险控制。
2. 分布式缓存 Memcache 全面接入天翼云 VPC，可基于天翼云构建出隔离的网络环境。
3. 采用 SASL 进行用户身份认证鉴权，保障数据访问安全性。

- 实时监控告警

为用户提供为用户提供实时的内存使用量、缓存命中率、网络流量、数据淘汰率、CPU 使用率等信息，以及 7 天内的的历史监控信息。

- 可视化数据管理

提供可视化数据管理工具，轻松搞定数据读写操作。

1.5. 应用场景

1. 静态页面缓存

Web 页面的内容片段，包括 HTML，CSS 和图片等静态数据，内容修改操作少，读取频繁，可以缓存到分布式缓存 Memcache 实例，提高网站的访问性能

2. 数据库缓存业务

在动态系统中存在对大量数据读多写少的场景，如社交、博客网站大量查询用户信息、好友信息、文章信息等。为了减少磁盘数据库负载，提升性能，这些经常需要从数据库读取的数据，可以缓存在分布式缓存 Memcache 存储

适宜缓存的数据主要有：

- 经常被读取，实时性要求不高，且可以自动过期的数据。

例如网站首页最新文章列表、某某排行等数据，虽然新数据不断产生，但对用户体验影响比较小。这类数据可使用典型的缓存策略，设置合理的过期时间，当数据过期以后再从数据库中读取。为了让编辑或者其它人员能马上看到效果，可以再定一个缓存清除/刷新策略。

- 经常被读取并且实时性要求强的数据。

比如用户的好友列表，用户文章列表，用户阅读记录，游戏装备等。这类数据首先被载入到 Memcache 中，当发生更改（添加、修改、删除）时就刷新缓存数据。

3. 秒杀业务

关系商品下单操作，牵涉数据库读取，写入订单，更改库存，及事务一致性要求， 对于使用传统型数据库的平台来说，秒杀活动阶段要避免订单创建后库存缺货，同时提供流畅的用户体验，压力巨大。

可以利用 Memcache 的 incr/decr 功能， 在内存中存储商品的库存量， 秒杀的抢单过程主要在内存中完成，速度非常快，抢单成功即得一个订单号，这时再去支付页面完成订单的后续操作。

1.6. 名词解释

术语	说明
Memcached	Memcached 是一个高性能的分布式内存对象缓存系统。 Memcached 官方介绍可参见这里。分布式缓存 Memcache 兼容 Memcached 二进制协议和文本协议两种方式。
命中率	用户读取成功的成功次数/用户读取次数。

免用户名密码访问	指用户可以在已获授权的天翼云云主机上，无需用户名密码即可访问对应的分布式缓存 Memcache。
SASL	SASL 全称 Simple Authentication and Security Layer，是一种用来扩充 C/S 模式验证能力的机制。Memcached 从 1.4.3 版本开始，支持 SASL 认证。由于分布式缓存 Memcache 的多租户共享特性，也采用 SASL 作为鉴权机制。SASL 本质上是使用密码保证的缓存数据安全，建议采用强密码和定期修改密码的策略。分布式缓存 Memcache 将每 60 秒自动进行一次鉴权。

2. 操作指南

2.1. 登录

在创建分布式缓存 Memcache 实例前，您需要创建一个天翼云账号。

1. 登录成功后，导航栏选择“云计算>数据库>分布式缓存 Memcache”，即可开始创建分布式缓存 Memcache 实例。

2.2 订购相关

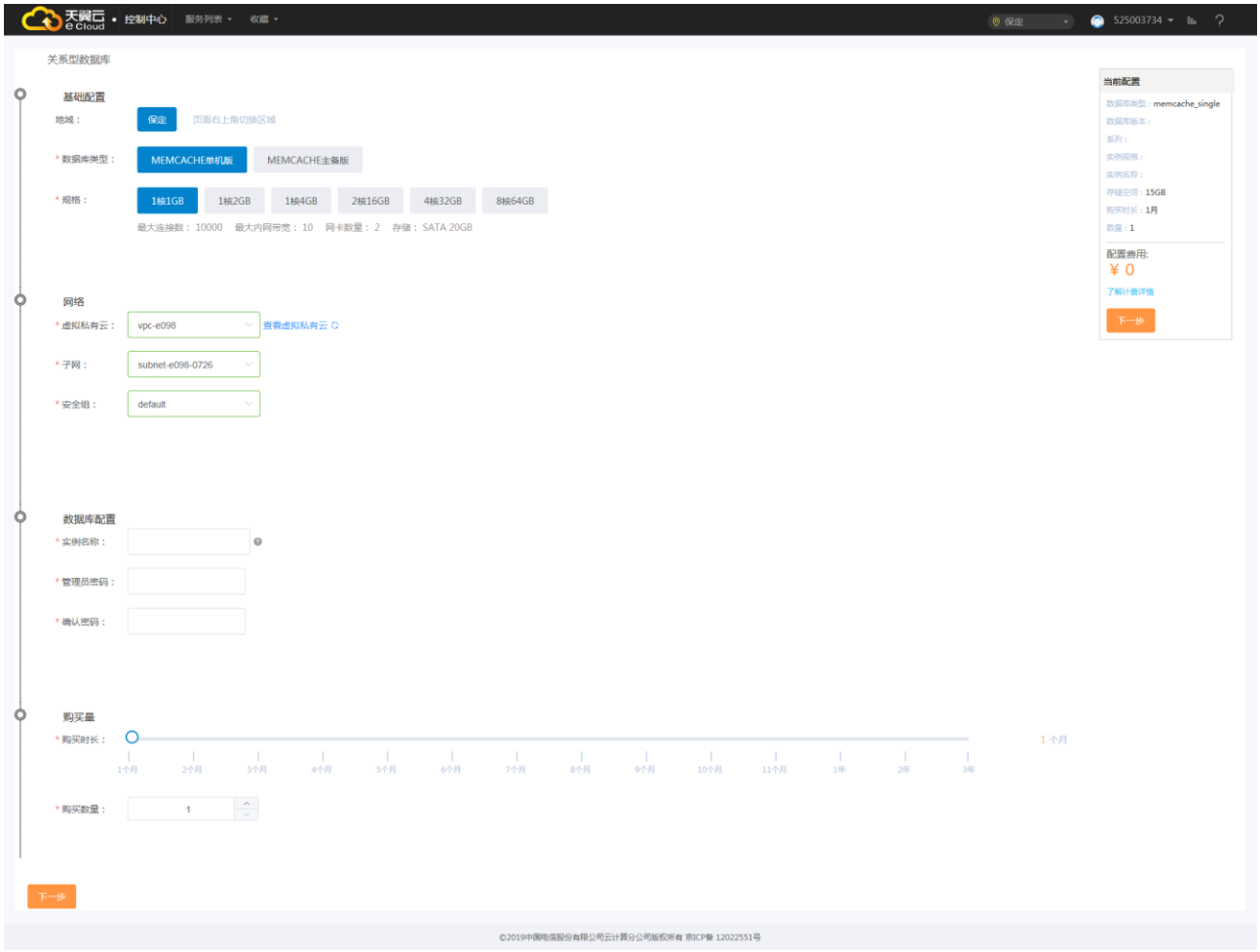
2.2.1 使用须知

用户可以根据自己的业务需求，选择创建 Memcache 实例，规格如下表所述。

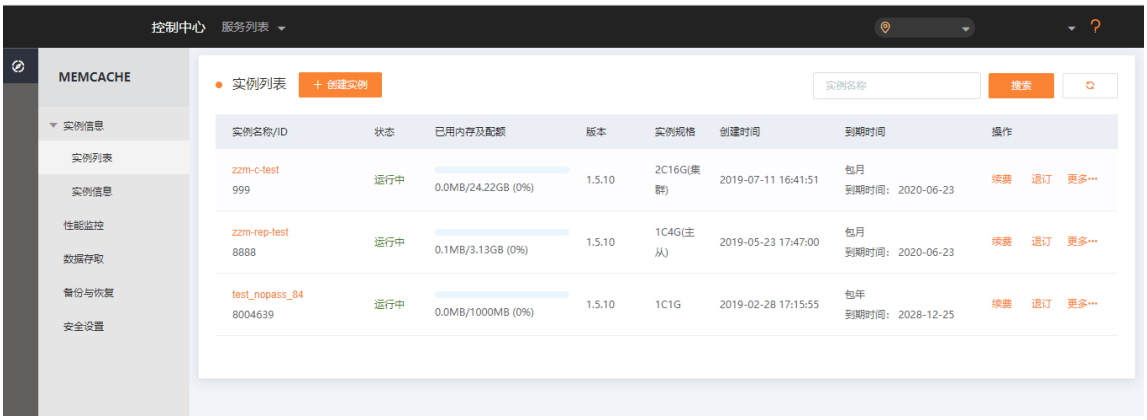
项目	描述
缓存容量	Memcache 支持 1GB、2GB、4GB、8GB、16GB、32GB、64GB
端口	Memcache 默 认 11211

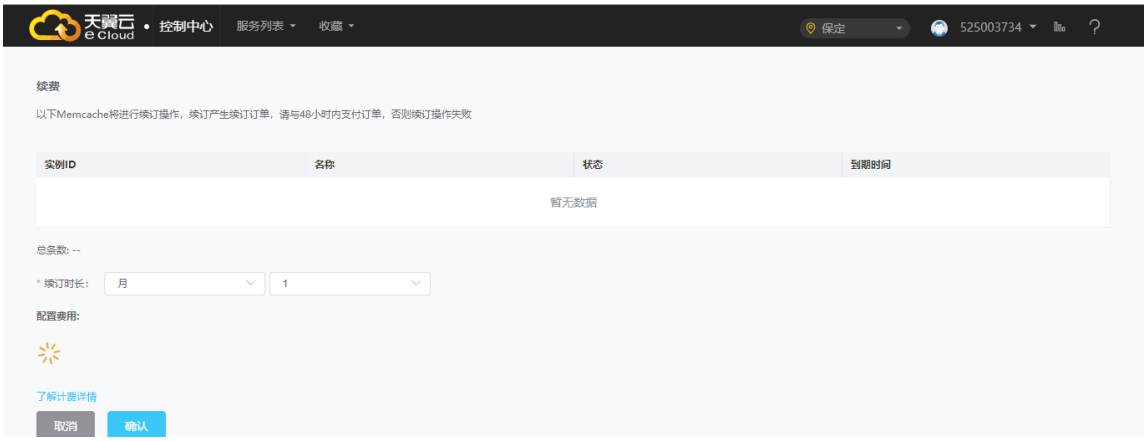
注意：创建的 memcache 实例需要在租户的云主机可以访问的条件下才能使用。

2.2.2 创建 Memcache 实例


关系型数据库
基础配置
地域：
保定
* 数据库类型：
MEMCACHE单机版
MEMCACHE主备版
* 规格：
1核1GB
1核2GB
1核4GB
2核16GB
4核32GB
8核64GB
网络
* 虚拟私有云：
vpc-e098
* 子网：
subnet-e098-0726
* 安全组：
default
数据库配置
* 实例名称：
* 管理员密码：
* 确认密码：
购买量
* 购买时长：
* 购买数量：
下一步
当前配置
配置费用:
¥ 0

2.2.3 续费


控制中心
服务列表
MEMCACHE
实例信息
实例列表
实例信息
性能监控
数据存取
备份与恢复
安全设置
实例列表
+ 创建实例
搜索
实例名称/ID
状态
已用内存及配额
版本
实例规格
创建时间
到期时间
操作
zzm-c-test
999
运行中
0.0MB/24.22GB (0%)
1.5.10
2C16G(集群)
2019-07-11 16:41:51
包月
到期时间：2020-06-23
续费
退订
更多
zzm-rep-test
8888
运行中
0.1MB/3.13GB (0%)
1.5.10
1C4G(主从)
2019-05-23 17:47:00
包月
到期时间：2020-06-23
续费
退订
更多
test_nopass_84
8004639
运行中
0.0MB/1000MB (0%)
1.5.10
1C1G
2019-02-28 17:15:55
包年
到期时间：2028-12-25
续费
退订
更多



2.2.4 规格变更

1. 支持对单机版和主备版的实例扩容，只允许扩容，不能缩容。实例扩容完成后，即刻按照新的实例规格进行计费。

2.3 管理 Memcache 实例

2.3.1 修改实例信息

1. 进入“实例列表”，选择需要修改的实例，进入实例“信息界面”。
2. 点击“修改”可以修改实例名称，点击“修改密码”可以修改实例的连接密码。

2.3.2 清空实例

1. 进入“实例列表”页面，下拉“更多...”选择需要清空实例，点击清除数据，二次确认后可以清空实例。

说明： 清空实例后，实例数据将无法恢复且实例状态恢复为运行中，请谨慎操作。

2.3.3 性能监控

监控项	描述
key 信息	当前总数、历史总数、历史过期数、历史淘汰数
内存使用量	用户程序使用缓存大小的实时信息。
连接数	连接到缓存的连接数。
CPU 使用率	运行中的程序占用的 CPU 资源与 CPU 的峰值 计算能力的比。
key 命中率	用户程序运行时，从缓存内调取的数据量 与调用的数据总量的比。
网络流量	缓存部分与其他服务之间数据交换的数据量。

1. 点开“性能监控”页面，查看该实例的各项指标是否符合要求。

2. 点击“实时监控信息”页签 ，查看实例的各项指标是否符合要求。

“实时更新”每隔五秒钟页面会更新数据，

“图表联动”拖动任何一个时间轴，所有时间轴跟着联动。

3. 点击“历史监控信息”页签，查看实例的各项指标的历史情况。用户可以自定义指标参数。

2.3.4 数据存取

1. 点开"数据存取"页面，进入“读取数据”tab 页，输入 KEY 值，点击查询按钮，查询出 KEY 对应的 VALUE 值，如果 KEY 不存在，则返回空。

2. 点开"数据存取"页面，进入“写入数据”tab 页，输入 KEY 值和 VALUE 值，选择或输入过期时间，点击保存按钮。如果 KEY 不存在，则保存成功，如果 KEY 存在则弹出提示用户是否继续更新保存。

3. 点开"数据存取"页面，进入“计数命令”tab 页，递增操作：输入 KEY 值和 VALUE 值，点击保存按钮，新 VALUE=旧 VALUE+递增 VALUE。递减操作：输入 KEY 值和 VALUE 值，点击保存按钮，新 VALUE=旧 VALUE-递减 VALUE。条件限制：VALUE 必需为数值，递减的时候 VALUE 最小只能到达 0。

2.3.5 备份与恢复

1. 创建手动备份，可以实时备份数据实例数据。

2. 自动设置备份时间节点，可以定时佩服实例数据。

3. 选择已经备份的文件进行数据恢复。

2.3.6 安全设置

用户在申请 Memcache 实例后，可通过进入“安全设置”页面对实例进行访问管理控制。

通过以下步骤即可快速添加本实例列表访问的白名单，此白名单只能针对用户开通的 ECS 云主机进行设置。

1. 点击“添加白名单分组”。输入分组名，添加 ECS 云主机 IP。

2. 在白名单分组列表中，显示用户已经添加好的 ECS 云主机 IP。点击“修改”按钮可以修改选中的 ECS 云主机 IP，点击“新增 IP”按钮可以增加新的 ECS 云主机 IP。

3. 在白名单分组列表中，选择需要删除的 ECS 云主机 IP 分组，点击“删除”按钮，从白名单中删除选中的分组。此时弹出提示框“删除后，这个分组内的所有 IP 都无法访问该实例，是否确认删除?”，确定删除点击“确定”，放弃删除点击“取消”。

3. 连接方式

3.1 JAVA: Spymemcache

3.1.1 JAVA 代码示例

Memcache 集群模式在客户端控制，一致性哈希算法可以有效防止单机或者多机掉线引发的大量数据迁移。

需要用户名密码

```
import lombok.extern.slf4j.Slf4j;
import net.spy.memcached.ConnectionFactoryBuilder;
import net.spy.memcached.DefaultHashAlgorithm;
import net.spy.memcached.MemcachedClient;
import net.spy.memcached.auth.AuthDescriptor;
import net.spy.memcached.auth.PlainCallbackHandler;
import net.spy.memcached.internal.OperationFuture;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.util.ArrayList;
import java.util.concurrent.ExecutionException;

@Slf4j
public class CtyunMemTest {

    public static void main(String[] args) {
        String host1 = "****";   //1 号机 ip
        int port1 = 11211;
        InetSocketAddress inetSocketAddress1 = new InetSocketAddress(host1, port1);
        String hostn = "****";   //n 号机 ip
        int portn = 11211;
        InetSocketAddress inetSocketAddressn = new InetSocketAddress(hostn, portn);
        ArrayList<InetSocketAddress> addressList = new ArrayList<>();
        addressList.add(inetSocketAddress1);
```

```
        addressList.add(inetSocketAddressn);
        ConnectionFactoryBuilder cBuilder = new
ConnectionFactoryBuilder().setProtocol(ConnectionFactoryBuilder.Protocol.BINARY).setHashAlg
(DefaultHashAlgorithm.KETAMA_HASH);     //这里通过设置
setHashAlg(DefaultHashAlgorithm.KETAMA_HASH)，将集群模式选择为一致性 hash。单机情况不用设置
setHashAlg

        //集群中不同主机的用户名和密码需要一致
        String username = "****";  //用户名
        String password = "****";  //密码
        AuthDescriptor ad = new AuthDescriptor(new String[]{"PLAIN"},
        new PlainCallbackHandler(username,password));
        cBuilder.setAuthDescriptor(ad);
        MemcachedClient memcachedClient = null;
        try {
            memcachedClient = new MemcachedClient(cBuilder.build(), addressList);
            OperationFuture<Boolean> operationFuture = memcachedClient.set("ctyun", 60,
"from ctyun");
            Boolean result = operationFuture.get();
            if(result){
                log.info("set success");
                log.info("get value is : {}", memcachedClient.get("ctyun"));
            }else{
                log.info("set fault");
            }

        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
```

```
        } finally {
            if(null != memcachedClient){
                memcachedClient.shutdown();
            }
        }
    }
}
```

不需要用户名密码

```
import lombok.extern.slf4j.Slf4j;
import net.spy.memcached.ConnectionFactoryBuilder;
import net.spy.memcached.DefaultHashAlgorithm;
import net.spy.memcached.MemcachedClient;
import net.spy.memcached.internal.OperationFuture;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.util.ArrayList;
import java.util.concurrent.ExecutionException;

@Slf4j
public class CtyunMemTest {

    public static void main(String[] args) {
        String host1 = "****";   //1 号机 ip
        int port1 = 11211;
        InetSocketAddress inetSocketAddress1 = new InetSocketAddress(host1, port1);
        String hostn = "****";   //n 号机 ip
        int portn = 11211;
        InetSocketAddress inetSocketAddressn = new InetSocketAddress(hostn, portn);
        ArrayList<InetSocketAddress> addressList = new ArrayList<>();
        addressList.add(inetSocketAddress1);
```

```
        addressList.add(inetSocketAddressn);
        ConnectionFactoryBuilder cBuilder = new
ConnectionFactoryBuilder().setHashAlg(DefaultHashAlgorithm.KETAMA_HASH);
                    //这里通过设置 setHashAlg(DefaultHashAlgorithm.KETAMA_HASH)，将集群模式选
择为一致性 hash 模式。单机情况不用设置 setHashAlg


        MemcachedClient memcachedClient = null;
        try {
            memcachedClient = new MemcachedClient(cBuilder.build(), addressList);
            OperationFuture<Boolean> operationFuture = memcachedClient.set("ctyun", 60,
"from ctyun");
            Boolean result = operationFuture.get();
            if(result){
                log.info("set success");
                log.info("get value is : {}", memcachedClient.get("ctyun"));
            }else{
                log.info("set fault");
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        } finally {
            if(null != memcachedClient){
                memcachedClient.shutdown();
            }
        }
    }
}
```

输出结果

```
set success
get value is : from ctyun
```

3.2PHP Memcached example

PHP 使用 memcached 需要安装 php-memcached 扩展，安装配置以及接口文档可以参考 https://www.php.net/manual/zh/book.memcached.php

3.2.1.PHP Memcached example

```
<?php
    $mem  = new Memcached();
    $mem->addServer('127.0.0.1',11211);
    $mem->set('foo', 100);
        var_dump($mem->get('foo'));
    ?>
```

3.1.3.PHP Memcached SASL example

```
<?php
        /**
        *当 memcached 设置了 sasl 加密时使用 setSaslAuthData 设置 sasl 校验信息，
Memcached::setSaslAuthData ( string $username , string $password )，$username 用户名，
$password 密码
        **/
    $mem  = new Memcached();
    $mem->addServer('127.0.0.1',11211);
        $mem->setSaslAuthData ( 'memcache' , '11111111' );
        $mem->setOption(Memcached::OPT_BINARY_PROTOCOL, true);
    $mem->set('foo', 100);
        var_dump($mem->get('foo'));
?>
```

3.3 Python

需要安装 bmemcached Package，pip install python-binary-memcached

3.3.1. Python 代码示例

```
import bmemcached

client = bmemcached.Client(('127.0.0.1:11211', ))

client.set('foo', '100')

print(client.get('foo'))
```

3.3.2. Python 链接配置了 sasl 验证的 memcached 示例

```
import bmemcached

# set sasl auth info username and password

client = bmemcached.Client(('127.0.0.1:11211'), 'username', 'password')

client.set('foo', '100')

print(client.get('foo'))
```

3.4 C#/.NET: EnyimMemcached

3.4.1. C#/.NET 代码示例

```
namespace ECS.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_ecs_host").AddressList[0].ToString()); //your_ecs_host 替换为 ECS 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);
                  }
          }
}
```

3.5 C++

需要安装 libmemcached

3.5.1. C++ 代码示例

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

int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;
  const char *key= "foo";
  const char *value= "100";
  size_t value_length;
  time_t expiration = (time_t)0;
  uint32_t  flags = 0;

  memcached_server_st *memcached_servers_parse (char *server_strings);
  memc= memcached_create(NULL);
  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
  rc= memcached_server_push(memc, servers);
  if (rc == MEMCACHED_SUCCESS)
      cout<<"Added server successfully!"<<endl;
  else
      cout<<"Couldn't add server:"<<memcached_strerror(memc, rc)<<endl;

  rc= memcached_set(memc, key, strlen(key), value, strlen(value), expiration, flags);
  if (rc == MEMCACHED_SUCCESS)
      cout<<"Key stored successfully!"<<endl;
  else
      cout<<"Couldn't store key:"<<memcached_strerror(memc, rc)<<endl;
```

```
    char* result = memcached_get(memc,key,strlen(key),&value_length,&flags,&rc);
    if(rc == MEMCACHED_SUCCESS)
        cout<<"Get value:"<<result<<" sucessful!"<<endl;
    else
        cout<<"Couldn't Get value:"<<memcached_strerror(memc, rc)<<endl;
    memcached_free(memc);
    return 0;
}
```

4. 常见问题

4.1 性能类问题

4.1.1 已经使用了存储服务，为什么还要使用缓存服务？

在数据请求相同的条件下，缓存服务的响应速度远远高于存储服务，但是其容量相对较小， 且价格相对较高。缓存服务一般用于热点数据的访问加速

4.1.2 使用天翼云 Memcache，原来的系统会有多大的修改？

Memcache 兼容原生 API，让代码的改动量降至最低。

4.1.3 天翼云 Memcache 实例扩缩容时有什么限制和注意事项？

客户在原来 Memcache 实例扩容内存，客户确定扩容后，由天翼云后端对 Memcache 主机升级，升级过程中主机需要重启，Memcache 数据会丢失。

4.2 安全类问题

4.2.1 会不会有人在 Memcache 上恶意获取属于我的数据？

不会。 用户需通过白名单的方式进行访问，且白名单中只能添加内网的 ECS 主机，防控外网攻击与恶意获取。 在 VPC 内部，实行严格的单用户数据隔离。每个用户只能访问自己的缓存数据。

4.2.2 Memcache 崩溃了怎么办，会不会影响到正常运营？

单机版：会。

主从版：不会，"主"崩溃了，"从"立即提升为"主"对外提供服务，后端服务会重启崩溃的 Memcache 实例为"新从"，并且从"新主"同步数据到"新从"。

4.2.3 为什么天翼云 Memcache 不提供 SSH 直接登录访问？

天翼云的分布式缓存是以实例的方式提供使用，用户直接进行读写操作即可。

4.2.4 连接地址

1、用于连接分布式缓存 Memcache 的 Host 地址以 IP 展示，可在实例信息>基本信息>连接信息>连接地址中查询到。
2、端口 11211 不能修改。

4.2.5 连接密码

用于连接分布式缓存 Memcache 的密码。可在购买时设置，或者在购买后重置密码。

4.3 其他问题

4.1.1 使用 Memcache 会增加多少成本？

1.根据不同的使用容量，Memcache 服务提供包月计费。相对于自己搭建 Memcache，省去了搭建、维护等诸多成本，同时 Memcache 还能提供完备的主备故障处理等成熟技术。

2.如果存在热点数据，相较于磁盘，Memcache 的成本低效果好。