Memcache 缓存服务器

一、Memcache 简述

1、Memcache 定义：

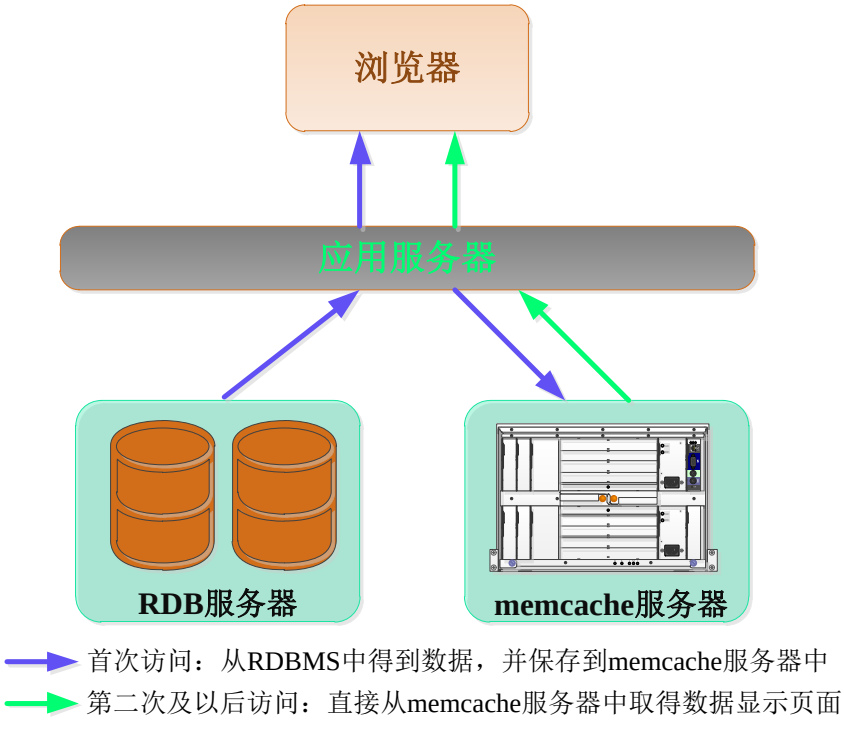
memcache 是一款开源、高性能、分布式内存对象缓存系统，可应用各种需要缓存的场景，其主要目的是通过降低对 Database 的访问来加速 web 应用程序。它是一个基于内存的“键值对(Key/Value)”存储，它的工作机制是在内存中开辟一块空间，用于存储数据库调用、API 调用或页面引用结果的直接数据，如字符串、图片等，并且将其建立为 Hash-Table，由 memcached 自身管理。

memcached 是 memcache 系统的主程序文件，以守护程序方式运行于一个或多个服务器中，随时接受客户端的连接操作，使用共享内存存取数据。

2、Memcache 出现的背景：

许多 Web 应用都将数据保存到 RDBMS(关系型数据库管理系统)中，应用服务器从中读取数据并在浏览器中显示。 但随着数据量的增大、访问的集中，就会出现 RDBMS 的负担加重、数据库响应恶化、 网站显示延迟等重大影响。

在大型系统中，访问同样的数据是很频繁的，memcached 可以大大降低数据库压力，通过缓存数据库查询结果，减少对数据库访问次数，来加速 Web 应用程序的速度、提高可扩展性。使系统执行效率提升。



3、Memcache 特性：

Memcached 是一款开发工具，它既不是一个代码加速器，也不是数据库中间件。其设计思想主要反映在如下方面：

(1)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；

(2)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；

(3) memcached 不互相通信的分布式即不在服务器间进行数据同步；

(4)O(1)的执行效率

(5)清理超期数据：默认情况下，Memcached 是一个 LRU(最近最少使用)缓存，同时，它按事先预订的时长清理超期数据；但事实上，memcached 不会删除任何已缓存数据，只是在其过期之后不再为客户所见；而且，memcached 也不会真正按期限清理缓存，而仅是当 get 命令到达时检查其时长；

Memcached 的服务器客户端通信并不使用复杂的 XML 等格式， 而使用简单的基于文本格式和二进制格式。因此，通过 telnet 也能在 memcached 上保存数据、取得数据。

二、Memcached 内存分配机制

1、Slab Allocation 机制

memcached 默认情况下采用了名为 Slab Allocator 的机制分配、管理内存。在该机制出现以前，内存的分配是通过对所有记录简单地进行 malloc()和 free()来进行的。但是，这种方式会导致内存碎片，加重操作系统内存管理器的负担，最坏的情况下，会导致操作系统比 memcached 进程本身还慢。Slab Allocator 就是为解决该问题而诞生的。

Slab Allocator 的基本原理是按照预先规定的大小，将分配的内存分割成特定长度的块，以完全解决内存碎片问题。

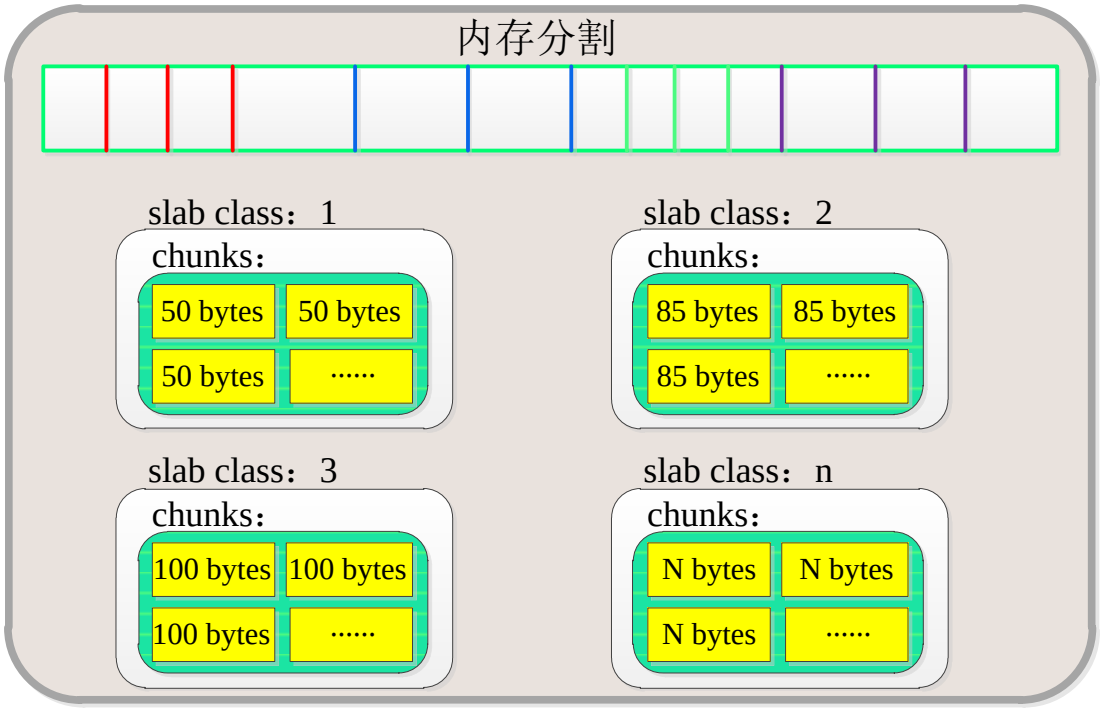
2、Slab Allocation 的主要术语

(1)page：分配给 slab 用于切割的内存空间，默认是 1MB。分配给 slab 之后根据 slab 的大小切分成 chunk

(2)chunk：存放数据的最小单元。用户数据 item（key、value 等）最终会保存在 chunk 中。

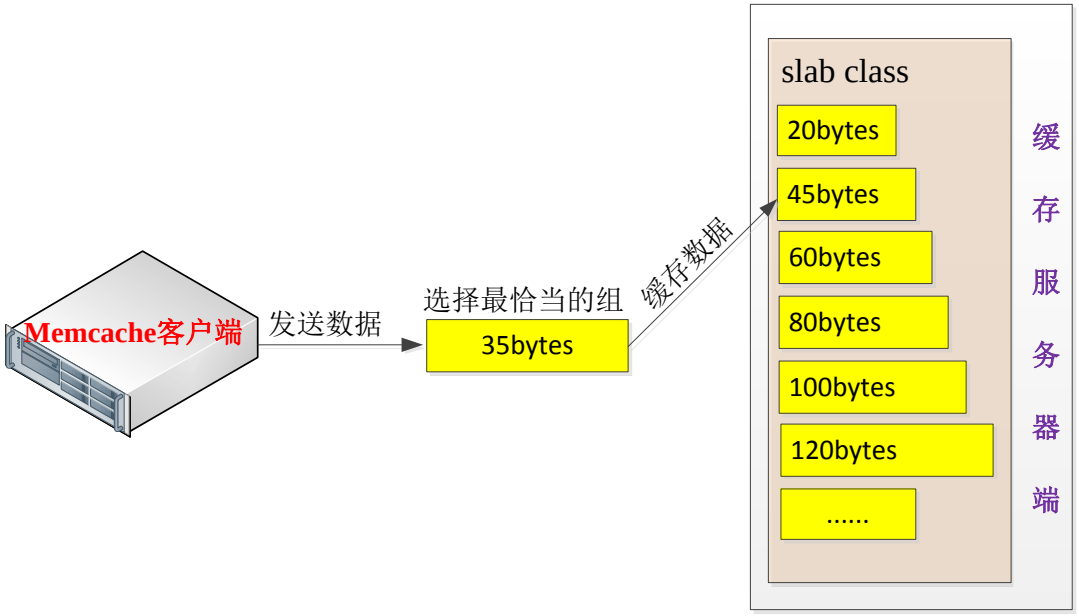
(3)slab class：特定大小的 chunk 的组，在 memcached 中，对数据的管理是以 slab 为单元进行管理的。

每个 slab class 对应一个或多个空间大小相同的 chunk。参考下图：



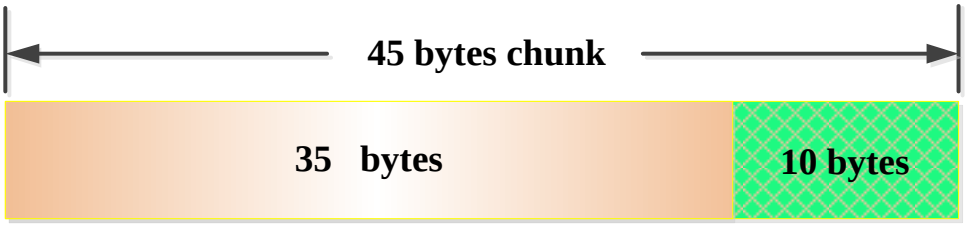
3、在 Slab 中缓存记录的原理

memcached 对客户端发送的数据选择最适合数据大小的 slab 并缓存到 chunk 中，memcached 服务器中保存着 slab 内空闲 chunk 的列表，根据该列表选择 chunk，然后将数据缓存其中。



4、Slab Allocator 还存在的问题：

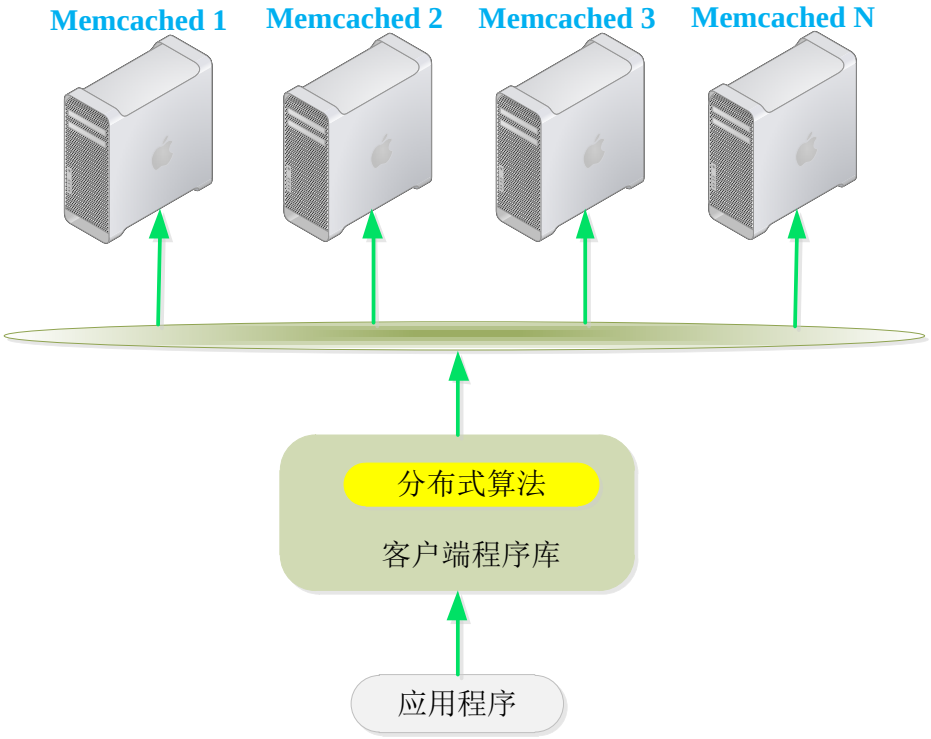
Slab Allocator 解决了当初的内存碎片问题，但新的机制也给 memcached 带来了新的问题。这个问题就是，由于分配的是特定长度的内存，因此无法有效利用分配的内存。例如，将 35 字节的数据缓存到 45 字节的 chunk 中，剩余的 10 字节浪费了，见下图：



在后续的学习中可知：可以通过设置合适的 growth factor（增长系数/因子）选项来调节 slab class 的大小差别，从而来减少内存资源的浪费

四、分布式缓存中三种负载均衡的方法

基于客户端的 memcached 的分布式架构(见下图)，memcached 不同的组合方式会严重影响缓存的有效性，基于此作者整理了以下三种关于 memcached 服务器的分布方式。



1、传统的数据分布方法,将 key 的 hash 值对机器数取模

Hash，翻译做“散列”，就是把任意长度的输入（又叫做预映射， pre-image），通过散列算法，变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值

来唯一的确定输入值。简单的描述就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。

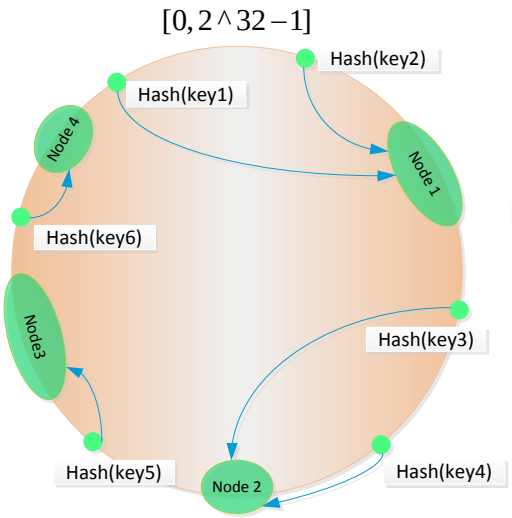
对用户的检索关键词分词后取 hash 值，计算 hash(key)/n，n 为机器数，得到的值就是该 key 需要路由到的服务器编号了。优点很明显，缺点是在在服务器数量发生变化的时候，缓存会大量失效，举例来说，共有 4 个 memcache 缓存服务器，当客户端 A 浏览的数据都缓存在 1 号服务器，且当 1 号服务器宕机时，由于缓存服务器个数由 4 减为 1，那么 A 的数据不仅将会全部丢失，对于后续浏览器的缓存数据也会造成雪崩式的丢失(由于取模结果与之前不同)。

2、一致性 hash 闭环

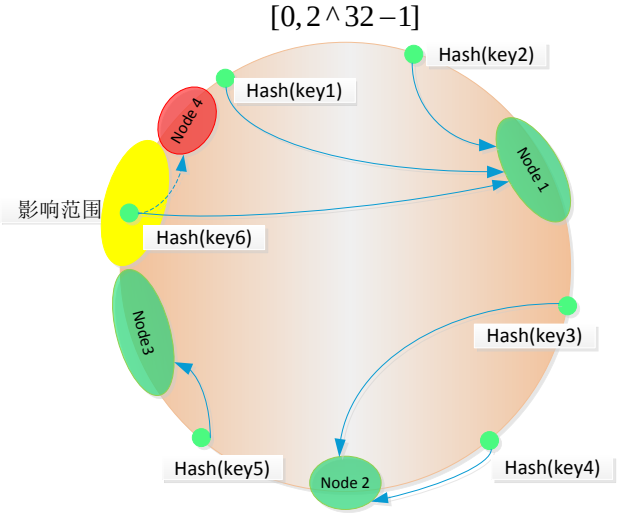
考虑到传统数据分布方法对缓存数据具有潜在的灾难性后果，一致性 hash 则可以解决部分问题，通常可以考虑[0,(2^32-1)]为 hash 值的取值范围。我们可以把取值范围想象成一个闭环，2^32-1 和 0 相连接。如下图所示。

假设有 4 个缓存服务器节点(node1~node4)，首先计算这四个节点的 hash 值，这四个点占据了闭环的四个位置。需要根据 key 来路由缓存服务器，首先计算 hash(key)值（这个 hash 算法需要保证 hash(key)的值落在在区间[0,2^32-1]中，hash(key)对 2^32-1 取模可保证在区间内）。然后 hash（key）值必定位于闭环中的某一个点，然后按照顺时针方向，选择距离最近的 node，命中的 node 就是 key 需要路由的服务器。如上图，需要找到 key2 的缓存值所在的服务器，首先计算 hash(key2)，落在如图的闭环中，然后顺时针找到离这个 hash 值最近的 node，可以看到是 node1。key2 对应的缓存值就存在 node1 中。

当四个节点中有一个节点失效时，影响的缓存数据仅仅为上图中黄色区域(node3~node4)，大大的减少了缓存失效的范围。



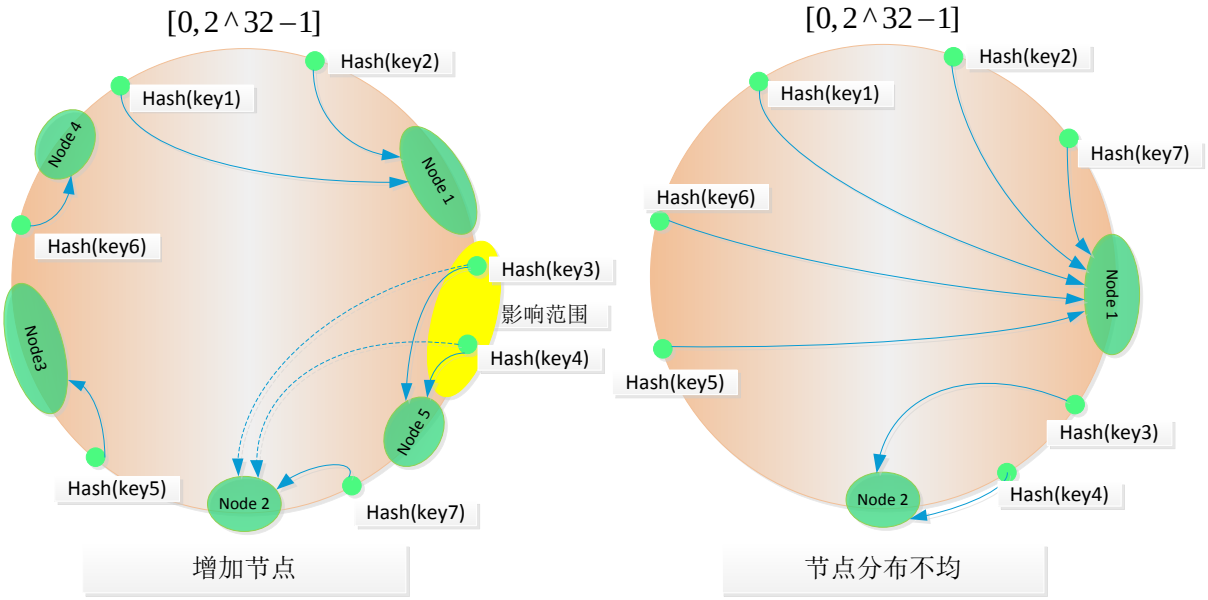
一致性哈希算法



节点失效

如下图所示，为了提高查询效率，当增加了一个节点时：只有 hash(key)落在 node1~node5 之间的 key 才会有影响，在未增加 node5 之前落在这个范围的 key 最终路由到 node2,增加 node5 之后路由到 node5 了，缓存失效了。除此之外，落在其他范围的 hash 值就不受影响。如上图，增加 node5 之后 key3 和 key4 原来会路由到 node2 的，现在路由到 node5。而 key7 不受影响，还是路由到 node2。

一致性哈希闭环算法也存在缺点，如下图，当节点机数量少且分布不均衡时，会造成缓存分布不均，当其中缓存两=量较大的节点失效时，也会造成比较大的缓存失效的后果，因此在最初设计时因避免节点分布不均的情况：



针对当节点机数量少造成缓存分布不均的情况，引入一个虚拟节点的概念，即在现有节点的基础上，在每一个节点内部虚拟出几个 memcache 服务器。这样就可以把压力尽量分配到各个机器上。另外需要维护一张表格，用来记录虚拟节点指向的真正节点。

优点：相比简单的对机器数取模算法，当节点变动的时候只有相对较少的 key 失效，实现也相对简单。不需要进行数据迁移，每个服务器是独立的。

缺点：还是会有部分的 key 失效，当访问量非常大的时候，若访问到失效的 key，则会直接连接数据库，可能造成数据服务器宕机。

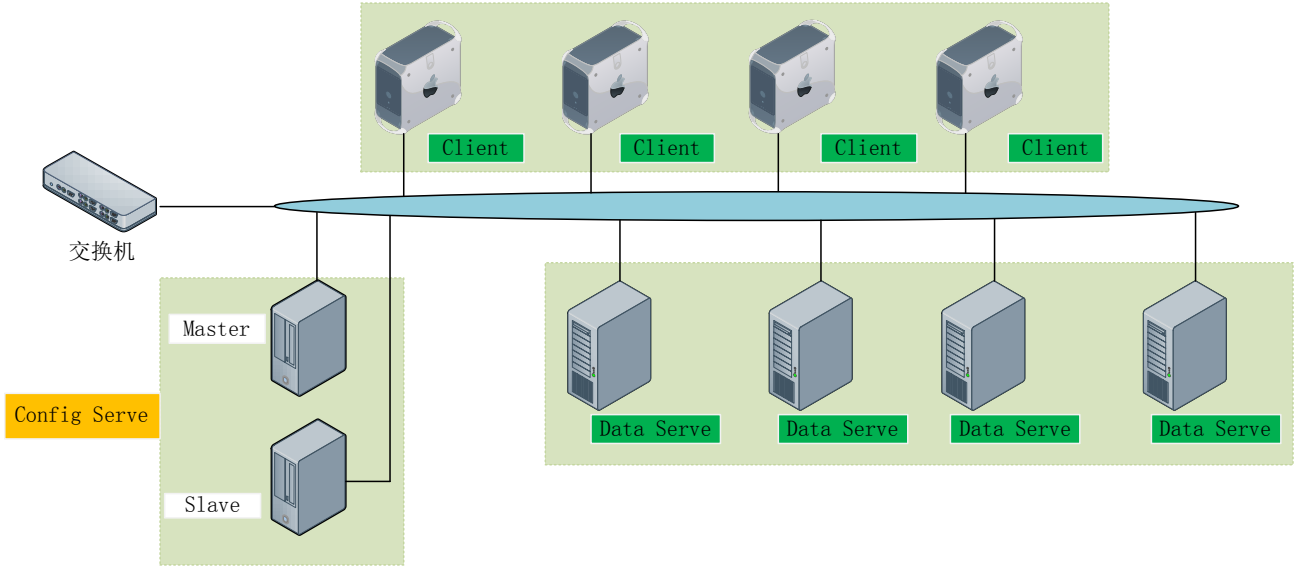
3、tair 负载均衡算法：(详细参考：https://www.lvtao.net/database/tair.html)

tair 是淘宝开发的一个分布式 key/value 存储引擎。tair 分为持久化和非持久化两种使用方式：非持久化的 tair 可以看成是一个分布式缓存；持久化的 tair 将数据存放于磁盘中。为了

解决磁盘损坏导致数据丢失，tair 可以配置数据的备份数目，tair 自动将数据备份到不同的主机上，当有主机发生异常，无法正常提供服务的时候，其于的备份会继续提供服务。

(1)tair 的总体结构

tair 作为一个分布式系统，是由一个中心控制节点和一系列的服务节点组成。称中心控制节点为 config server，服务节点 data server。config server 负责管理所有的 data server，维护 data server 的状态信息。data server 对外提供各种数据服务，并以心跳的形式将自身状况汇报给 config server。config server 是控制点，而且是单点，目前采用一主一备的形式来保证其可靠性。所有的 data server 地位都是等价的。



五、memcached 数据过期清理方式：Lazy Expiration + LRU

1、Lazy Expiration（惰性过期）

根据缓存对象的 ttl 进行清理，memcached 内部不会监视记录是否过期，而是在 get 时查看记录的时间戳，检查记录是否过期。这种技术被称为 lazy（惰性）expiration。因此，memcached 不会在过期监视上耗费 CPU 时间。

2、LRU（最近最少使用算法）

memcached 会优先使用已超时的记录的空间，但即使如此，也会发生追加新记录时空间不足的情况，此时就要使用名为 Least Recently Used（LRU）机制来分配空间。当 memcached 的内存空间不足时（无法从 slab class 获取到新的空间时），就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。可以支持禁用此功能。

六、memcached 安装配置

1、memcached 依赖于 libevent 库

memcached 基于libevent的事件处理，libevent是个程序库，它将Linux的epoll、epool、select、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 O(1)的性能。 memcached 依赖于 libevent API，因此要需首先安装，libevent 官方主页：http://libevent.org/

```
[root@web ~]#tar zxvf libevent-2.0.22-stable.tar.gz
[root@web ~]#cd libevent-2.0.22-stable
[root@web ~]#./configure --prefix=/usr/local/libevent
[root@web ~]#make && make install
##添加配置文件：
[root@web ~]#echo "/usr/local/libevent/lib" > /etc/ld.so.conf.d/libevent.conf
##将 liebevent 库文件追击到系统库内，使其生效：
[root@web ~]#ldconfig -a|head
```

2、memcached 的安装

(1)yum 安装

```
[root@web ~]#yum install memcached -y
[root@web ~]#rpm -ql memcached
/etc/rc.d/init.d/memcached          ##服务脚本
/etc/sysconfig/memcached            ##配置文件
/usr/bin/memcached                  ##可执行文件
/usr/bin/memcached-tool             ##memcached 工具
......
```

(2)编译安装 memcached

官方主页：https://memcached.org。前期已将 libevent 安装在/usr/local/libevent 目录下，以下是编译安装配置 memcached：

```
[root@web ~]#wget http://www.memcached.org/files/memcached-1.4.33.tar.gz
[root@web ~]#tar zxvf memcached-1.4.33.tar.gz
[root@web ~]#cd memcached-1.4.33
[root@web ~]#./configure --prefix=/usr/local/memcached --with-libevent=/usr/local/libevent
[root@web ~]#make && make install
##环境配置
[root@web ~]#cat /etc/profile.d/memcache.sh
export PATH=$PATH:/usr/local/memcached/bin/
[root@web ~]#source /etc/profile.d/memcache.sh
```

3、memcached 常用选项

- -p TCP 监听端口 (default: 11211)，小写
- -U UDP 监听端口 (default: 11211, 0 默认关闭)，大写
- -l 监听的服务器 IP 地址 (default: all addresses)
- -d 运行为守护进程，后台运行
- -u 运行 memcached 用户
- -m #：最大的内存使用 (default: 64 MB)，也不宜过大
- -c 最大并发连接 (default: 1024)
- -v 输出警告和错误信息
- -vv 同时打印客户端请求和返回信息
- -vvv 打印内部状态转换信息
- -P 设置保存 pid 文件目录，仅当使用-d 选项时
- **-f 指定增长因子|倍数 (default: 1.25)**
- -n key+value+flags 最小分配空间(default: 48)
- -L 尝试使用大内存页。增加内存页大小可以减少失误的 TLB 数量，提高性能。
- -t 使用的线程数量 (default: 4)
- -R 每个事件的最大请求数 (default: 20)
- -o 配置额外选项

4、memcached 的使用

(1)启动 memcached：

```
[root@web ~]#id memcached          ##使用 yum 安装，自动添加 memcached 用户
uid=495(memcached) gid=493(memcached) groups=493(memcached)
[root@web ~]# memcached -u memcached -f 4 -vvv
slab class   1: chunk size       96 perslab   10922
slab class   2: chunk size      384 perslab    2730
slab class   3: chunk size     1536 perslab     682
slab class   4: chunk size     6144 perslab     170
slab class   5: chunk size    24576 perslab      42
slab class   6: chunk size    98304 perslab      10
slab class   7: chunk size  1048576 perslab       1
<26 server listening (auto-negotiate)
<27 server listening (auto-negotiate)
<28 send buffer was 229376, now 268435456
<29 send buffer was 229376, now 268435456
##从输出结果可以看到 memcached 的内存分配过程：
[root@web ~]#service memcached start
注：编译安装的 memcached 后文提供了 startup 脚本以及自定义的配置文件
[root@web ~]#ss -tnl                ##监听 11211TCP 端口
```

```
State       Recv-Q Send-Q          Local Address:Port            Peer Address:Port
LISTEN      0      128                    :::11211                     :::*
LISTEN      0      128                     *:11211                      *:*
```

(2)memcached 基本命令

memcached 提供了几个命令来完成与服务器端的交互，这些命令基于 memcached 的协议实现。memcached 可以利用 telnet 进行交互式连接，以下的命令均可在 Telnet 连接下执行：

存储类命令：set, add, replace, append（在一个缓存后附加内容）, prepend（在一个缓存之前附加内容）

获取数据类命令：get, delete, incr/decr（自加 1/自减 1）

统计类命令：stats, stats items, stats slabs, stats sizes

清理命令：flush_all

```
##命令详解：
##set:   add keyname flag（修饰符）  timeout  datasize         ##添加或者更新 key
##add:   add keyname flag（修饰符）  timeout  datasize         ##key 不存在时添加一个 key
##示例:  add mykey 0（#表示没有修饰符） 10（#生成周期） 12（#数据大小）
         Hello World(#键值(value))
##get:   get keyname  ##获取键所对应的值,gets 命令比 get 返回的值多一个数字,用来判断数据是否发生过改变。
##replace: replace keyname flag（修饰符）  timeout  datasize  ##数据存在时替换，不存在时返回错误
##delete:  delete keyname ##删除 key 及其对应的值
##stats:   stats  ##状态命令，显示 memcached 当前状态
[root@web ~]# telnet 192.168.88.66 11211
Trying 192.168.88.66...
Connected to 192.168. 88.66.
Escape character is '^]'.
stats
STAT pid 8279                       #进程 ID
STAT uptime 8000                    #服务器运行秒数
STAT time 1378284623                #服务器当前 unix 时间戳,秒
STAT version 1.4.15                 #服务器版本
STAT libevent 2.0.21-stable         #libevent 版本号
STAT pointer_size 64                #操作系统指针大小(这台服务器是 64 位的)
STAT rusage_user 0.000999           #计用户时间
STAT rusage_system 0.003999         #进程累计系统时间
STAT curr_connections 10            #当前打开连接数
STAT total_connections 11           #曾打开的连接总数
STAT connection_structures 11       #服务器分配的连接结构数
STAT reserved_fds 20                #内部使用的 FD 数
STAT cmd_get 0                      #执行 get 命令总数
STAT cmd_set 0                      #执行 set 命令总数
STAT cmd_flush 0                    #执行 flush 命令总数
STAT cmd_touch 0                    #执行 touch 命令总数
STAT get_hits 0                     #get 命中次数
STAT get_misses 0                   #get 未命中次数
STAT delete_misses 0                #delete 未命中次数
```

```
STAT delete_hits 0                    #delete 命中次数
STAT incr_misses 0                    #incr 未命中次数
STAT incr_hits 0                      #incr 命中次数
STAT decr_misses 0                    #decr 未命中次数
STAT decr_hits 0                      #decr 命中次数
STAT cas_misses 0                     #cas 未命中次数
STAT cas_hits 0                       #cas 命中次数
STAT cas_badval 0                     #使用擦拭次数
STAT touch_hits 0                     #touch 命中次数
STAT touch_misses 0                   #touch 未命中次数
STAT auth_cmds 0                      #认证处理的次数
STAT auth_errors 0                    #认证失败次数
STAT bytes_read 7                     #读取字节总数
STAT bytes_written 0                  #写入字节总数
STAT limit_maxbytes 134217728         #现在的内存大小为 128M
STAT accepting_conns 1                #目前接受的新接数
STAT listen_disabled_num 0            #失效的监听数
STAT threads 4                        #当前线程数
STAT conn_yields 0                    #连接操作主支放弃数目
STAT hash_power_level 16              #hash 等级
STAT hash_bytes 524288                #当前 hash 表等级
STAT hash_is_expanding 0              #hash 表扩展大小
STAT bytes 0                          #当前存储占用的字节数
STAT curr_items 0                     #当前存储数据总数
STAT total_items 0                    #启动以来存储的数据总数
STAT expired_unfetched 0              #已过期但未获取的对象数目
STAT evicted_unfetched 0              #已驱逐但未获取的对象数目
STAT evictions 0                      #LRU 释放的对象数目
STAT reclaimed 0                      #用已过期的数据条目来存储新数据的数目
END
##flush_all       ##清空所有选项，flush_all 实际上没有立即释放项目所占用的内存，而是在随后陆续有新的项目被储存时执行（这是由 memcached 的懒惰检测和删除机制决定的）。
##append 和 prepend          ##在 Value 后续追加或在其前面添加
```

5、memcached 图形工具(memcachephp)

```
##下载 memcachephp
[root@web ~]#wget https://codeload.github.com/lagged/memcache.php/zip/master
[root@web ~]#unzip memcache.php-master.zip
[root@web ~]#mv memcahch.php-master /var/www/html/memadmin
[root@web www]# ls memadmin/
apps  config.php  images  include  index.php  langs  LICENSE.txt  README.txt  views
##config.php 文件定义了管理员默认登录用户和密码，并设置要连接 memcached 服务器
```

火狐或者谷歌浏览器中输入：http://192.168.88.66/memadmin 即可看到如下图形化的 web 界面：

6、使用 libmemcached 的客户端工具:

访问 memcached 的传统方法是使用基于 perl 语言开发的 Cache::memcached 模块，这个模块在大多数 perl 代码中都能良好的工作，但也有着性能方面的问题。libMemcached 则是基于 C 语言开发的开源的 C/C++代码，它还提供了数个可以远程使用的 memcached 管理工具，如 memcat, memping，memstat，memslap 等。

(1) 编译安装 libmemcached，访问官网：http://libmemcached.org/libMemcached.html，下载相应版本的软件包

```
##下载 libmemcached
[root@web ~]#wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0
18.tar.gz
[root@web ~]# tar xf libmemcached-1.0.18.tar.gz
[root@web ~]#cd libmemcached-1.0.18
[root@web ~]#./configure
[root@web ~]#make && make install
[root@web ~]#ldconfig -v
```

(2) 客户端工具

```
[root@web ~]# memcat --servers=127.0.0.1:11211 mykey
[root@web ~]#memping
[root@web ~]#memslap --servers=127.0.0.1
[root@web ~]#memstat --servers=127.0.0.1
```

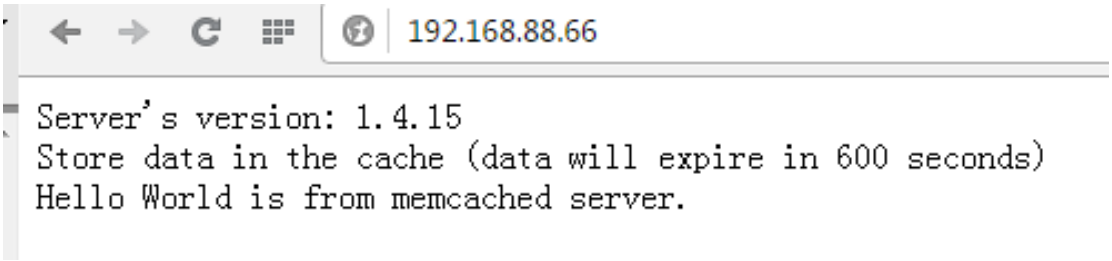
7、memcached 整合其他软件的运用：

(1)Memcached 的 PHP 扩展

官网 https://pecl.php.net/package/memcache

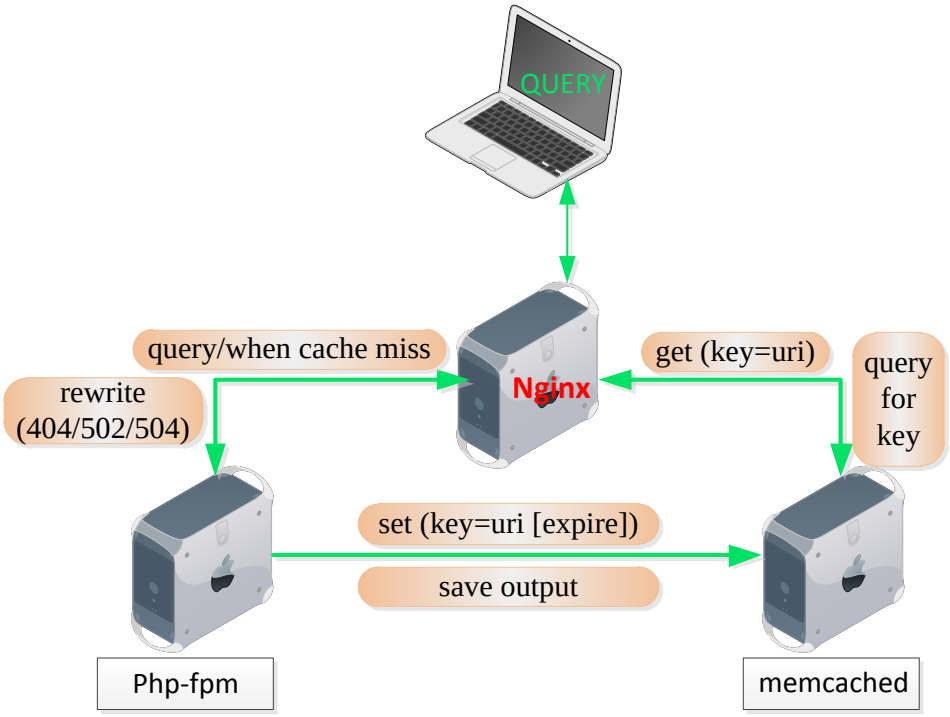
```
[root@web ~]#wget https://pecl.php.net/get/memcache-3.0.0.tgz
[root@web ~]#tar xvf memcache-3.0.0.tgz
[root@web ~]#cd memcache-3.0.0
[root@web ~]#/usr/local/php/phpize #采用 php 的命令生成 configure 文件
[root@web ~]#./configure --with-php-config=/usr/local/php/bin/php-config --enable-memcach
[root@web ~]#make && make install  ##在安装的最后会有如下提示信息输出
.............................
Installing shared extensions:/usr/local/php/lib/php/extensions/no-debug-non-zts-20131226/m
emcached.so
##将上述信息添加到下述文件中，使 memcache 加载进 php 中：
[root@web ~]#cat /etc/php.d/php.ini
extension=/usr/local/php/lib/php/extensions/no-debug-non-zts-20131226/memcached.so
[root@web ~]#service php-fpm restart
##而后对 memcached 功能进行测试，在网站目录中建立测试页面 test.php，添加如下内容：
[root@web ~]# vim /var/www/html/index.php
<?php
    $mem = new Memcache;
    $mem->connect("192.168.88.11", 11211)  or die("Could not connect");
    $mem->set('hellokey', 'Hello World', 0, 600) or die("Failed to save data at the memcac
hed server");
    echo "Store data in the cache (data will expire in 600 seconds)<br/>\n";
    $get_result = $mem->get('hellokey');
    echo "$get_result is from memcached server.";
?>
```

浏览器中输出测试页面：



(2)Nginx 整合 memcached:

nginx 的 memcached_module 模块可以直接从 memcached 服务器中读取内容后输出，后续的请求不再经过应用程序(如 php-fpm、django)处理,大大的提升动态页面的速度。nginx 只负责从 memcached 服务器中读取数据，要往 memcached 写入数据还得需要后台的应用程序来完成，主动的将要缓存的页面缓存到 memcached 中，可以通过 404 重定向到后端去处理的。



memcached 的 key 可以通过 memcached_key 变量来设置，如以$uri。如果命中，那么直接输出内容，没有命中就意味着 nginx 需要从应用程序请求页面。同时，还希望该应用程序将键值对写入到 memcached 服务器中，以便下一个请求可以直接从 memcached 获取。如果键值不存在，nginx 将报告 not found 错误(404)。最好的方法是使用 error_page(指定和 location 请求处理。同时包含”Bad Gateway” 502)错误和”Gateway Timeout”(504)错误，如：error_page 404 502 504 = @app;。注意：需要设置 default_type，否则可能会显示不正常。

1)Nginx 整合 memcached 的配置文件(/etc/nginx/nginx.conf)修改：

```
[root@web ~]# vim /etc/nginx/nginx.conf
.........................
server {
    location / {
             set $memcached_key "$uri?$args";
             memcached_pass     192.168.88.11:11211;
            # default_type       text/html;
             error_page         404 @fallback;    #调用一个叫 fallback 的 location
      }
     location @fallback {                        #定义一个叫 fallback 的 location
             proxy_pass  http://127.0.0.1;
      }
}
.......................
```

从配置文件可知，一个请求到达后，将 uri 作为 key 去 Memcached 服务器 192.168.88.11：11211 上查找 value，如果没有命中，则返回 404。这时通过 error_page 将 404 接收转到@fallback，返回给 Web 服务去处理请求。

2)Nginx 中关于 memcache 的模块说明：

模块名称	语法	模块含义	默认值
memcached_bind	memcached_bind address | off;	指定从哪个 IP 来连接 memcached 服务器	none
memcached_pass	**memcached_pass address:port or socket；**	**memcached 服务器地址:端口号**	**none**
memcached_buffer_size	memcached_buffer_size size;	从 memcached 服务器接收到响应的缓冲大小	4k|8k;
memcached_connect_timeout	memcached_connect_timeout time;	与 memcached 服务器建立连接的超时时间	60s
memcached_read_timeout	memcached_read_timeout time;	从 memcached 服务器读取响应超时时间	60s
memcached_send_timeout	memcached_send_timeout time；	发送请求到 memcached 服务器的超时时间	60s

模块名称：memcached_next_upstream

语法: memcached_next_upstream error | timeout | invalid_response | not_found | off …;

默认值： error timeout;

模块含义:哪些状态下请求将转发到另外的负载均衡服务器上(memcached_pass 有多个适用)。

注：以上各个模块在配置文件中的位置：http->server->location

附件：

```
##编译安装 memcached 的启动启动脚本
[root@web ~]# vim /etc/rc.d/init.d/memcached
#!/bin/bash
#
# chkconfig: 35 99 05
```

###35: runlevel 下启动，”-”代表关闭；

###99：在 rc3.d 和 rc5.d 下产生 S99fmemcached （越小优先权越高）

###05: 在 rc0.d、rc1.d、rc2.d、rc4.d、rc6.d 下产生 K05fmemcached（越小优先权越高）

####S:代表 Start

####K:代表 Kill

```
# description:  The memcached daemon is a network memory cache service.
# processname: memcached
# config: /etc/sysconfig/memcached
# pidfile: /var/run/memcached/memcached.pid
. /etc/init.d/functions
PORT=11211
USER=memcached
MAXCONN=1024
CACHESIZE=64
```

```
OPTIONS=""
[ -f /etc/sysconfig/memcached ]&&. /etc/sysconfig/memcached
. /etc/sysconfig/network
if [ "$NETWORKING" = "no" ]
then
        exit 0
fi
RETVAL=0
prog="/usr/local/memcache/bin/memcached"
pidfile=${PIDFILE-/var/ruan/memcached/memcached.pid}
lockfile=${LOCKFILE-/var/lock/subsys/memcached}
start () {
        echo -n $"Starting $prog: "
        # Ensure that /var/run/memcached has proper permissions
        if [ "`stat -c %U /var/run/memcached`" != "$USER" ]; then
                chown $USER /var/run/memcached
        fi
        daemon --pidfile ${pidfile} memcached -d -p $PORT -u $USER  -m $CACHESIZE -c $MAXCONN
-P ${pidfile} $OPTIONS
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && touch ${lockfile}
}
stop () {
        echo -n $"Stopping $prog: "
        killproc -p ${pidfile} /usr/bin/memcached
        RETVAL=$?
        echo
        if [ $RETVAL -eq 0 ] ; then
                rm -f ${lockfile} ${pidfile}
        fi
}
restart () {
       stop
       start
}
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  status)
        status -p ${pidfile} memcached
        RETVAL=$?
        ;;
  restart|reload|force-reload)
        restart
        ;;
  condrestart|try-restart)
        [ -f ${lockfile} ] && restart || :
        ;;
  *)
        echo $"Usage: $0
{start|stop|status|restart|reload|force-reload|condrestart|try-restart}"
        RETVAL=2
       ;;
```

```
esac
exit $RETVAL
##以下是 memcached 启动时的配置信息，可手动编辑：
[root@web ~]#cat /etc/sysconfig/memcached
PORT="11211"
USER="memcached"
MAXCONN="1024"
CACHESIZE="64"
OPTIONS=""
[root@web ~]#chmod +x /etc/rc.d/init.d/memcached
[root@web ~]#chkcon --add memcached
[root@web ~]#service memcached start
```