第四十四章：Memcached高性能对象缓存

**一、NoSQL概述；**

**二、Memcached概述；**

**三、运行架构；**

**四、Memcached缓存存储策略；**

**五、Memcached优缺点；**

**六、Memcached的使用；**

**七、案例：Memcached缓存结合lamp缓存搭建；**

**八、Memcached集群原理；**

**九、Memcached主从复制集群；**

**一、NoSQL概述：**

**概述：**NoSQL 数据存储不需要固定的表结构，通常也不存在连接操作。在大数据存取上具备关系型数据库无法比拟的性能优势。随着互联网web2.0网站的兴起，NoSQL数据库现在成了一个极其热门的新领域，NoSQL产品的发展非常迅速.目前Google的 BigTable 和Amazon 的Dynamo使用的就是NoSQL型数据库。

**产品：**

1.HBase：是 Apache Hadoop 中的一个子项目，属于 bigtable 的开源版本，所实现的语言为Java（故依赖 Java SDK）。HBase 依托于 Hadoop 的 HDFS（分布式文件系统）作为最基本存储基础单元；

2.Redis：是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。目前由VMware主持开发工作；

3.MongoDB：是一个高性能，开源，无模式的文档型数据库，开发语言是C++。它在许多场景下可用于替代传统的关系型数据库或键/值存储方式；

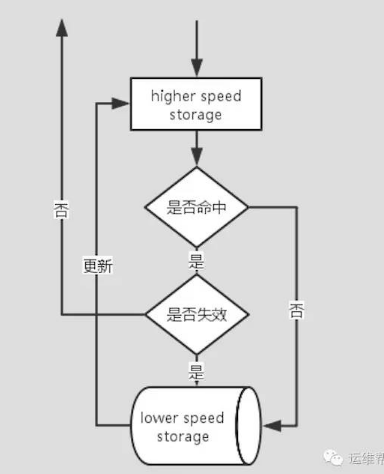
4.Memcached：如下；

**二、Memcached概述：**

**概述：**Memcached是国外社区网站 LiveJournal 的开发团队开发的一套高性能的分布式内存对象缓存服务器。它将所有的数据统统保存在内存中，在内存中会维护一个巨大的hash表，支持任意存储类型的数据，很多网站通过Memcached提高网站的访问速度，尤其是对于大型的需要频繁访问的网站，减少查询效率，提高查询速度；



**缓存应用系统：**计算机体系存储系统模型扩展到应用也是一样，应用需要数据，数据哪里来？缓存(更快的存储)->DB（较慢的存储），他们的工作流程大致如下图所示：



**架构：**C/S架构

服务端：Memcached服务端，通过C语言编写而成；

客户端：Memcached API客户端，可以通过任何语言编写，如php、py等；

**特点：**

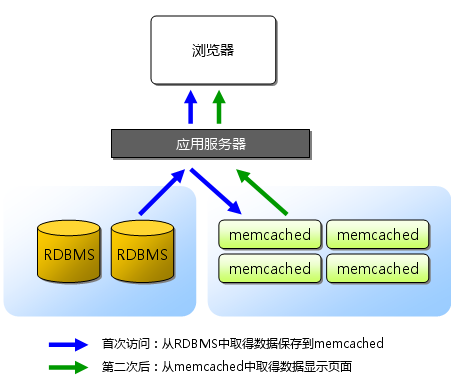
1.为了提高性能，memcached中保存的数据都存储在memcached内置的内存存储空间中。由于数据仅存在于内存中，因此重启memcached、重启操作系统会导致全部数据消失；

2.基于libevent的事件处理：libevent是个程序库，它将Linux的epoll、BSD类操作系统的kqueue等事件处理功能封装成统一的接口。即使对服务器的连接数增加，也能发挥I/O的性能。memcached使用这个libevent库，因此能在Linux、BSD、Solaris等操作系统上发挥其高性能；

3. 简单key/value存储：服务器不关心数据本身的意义及结构，只要是可序列化数据即可。存储项由“键、过期时间、可选的标志及数据”四个部分组成；

4.功能的实现一半依赖于客户端，一半基于服务器端：客户负责发送存储项至服务器端、从服务端获取数据以及无法连接至服务器时采用相应的动作；服务端负责接收、存储数据，并负责数据项的超时过期；

**三、运行架构：**

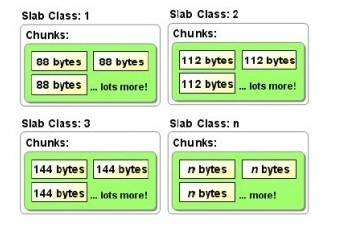


**四、Memcached缓存存储策略；**

**内存缓存策略：** Slab Allocation机制

**Slab Allocator机制基本原理**：按照预先规定的大小，将分配的内存分割成特定长度的块(chunk)，并把尺寸相同的块分成组，以完全解决内存碎片问题。但由于分配的是特定长度的内存，因此无法有效利用分配的内存。比如将100字节的数据缓存到128字节的chunk中，剩余的28字节就浪费了；

按照预先规定的大小，将分配的内存分割成特定长度的内存块（chunk），再把尺寸相同的内存块分层组（chunk集合），这些内存不会释放，可以反复利用；

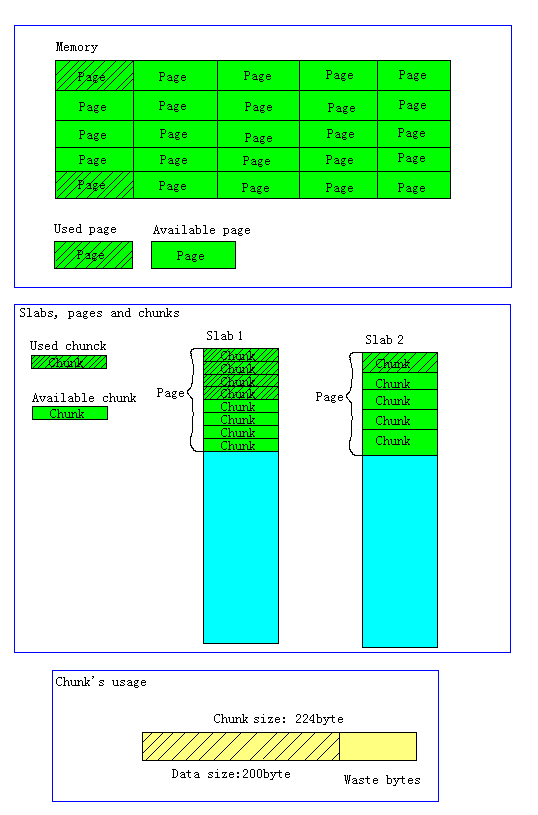


**Slab Allocation机制角色：**

1.Chunk为固定大小的内存空间，默认为96Byte。

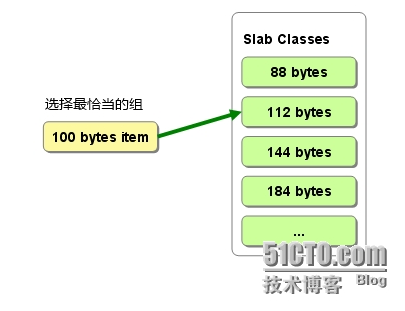
2.page对应实际的物理空间，1个page为1M。

3.同样大小的chunk又称为slab。



**客户端选择slab机制：**

下面说明memcached如何针对客户端发送的数据选择slab并缓存到chunk中。memcached根据收到的数据的大小，选择最适合数据大小的slab。 memcached中保存着slab内空闲chunk的列表，根据该列表选择chunk， 然后将数据缓存于其中。如下图；



**内存释放机制：**

Laxzy Expiration：Memcached每个被存取的对象都有唯一的标识符key，存取操作均通过key进行，例如可以把后端数据库中的select操作提取出来，然后对相应的SQL进行hash计算得出key，然后以这个key在memcached中查找数据，如果数据不存在，说明其尚未被写入缓存中，并设置一个失效时间（比如1小时），在失效时间内的数据都是从缓存中提取，这样就有效地减少了数据库的压力；

Least Recently Used（LRU）：删除“最近最少使用”的记录的机制。当memcached的内存空间不足时，从最近未被使用的记录中搜索，并将其空间分配给新的记录。-M 参数禁止LRU功能，内存用尽时，memcached会返回错误，不建议使用memcached -M -m 1024；

**五、Memcached优缺点：**

**优点：**

1.读写性能优异，特别是高并发时和文件缓存比有明显优势；

2.memcached组建支持集群，并且是自动管理负载均衡；

3.开源，占用资源小，协议简单的软件，实现了数据库和web之间的数据缓存功能,减少数据库的检索次数,减少数据库的I/O，解决了架构数据库端的压力；

4.存储方式：内置于内存存储方式，存取的效率高，执行的速度快；

**缺点：**

1.缓存空间有限：据说一台电脑的mem缓存开到2g以上会出现不稳定，数据无故丢失的现象；

2.掉电丢失数据：由于是把数据放在内存里的，所有一旦机器掉电，数据也就全部丢失了。

一般建议：而mem则适合放一些频繁更改的数据，比如可以把session数据放进mem；

**六、Memcached的使用；**

1.Memcache的协议的错误部分主要是三个错误提示之提示指令：

普通错误信息： ERROR

客户端错误： CLIENT\_ERROR <错误信息>

服务器端错误： SERVER\_ERROR <错误信息>

2.数据存取命令：

set：是保存一个叫做key的数据到服务器上；

add：是添加一个数据到服务器，但是服务器必须这个key是不存在的，能够保证数据不会被覆盖；

replace：是替换一个已经存在的数据，如果数据不存在，就是类似set功能；

get：格式是：get <键>\*

delete：删除指令；

flush\_all：这个指令执行后，服务器上所有缓存的数据都被删除，并且返回在php里也可以用getStats()来查看；

quit：退出

3.数据保存指令：

指令格式：<命令> <键> <标记> <有效期> <数据长度>

set key 0 60 10

<键key>：就是保存在服务器上唯一的一个表示符；

<标记flag>：一个16位的无符号整形，用来设置服务器端跟客户端一些交互的操作；

<有效期>：是数据在服务器上的有效期限，如果是0，则数据永远有效，单位是秒，Memcache服务器端会把一个数据的有效期设置为当前Unix时间+设置的有效时间；

<数据长度>：块数据的长度，一般在这个个长度结束以后下一行跟着block data数据内容，发送完数据以后，客户端一般等待服务器端的返回，服务器端的返回：数据保存成功 （STORED）,数据保存失败（NOT\_STORED），一般是因为服务器端这个数据key已经存在了；

**七、案例：Memcached缓存结合lamp缓存搭建；**

**案例环境：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统类型 | IP地址 | 主机名 | 所需软件 | 内存 |
| Centos 7.4 1708 64bit | 192.168.100.101 | memser.linuxfan.cn | libevent-release-1.4.15-stable.tar.gz  memcached-1.5.10.tar.gz | 4G |
| Centos 7.4 1708 64bit | 192.168.100.102 | memcli.linuxfan.cn | libmemcached-1.0.18.tar.gz  memcached-2.2.0.tgz  httpd mariadb mysql php php-devel php-mysql | 1G |

**下载位置：**

server端：

http://memcached.org/downloads ##主程序memcached-1.5.10.tar.gz

https://codeload.github.com/libevent/libevent/tar.gz/release-1.4.15-stable ##libevent程序

client端：

https://launchpad.net/libmemcached/+download ##libmemcached程序下载

http://pecl.php.net/get/memcached-2.2.0.tgz ##memcached扩展程序（api）

**案例步骤：**

* 在server节点安装Libevent程序；
* 在server节点安装Memcached主程序；
* 在server节点配置优化Memcached服务；
* 在client节点安装lamp应用平台；
* 在client节点安装LibMemcached程序；
* 在client节点安装Memcached扩展组件（Memcached api）；
* 在client节点配置php支持Memcached组件；
* 测试访问Memcached API功能；
* Memcached数据库的操作与管理；
* **在server节点安装Libevent程序；**

[root@memser ~]# ls

libevent-release-1.4.15-stable.tar.gz memcached-1.5.10.tar.gz

[root@memser ~]# tar zxvf libevent-release-1.4.15-stable.tar.gz -C /usr/src/

[root@memser ~]# cd /usr/src/libevent-release-1.4.15-stable/

[root@memser libevent-release-1.4.15-stable]# ./autogen.sh

[root@memser libevent-release-1.4.15-stable]# ./configure --prefix=/usr/local/libevent

[root@memser libevent-release-1.4.15-stable]# make && make install

[root@memser libevent-release-1.4.15-stable]# cd

[root@memser ~]# ln -s /usr/local/libevent/lib/libevent\* /usr/lib64/

* **在server节点安装Memcached程序；**

[root@memser ~]# tar zxvf memcached-1.5.10.tar.gz -C /usr/src/

[root@memser ~]# cd /usr/src/memcached-1.5.10/

[root@memser memcached-1.5.10]# ./configure --prefix=/usr/local/memcached --with-libevent=/usr/local/libevent/

[root@memser memcached-1.5.10]# make && make install

[root@memser memcached-1.5.10]# cd

* **在server节点配置优化Memcached服务；**

[root@memser ~]# ln -s /usr/local/memcached/bin/\* /usr/local/bin/

[root@memser ~]# memcached -d -m 128m -u root ##启动memcached服务

[root@memser ~]# netstat -utpln |grep mem

tcp 0 0 0.0.0.0:11211 0.0.0.0:\* LISTEN 10748/memcached



* **在client节点安装lamp应用平台；**

[root@memcli ~]# wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo

[root@memcli ~]# yum -y install httpd mariadb-server mysql php php-devel php-mysql

* **在client节点安装LibMemcached程序；**

[root@memcli ~]# ls

libmemcached-1.0.18.tar.gz memcached-2.2.0.tgz

[root@memcli ~]# tar zxvf libmemcached-1.0.18.tar.gz -C /usr/src/

[root@memcli ~]# cd /usr/src/libmemcached-1.0.18/

[root@memcli libmemcached-1.0.18]# ./configure --prefix=/usr/local/libmemcached --with-memcached=/usr/local/memcached ##需要指定mem扩展组件的安装位置

[root@memcli libmemcached-1.0.18]# make && make install

[root@memcli libmemcached-1.0.18]# cd

* **在client节点安装Memcached扩展组件（Memcached api）；**

[root@memcli ~]# yum -y install zlib zlib-devel

[root@memcli ~]# tar zxvf memcached-2.2.0.tgz -C /usr/src/

[root@memcli ~]# cd /usr/src/memcached-2.2.0/

[root@memcli memcached-2.2.0]# phpize

[root@memcli memcached-2.2.0]# ./configure --enable-memcached --with-php-config=/usr/bin/php-config --with-libmemcached-dir=/usr/local/libmemcached/ --disable-memcached-sasl

注解：

--enable-memcached ##开启memcached api功能

--with-php-config=/usr/bin/php-config ##指定php扩展命令所在位置

--with-libmemcached-dir=/usr/local/libmemcached/ ##指定上述libmemcached软件的安装位置

--disable-memcached-sasl ##关闭memcached的sasl功能

[root@memcli memcached-2.2.0]# make

[root@memcli memcached-2.2.0]# make test

[root@memcli memcached-2.2.0]# make install

Installing shared extensions: /usr/lib64/php/modules/

[root@memcli memcached-2.2.0]# ls /use/lib/64/php/modules/

curl.so json.so mysqli.so pdo\_mysql.so pdo\_sqlite.so sqlite3.so

fileinfo.so memcached.so mysql.so pdo.so phar.so zip.so

[root@memcli memcached-2.2.0]# cd

* **在client节点配置php支持Memcached组件；**

[root@memcli ~]# vi /etc/php.ini ##末尾追加

extension\_dir = /usr/lib64/php/modules/

extension = memcached.so

[root@memcli ~]# systemctl start httpd

[root@memcli ~]# netstat -utpln |grep 80

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 35528/httpd

[root@memcli ~]# cat <<END >/var/www/html/index.php

<?php

phpinfo();

?>

END

[root@memcli ~]# vi /var/www/html/test.php

<?php

$memcache = new Memcached();

$memcache->addServer('192.168.100.101',11211);

$memcache->set('key','Memcache test sussessful!',0,60);

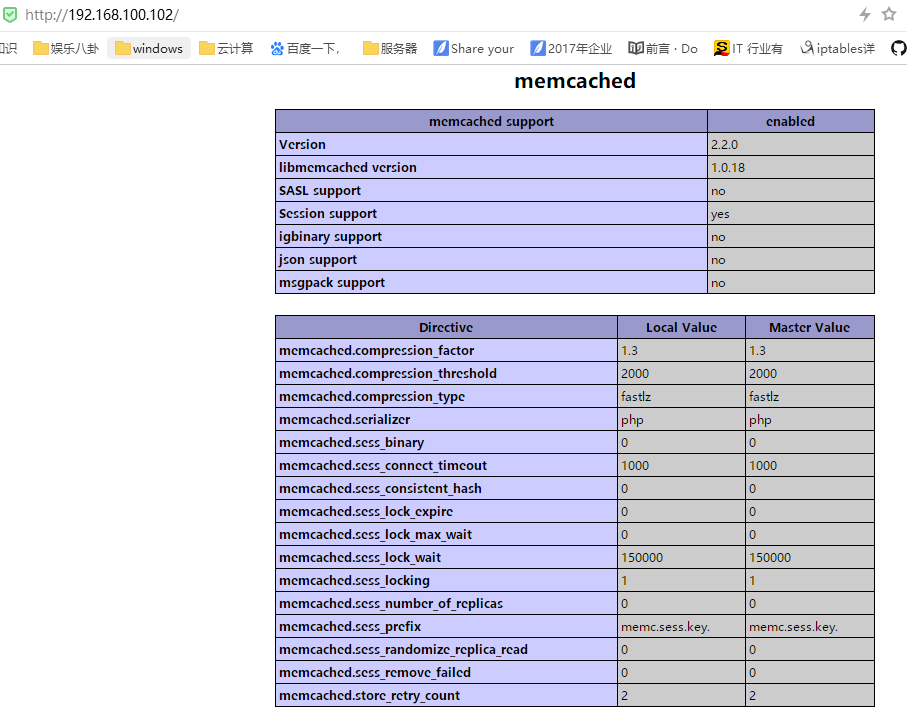
$result = $memcache->get('key');

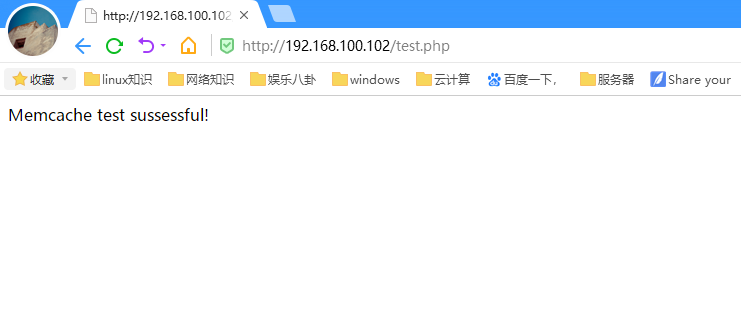
unset($memcache);

echo $result;

?>

* **测试访问Memcached API功能；**





[root@memser ~]# yum -y install telnet

[root@memser ~]# telnet 192.168.100.101 11211

get key

VALUE key 3932160 25

Memcache test sussessful!

END

quit

* **Memcached数据库的操作与管理；**

[root@memser ~]# telnet 192.168.200.203 11211

Trying 192.168.200.203...

Connected to 192.168.200.203.

Escape character is '^]'.

stats ##查看状态

STAT pid 12863 //memcached启动的进程ID

STAT uptime 4008 //到目前为止启动了多少秒

STAT time 1464713917 //服务器当前的unix时间戳

STAT version 1.2.6 //memcached的版本信息

STAT pointer\_size 64 //当前操作系统的指针大小（64位系统一般是64bit）

STAT rusage\_user 0.278957 //进程的累计用户时间

STAT rusage\_system 0.371943 //进程的累计系统时间

STAT curr\_items 1 //服务器当前存储的items数量

STAT total\_items 2 //从服务器启动以后存储的items总数量

STAT bytes 79 //当前服务器存储items占用的字节数

STAT curr\_connections 2 //当前的并发连接数

STAT total\_connections 4 //从服务器启动以后曾经打开过的连接数

STAT connection\_structures 3 //服务器分配的连接构造数

STAT cmd\_get 2 //执行get命令的次数

STAT cmd\_set 2 //执行set命令的次数

STAT get\_hits 2 //get的命中次数

STAT get\_misses 0 //get的非命中数

STAT evictions 0 //为获取空闲内存而删除的items数（分配给memcache的空间用满后需要删除旧的items来得到空间分配给新的items）

STAT bytes\_read 98 //总读取字节数（请求字节数）

STAT bytes\_written 97 //总发送字节数（结果字节数）

STAT limit\_maxbytes 33554432 //分配给memcache的内存大小（字节）

STAT threads 1 //当前进程数

END

set test\_key 0 0 10 ##设置test\_key指令格式：<命令> <键> <标记> <有效期> <数据长度>”

test\_value ##输入test\_key所对应的值

STORED

get test\_key ##查看test\_key键的内容

VALUE test\_key 0 10

test\_value

END

replace test\_key 0 0 11 ##修改test\_key键的数据长度

test\_value2 ##存储新的值

STORED

get test\_key ##查看验证

VALUE test\_key 0 11

test\_value2

END

stats items ##查询memcahed的key状态

STAT items:1:number 2

...

stats cachedump 1 100 ##查询memcached的key列表

append test\_key 0 0 5 ##在test\_key后边追加5个字符

,test ##添加的值

STORED

get test\_key ##查看验证

VALUE test\_key 0 16

test\_value2,test

END

prepend test\_key 0 0 6 ##在test\_key前面追加6个字符

test1, ##添加的值

STORED

get test\_key ##查看验证

VALUE test\_key 0 22

test1,test\_value2,test

END

delete test\_key ##删除键

DELETED

set myconter 0 0 1 ##定义一个myconter的键

1 ##设置上述键的值

STORED

incr myconter 1 ##值增加1

2

get myconter ##查看验证

VALUE myconter 0 1

2

END

decr myconter 1 ##值减少1

1

get myconter

VALUE myconter 0 1

1

END

flush\_all ##删除所有的键值对

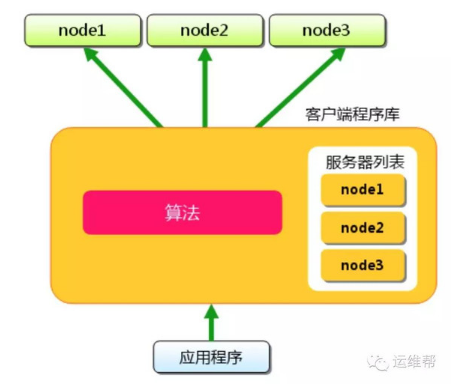
OK

quit ##退出

Connection closed by foreign host.

**八、Memcached集群原理；**

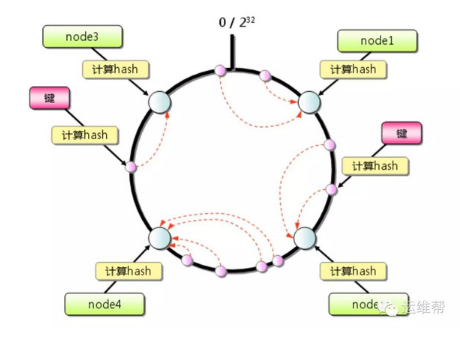
**概述：**memcached尽管是“分布式”缓存服务器，但服务器端并没有分布式功能，各个memcached不会互相通信以共享数据，这完全取决memcached api的客户端所使用的路由算法；



**客户端路由算法：**

1.求余数hash算法：先用key做一个hash运算得到一个整数，再去做hash算法，根据余数进行路由选择，这种算法适用于大多数据需求，但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：适用于动态变化的环境中，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算法将节点机器也映射到环中，按顺时针方向将所有对象存储到离客户端最近的node节点上，如下图；



**九、Memcached主从复制集群；**

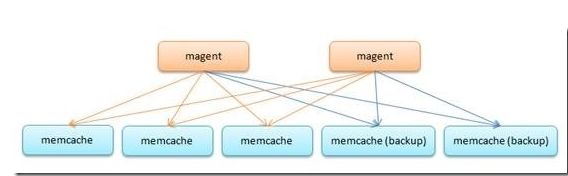
**案例环境：**

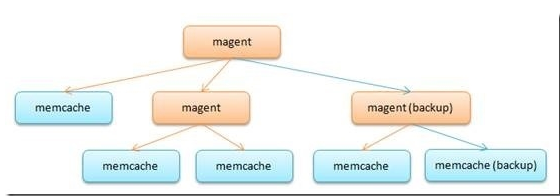
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统类型 | IP地址 | 主机名 | 所需软件 | 内存 |
| Centos 7.4 1708 64bit | 192.168.100.101 | mastermem.linuxfan.cn | memcached-1.5.10.tar.gz  libevent-release-1.4.15-stable.tar.gz  magent-0.5.tar.gz  keepalived | 2G |
| Centos 7.4 1708 64bit | 192.168.100.102 | slavemem.linuxfan.cn | memcached-1.5.10.tar.gz  libevent-release-1.4.15-stable.tar.gz  magent-0.5.tar.gz  keepalived | 2G |
| Centos 7.4 1708 64bit | 192.168.100.103 | client.linuxfan.cn | telnet | 1G |

**注意点：**

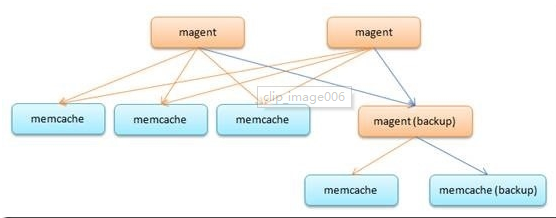
1.magent概述：Magent是一款开源的Memcached代理服务器软件，采用 Magent 缓存代理，防止单点现象，缓存代理也可以做备份，通过客户端连接到缓存代理服务器，缓存代理服务器连接缓存服务器，缓存代理服务器可以连接多台Memcached机器；

2.常见memcached+magent运行架构：





上图此模型已经能够很好的解决一个节点，一组服务器的缓存数据服务，但是如果在北方网通架设了一组服务器，同时在南方电信又架设了另外一组服务器，那么这两组相对独立的节点之间如何做到数据的同步与共享，基于magent与memcached的解决方案如下：



架构详解：

http://blog.51cto.com/ultrasql/1633897

3.为避免magent节点单点故障问题，可以使用keepalived服务为其实现高可用；

**案例步骤：**

* 部署两个节点的Memcached程序（在此两个节点的安装一致，在此列举master节点的配置）；
* 部署master节点的magent程序，部署完成传送给slave节点；
* 安装两个节点的Keepalived程序（在此两个节点的安装一致，在此列举master节点的配置）；
* 配置master节点的keepalived服务；
* 配置slave节点的keepalived服务；
* client节点测试主从复制集群；
* **部署两个节点的Memcached程序（在此两个节点的安装一致，在此列举master节点的配置）；**

[root@mastermem ~]# ls

libevent-release-1.4.15-stable.tar.gz memcached-1.5.10.tar.gz

[root@mastermem ~]# tar zxvf libevent-release-1.4.15-stable.tar.gz -C /usr/src/

[root@mastermem ~]# cd /usr/src/libevent-release-1.4.15-stable/

[root@mastermem libevent-release-1.4.15-stable]# ./autogen.sh

[root@mastermem libevent-release-1.4.15-stable]# ./configure --prefix=/usr/local/libevent

[root@mastermem libevent-release-1.4.15-stable]# make && make install

[root@mastermem libevent-release-1.4.15-stable]# cd

[root@mastermem ~]# ln -s /usr/local/libevent/lib/libevent\* /usr/lib64/

[root@mastermem ~]# tar zxvf memcached-1.5.10.tar.gz -C /usr/src/

[root@mastermem ~]# cd /usr/src/memcached-1.5.10/

[root@mastermem memcached-1.5.10 ~]# ./configure --prefix=/usr/local/memcached --with-libevent=/usr/local/libevent/

[root@mastermem memcached-1.5.10 ~]# make && make install

[root@mastermem memcached-1.5.10 ~]# cd

[root@mastermem ~]# ln -s /usr/local/memcached/bin/\* /usr/local/bin/

* **部署master节点的magent程序，部署完成传送给slave节点；**

[root@mastermem ~]# ls magent-0.5.tar.gz

magent-0.5.tar.gz

[root@mastermem ~]# mkdir magent

[root@mastermem ~]# tar zxvf magent-0.5.tar.gz -C magent/

ketama.c

magent.c

ketama.h

Makefile

[root@mastermem ~]# cd magent

[root@mastermem magent]# ls

ketama.c ketama.h magent.c Makefile

[root@mastermem magent]# vi ketama.h ##在文件开头添加

#ifndef SSIZE\_MAX

#define SSIZE\_MAX 32767

#endif

[root@mastermem magent]# vi Makefile

1 LIBS = -levent -lm -L /usr/local/libevent/lib ##选项为小-L

2 INCLUDE= -I /usr/local/libevent/include ##选项为大-i

[root@mastermem magent]# make

gcc -Wall -O2 -g -I /usr/local/libevent/include -c -o magent.o magent.c

gcc -Wall -O2 -g -I /usr/local/libevent/include -c -o ketama.o ketama.c

gcc -Wall -O2 -g -o magent magent.o ketama.o -levent -lm -L /usr/local/libevent/lib

[root@mastermem magent]# ls

ketama.c ketama.h ketama.o magent magent.c magent.o Makefile

[root@mastermem magent]# cp magent /usr/bin/

[root@mastermem magent]# scp magent 192.168.100.102:/usr/bin/

[root@mastermem magent]# cd

* **安装两个节点的Keepalived程序（在此两个节点的安装一致，在此列举master节点的配置）；**

[root@mastermem ~]# yum -y install keepalived

* **配置master节点的keepalived服务；**

[root@mastermem ~]# vi /etc/keepalived/keepalived.conf

global\_defs {

router\_id R1

}

vrrp\_instance VI\_1 {

state MASTER

interface eth0

virtual\_router\_id 1

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.100.250

}

}

[root@mastermem ~]# vi /opt/check.sh

#!/bin/bash

while true;do

K=$(ip a|grep 192.168.100.250|wc -l)

if [ $K -ne 0 ];then

magent -u root -n 51200 -l 192.168.100.250 -p 12000 -s 192.168.100.101:11211 -b 192.168.100.102:11211

else

pkill -9 magent

fi

M=$(netstat -utpln |grep mem|wc -l)

if [ $M -eq 0 ];then

systemctl stop keepalived

fi

done

注解：magent参数详解：

-u ##指定运行用户

-n ##最大的连接数，默认为4096

-l ##小写L，magent监听的ip地址

-p ##magent监听的端口

-s ##设置memcached主缓存的ip地址和端口

-b ##设置memcached备缓存的ip地址和端口

[root@mastermem ~]# chmod +x /opt/check.sh

[root@mastermem ~]# memcached -u root -d -m 128m

[root@mastermem ~]# netstat -utpln |grep memcached

tcp 0 0 0.0.0.0:11211 0.0.0.0:\* LISTEN 7117/memcached

[root@mastermem ~]# /opt/check.sh &

[root@mastermem ~]# jobs -l

[1]+ 16273 运行中 /opt/check.sh &

[root@mastermem ~]# systemctl start keepalived

[root@mastermem ~]# ip a|grep 192.168.100.250

inet 192.168.100.250/32 scope global eth0

[root@mastermem ~]# netstat -utpln |grep magent

tcp 0 0 192.168.100.250:12000 0.0.0.0:\* LISTEN 22834/magent

* **配置slave节点的keepalived服务；**

[root@slavemem ~]# vi /etc/keepalived/keepalived.conf

global\_defs {

router\_id R2

}

vrrp\_instance VI\_1 {

state BACKUP

interface eth0

virtual\_router\_id 1

priority 99

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.100.250

}

}

[root@slavemem ~]# vi /opt/check.sh

#!/bin/bash

while true;do

K=$(ip a|grep 192.168.100.250|wc -l)

if [ $K -ne 0 ];then

memcached -u root -d -m 128m

magent -u root -n 51200 -l 192.168.100.250 -p 12000 -s 192.168.100.102:11211 -b 192.168.100.101:11211

else

pkill magent

fi

done

[root@slavemem ~]# chmod +x /opt/check.sh

[root@slavemem ~]# /opt/check.sh &

[1] 18066

[root@slavemem ~]# jobs -l

[1]+ 18066 完成 /opt/check.sh

[root@slavemem ~]# memcached -u root -d -m 128m

[root@slavemem ~]# netstat -utpln |grep mem

tcp 0 0 0.0.0.0:11211 0.0.0.0:\* LISTEN 17920/memcached

[root@slavemem ~]# systemctl start keepalived

[root@slavemem ~]# ip a|grep 192.168.100.250

[root@slavemem ~]# netstat -utpln |grep magent

* **client节点测试主从复制集群；**

**此时192.168.100.101节点为master，VIP在master节点，运行memcahed、magent进程；**

[root@client ~]# yum -y install telnet

[root@client ~]# telnet 192.168.100.250 12000

Trying 192.168.100.250...

Connected to 192.168.100.250.

Escape character is '^]'.

set k1 0 0 5

hello

STORED

get k1

VALUE k1 0 5

hello

END

quit

Connection closed by foreign host.

**将192.168.100.101节点的memcached进程关闭，模拟故障，后台运行的脚本导致keepalived服务关闭，keepalived服务关闭，导致magent进程关闭；**

[root@mastermem ~]# netstat -utpln |grep mem

tcp 0 0 0.0.0.0:11211 0.0.0.0:\* LISTEN 63140/memcached

[root@mastermem ~]# kill -9 63140

[root@mastermem ~]# netstat -utpln |grep mem

[root@mastermem ~]# netstat -utpln |grep magent

[root@mastermem ~]# ip a |grep 192.168.100.250

**此时验证slave节点的进程情况，VIP转移到slave节点上；**

[root@slavemem ~]# ip a|grep 192.168.100.250

inet 192.168.100.250/32 scope global eth0

[root@slavemem ~]# netstat -utpln |grep mem

tcp 0 0 0.0.0.0:11211 0.0.0.0:\* LISTEN 17920/memcached

[root@slavemem ~]# netstat -utpln |grep magent

tcp 0 0 192.168.100.250:12000 0.0.0.0:\* LISTEN 53789/magent

**客户端测试读写数据；**

[root@client ~]# telnet 192.168.100.250 12000

Trying 192.168.100.250...

Connected to 192.168.100.250.

Escape character is '^]'.

get k1

VALUE k1 0 5

hello

END

set k2 0 0 2

ha

STORED

get k2

VALUE k2 0 2

ha

END

quit

Connection closed by foreign host.