Memcahced操作记录

一memcached

1. memcached理论知识补充
   * 1. 简介

（1）   Memcache是一个高性能的分布式的内存对象缓存系统，通过在内存里维护一个统一的巨大的hash表，它能够用来存储各种格式的数据，包括图像、视频、文件以及数据库检索的结果等。简单的说就是将数据调用到内存中，然后从内存中读取，从而大大提高读取速度（即用于在动态应用中减少数据库负载，提升访问速度）。

（2）  Memcache是以守护进程方式运行于一个或多个服务中，随时会接收客户端的连接和操作。

（3）  Memcached是一种无阻塞的socket通信方式服务，基于libevent库，由于无阻塞通信，对内存读写速度非常之快。

（4）  Memcached分服务器端和客户端，可以配置多个服务器端和客户端，应用于分布式的服务非常广泛。

（5）  Memcached是键值一一对应，key默认最大不能超过128个字 节，value默认大小是1M，也就是一个slabs，如果要存2M的值（连续的），不能用两个slabs，因为两个slabs不是连续的，无法在内存中 存储，故需要修改slabs的大小，多个key和value进行存储时，即使这个slabs没有利用完，那么也不会存放别的数据。

* + 1. 优点

1. 高性能
2. 分布式
   * 1. 局限性
3. 数据结构单一，只支持key/value,不像redis支持set ，list，等。
4. 无法进行持久化，数据不能备份，只能用于缓存使用。重启丢失数据。
5. 无法进行数据同步，不能将memcached中数据迁移到其他memcached示例中。（不支持主从复制）
6. value值-redis最大可以达到1GB，而memcache只有1MB。
   * 1. 使用场景
     2. 对比redis

1.1安装libevent

1006 sudo wget https://github.com/libevent/libevent/releases/download/release-2.1.8-stable/libevent-2.1.8-stable.tar.gz -c -O libevent-2.1.8-stable.tar.gz

1007 ls

1008 tar -zxvf libevent-2.1.8-stable.tar.gz

1010 cd libevent-2.1.8-stable

1012 ./configure

1013 ./configure --prefix=/usr/local/libevent

1014 make

1016 make install

1.2 安装memcahced

1023 wget http://www.memcached.org/files/memcached-1.5.16.tar.gz

1024 ls

1025 tar -zxvf memcached-1.5.16.tar.gz

1026 ls

1027 cd memcached-1.5.16

1028 ls

1029 ./configure --prefix=/usr/local/memcached --with-libevent=/usr/local/libevent

1030 make

1035 make install

1038 ./memcached -m 64 -p 11211 –vvv (这里提示root不行，加 –u 参数)

1039 ./memcached -m 64 -p 11211 -vvv -u nobody

二．使用

在启动memcached以后，11211端口就启动了监听。Libmemcached 是一个开源的memcached客户端库，其内部实现了分布式管理，内存池等功能。通过api的形式提供出来，使用程序员可以专心上层业务逻辑，避免底层与memcached交互的细节，所以Libmemcached编译，安装之后就会以动态库的方式提供出来给程序员调用。

2．1 libmemcached特性：

(1) 异步和同步传输自持。

(2)支持一致性hash分布式算法。

(3)可调哈希算法来匹配密钥。

(4)访问大对象支持。

(5)本地复制。

(6)提供了一些管理memcached服务器的工具命令。

2.2 安装libmemcached库。

wget <https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz>

./configure --prefix=/usr/local/libmemcached --with-memcached

Make

Make install

编译完成以后，头文件/usr/local/libmemcached/include

库文件/usr/local/libmemcached/lib

2.3 使用示例

2.3.1 代码见附录一

编译

g++ -std=c++11 -g -I/usr/local/libmemcached/include/

-L/usr/local/libmemcached/lib -lmemcached -o test memcachedtest.cpp

结果

[root@syycdz test]# ./test

key: key1 rc:SUCCESS

Get value:key1Valu sucessful!

key: key2 rc:SUCCESS

Get value:key2Valu sucessful!

key: key3 rc:CONNECTION FAILURE

SERVER IS MARKED DEAD

key: key4 rc:SUCCESS

Get value:key4Valu sucessful!

key: key5 rc:SUCCESS

Get value:key5Valu sucessful!

key: key6 rc:SUCCESS

Get value:key6Valu sucessful!

key: key7 rc:SUCCESS

Get value:key7Valu sucessful!

time: 0

附录1

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <unistd.h>

#include <string>

#include <iostream>

#include <libmemcached/memcached.h>

#include <libmemcached/util.h>

#include<libmemcached/memcached.hpp>

using namespace std;

/\*

\*/

int main(int argc, char \*argv[])

{

memcached\_st \*memc;

memcached\_return rc;

memcached\_server\_st \*servers;

memc = memcached\_create(NULL);

servers = memcached\_server\_list\_append\_with\_weight(NULL, (char\*)"localhost", 11211, 50 , &rc);

servers = memcached\_server\_list\_append\_with\_weight(servers, (char\*)"localhost", 11212 , 50 , &rc);

rc = memcached\_server\_push(memc, servers);

memcached\_server\_free(servers);

memcached\_behavior\_set(memc,MEMCACHED\_BEHAVIOR\_DISTRIBUTION,MEMCACHED\_DISTRIBUTION\_CONSISTENT\_KETAMA);

memcached\_behavior\_set(memc, MEMCACHED\_BEHAVIOR\_REMOVE\_FAILED\_SERVERS, true) ;

memcached\_behavior\_set(memc , MEMCACHED\_BEHAVIOR\_DEAD\_TIMEOUT , 5);

int time\_sl = 0;

int times = 0;

while(times++ < 1)

{

//save data

const char \*keys[]= {"key1", "key2", "key3","key4" ,"key5","key6","key7"};

const size\_t key\_length[]= {4, 4, 4, 4 ,4 ,4 ,4 };

const char \*values[] = {"key1Value", "key2Value", "key3Value","key4Value" , "key5Value" , "key6Value" , "key7Value"};

size\_t val\_length[]= {sizeof(values[0]), sizeof(values[1]), sizeof(values[2]), sizeof(values[3]),sizeof(values[4]),sizeof(values[5]),sizeof(values[6])};

for (int i = 0 ; i < 7; i++)

{

rc = memcached\_set(memc, keys[i], key\_length[i], values[i], val\_length[i], (time\_t)180,(uint32\_t)0);

printf("key: %s rc:%s\n", keys[i], memcached\_strerror(memc, rc)); //

char\* result = memcached\_get(memc, keys[i], key\_length[i], NULL, NULL, &rc);

if (rc == MEMCACHED\_SUCCESS)

{

cout << "Get value:" << result << " sucessful!" << endl;

}else

cout << memcached\_strerror(memc, rc) << endl;

}

printf("time: %d\n", time\_sl++);

sleep(1);

}

memcached\_free(memc);

return 0;

}

附录2

1. #include <iostream>
2. #include <string>
3. #include <sys/time.h>
4. #include <libmemcached/memcached.h>
5. using namespace std;
6. int main(int argc,char \*argv[])
7. {
8. *//connect server*
9. memcached\_st \*memc;
10. memcached\_return rc;
11. memcached\_server\_st \*servers;
12. time\_t expiration = 180;
13. uint32\_t flags = 0;
14. memc = memcached\_create(NULL);
15. servers = memcached\_server\_list\_append(NULL, "localhost", 8000, &rc);
16. rc = memcached\_server\_push(memc, servers);
17. memcached\_server\_list\_free(servers);
18. string key = "key";
19. string value = "value";
20. size\_t value\_length = value.length();
21. size\_t key\_length = key.length();
23. struct timeval start;
24. struct timeval end;
25. gettimeofday(&start, NULL);
27. for(int i=0; i < 1000; ++i)
28. {
29. *//Save data*
30. rc = memcached\_set(memc, key.c\_str(), key.length(), value.c\_str(), value.length(), expiration, flags);
31. if(rc == MEMCACHED\_SUCCESS)
32. {
33. cout<<"Save data: "<<value<<" successful!"<<endl;
34. }
35. *//Get data*
36. char\* result = memcached\_get(memc, key.c\_str(), key\_length, &value\_length, &flags, &rc);
37. if(rc == MEMCACHED\_SUCCESS)
38. {
39. cout<<"Get value: "<<value<<"by key "<<key<<"successful!"<<endl;
40. }
41. *//Delete data*
42. rc = memcached\_delete(memc, key.c\_str(), key\_length, expiration);
43. if(rc == MEMCACHED\_SUCCESS)
44. {
45. cout<<"Delete data: "<<value<<" successful!"<<endl;
46. }
48. }
49. gettimeofday(&end, NULL);
50. cout<<"Use Time"<<start.tv\_sec-end.tv\_sec<<"sec, "<<start.tv\_usec-end.tv\_usec<<"usec";
52. *//free*
53. memcached\_free(memc);
54. return 0;
55. }