

## 第 2 章

# 高性能内存对象缓存 Memcached

### 技能目标

- 理解 Memcached 核心概念
- 会进行 Memcached 相关部署操作
- 会进行 Memcached 主主复制操作
- 会进行 Memcached 服务高可用配置

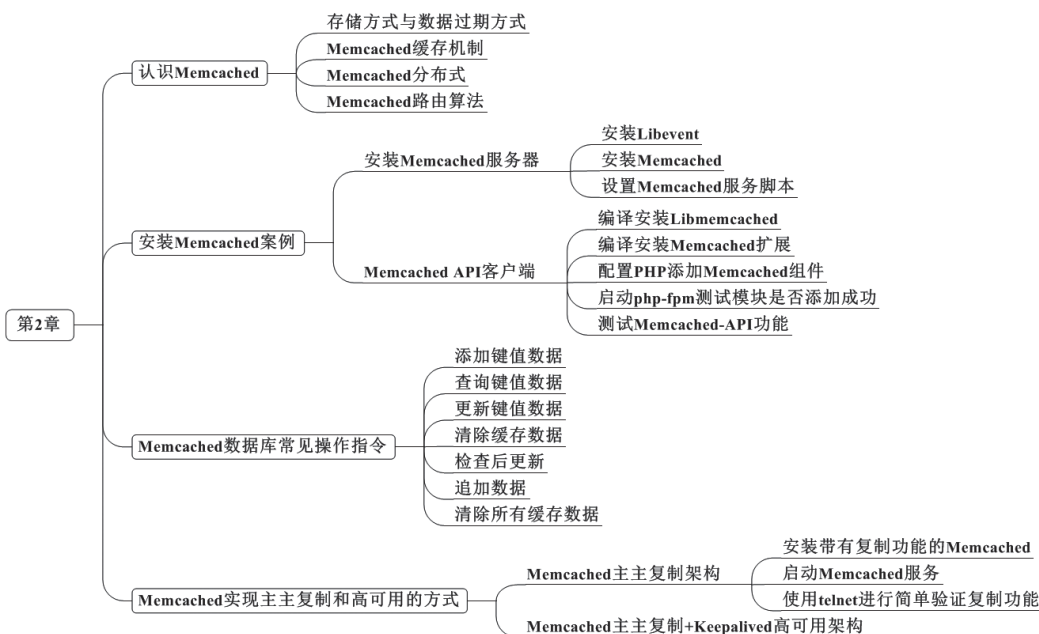
### 本章导读

Memcached 是一套开源的高性能分布式内存对象缓存系统，它将所有的数据都存储在内存中，因为在内存中会统一维护一张巨大的 Hash 表，所以支持任意存储类型的数据。很多网站使用 Memcached 以提高网站的访问速度，尤其是对于大型的需要频繁访问数据的网站。

Memcached 主主复制是指在任意一台 Memcached 服务器修改数据都会被同步到另外一台，可以使用 keepalived 提供高可用架构。

### 知识服务





## 2.1 认识 Memcached

Memcached 是一套开源的高性能分布式内存对象缓存系统，它将所有的数据都存储在内存中，因为在内存中会统一维护一张巨大的 Hash 表，所以支持任意存储类型的数据。很多网站通过使用 Memcached 提高网站的访问速度，尤其是对于大型的需要频繁访问数据的网站。

Memcached 是典型的 C/S 架构，因此需要安装 Memcached 服务端与 Memcached API 客户端。Memcached 服务端是用 C 语言编写的，而 Memcached API 客户端可以用任何语言来编写，如 PHP、Python、Perl 等，并通过 Memcached 协议与 Memcached 服务端进行通信。常用典型架构如图 2.1 所示。

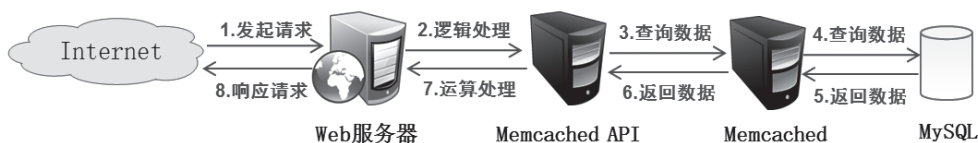


图 2.1 Memcached 常用架构

当 Web 客户端发送请求到 Web 服务器的应用程序时，应用程序会通过调用 Memcached API 客户端程序库接口去连接 Memcached 服务器，进而查询数据。如果此时 Web 客户端所请求的数据已经在 Memcached 服务端中缓存，则 Memcached 服务端会将数据返回给 Web 客户端；如果数据不存在，则会将 Web 客户端请求发送至 MySQL 数据库，由数据库将请求的数据返回给 Memcached 以及 Web 客户端，与此同

时 Memcached 服务器也会将数据进行保存，以方便用户下次请求使用。

### 1. 存储方式与数据过期方式

Memcached 具有独特的存储方式和数据过期方式。

#### (1) 数据存储方式：Slab Allocation

Slab Allocation 即按组分配内存，每次先分配一个 Slab，相当于一个大小为 1MB 的页，然后在 1MB 的空间里根据数据划分大小相同的 Chunk，如图 2.2 所示。该方法可以有效解决内存碎片问题，但可能会对内存空间有所浪费。

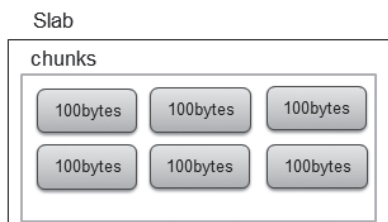


图 2.2 Slab Allocation

#### (2) 数据过期方式：LRU、Laxzy Expiration

LRU 是指追加的数据空间不足时，会根据 LRU 的情况淘汰最近最少使用的记录。Laxzy Expiration 即惰性过期，是指使用 get 时查看记录时间，从而检查记录是否已经过期。

### 2. Memcached 缓存机制

缓存是常驻在内存的数据，能够快速进行读取。而 Memcached 就是这样一款非常出色的缓存软件，当程序写入缓存数据请求时，Memcached 的 API 接口将 Key 输入路由算法模块路由到集群中一台服务器，之后由 API 接口与服务器进行通信，完成一次分布式缓存写入，如图 2.3 所示。

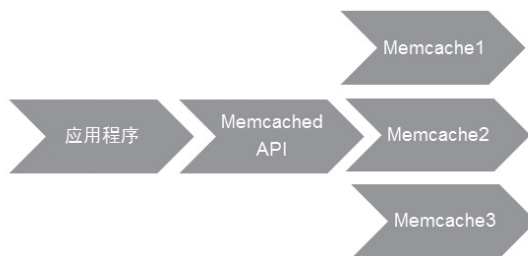


图 2.3 Memcached 缓存机制

### 3. Memcached 分布式

Memcached 分布式部署主要依赖于 Memcached 的客户端来实现，多个 Memcached 服务器是独立的。分布式数据如何存储是由路由算法所决定的。

当数据到达客户端程序库时，客户端的算法就依据路由算法来决定保存的 Memcached 服务器。读取数据时，客户端依据保存数据时的路由算法选中和存储数据时相同的服务器来读取数据，如图 2.4 所示。

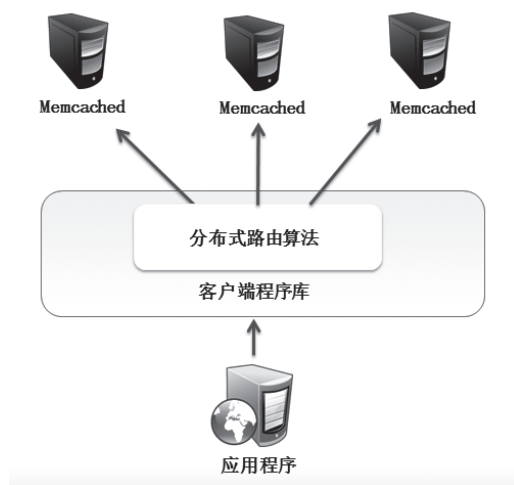


图 2.4 Memcached 分布式

#### 4. Memcached 路由算法

##### (1) 求余数 hash 算法

求余数 hash 算法先用 key 做 hash 运算得到一个整数，再去做 hash 算法，根据余数进行路由。这种算法适合大多数数据需求，但是不适合用在动态变化的环境中，比如有大量机器添加或者删除时，会导致大量对象的存储位置失效。

##### (2) 一致性 hash 算法

一致性 hash 算法适合在动态变化的环境中使用。原理是按照 hash 算法把对应的 key 通过一定的 hash 算法处理后，映射形成一个首尾相接的闭合循环，然后通过使用与对象存储一样的 hash 算法将机器也映射到环中，按顺时针方向将所有对象存储到离自己最近的机器中，如图 2.5 所示。

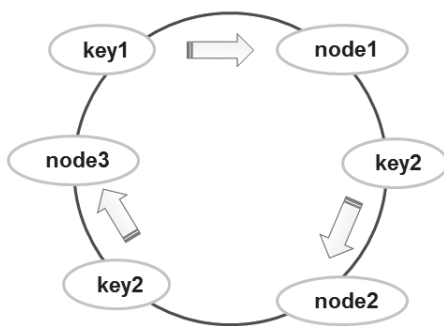


图 2.5 一致性 hash 算法

## 2.2 安装 Memcached 案例

本案例使用两台 CentOS7.3 系统完成，一台是 Memcached 服务器，另一台是基于 LAMP 架构进行 Memcached 扩展的 Memcached API 客户端，可以根据企业需求进行架构调整。案例环境如表 2-1 所示。

表 2-1 案例环境

名称	IP 地址	角色	主要软件包
Memcached	192.168.1.105	Memcached 服务器	libevent-1.4.12-stable.tar.gz memcached-1.4.31.tar.gz
Memcached API	192.168.1.102	Memcached API 客户端	httpd-2.4.25.tar.gz php-5.6.30.tar.gz libmemcached-1.0.18.tar.gz memcached-2.2.0.tgz

### 2.2.1 安装 Memcached 服务器

#### 1. 安装 Libevent

Libevent 是一款跨平台的事件处理接口的封装，可以兼容多个操作系统的事件访问。Memcached 的安装依赖于 Libevent，因此需要先完成 Libevent 的安装。

```
[root@memcached ~]# wget https://github.com/downloads/libevent/libevent/libevent-1.4.14b-stable.tar.gz
[root@memcached ~]# tar xzvf libevent-1.4.14b-stable.tar.gz
[root@memcached libevent-1.4.14b-stable]# ./configure --prefix=/usr/local/libevent
[root@memcached libevent-1.4.14b-stable]# make && make install
```

到此 Libevent 安装完毕，接下来就可以开始安装 Memcached。

#### 2. 安装 Memcached

采用源码的方式进行 Memcached 的编译安装，安装时需要指定 Libevent 的安装路径。

```
[root@memcached ~]# wget http://www.memcached.org/files/memcached-1.4.31.tar.gz
[root@memcached ~]# tar xzvf memcached-1.4.31.tar.gz
[root@memcached ~]# cd memcached-1.4.31/
[root@memcached memcached-1.4.31]# ./configure --prefix=/usr/local/memcached --with-libevent=/usr/local/libevent
[root@memcached memcached-1.4.31]#make && make install
```

#### 3. 设置 Memcached 服务脚本

Memcached 服务器安装完成后，可以使用安装目录下的 bin/memcached 来启动服

务，但是为了更加方便地管理 Memcached，还是编写脚本来管理 Memcached 服务。

```
[root@memcached ~]# vi /usr/local/memcached/memcached_service.sh
#!/bin/bash
CMD="/usr/local/memcached/bin/memcached"
start(){
    $CMD -d -m 128 -u root
}
stop(){
    killall memcached;
}

ACTION=$1
case $ACTION in
'start')
    start;;
'stop')
    stop;;
'restart')
    stop
    sleep 2
    start;;
*)
    echo 'Usage: {start|stop|restart}'
esac
esac
```

其中启动 Memcached 时，`-d` 选项的作用是以守护进程的方式运行 Memcached 服务；`-m` 是为 Memcached 分配 128MB 的内存，应根据企业需要进行调整；`-u` 指定运行的用户账号。

之后设置脚本权限，启动 Memcached 服务。

```
[root@memcached ~]# chmod 755 /usr/local/memcached/memcached_service.sh
[root@memcached ~]# /usr/local/memcached/memcached_service.sh start
```

服务启动之后，监听 11211/tcp 端口。

```
[root@memcached ~]# netstat -antp |grep memcached
tcp    0    0 0.0.0.0:11211  0.0.0.0:*    LISTEN 10196/memcached
tcp6   0    0 :::11211      :::*         LISTEN  10196/memcached
```

## 2.2.2 Memcached API 客户端

为了使得程序可以直接调用 Memcached 库和接口，可以使用 Memcached 扩展组件将 Memcached 添加为 PHP 的一个模块。此扩展使用了 Libmemcached 库提供的 API

与 Memcached 服务端进行交互。

### 1. 编译安装 Libmemcached

在编译 Memcached 扩展组件时，需要指定 Libmemcached 库的位置，所以先安装 Libmemcached 库。

```
[root@memcached-api ~]# wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz
[root@memcached-api ~]# tar xzvf libmemcached-1.0.18.tar.gz
[root@memcached-api ~]# cd libmemcached-1.0.18/
[root@memcached-api libmemcached-1.0.18]# ./configure --prefix=/usr/local/libmemcached --with-memcached=/usr/local/memcached
[root@memcached-api libmemcached-1.0.18]# make && make install
```

### 2. 编译安装 Memcached 扩展

然后就可以进行 PHP 的 Memcached 扩展组件安装。

```
[root@memcached-api ~]# wget http://pecl.php.net/get/memcached-2.2.0.tgz
[root@memcached-api ~]# tar xzvf memcached-2.2.0.tgz
[root@memcached-api ~]# cd memcached-2.2.0/
```

注意配置 Memcached API 时，memcached-2.2.0.tgz 源码包中默认没有 configure 配置脚本，需要使用 PHP 的 phpize 脚本生成配置脚本 configure。

```
[root@memcached-api memcached-2.2.0]# /usr/local/php/bin/phpize
Configuring for:
PHP Api Version: 20131106
Zend Module Api No: 20131226
Zend Extension Api No: 220131226
[root@memcached-api memcached-2.2.0]# ./configure --enable-memcached --with-php-config=/usr/local/php/bin/php-config --with-libmemcached-dir=/usr/local/libmemcached --disable-memcached-sasl
```

#### 注意

配置时使用 `--disable-memcached-sasl` 选项关闭 Memcached 的 SASL 认证功能，否则会报错。

```
[root@memcached-api memcached-2.2.0]# make
[root@memcached-api memcached-2.2.0]# make test
[root@memcached-api memcached-2.2.0]# make install
Installing shared extensions: /usr/local/php/lib/php/extensions/no-debug-zts-20131226/
// 共享组件的位置
```

### 3. 配置 PHP 添加 Memcached 组件

编辑 PHP 配置文件 `php.ini`，添加 Memcached 组件。

```
[root@memcached-api ~]# cd /usr/local/php/
[root@memcached-api php]# vi etc/php.ini
添加如下内容
extension_dir = "/usr/local/php/lib/php/extensions/no-debug-zts-20131226/"
extension=memcached.so
```

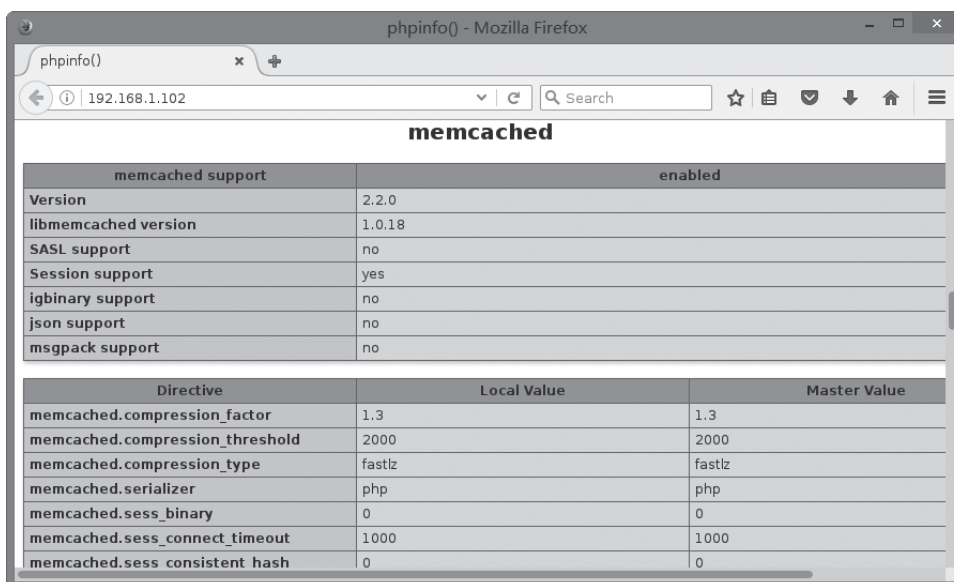
#### 4. 启动 php-fpm 测试模块是否添加成功

```
[root@memcached-api php]# cp etc/php-fpm.conf.default etc/php-fpm.conf
[root@memcached-api php]# ./sbin/php-fpm -c etc/php.ini -y etc/php-fpm.conf
[root@memcached-api php]# ps aux |grep php
root    83801  0.0  0.1 159340 3780 ?      Ss   08:51   0:00 php-fpm: master process (etc/php-fpm.conf)
nobody  83802  0.0  0.1 161424 3700 ?      S   08:51   0:00 php-fpm: pool www
nobody  83803  0.0  0.1 161424 3700 ?      S   08:51   0:00 php-fpm: pool www
root    83813  0.0  0.0 112648  964 pts/1  S+   08:51   0:00 grep --color=auto php
```

可以通过 `phpinfo()`，查看是否已经添加 Memcached 扩展模块。

```
[root@memcached ~]# vi /usr/local/apache/htdocs/index.php
<?php
phpinfo();
?>
```

使用浏览器进行访问，结果如图 2.6 所示，已经添加成功。



memcached support		enabled	
Version	2.2.0		
libmemcached version	1.0.18		
SASL support	no		
Session support	yes		
igbinary support	no		
json support	no		
msgpack support	no		

Directive	Local Value	Master Value
memcached.compression_factor	1.3	1.3
memcached.compression_threshold	2000	2000
memcached.compression_type	fastlz	fastlz
memcached.serializer	php	php
memcached.sess_binary	0	0
memcached.sess_connect_timeout	1000	1000
memcached.sess_consistent_hash	0	0

图 2.6 phpinfo 信息

#### 5. 测试 Memcached API 功能

通过编写简单的 PHP 测试代码调用 Memcache 程序接口，来测试是否与



Memcached 服务器协同工作，代码如下：

```
[root@memcached-api ~]# vi /usr/local/apache/htdocs/test.php
<?php
$memcache = new Memcached();
$memcache->addServer('192.168.1.105', 11211);
$memcache->set('key', 'Memcache test successful!', 0, 60);
$result = $memcache->get('key');
unset($memcache);
echo $result;
?>
```

此段代码的作用是在客户端连接 Memcached 服务器，设置名为 'key' 的键的值为 'Memcache test successful!'，并读取显示。显示成功，则表示服务器与客户端协同工作正常。使用浏览器进行访问，测试结果如图 2.7 所示。

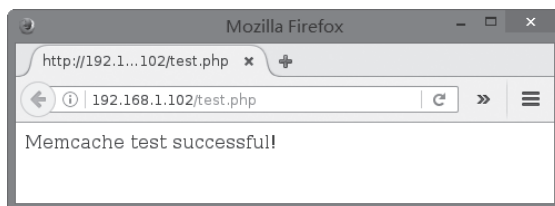


图 2.7 测试页面

## 2.3 Memcached 数据库操作与管理

Memcached 协议简单，可直接使用 telnet 连接 Memcached 的 11211 端口，对 Memcached 数据库进行操作与管理。

```
[root@memcached ~]# telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
// 输入操作指令
```

操作命令格式：

```
<command name><key><flags><exptime><bytes><data block>
```

### 1. 常见操作指令

#### (1) 添加一条键值数据

```
add username 0 0 7
example
STORED
```

其中 `add username 0 0 7` 表示键值名为 `username`，标记位表示自定义信息为 `0`，过期时间为 `0`（永不过期，单位为秒），字节数为 `7`。`example` 为键值，注意输入长度为 `7` 字节，与设定值相符合。

### （2）查询键值数据

```
get username
VALUE username 0 7
example
END
gets username
VALUE username 0 7 4
example
END
```

其中 `get` 后跟键值名，如果检查最近是否更新，可以使用 `gets`，最后一位显示的是更新因子，每更新一次更新因子数会加 `1`。

### （3）更新一条键值数据

```
set username 0 0 10
everything
STORED
get username
VALUE username 0 10
everything
END
```

其中 `set` 后跟需要更新的键值名、标记位、过期时间、字节数。如果键值名不存在，`set` 相当于 `add`。如果仅仅是想单纯地更新没有添加的功能，使用 `replace`。此时更新的键值名必须存在，如果键值名不存在，就会报 `NOT_STORED` 的错误。

```
replace username 0 0 7
lodging
STORED
gets username
VALUE username 0 7 6
lodging
END
replace username1 0 0 7
example
NOT_STORED
```

### （4）清除一条缓存数据

```
delete username
DELETED
get username
END
```

使用 `delete` 删除一条键值为 `username` 的缓存数据，使用 `get` 查看发现没有内容存在。

#### (5) 检查后更新

```
gets username
VALUE username 0 7 7
example
END
cas username 0 0 7 1
lodging
EXISTS
cas username 0 0 7 7
lodging
STORED
gets username
VALUE username 0 7 8
lodging
END
```

如果 `cas` 的最后一个更新因子数与 `gets` 返回的更新因子数相等，则更新，否则返回 `EXISTS`。

#### (6) 追加数据

```
append username 0 0 7 // 后追加 7 字节
example
STORED
get username
VALUE username 0 14
lodgingexample
END
```

在键值名 `username` 的原键值后追加数据使用 `append`。

```
prepend username 0 0 2 // 前追加 2 字节
un
STORED
get username
VALUE username 0 16
unlodgingexample
END
```

在键值名 `username` 的原键值前追加数据使用 `prepend`。

#### (7) 清除所有缓存数据

```
flush_all
OK
```

#### (8) 查看服务器统计信息

```
stats
stats items // 返回所有键值对的统计信息
```

```
stats cachedump 1 0 // 返回指定存储空间的键值对
stats slabs // 显示各个 slab 的信息，包括 chunk 的大小、数目、使用情况等
stats sizes // 输出所有 item 的大小和个数
stats reset // 清空统计数据
```

## 2.4 Memcached 实现主主复制和高可用的方式

Memcached 主主复制是指在任意一台 Memcached 服务器修改数据都会被同步到另外一台，但是 Memcached API 客户端是无法判断连接到哪一台 Memcached 服务器的，所以需要设置 VIP 地址，提供给 Memcached API 客户端进行连接。可以使用 keepalived 产生的 VIP 地址连接主 Memcached 服务器，并且提供高可用架构。

本案例使用两台 Memcached 服务器来完成，实验环境如表 2-2 所示。

表 2-2 案例环境

名称	IP 地址	操作系统	主要软件包
Memcached1	192.168.1.100	Centos 7.3	libevent-1.4.12-stable.tar.gz memcached-1.2.8-repcached-2.2.tar.gz keepalived-1.2.13-8.el7.x86_64
Memcached2	192.168.1.105	Centos 7.3	libevent-1.4.12-stable.tar.gz memcached-1.2.8-repcached-2.2.tar.gz keepalived-1.2.13-8.el7.x86_64

### 2.4.1 Memcached 主主复制架构

Memcached 的复制功能支持多个 Memcached 之间进行相互复制（双向复制，主备都是可读可写的），可以解决 Memcached 的容灾问题。

要使用 Memcached 复制架构，需要重新下载支持复制功能的 Memcached 安装包。  
<http://downloads.sourceforge.net/repcached/memcached-1.2.8-repcached-2.2.tar.gz>

安装过程与之前安装的 Memcached 方法相同，下面简略描述一下。

#### 1. 安装带有复制功能的 Memcached

安装完成 Libevent 之后，将下载的 memcached-1.2.8-repcached-2.2.tar.gz 进行解压，然后完成编译安装。

```
[root@memcached1 ~]# cd memcached-1.2.8-repcached-2.2/
[root@memcached1 memcached-1.2.8-repcached-2.2]# ./configure --prefix=/usr/local/
memcached_replication --enable-replication --with-libevent=/usr/local/libevent
[root@memcached1 memcached-1.2.8-repcached-2.2]# make && make install
```

#### 2. 启动 Memcached 服务

支持复制功能的 Memcached 安装完成之后，需要将编译安装的 libevent-1.4.so.2

模块复制到 /usr/lib64 目录下，否则在启动带有复制功能的 Memcached 服务时会报错。

```
[root@memcached1 ~]# ln -s /usr/local/libevent/lib/libevent-1.4.so.2 /usr/lib64/libevent-1.4.so.2
```

启动服务时，使用 -x 选项指向对端。

```
[root@memcached1 ~]# /usr/local/memcached_replication/bin/memcached -d -u root -m 128
-x 192.168.1.105
```

```
[root@memcached1 ~]# netstat -antp |grep memcached
tcp    0    0 0.0.0.0:11211      0.0.0.0:*        LISTEN  8163/memcached
tcp    0    0 0.0.0.0:11212      0.0.0.0:*        LISTEN  8163/memcached
tcp6   0    0 :::11211          :::*              LISTEN  8163/memcached
```

同样启动 Memcached2 服务器，注意启动 Memcached 服务时指向对端。

### 3. 使用 telnet 进行简单验证复制功能

(1) 在 Memcached1 上插入一条具有特点的键值

```
[root@memcached1 ~]# telnet 192.168.1.100 11211
Trying 192.168.1.100...
Connected to 192.168.1.100.
Escape character is '^]'.
set username 0 0 8
20170226
STORED
get username
VALUE username 0 8
20170226
END
quit
Connection closed by foreign host.
```

(2) 在 Memcached2 上查看刚刚插入的键值

```
[root@memcached2 ~]# telnet 192.168.1.105 11211
Trying 192.168.1.105...
Connected to 192.168.1.105.
Escape character is '^]'.
get username
VALUE username 0 8
20170226
END
get username2
END
quit
Connection closed by foreign host.
```

同理，在 Memcached2 上插入的数据，在 Memcached1 上也可以查看到。这就是 Memcached 的主主复制。

## 2.4.2 Memcached 主主复制 +Keepalived 高可用架构

因为 Memcached 主主复制这种架构，在程序连接时不知道应该连接哪个主服务器，所以需要在前端加 VIP 地址，实现高可用架构。这里用 Keepalived 实现，因而 Keepalived 的作用是用来检测 Memcached 服务器的状态是否正常，如图 2.8 所示。

Keepalived 不断检测 Memcached 主服务器的 11211 端口，如果检测到 Memcached 服务发生宕机或者死机等情况，就会将 VIP 从主服务器移至从服务器，从而实现 Memcached 的高可用性。

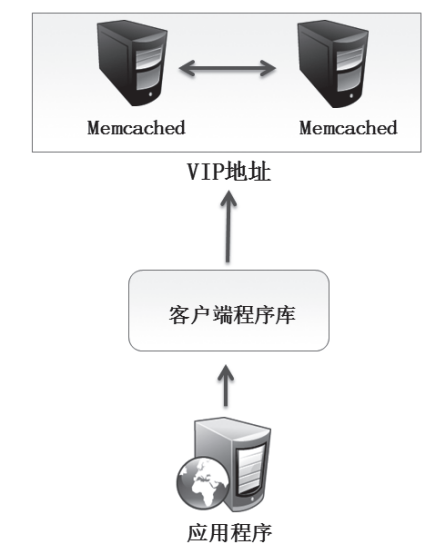


图 2.8 Memcached 高可用架构

### 1. 安装配置 keepalived

```
[root@memcached1 ~]# yum install keepalived
```

#### (1) 配置主 keepalived

```
[root@memcached1 ~]# vi /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
    notification_email {
        admin@example.com
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL // 路由标识，主从保持一致
}
```

```

vrrp_sync_group cluster {
  group {
    mem_ha
  }
}
vrrp_instance mem_ha {
  state MASTER           // 主备状态均为 MASTER
  interface ens33
  virtual_router_id 51   // 虚拟路由 ID, 主从相同
  priority 100           // 优先级, 主的高于备
  advert_int 1
  nopreempt              // 不主动抢占资源, 只在 Master 或者高优先级服务器上进行设置

  authentication {
    auth_type PASS
    auth_pass 1111
  }
  virtual_ipaddress {    // 定义 VIP 地址
    192.168.1.200
  }
}

virtual_server 192.168.1.200 11211 { //VIP 故障检测
  delay_loop 6
  persistence_timeout 20
  protocol TCP
  sorry_server 192.168.1.100 11211 // 对端
  real_server 192.168.1.105 11211 { // 本机
    weight 3
    notify_down /root/memcached.sh // 当 memcached 宕机, 停止 keepalived 服务

    TCP_CHECK {
      connect_timeout 3
      nb_get_retry 3
      delay_before_retry 3
      connect_port 11211
    }
  }
}

```

设置执行脚本如下:

```

[root@memcached1 ~]# echo "/usr/bin/systemctl stop keepalived" > memcached.sh
[root@memcached1 ~]# chmod +x memcached.sh

```

## (2) 配置备 keepalived

主从 keepalived 配置文件内容差不多, 可以直接复制进行修改, 以下只把不一样

的地方整理出来。

```
[root@memcached1 ~]# scp /etc/keepalived/keepalived.conf 192.168.1.105:/etc/keepalived/
// 省略
vrrp_instance mem_ha {
    state MASTER           // 从也使用 MASTER
    interface ens33
    virtual_router_id 51
    priority 90            // 优先级低
    advert_int 1
                            // 去掉 nopreempt

    authentication {
        auth_type PASS
        auth_pass 1111
    }
// 省略
virtual_server 192.168.1.200 11211 {
    delay_loop 6
    persistence_timeout 20
    protocol TCP
    sorry_server 192.168.1.100 11211 // 对端
    real_server 192.168.1.105 11211 { // 本机
        weight 3
        notify_down /root/memcached.sh
    }
// 省略
```

同样设置脚本如下：

```
[root@memcached2 ~]# echo "/usr/bin/systemctl stop keepalived" > memcached.sh
[root@memcached2 ~]# chmod +x memcached.sh
```

## 2. 测试验证

分别启动主从的 keepalived 服务。

```
[root@memcached1 ~]# systemctl start keepalived
[root@memcached2 ~]# systemctl start keepalived
```

(1) 验证主 keepalived 获取 VIP 地址

使用 ip address show 命令查看 VIP 地址（使用 ifconfig 查看不到）。

```
[root@memcached1 ~]# ip address
// 省略
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:1c:8c:62 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.100/24 brd 192.168.1.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet 192.168.1.200/32 scope global ens33 // 已获得 VIP 地址
```



```
valid_lft forever preferred_lft forever
inet6 fe80::20c:29ff:fe1c:8c62/64 scope link
valid_lft forever preferred_lft forever
```

## (2) 验证高可用性

关闭 Memcached1 服务器的 Memcached 服务，在 Memcached2 服务器上查看地址信息。

```
[root@memcached1 ~]# killall memcached
[root@memcached2 ~]# ip addr
// 省略
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
link/ether 00:0c:29:db:af:6a brd ff:ff:ff:ff:ff:ff
inet 192.168.1.105/24 brd 192.168.1.255 scope global ens33
    valid_lft forever preferred_lft forever
inet 192.168.1.200/32 scope global ens33           // 已获取 VIP 地址
    valid_lft forever preferred_lft forever
inet6 fe80::20c:29ff:fedb:af6a/64 scope link
    valid_lft forever preferred_lft forever
```

## 本章总结

- Memcached 是分布式内存对象缓存系统，因为所有数据都存储在内存中，从而常用于网站加速。
- Memcached 分布式实现不是在服务端实现的而是在客户端实现的。
- Memcached 可以通过 keepalived 实现 Memcached 服务的高可用性。