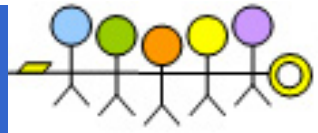
目录



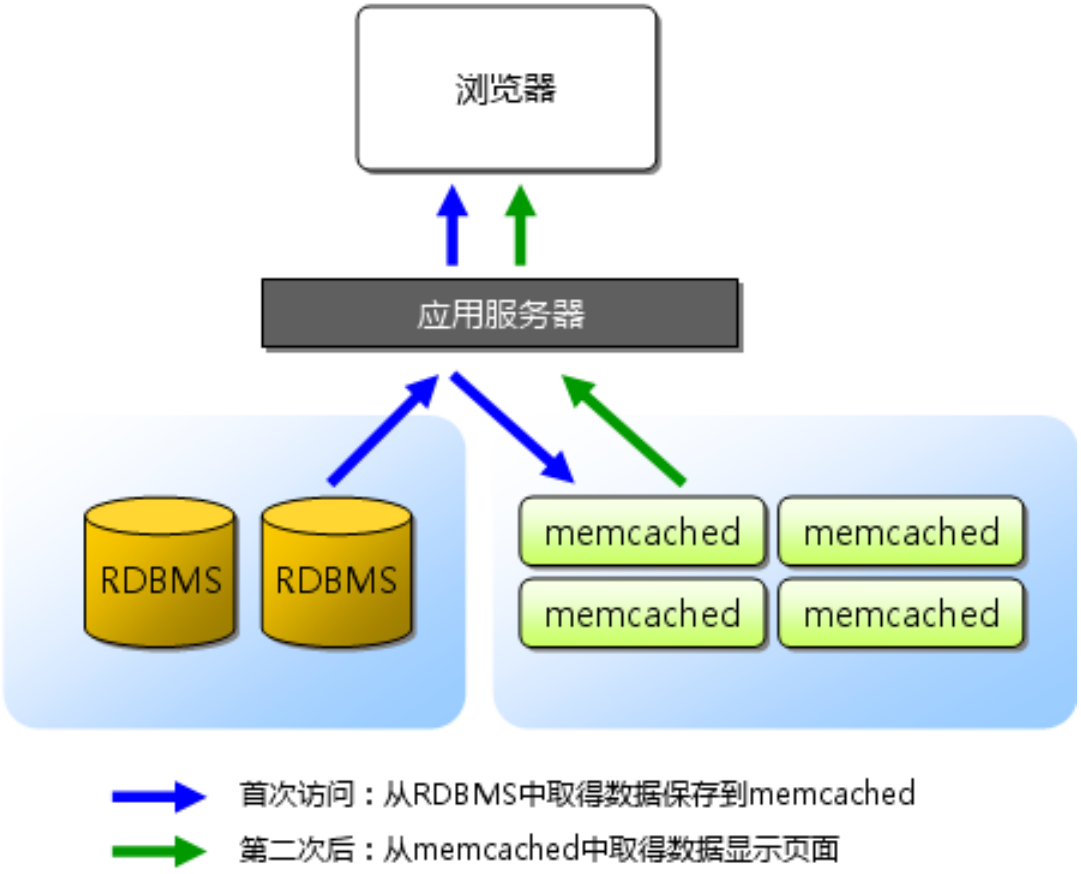
介绍

Memcached是国外社区网站 LiveJournal 的开发团队开发的高性能的分布式内存缓存服务器。一般的使用目的是，通过缓存数据库查询结果，减少数据库访问次数，以提高动态Web应用的速度、提高可扩展性。

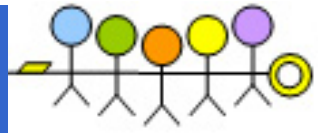
memcache是一个自由和开放源代码、高性能、分布式的内存对象缓存系统。用于加速动态web应用程序，减轻数据库负载。

Memcached运行图

Memcached运行图



Meacached特征



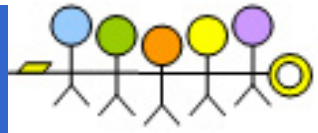
I. Memcached特征

1. 基于C/S架构，协议简单；
2. 基于libevent的事件处理；
3. 内置内存存储方式；
4. 基于客户端的memcached分布式。

II. 适用场景

1. 需要分布式部署的；
2. 需要频繁访问相同数据的；
3. 需要数据共享的。

介绍C/S架构



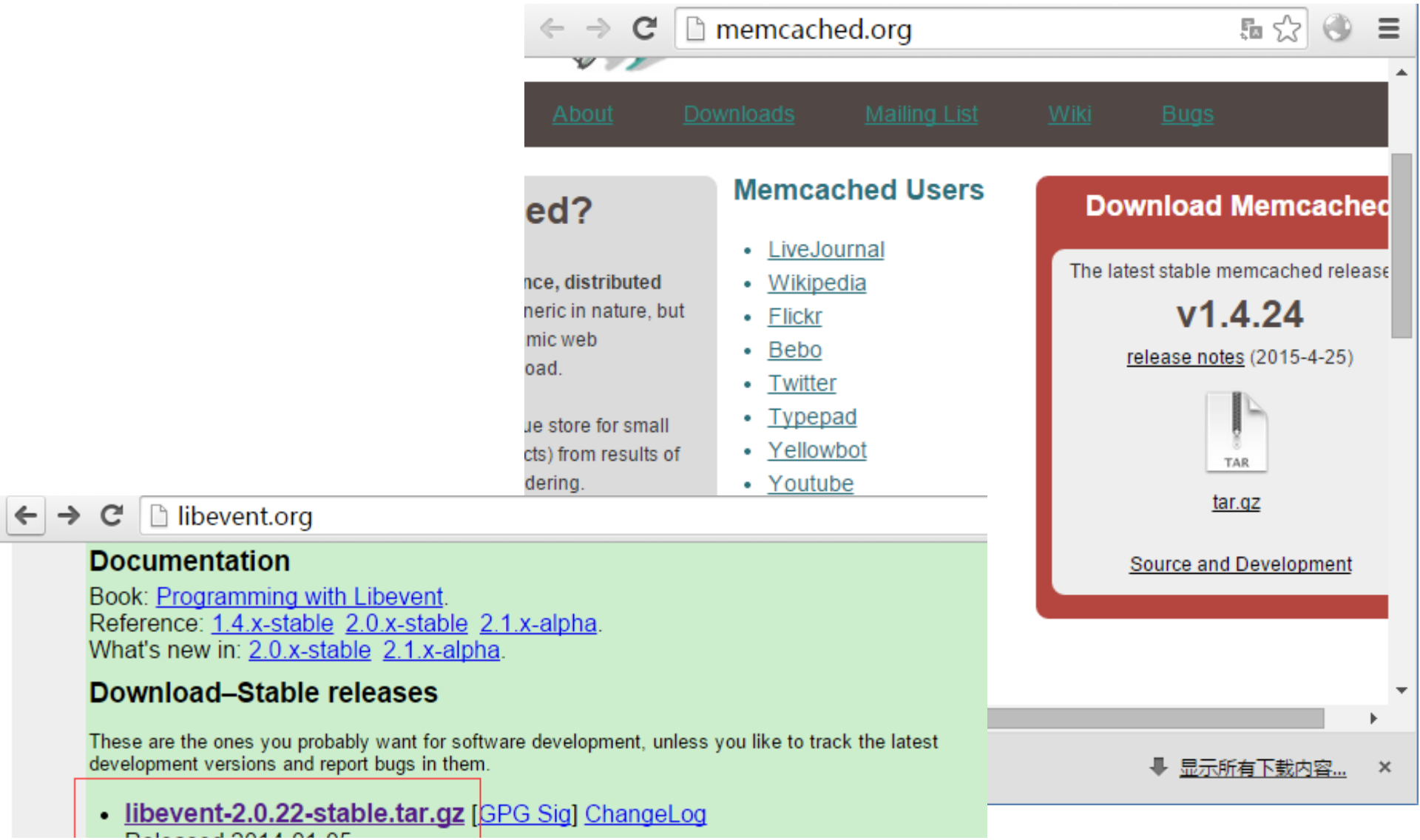
I. 基于C/S架构，协议简单

1. 服务端启动memcached进程；
2. 客户端可以通过telnet操作，也可以通过各种编程语言实现的客户端程序存取数据及查询状态；
3. memcached的服务器与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行的协议

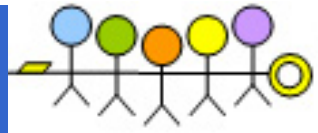
II. 基于libevent的事件处理

1. libevent是一套跨平台的事件处理接口的封装，能够兼容包括这些操作系统：Windows/Linux/BSD/Solaris 等操作系统的的事件处理；
2. 包装的接口包括：poll、select(Windows)、epoll(Linux)、kqueue(BSD)、/dev/pool(Solaris)；
3. Memcached 使用libevent来进行网络并发连接的处理，能够保持在很大并发情况下，仍旧能够保持快速的响应能力。

安装



安装



火龙果·整理
uml.org.cn

Downloads

Latest stable

memcached-1.4.24.tar.gz (2015-4-25) (release notes) (sha1: 32a798a37ef782da10a09d74aa1e5be91f2861db)

Older releases

Full list of releases

Installation

Debian/Ubuntu: apt-get install libevent-dev Redhat/Centos: yum install libevent-devel

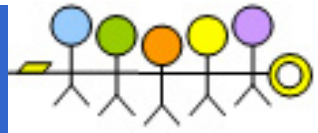
```
wget http://memcached.org/latest
tar -zxvf memcached-1.x.x.tar.gz
cd memcached-1.x.x
./configure && make && make test && sudo make install
```

See the wiki for further information

```
checking for library containing gethostbyname... none required
checking for libevent directory... configure: error: libevent is required.  You can get it from http://www.monkey.org/~provos/libevent/

      If it's already installed, specify its path using --with-libevent=/dir/

[root@AY140102162314127f09Z memcached-1.4.24]# make
```

到http://www.monkey.org/~provos/libevent/ 下

MEMCACHED启动



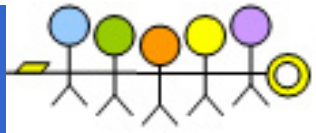
火龙果·整理
uml.org.cn

```
memcached -d -m 1024 -u root -l 115.29.47.23 -p 11211 -c 1024 -P /data/tmp/memcached.pid

ps -ef|grep mem
memcached -d -m 1024 -u root -l 115.29.47.23 -p 11211 -c 1024 -P /data/tmp/memcached.pid
```

这算启动成功

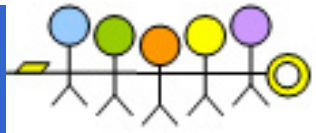
```
[root@AY140102162314127f09Z libevent-2.0.22-stable]# ps -ef|grep mem
root      3019     1  0 14:34 ?        00:00:00 memcached -d -m 1024 -u root -l 115.29.47.23 -p 11211 -c 1024 -P /data/tmp/memcached.pid
root      3692 24054  0 14:43 pts/0    00:00:00 grep mem
[root@AY140102162314127f09Z libevent-2.0.22-stable]#
```

MEMCACHED启动成功




```
WARNING! The remote SSH server rejected X11 forwarding request.
Last login: Fri Jun  8 17:14:11 2012 from 172.16.93.56
This is a private network server, in monitoring state.
It is strictly prohibited to unauthorized access and used.
[op1@SVR1693HP360 ~]$ memcached -h
memcached 1.4.13
-p <num>      TCP port number to listen on (default: 11211)
-U <num>      UDP port number to listen on (default: 11211, 0 is off)
-s <file>     UNIX socket path to listen on (disables network support)
-a <mask>     access mask for UNIX socket, in octal (default: 0700)
-l <addr>     interface to listen on (default: INADDR_ANY, all addresses)
              <addr> may be specified as host:port. If you don't specify
              a port number, the value you specified with -p or -U is
              used. You may specify multiple addresses separated by comma
              or by using -l multiple times
-d            run as a daemon
-r            maximize core file limit
-u <username> assume identity of <username> (only when run as root)
-m <num>      max memory to use for items in megabytes (default: 64 MB)
-M            return error on memory exhausted (rather than removing items)
-c <num>      max simultaneous connections (default: 1024)
-k            lock down all paged memory.  Note that there is a
              limit on how much memory you may lock.  Trying to
              allocate more than that would fail, so be sure you
              set the limit correctly for the user you started
              the daemon with (not for -u <username> user;
              under sh this is done with 'ulimit -S -l NUM_KB').
-v            verbose (print errors/warnings while in event loop)
-vv           very verbose (also print client commands/reponses)
-vvv          extremely verbose (also print internal state transitions)
-h            print this help and exit
-i            print memcached and libevent license
-P <file>     save PID in <file>, only used with -d option
-f <factor>   chunk size growth factor (default: 1.25)
-n <bytes>    minimum space allocated for key+value+flags (default: 48)
```

MEMCACHED启动



启动方式：

-d　　以守护程序（daemon）方式运行

-u root　　指定用户，如果当前为 root ，需要使用此参数指定用户

-P /tmp/a.pid　　保存PID到指定文件

内存设置：

-m 1024　　数据内存数量，不包含memcached本身占用，单位为 MB

-M　　内存不够时禁止LRU，报错

-n 48　　初始chunk=key+suffix+value+32结构体，默认48字节

-f 1.25　　增长因子，默认1.25

-L　　启用大内存页，可以降低内存浪费，改进性能

连接设置：

-l 127.0.0.1　　监听的 IP 地址，本机可以不设置此参数

-p 11211　　TCP端口，默认为11211，可以不设置

-U 11211　　UDP端口，默认为11211，0为关闭

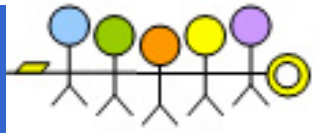
并发设置：

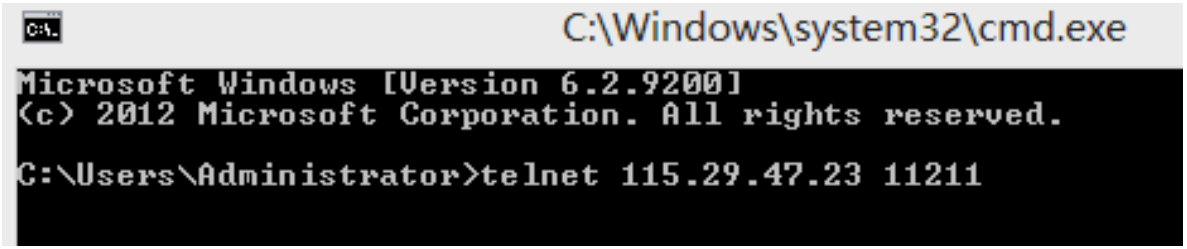
-c 1024　　最大并发连接数，默认1024，最好是200

-t 4　　线程数，默认4。由于memcached采用NIO，所以更多线程没有太多作用

-R 20　　每个event连接最大并发数，默认20

客户端连接





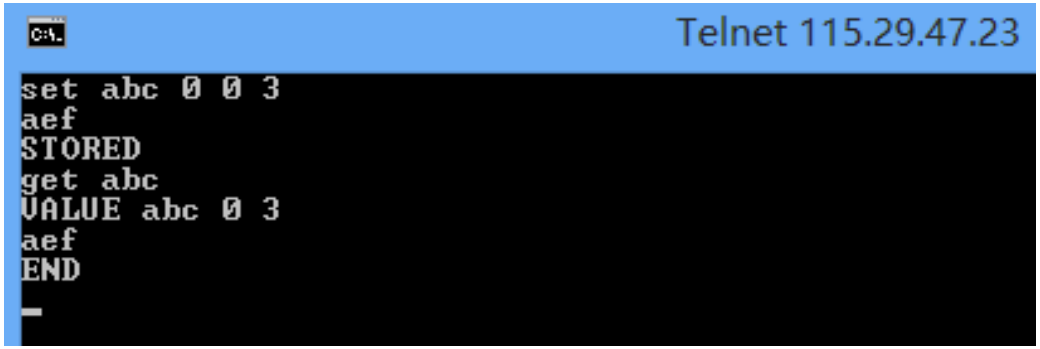
这是因为存储的字节长度与指定的长度不匹配造成的，如：

```
1. set username 0 0 2
```

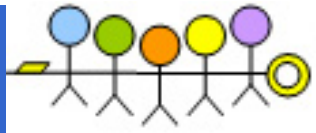
你是打算存储两个字节，但如果你输入不等于2个字节就会报

> CLIENT_ERROR bad data chunk

错误，注意必须是2个，多于或少于2个字节都会报这个错误。



客户端常用的命令



火龙果 · 整理
uml.org.cn

一、存储命令

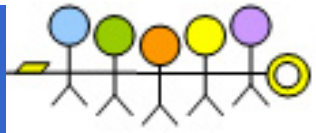
存储命令的格式：

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

参数说明如下：

<command name>	set/add/replace
<key>	查找关键字
<flags>	客户机使用它存储关于键值对的额外信息
<exptime>	该数据的存活时间，0表示永远
<bytes>	存储字节数
<data block>	存储的数据块（可直接理解为key-value结构中的value）

客户端常用的命令



火龙果 · 整理
uml.org.cn

1、添加

(1)、无论如何都存储的set

```
set username 0 0 8
jeffwong
STORED
get username
VALUE username 0 8
jeffwong
END
set username 0 0 7
jeffery
STORED
get username
VALUE username 0 7
jeffery
END
```

这个set的命令在memcached中的使用频率极高。set命令不但可以简单添加，如果set的key已经存在，

可以通过"get 键名"的方式查看添加进去的记录：

```
set username 0 10 8
jeffwong
STORED
get username
VALUE username 0 8
jeffwong
END
```

如你所知，我们也可以通过delete命令删除掉，然后重新添加。

```
set username 0 0 7
jeffery
STORED
delete username
DELETED
delete username
```

(2)、只有数据**不存在**时进行添加的add

```
add newuser 0 0 7
jeffery
STORED
add newuser 0 0 7
jackson
NOT_STORED
```

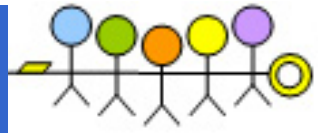
(3)、只有数据**存在**时进行替换的replace

```
replace notexist 0 0 7
jeffery
NOT_STORED
set notexist 0 0 7
jackson
STORED
replace notexist 0 0 7
jeffery
STORED
get notexist
VALUE notexist 0 7
jeffery
END
```

2、删除

```
set username 0 0 7
jeffery
STORED
delete username
DELETED
delete username
NOT_FOUND
```

客户端常用的命令



二、读取命令

1、get

get命令的key可以表示一个或者多个键，键之间以空格隔开

```
get username
VALUE username 0 7
jeffery
END
get notexist
VALUE notexist 0 8
jeffwong
END
get username notexist
VALUE username 0 7
jeffery
VALUE notexist 0 8
jeffwong
END
```

2、gets

```
get username
VALUE username 0 7
jeffery
END
gets username
VALUE username 0 7 13
jeffery
END
replace username 0 0 8
jeffwong
STORED
gets username
VALUE username 0 8 15
jeffwong
END
```

3、cas

cas即checked and set的意思，只有当最后一个参数和gets所获取的参数匹配时才能存储，否则返回"EXISTS"。

```
gets username
VALUE username 0 8 15
jeffwong
END
cas username 0 0 8 16
abcdefgh
EXISTS
cas username 0 0 8 15
abcdefgh
STORED
get username
VALUE username 0 8
abcdefgh
END
```

可以看到，gets命令比普通的get命令多返回了一个数字（上图中为13）。这个数字可以检查数据是否发生改变。

客户端常用的命令

火龙果 · 整理
uml.org.cn

三、状态命令

1、stats

```
stats
STAT pid 1988
STAT uptime 4084
STAT time 1320565332
STAT version 1.2.4
STAT pointer_size 32
STAT curr_items 3
STAT total_items 16
STAT bytes 193
STAT curr_connections 1
STAT total_connections 3
STAT connection_structures 2
STAT cmd_get 25
STAT cmd_set 24
STAT get_hits 19
STAT get_misses 6
STAT evictions 0
STAT bytes_read 1324
STAT bytes_written 1103
STAT limit_maxbytes 67108864
STAT threads 1
END
```

2、stats items

```
stats items
STAT items:1:number 3
STAT items:1:age 4095
END
```

执行stats items，可以看到STAT items行，如果memcached存储内容很多，那么

四、其他常见命令

1、append

```
get username
VALUE username 0 8
jeffwong
END
append username 0 0 8
 cnblogs
STORED
get username
VALUE username 0 16
jeffwong cnblogs
END
```

在现有的缓存数据后添加缓存数据，如现有缓存的key不存在服务器响应为NOT_STORED。

2、prepend

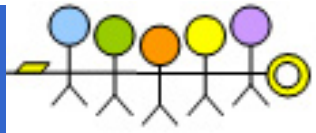
和append非常类似，但它的作用是在现有的缓存数据前添加缓存数据。

```
get username 0 0 8
jeffwong
STORED
get username
VALUE username 0 8
jeffwong
END
prepend username 0 0 8
cnblogs
STORED
get username
VALUE username 0 16
cnblogs jeffwong
END
```

3、flush_all

```
get username 0 0 8
jeffwong
STORED
get username
VALUE username 0 8
jeffwong
END
flush_all
OK
get username
```

memcached调试

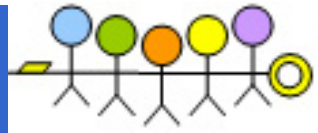


- -v +输出error/warning
- -vv +输出命令/响应
- -vvv +输出内部状态

```
[@10_10_82_80 ~]# memcached -d -u root -m 1024 -p 11210 -vvv
[@10_10_82_80 ~]# memcached -d -u root -m 1024 -p 11211  – vvv
slab class   1: chunk size       96 perslab   10922
slab class   2: chunk size      120 perslab    8738
......
slab class  42: chunk size   1048576 perslab       1
<36 server listening (auto-negotiate)
<37 send buffer was 126976, now 268435456
<37 server listening (udp)
<37 server listening (udp)
<37 server listening (udp)
<37 server listening (udp)
<38 new auto-negotiating client connection
38: Client using the ascii protocol
<38 stats
<38 get abc
>38 END
<38 quit
```

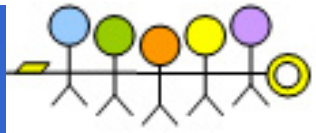
```
telnet localhost 11210/11211
stats
get abc
quit
```

memcached命令列表



- 存储命令set/add/replace/append/prepend/cas
- 读取命令get=bget?/gets
- 删除命令delete
- 计数命令incr/decr
- 统计命令stats/settings/items/sizes/slabs
- 工具memcached-tool

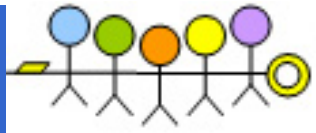
存储命令



格式：
<command> <key> <flags> <exptime> <bytes> [<version>]\r\n
<datablock>\r\n
<status>\r\n

command	set无论如何都进行存储 add只有数据不存在时进行添加 repalce只有数据存在时进行替换 append往后追加：append <key> datablock <status>? prepend往前追加：prepend <key> datablock <status> cas按版本号更改
key	字符串，<250个字符，不包含空格和控制字符
flags	客户端用来标识数据格式的数值，如json,xml,压缩等
exptime	存活时间s，0为永远，<30天60*60*24*30为秒数，>30天为unixtime
bytes	byte字节数，不包含\r\n，根据长度截取存/取的字符串，可以是0，即存空串
datablock	文本行，以\r\n结尾，当然可以包含\r或\n
status	STORED/NOT_STORED/EXISTS/NOT_FOUND ERROR/CLIENT_ERROR/SERVER_ERROR服务端会关闭连接以修复

存储命令set/add/replace



datablock长度必须正确

```
set liu 32 0 4
java
STORED//正确

get liu
VALUE abc 32 4
java
END

set liu 32 0 4
cplus
CLIENT_ERROR bad data
chunk
ERROR//长度错误
```

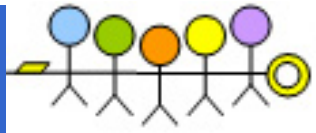
add只能添加不存在的key

```
set liu 32 0 4
java
STORED

add liu 32 0 5
cplus
NOT_STORED
//已存在不能add

get liu
VALUE abc 32 4
java
END

add song 32 0 5
cplus
STORED
//不存在可以add
```

replace只能替换已有的key

```
set liu 32 0 4
java
STORED

replace liu 32 0 5
cplus
STORED
//已存在可以replace

get  liu
VALUE cplus 32 5
liu
END

replace yang 32 0 5
cplus
NOT_STORED
//不存在不能replace
```

读取命令get/gets



格式:
<command> <key>*\r\n
VALUE <key1> <flags> <bytes> [<version>]\r\n
<datablock>\r\n
…
VALUE <keyn> <flags> <bytes> [<version>]\r\n
<datablock>\r\n
END\r\n

command: get普通查询，gets用于查询带版本的值

```
get liu song yang
VALUE liu 32 4
java
VALUE song 32 5
cplus
END
//查询多个键值
```

```
gets liu
VALUE liu 32 4 12
java
END
//取得版本号

replace liu 32 0 4
java
STORED
//增加版本号

get liu
VALUE liu 32 4
java
END

gets liu
VALUE liu 32 4 13
java
END
```

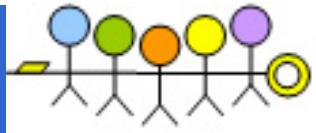
版本号

检查存储命令cas

cas即check and set，只有版本号相匹配时才能存储，否则返回EXISTS

设计意图：解决多客户端并发修改同一条记录的问题，防止使用经过改变了的value/key对

计数命令incr/decr



格式：
incr/decr<key> <int>
<int>
要求：key必须存在，value必须是数字

实现计数器

```
set count 32 0 1
1
STORED

incr count 8
9

decr count 2
7
```

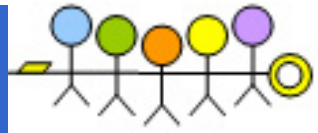
key不存在不能计数

```
delete count
DELETED

incr count 1
NOT_FOUND
```

value不是数字不能计数

```
incr liu 2
CLIENT_ERROR cannot
increment or decrement
non-numeric value
```

删除命令delete

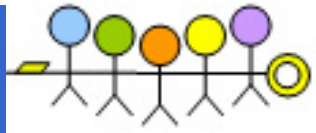


格式:
delete <key> [<time>]
DELETE\r\n

time: 秒数或Unixtime，在time时间内不能add或replace，但能set，不能get。过期后才能够重新set有效并能get

```
delete liu
DELETED
get liu
END
```

统计命令stats

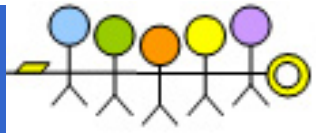


格式:
stats [<args>]\r\n
STAT <name> <value>\r\n
END\r\n

```
stats
STAT pid 23178
STAT uptime 1039318
STAT time 1292036037
STAT version 1.4.2
STAT pointer_size 64
STAT rusage_user 1011.574217
STAT rusage_system 1677.713948
STAT curr_connections 114
STAT total_connections 73801
STAT connection_structures 149
STAT cmd_get 79114939
STAT cmd_set 27302514
STAT cmd_flush 0
STAT get_hits 79114939
STAT get_misses 24322507
STAT delete_misses 133928
STAT delete_hits 402569
STAT incr_misses 0
STAT incr_hits 0
```

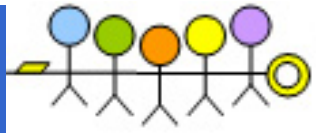
```
STAT decr_misses 0
STAT decr_hits 0
STAT cas_misses 0
STAT cas_hits 0
STAT cas_badval 0
STAT auth_cmds 0
STAT auth_errors 0
STAT bytes_read 59348603658
STAT bytes_written 425549797158
STAT limit_maxbytes 4294967296
STAT accepting_conns 1
STAT listen_disabled_num 0
STAT threads 4
STAT conn_yields 0
STAT bytes 3832761746
STAT curr_items 2854731
STAT total_items 27302514
STAT evictions 18456987
STAT reclaimed 0
END
```

stats统计项



名称	描述	
pid	Memcached进程ID	
uptime	Memcached运行时间，单位：秒	
time	Memcached当前的UNIX时间	
version	Memcached的版本号	
rusage_user	该进程累计的用户时间，单位：秒	分析CPU占用是否高
rusage_system	该进程累计的系统时间，单位：秒	
curr_connections	当前连接数量	分析连接数是否太多
total_connections	Memcached运行以来接受的连接总数	
connection_structures	Memcached分配的连接结构的数量	
cmd_get	查询请求总数	分析命中率是否太低
get_hits	查询成功获取数据的总次数	
get_misses	查询成功未获取到数据的总次数	
cmd_set	存储（添加/更新）请求总数	
bytes	Memcached当前存储内容所占用字节数	分析字节数流量
bytes_read	Memcached从网络读取到的总字节数	
bytes_written	Memcached向网络发送的总字节数	
limit_maxbytes	Memcached在存储时被允许使用的字节总数	
curr_items	Memcached当前存储的内容数量	分析对象数LRU频率
total_items	Memcached启动以来存储过的内容总数	
evictions	LRU释放对象数，用来释放内存	

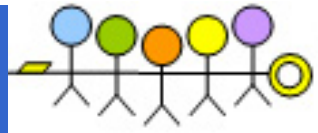
stats settings查看设置




```
stats settings
STAT maxbytes 0
STAT maxconns 1024
STAT tcpport 11213
STAT udpport 11211
STAT inter NULL
STAT verbosity 0
STAT oldest 0
STAT evictions on
STAT domain_socket NULL
STAT umask 700
STAT growth_factor 1.25
STAT chunk_size 48
STAT num_threads 4
STAT stat_key_prefix :
STAT detail_enabled no
STAT reqs_per_event 20
STAT cas_enabled yes
STAT tcp_backlog 1024
STAT binding_protocol auto-negotiate
STAT item_size_max 1048576
END
```

名称	描述
maxbytes	最大字节数限制，0无限制
maxconns	允许最大连接数
tcpport	TCP端口
udpport	UDP端口
inter	
verbosity	日志0=none,1=som,2=lots
oldest	最老对象过期时间
evictions	on/off，是否禁用LRU
domain_socket	socket的domain
umask	创建Socket时的umask
growth_factor	增长因子
chunk_size	key+value+flags大小
num_threads	线程数，可以通过-t设置，默认4
stat_key_prefix	stats分隔符
detail_enabled	yes/no，显示stats细节信息
reqs_per_event	最大IO吞吐量(每event)
cas_enabled	yes/no，是否启用CAS，-C禁用
tcp_backlog	TCP监控日志
auth_enabled_sasl	yes/no，是否启用SASL验证

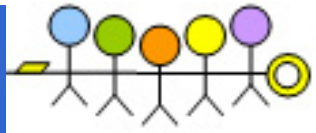
stats items数据项统计




```
stats items
STAT items:1:number 10922
STAT items:1:age 350988
STAT items:1:evicted 3829
STAT items:1:evicted_nonzero 0
STAT items:1:evicted_time 690209
STAT items:1:outofmemory 0
STAT items:1:tailrepairs 0
STAT items:2:number 375734
STAT items:2:age 898762
STAT items:2:evicted 2661399
STAT items:2:evicted_nonzero 0
STAT items:2:evicted_time 142500
STAT items:2:outofmemory 0
STAT items:2:tailrepairs 0
...
STAT items:40:number 14
STAT items:40:age 977359
STAT items:40:evicted 25
STAT items:40:evicted_nonzero 0
STAT items:40:evicted_time 60653
STAT items:40:outofmemory 0
STAT items:40:tailrepairs 0
END
```

名称	描述
number	该slab中对象数，不包含过期对象
age	LRU队列中最老对象的过期时间
evicted	LRU释放对象数
evicted_nonzero	设置了非0时间的LRU释放对象数
evicted_time	最后一次LRU秒数，监控频率
outofmemory	不能存储对象次数，使用-M会报错
tailrepairs	修复slabs次数
reclaimed	使用过期对象空间存储对象次数

stats sizes对象数量统计

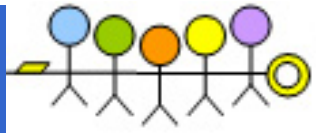



```
stats sizes
STAT 96 10922
STAT 128 375734
STAT 160 200416
STAT 192 816311
STAT 224 8685
STAT 256 3321
STAT 288 3549
STAT 320 826
STAT 352 427
END
```

格式：STAT <size> <count>

注意：会锁定服务，暂停处理请求

stats slabs区块统计

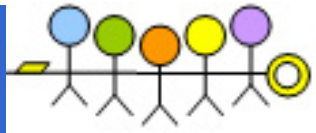



```
stats slabs
STAT 1:chunk_size 96
STAT 1:chunks_per_page 10922
...
STAT 40:chunk_size 1048576
STAT 40:chunks_per_page 1
STAT 40:total_pages 15
STAT 40:total_chunks 15
STAT 40:used_chunks 14
STAT 40:free_chunks 1
STAT 40:free_chunks_end 0
STAT 40:mem_requested 9348752
STAT 40:get_hits 9593
STAT 40:cmd_set 4828
STAT 40:delete_hits 40
STAT 40:incr_hits 0
STAT 40:decr_hits 0
STAT 40:cas_hits 0
STAT 40:cas_badval 0
STAT active_slabs 40
STAT total_malloced 4294496616
END
```

名称	描述	
chunk_size	chunk大小，byte	区块数量
chunks_per_page	每个page的chunk数量	
total_pages	page数量	
total_chunks	chunk数量*page数量	
get_hits	get命中数	命中率
cmd_set	set数	
delete_hits	delete命中数	
incr_hits	incr命中数	
decr_hits	decr命中数	
cas_hits	cas命中数	
cas_badval	cas数据类型错误数	
used_chunks	已被分配的chunk数	分析占用情况
free_chunks	剩余chunk数	
free_chunks_end	分完page浪费chunk数	
mem_requested	请求存储的字节数	
active_slabs	slab数量	
total_malloced	总内存数量	

被浪费内存数=(total_chunks * chunk_size) - mem_requested

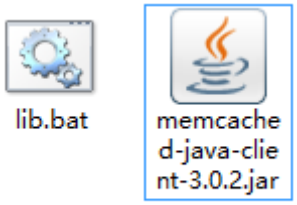
其他命令



- version
- flush_all
- quit
- echo/nc快捷方式

```
[@10_10_82_80 ~]# telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
version
VERSION 1.4.5
flush_all
OK
quit
Connection closed by foreign host.
You have new mail in /var/spool/mail/root

[@10_10_82_80 ~]# echo flush_all | nc localhost 11211
OK

[@10_10_82_80 ~]# echo get liu | nc localhost 11210
VALUE liu 32 4
java
END
```

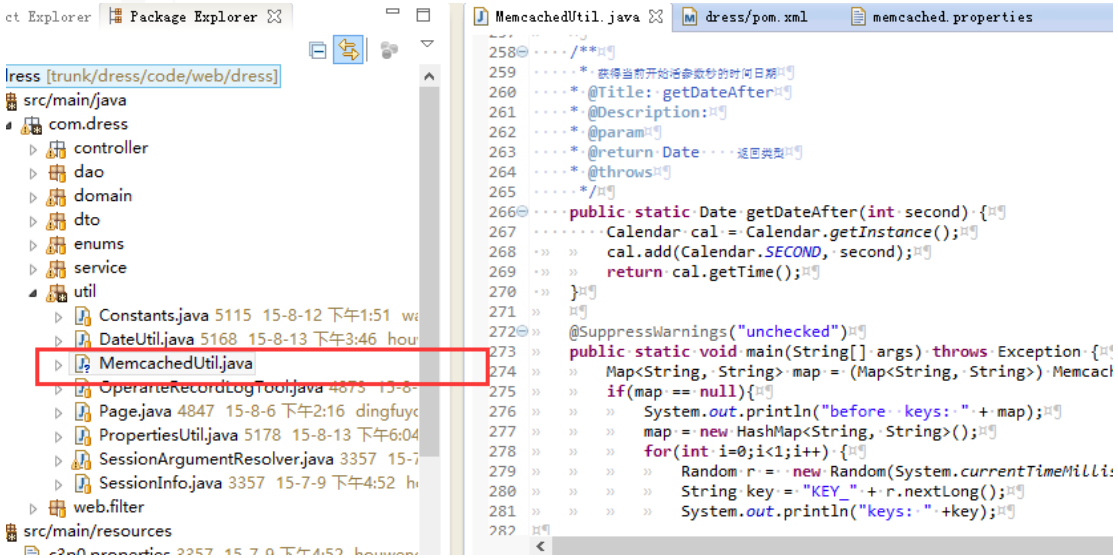
JAVA客户端的写法



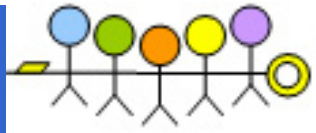
MemcachedUtil.java | dress/pom.xml | memcached.properties

```
1 memcache.cache.server=115.29.47.23:11211
2 cache.enable=true
```

```
<dependency>
    <groupId>com.danga</groupId>
    <artifactId>memcached</artifactId>
    <version>3.0.2</version>
</dependency>
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.2</version>
</dependency>
<dependency>
    <groupId>commons-pool</groupId>
    <artifactId>commons-pool</artifactId>
    <version>1.6</version>
</dependency>
```



JAVA客户端的写法



火龙果·整理
uml.org.cn

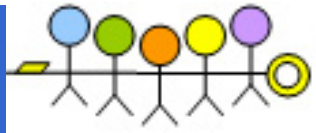
```
@SuppressWarnings("unchecked")
public static void main(String[] args) throws Exception {
    Map<String, String> map = (Map<String, String>) MemcachedUtil.getInstance().get("af");
    if(map == null){
        System.out.println("before keys: " + map);
        map = new HashMap<String, String>();
        for(int i=0;i<1;i++) {
            Random r = new Random(System.currentTimeMillis());
            String key = "KEY_" + r.nextLong();
            System.out.println("keys: " +key);

            byte[] bt = new byte[0];
            String value= new String(bt,"UTF-8");
            for(int j=0;j<10;j++) {
                value= value + "abc";
            }
            map.put(key, value);
        }

        boolean a =MemcachedUtil.getInstance().set("af",MemcachedUtil.TWO_HOUR_SECOND, map);

        boolean b = MemcachedUtil.getInstance().set("ac",MemcachedUtil.TWO_HOUR_SECOND, "2");
        System.out.println(a + ", " + b);
    }

    System.out.println( MemcachedUtil.getInstance().get("ac")+"after keys: " + map.size());
```

JAVA客户端的写法

MemcachedUtil.ja

```
public class MemcachedUtil {
    private static Object LOCK = new Object();
    private final static Log loger = LogFactory.getLog(MemcachedUtil.class);
    private Properties properties;
    private MemCachedClient memCachedClient;
    public final static int TWO_HOUR_SECOND=7200;
    public final static int FIVE_HOUR_SECOND=5*60*60;
    public final static int TEN_HOUR_SECOND=10*60*60;
    public final static String memcacheMapKey="MEMACACHE_MAP_KEY";

private MemcachedUtil(){
    init();
}

private boolean isCacheEnabled() {
    boolean useCache = false;
    try {
        useCache = Boolean.valueOf(properties.getProperty("cache.enable"));
    } catch (Exception e) {
        useCache = false;
        loger.info("Please enable memcached");
    }
    return useCache;
}

/**
 * 改用嵌套类静态实始化
 * @return
 */
public static MemcachedUtil getInstance() {
    if(instance == null){
        synchronized(LOCK) {
            if (instance==null) {
                instance=new MemcachedUtil();
            }
        }
    }
    return instance;

ource("memcached.properties");
tream());

roperty("memcache.cache.server").split(",");
stance("dataServer");

ISTENT_HASH);
client实例*/
ent("dataServer");
```

客户端常用的命令



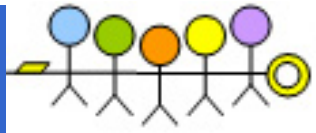

```
/**
 * 将KEY保存到memcache中
 *
 * @param key
 * @param exp
 * @param obj
 * @return
 */
public boolean set(String key,Date exp,Object obj){
    if(StringUtils.isEmpty(key)){
        return false;
    }
    if(obj==null){
        return true;
    }
    try{
        if (isCacheEnabled()) {
            boolean result = memCachedClient.set(key, obj, exp);
            loger.debug("SET A OBJECT: KEY:" + key + ", OBJ:" + obj + ", exp:" + exp + ", result:" + result);
            return result;
        }
        return true;
    }catch(Exception e) {
        e.printStackTrace();
        loger.error(e, e);
    }
    return false;
}

```

客户端常用的命令

火龙果·整理
uml.org.cn

```
public Object get(String key) {
    try{
        if (isCacheEnabled()) {
            Object obj = memCachedClient.get(key);
            loger.debug("GET A OBJECT: KEY:" + key + " OBJ:" + (obj==null)) ;
            return obj;
        }
    }catch(Exception e) {
        e.printStackTrace();
        loger.error(e, e);
    }
    return null;
}

/**
 * 取
 * @param key
 * @return
 */
public boolean remove(String key) {
    if(StringUtils.isEmpty(key)){
        return false;
    }
    try{
        if (isCacheEnabled()) {
            loger.info("delete memcached key: " + key);
            return memCachedClient.delete(key);
        }
    }catch(Exception e) {
        e.printStackTrace();
        loger.error(e, e);
    }
    return true;
```

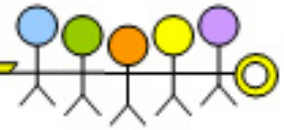
客户端常用的命令



实际使用

```
public class ProductInfoServiceImpl implements ProductInfoService {
    
    @Autowired
    private SysDictionaryMapper sysDictionaryMapper;
    @Autowired
    private ProductStyleMapper productStyleMapper;
    
    @SuppressWarnings("unchecked")
    public List<SysDictionary> clothesTypeList() {
        List<SysDictionary> result;
        String clothesTypeListKey="CLOTHESTYPELIST_KEY";
        Object value = MemcachedUtil.getInstance().get(clothesTypeListKey);
        if (value == null) {
            SysDictionary dic = new SysDictionary();
            dic.setType(SysDictionaryTypeEnum.CLOTHESTYPE.getCode());
            dic.setIsDelete("N");
            result = sysDictionaryMapper.selectByCondition(dic);
            MemcachedUtil.getInstance().set(clothesTypeListKey,MemcachedUtil.TWO_HOUR_SECOND, result);
        } else {
            result = (ArrayList<SysDictionary>) value;
        }
        
        return result;
    }

```

JAVA客户端使用注意事项

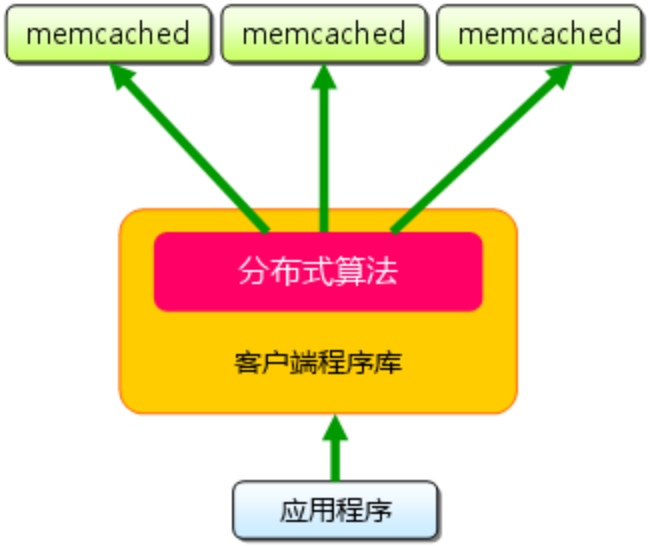


1. Memcached通过libevent完成socket 的通讯，理论上性能的瓶颈落在网卡上。
2. memcached 不是一个数据库，他只是内存，因此，如果 用 memcached，那么 缓存不是信息的唯一来源，不能把 memcached 当做数据库来使用，如果 memcached 宕 掉，可以 通过数据库进行 重新 加载 。
3. 使用简单的 key-value 存储的话， Memcached 的内存利用率更高。
4. 频繁使用小数据，将这些小数据进行缓存，定期持久化就可以了，查询操作一直都在内存中运行。
5. Memcached 仅限于缓存数据的应用，即使当机了，也不影响业务正常开展的这种

分布式和热备都是在客户端做的，当然，服务器端也可以做热备

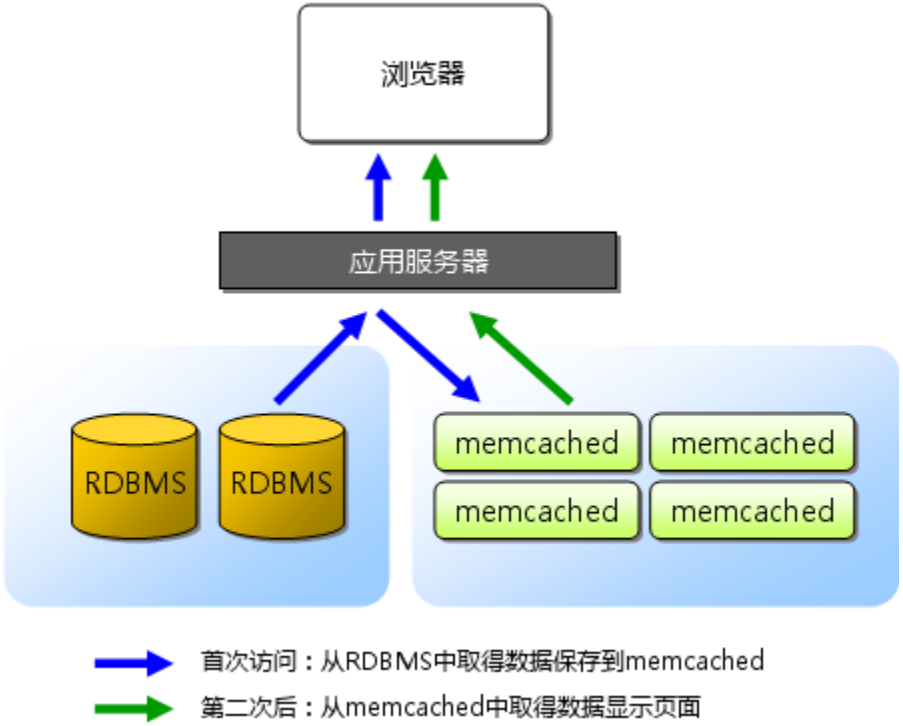
客户端实现原理

--客户端实现Hash

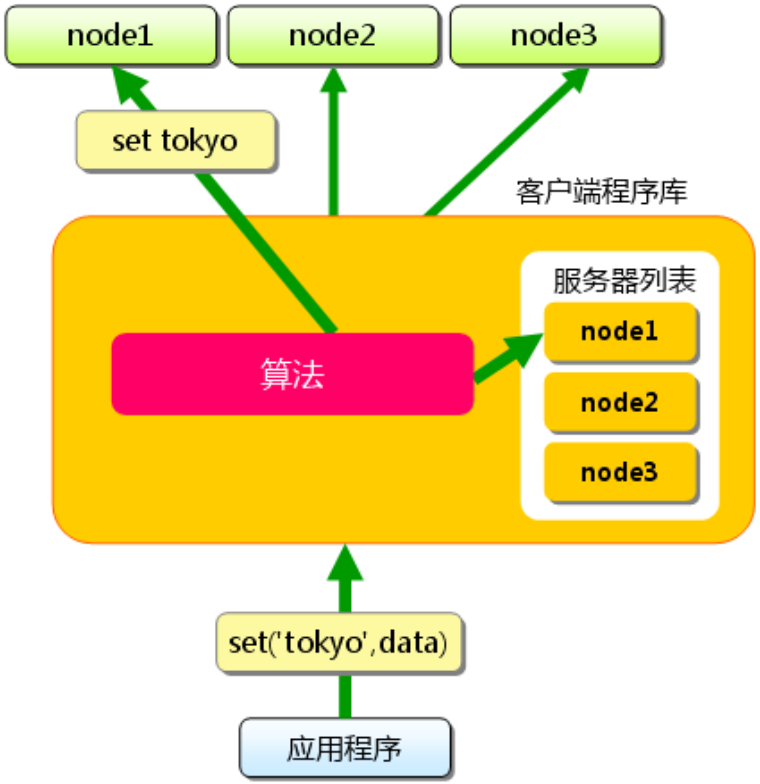


Memcached架构层次

--减少DB访问，提高Web速度

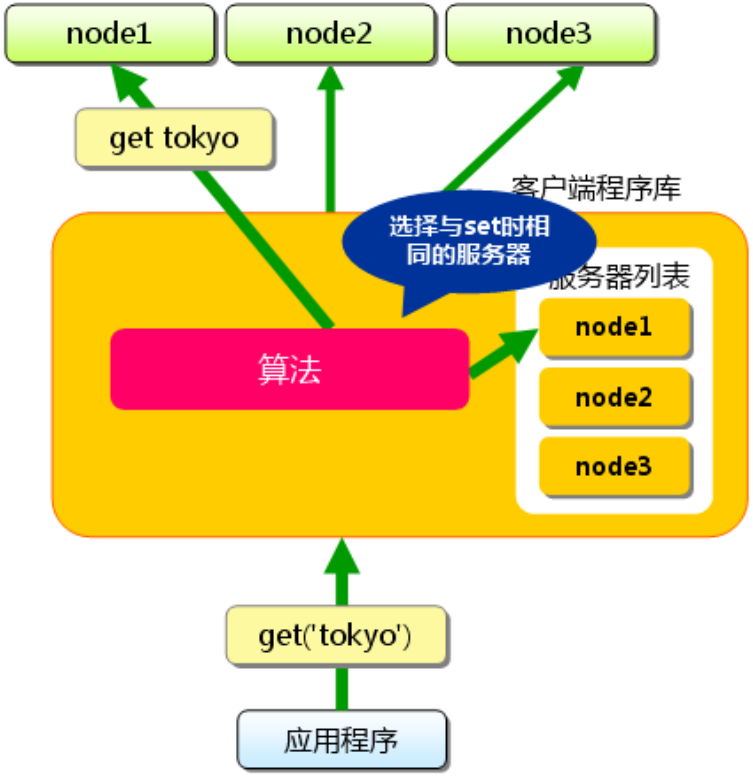


添加对象时

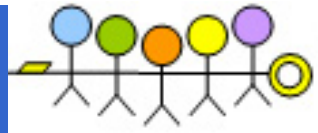

node1
node2
node3
set tokyo
客户端程序库
服务器列表
node1
node2
node3
算法
set('tokyo',data)
应用程序

获取对象时

中间的算法使用的hash算法



最常用的Hash算法



根据余数计算Hash

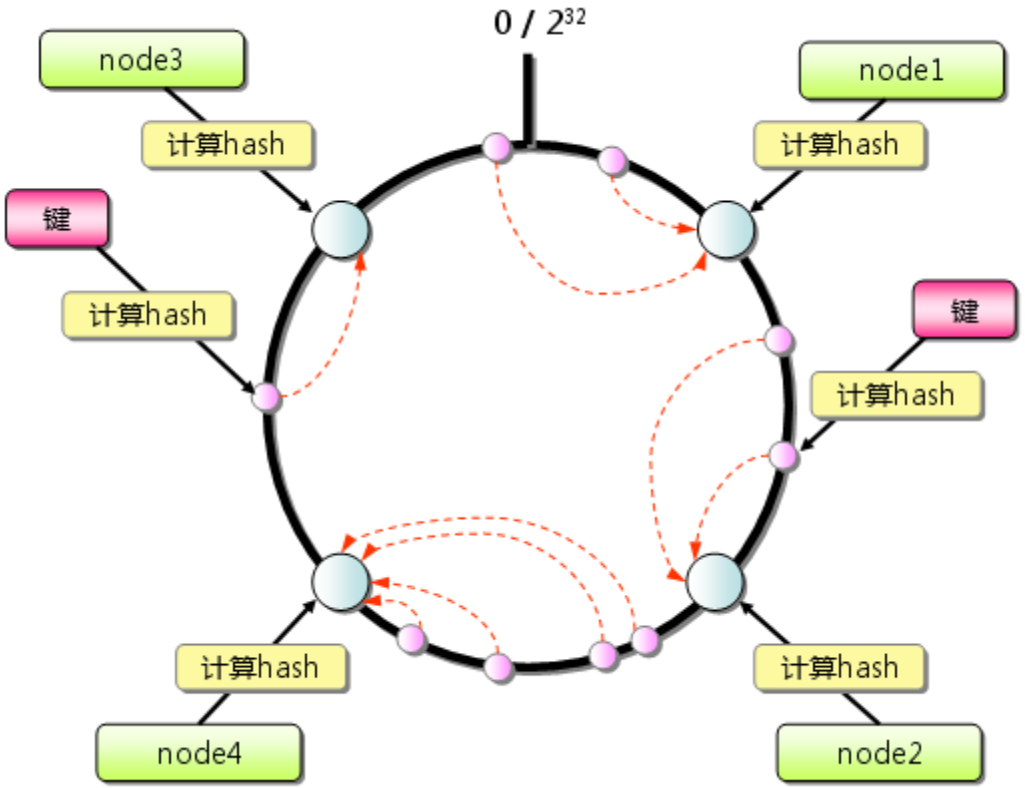
1. 如Perl函数库：Cache::Memcached
2. 优点：简单、分散性优秀
3. 缺点：添加/移除服务器时，缓存重组代价巨大，影响命中率
4. 26个字母由3个节点增加一个节点，命中率下降了23%

三个节点：node1 node2 nod3
node1: a,c,d,e,h,j,n,u,w,x
node2: g,i,k,l,p,r,s,y
node3: b,f,m,o,q,t,v,z

增加节点

四个节点：node1 node2 node3 node4
node1: **d**,f,m,o,t,v
node2: b,**i**,**k**,**p**,**r**,**y**
node3: e,g,l,n,u,w
node4: a,c,h,j,q,s,x,z

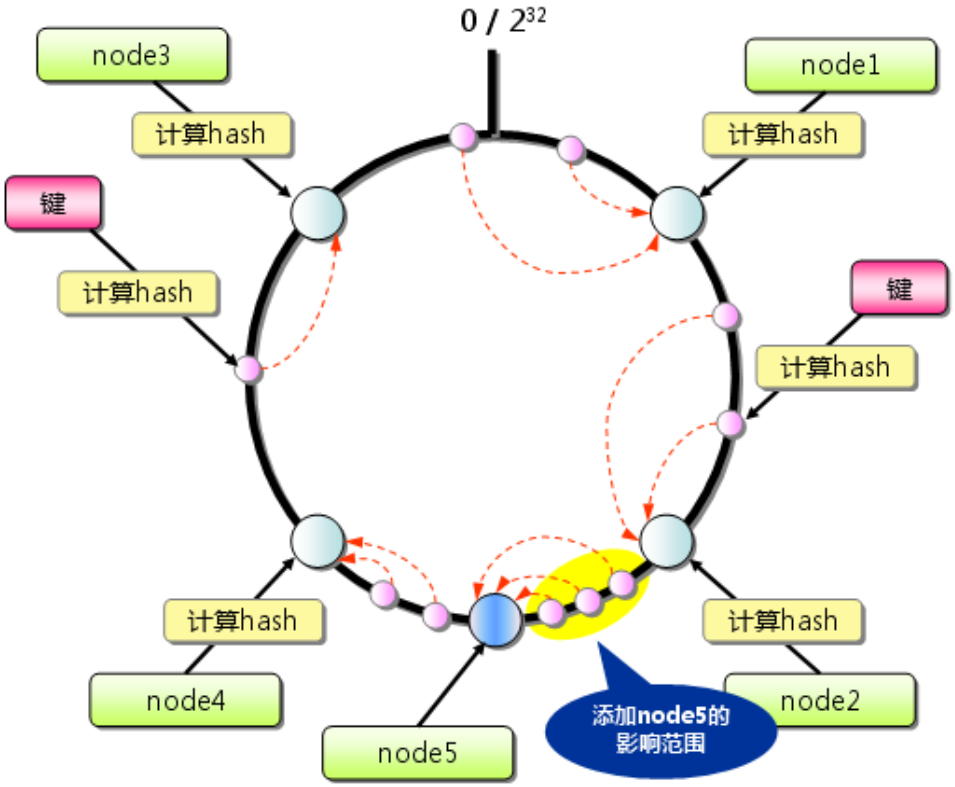
Consistent Hash



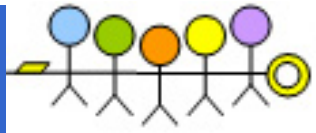
参考原文：
http://alpha.mixi.co.jp/blog/?p=158
http://www.hyuki.com/yukiwiki/wiki.cgi?ConsistentHashing

添加服务器时

- ✓Consistent Hashing最大限度地抑制了键的重新分布
- ✓有的Consistent Hashing的实现方法还采用了虚拟节点的思想



支持Consistent Hash的函数库

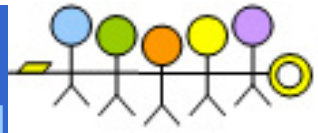


A. libketama的PHP库
   A. libketama（网站Last.fm）

B. Perl客户端
   A. Cache::Memcached::Fast（search.cpan.org）
   B. Cache::Memcached::libmemcached（search.cpan.org）

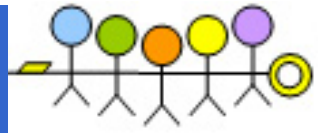
Memcached支持语言

参考：http://code.google.com/p/memcached/wiki/Cl



a) C/C++
   a) libmemcached
   b) libmemcache
   c) apr_memcache
   d) memcaheclient
   e) libketama
b) PHP
   a) PECL/memcached
   b) PECL/memcache
   c) PHP libmemcached
c) Java
   a) spymemcached
   b) Java memcached client/danga
   c) memcache-client-forjava/taobao
d) Python
e) Ruby
f) Perl
g) .NET
h) MySQL
i) PostgreSQL
j) Erlang
k) Lua
l) Lisp

Memcached一些特性和限制



① 在 Memcached 中可以保存的item数据量是没有限制的，只有内存足够

② Memcached单进程最大使用内存为2G，要使用更多内存，可以分多个端口开启多个Memcached进程

③ 最大30天的数据过期时间,设置为永久的也会在这个时间过期，常量REALTIME_MAXDELTA

④ 60*60*24*30 控制

⑤ 最大键长为250字节，大于该长度无法存储，常量KEY_MAX_LENGTH 250 控制

⑥ 单个item最大数据是1MB，超过1MB数据不予存储，常量POWER_BLOCK 1048576 进行控制，

⑦ 它是默认的slab大小

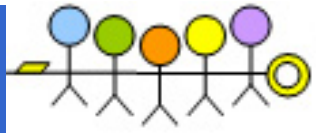
⑧ 最大同时连接数是200，通过 conn_init()中的freetotal 进行控制，最大软连接数是1024，通过

⑨ settings.maxconns=1024 进行控制

⑩ 跟空间占用相关的参数：settings.factor=1.25, settings.chunk_size=48, 影响slab的数据占用和步进方式

深入Memcached内部

基于libevent的事件处理



火龙果·整理
uml.org.cn

基于libevent的事件处理

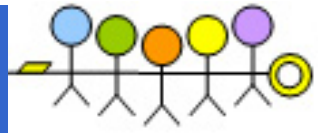
libevent是一套跨平台的事件处理接口的封装，能够兼容包括这些操作系统:
Windows/Linux/BSD/Solaris 等操作系统的的事件处理。

包装的接口包括:
poll、select(Windows)、epoll(Linux)、kqueue(BSD)、/dev/pool(Solaris)

Memcached 使用libevent来进行网络并发连接的处理，能够保持在很大并发情况下，仍旧能够保持快速的响应能力。

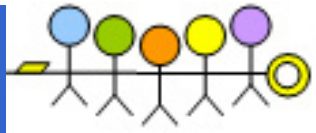
libevent: http://www.monkey.org/~provos/libevent/

UDP协议



A. 何时使用UDP?
   A. TCP连接客户端过多时选用
   B. 允许少量的操作失败
   C. 如get有少量丢失，可以当做没被cache处理
B. frame header: 8byte + tcppacket
   A. 0-1 Request ID
   B. 2-3 Sequence number
   C. 4-5 Total number of datagrams in this message
   D. 6-7 Reserved for future use; must be 0

memcached机制



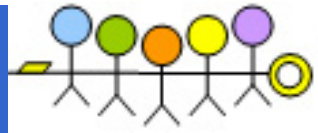
I. 守护进程机制
   I. UNIX daemon

II. Socket事件处理机制
   I. non-blocked：非阻塞
   II. libevent：异步事件处理
   III. epoll/kqueue

III. 内存管理机制
   I. slab：内存分配机制
   II. LRU：对象清除机制
   III. Hash机制：快速检索item

IV. 多线程处理机制：pthread(POSIX)线程模式
   I. 编译时开启：./configure –enable-threads
   II. 目前还比较粗糙，锁机制locking不够完善
   III. 负载过重时，可以开启(-t线程数为CPU核数)

memcached内存管理机制

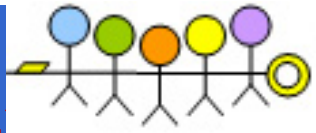


1. SLAB内存处理机制
   1. 提前分配大内存slab 1MB，再进行小对象填充chunk
   2. 避免大量重复的初始化和清理→减轻内存管理器负担
   3. 避免频繁malloc/free→系统碎片

2. 懒惰检测机制
   1. 不检测item对象是否超时
   2. get时检查item对象是否应该删除

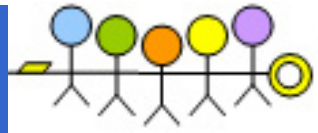
3. 懒惰删除机制
   1. 删除item对象时，不释放内存，作删除标记，指针放入slot回收插槽，下次分配的时候直接使用

重点看一下过期时间处理



1. 数据过期方式-Lazy Expiration
   1. 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. 数据过期方式-LRU
   1. 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。顾名思义，这是删除“最近最少使用”的记录的机制。因此，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。从缓存的实用角度来看，

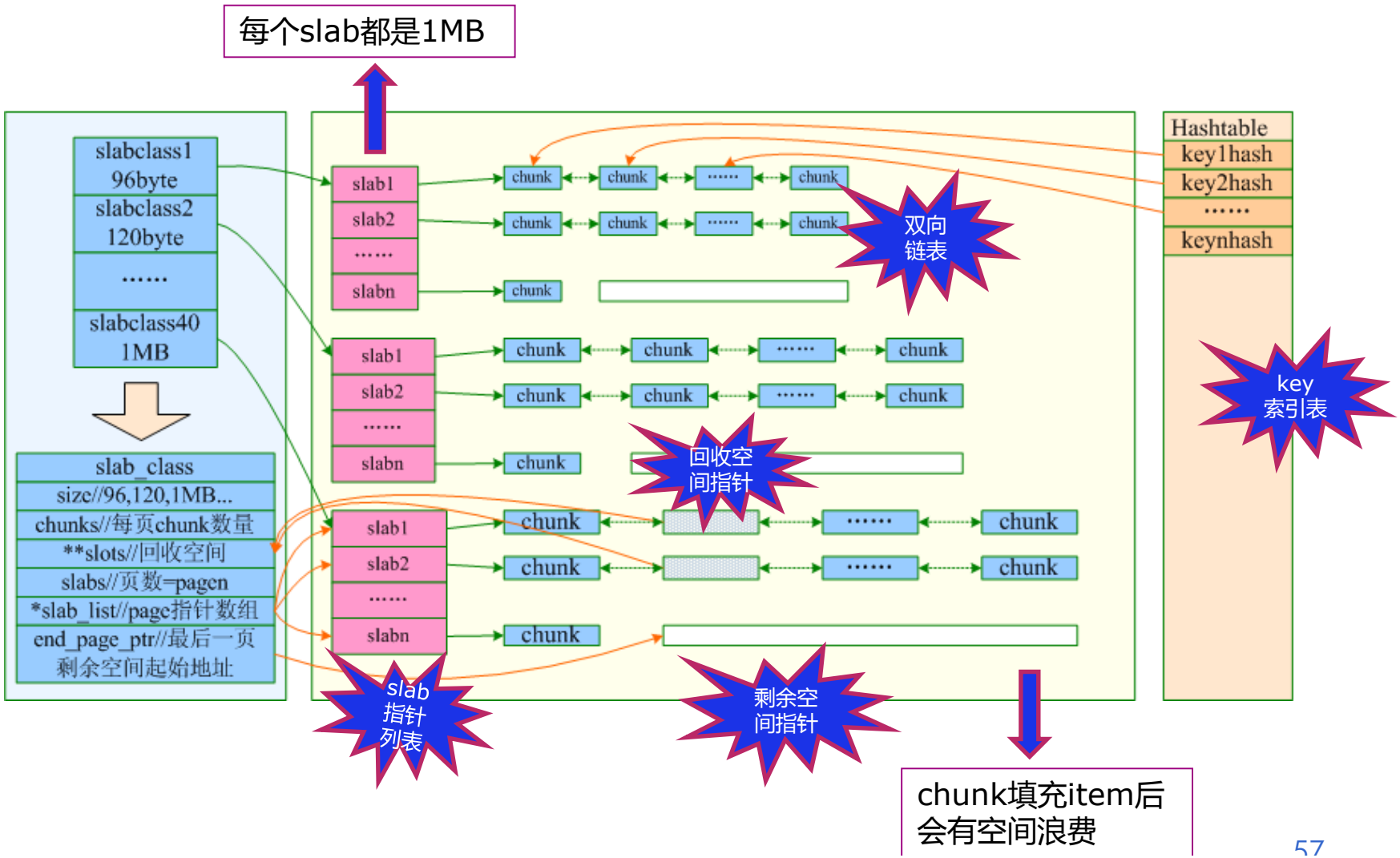
名词解释



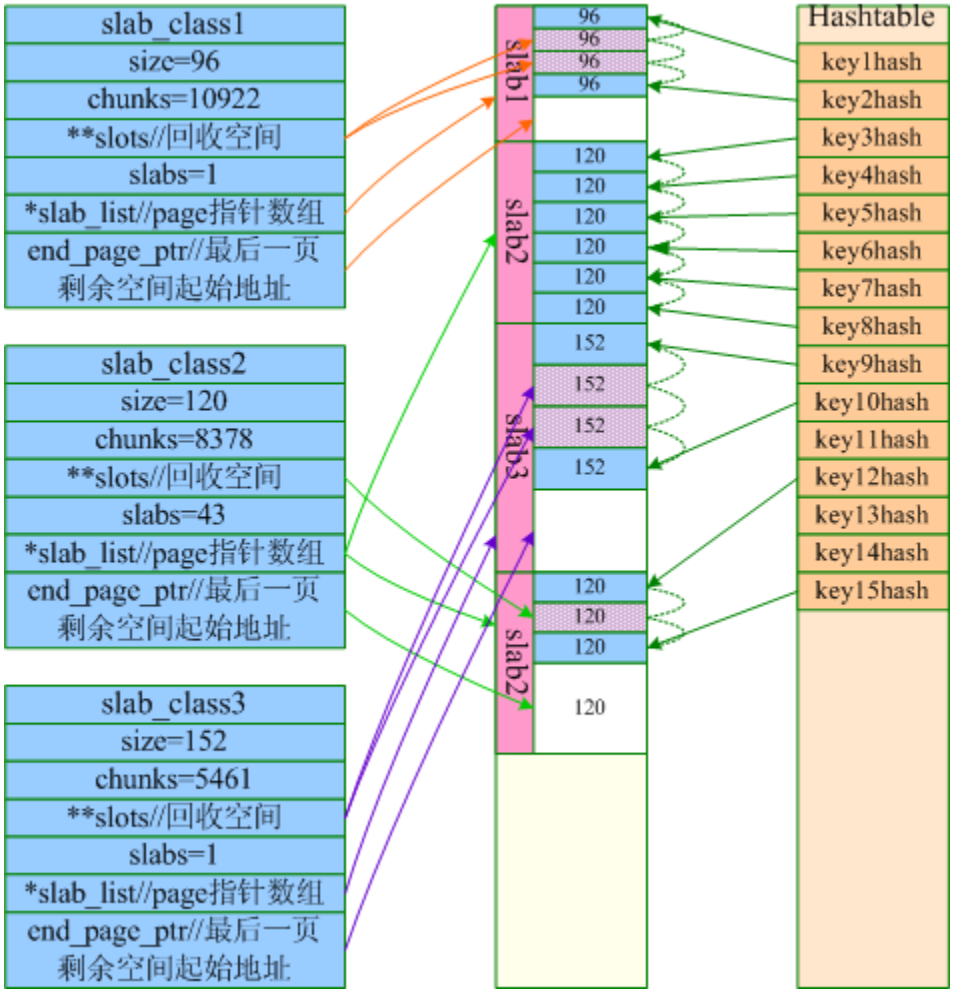
- slab class：内存区类别(48byte-1MB)
  - slab==page：动态创建的实际内存区
  - slab classid：slab class的ID
- chunk：数据区块，固定大小
  - item：实际存储在chunk中的数据项

slab内存分配

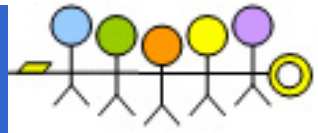
slab内存结构图：二维数组



slab内存分配实例

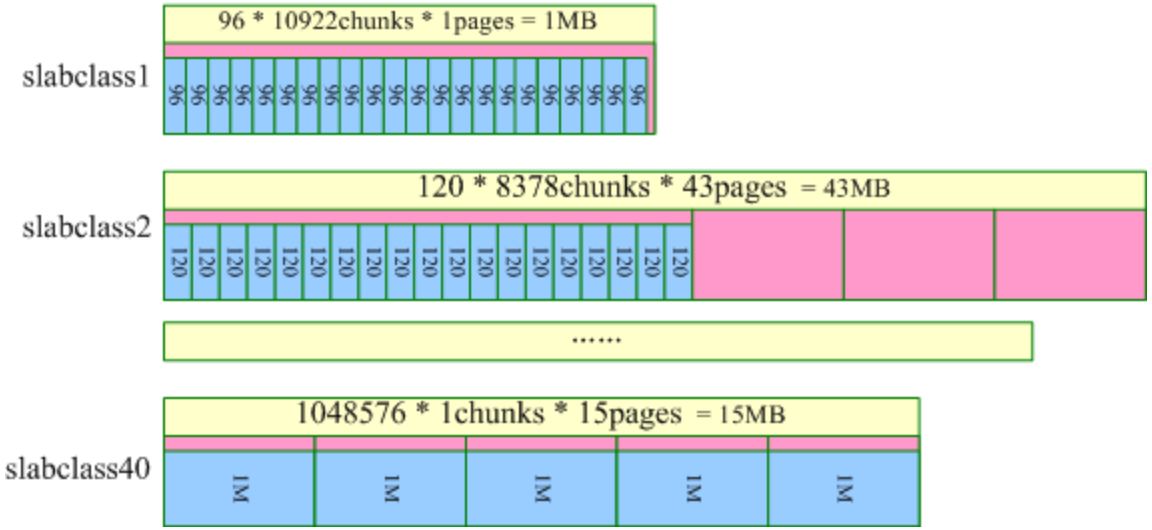

slab_class1
size=96
chunks=10922
**slots//回收空间
slabs=1
*slab_list//page指针数组
end_page_ptr//最后一页
剩余空间起始地址
slab_class2
size=120
chunks=8378
**slots//回收空间
slabs=43
*slab_list//page指针数组
end_page_ptr//最后一页
剩余空间起始地址
slab_class3
size=152
chunks=5461
**slots//回收空间
slabs=1
*slab_list//page指针数组
end_page_ptr//最后一页
剩余空间起始地址
slab1
slab2
slab3
slab2
96
96
96
96
120
120
120
120
120
120
152
152
152
152
120
120
120
120
Hashtable
key1hash
key2hash
key3hash
key4hash
key5hash
key6hash
key7hash
key8hash
key9hash
key10hash
key11hash
key12hash
key13hash
key14hash
key15hash

实例数据--195:11213 4GB

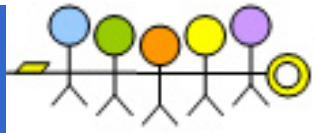


class	size	chunks	pages	大小
1	96	10922	1	1MB
2	120	8738	43	43MB
3	152	6898	1	1MB
4	192	5461	186	186MB
5	240	4369	2	2MB
6	304	3449	2	2MB
7	384	2730	9	9MB
8	480	2184	91	91MB
9	600	1747	104	104MB
10	752	1394	274	274MB
11	944	1110	173	173MB
12	1184	885	79	79MB
13	1480	708	67	67MB
14	1856	564	69	69MB
15	2320	451	75	75MB
16	2904	361	105	105MB
17	3632	288	128	128MB
18	4544	230	129	129MB
19	5680	184	111	111MB
20	7104	147	119	119MB
21	8880	118	157	157MB
22	11104	94	226	225MB
23	13880	75	315	313MB
24	17352	60	356	354MB
25	21696	48	330	328MB
26	27120	38	259	255MB
27	33904	30	181	176MB
28	42384	24	141	137MB
29	52984	19	111	107MB
30	66232	15	87	83MB
31	82792	12	65	62MB
32	103496	10	38	38MB
33	129376	8	25	25MB
34	161720	6	18	17MB
35	202152	5	11	11MB
36	252696	4	9	9MB
37	315872	3	8	8MB
38	394840	2	14	11MB
39	493552	2	6	6MB
40	1048576	1	15	15MB
合计				**4.1GB**

计算slab占用内存


96 * 10922chunks * 1pages = 1MB
slabclass1
120 * 8378chunks * 43pages = 43MB
slabclass2
.....
1048576 * 1chunks * 15pages = 15MB
slabclass40
1M

slab参数

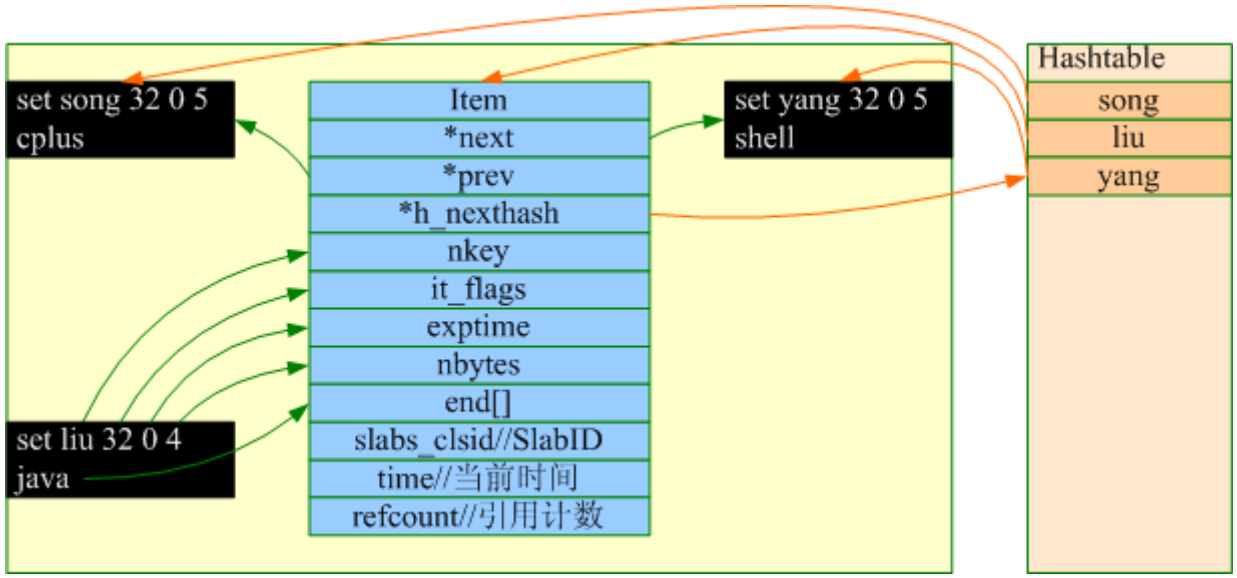


- 进程内存区
  - slabclass元信息：1.1中是21byte，1.2中是200byte
  - Hashtable：1.1中位41MB，1.2中位65MB
- 数据内存区
  - slab默认大小为1048576byte（1MB），大于1MB数据忽略
  - chunk初始大小，1.1中是1byte，1.2中是48byte
- 增长因子factor
  - 1.1中，chunk大小为初始大小*2^n，n为classid，即：
    - id为0的slab大小1byte，id为1的slab大小2byte，id为2的slab大小4byte...
    - id为20的slab，每chunk大小为1MB，只有一个chunk
  - 1.2中有一个factor值，默认为1.25
    - 96,120,152...

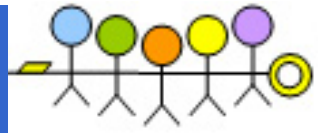
Item内存分配

Item数据格式

Item是保存在chunk中的实际数据



新建Item分配内存过程

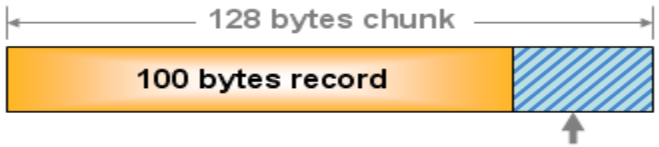


- 快速定位slab classid
  - 计算key+value+suffix+32结构体，如90byte
  - 如果>1MB，无法存储丢弃
  - 取最小冗余的slab class
    - 如：有48,96,120，存90会选择96

- 按顺序寻找可用chunk
  - slot：检查slab回收空间slot里是否有剩余chunk
    - delete：delete时标记到slot
    - exptime：get时检查的过期对象标记到slot
  - end_page_ptr：检查page中是否有剩余chunk
  - memory：内存还有剩余则开辟新的slab
  - LRU：Slab内部扫描Item双向链表50次

内存有浪费了!!

- slab尾部剩余空间
  - 如classid=40中，两个chunk占用了1009384byte，就有1048576-1009384=39192byte被浪费
  - 解决办法：规划slab=chunk*n整数倍
- slab中chunk利用率低：申请的slab只存放了一个Item
  - 解决办法：规划slab=chunk
- chunk存储Item浪费
  - 如I tem是100，存到128字节chunk，就有28字节浪费
  - ***解决办法：规划chunk=Item***



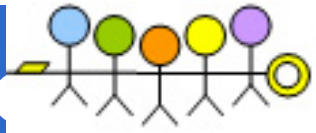
使用合适的factor，减

- -f参数：默认为1.25，曾经为2
- 值越小，slab中chunk size差距越小，内存浪费越小
- 1.25适合缓存几百字节的对象

factor=2	factor=1.25
slab class 1: chunk size 128 perslab 8192	slab class 1: chunk size 88 perslab 11915
slab class 2: chunk size 256 perslab 4096	slab class 2: chunk size 112 perslab 9362
slab class 3: chunk size 512 perslab 2048	slab class 3: chunk size 144 perslab 7281
slab class 4: chunk size 1024 perslab 1024	slab class 4: chunk size 184 perslab 5698
slab class 5: chunk size 2048 perslab 512	slab class 5: chunk size 232 perslab 4519
slab class 6: chunk size 4096 perslab 256	slab class 6: chunk size 296 perslab 3542
slab class 7: chunk size 8192 perslab 128	slab class 7: chunk size 376 perslab 2788
slab class 8: chunk size 16384 perslab 64	slab class 8: chunk size 472 perslab 2221
slab class 9: chunk size 32768 perslab 32	slab class 9: chunk size 592 perslab 1771
slab class 10: chunk size 65536 perslab 16	slab class 10: chunk size 744 perslab 1409
slab class 11: chunk size 131072 perslab 8	
slab class 12: chunk size 262144 perslab 4	
slab class 13: chunk size 524288 perslab 2	

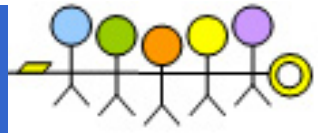
建议：计算一下数据的预期平均长度，调整factor，以获得最恰当的设置

根据数据分布调整fact



- 非均匀分布，即数据长度集中在几个区域内
  - 如保存用户Session
- 更极端的状态是等长数据
  - 如定长键值，定长数据
  - 多见于访问、在线统计或执行锁
- 计算Item长度
  - key键长 + suffix+value值长 + 结构大小（32字节）

调优的最高指示精神

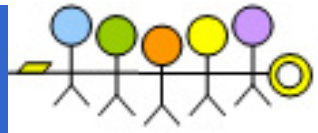


1. 提高内存利用率，减少内存浪费

2. 提高命中率(80%,95%?)

3. 调优方法：
   - f参数：factor增长因子
   - n参数：chunk初始值

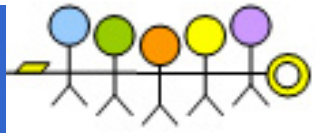
一切都是为了更快



- 低CPU消耗(瓶颈在于网络IO)
  - libevent事件机制
  - slab内存预分配机制

- 适合使用大量低CPU的机器搭建集群
  - 32位机器最大2GB，64GB无限制
  - -m分配内存为数据区，memcached本身也需要占用内存，因此不可将物理内存全部分配
  - 使用连接池维持连接

因为优秀，所以不足



火龙果 · 整理
uml.org.cn

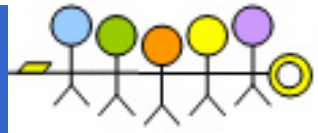
- A. Can’t dump
  - A. 无法备份，重启无法恢复
- B. Can’t iterate over keys
  - A. 无法查询
- C. Not persistent
  - A. 没有持久化，重启全部丢失
- D. Not redundant
  - A. 单点故障failover
- E. No Sessions
  - A. 崩溃没法查找原因
- F. No security
  - A. 任何机器都可以telnet，需要放在防火墙后
- G. 内存问题
  - A. LRU是slab局部，没有全局
  - B. 有空间浪费
- H. 日志问题
  - A. 没有合理的日志
- I. 集群问题
  - A. 集群增加机器成本高

改进计划

A. 扩展二进制协议
   A. 不需要文本解析，速度更快
   B. 减少文本协议的漏洞

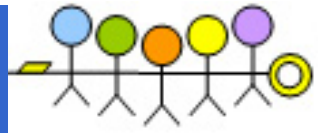
B. 外部引擎加载功能
   A. MySQL plugin

同款产品的类比



- redis
- Ehcache
- Guava Cache
- mongodb

QA!



火龙果 · 整理
uml.org.cn

1. 我们的项目里面哪里可以用一下?

2. 用过之后和用之前有什么区别，快在哪里?