<http://blog.csdn.net/xzknet/article/details/44066207>

# 1       Memcache是什么

Memcache是danga.com的一个项目，最早是为 LiveJournal 服务的，目前全世界不少人使用这个缓存项目来构建自己大负载的网站，来分担数据库的压力。

它可以应对任意多个连接，使用非阻塞的网络IO。由于它的工作机制是在内存中开辟一块空间，然后建立一个HashTable，Memcached自管理这些HashTable。

为什么会有Memcache和memcached两种名称？

其实Memcache是这个项目的名称，而memcached是它服务器端的主程序文件名，

Memcache官方网站：<http://www.danga.com/memcached>，

# 2       Memcache工作原理

首先 memcached 是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，客户端可以由各种语言编写，目前已知的客户端 API 包括 Perl/PHP/Python/Ruby/Java/C#/C 等等。客户端在与 memcached 服务建立连接之后，接下来的事情就是存取对象了，每个被存取的对象都有一个唯一的标识符 key，存取操作均通过这个 key 进行，保存到memcached 中的对象实际上是放置内存中的，并不是保存在 cache 文件中的，这也是为什么 memcached 能够如此高效快速的原因。注意，这些对象并不是持久的，服务停止之后，里边的数据就会丢失。

与许多 cache 工具类似，Memcached 的原理并不复杂。它采用了C/S的模式，在 server 端启动服务进程，在启动时可以指定监听的 ip，自己的端口号，所使用的内存大小等几个关键参数。一旦启动，服务就一直处于可用状态。Memcached 的目前版本是通过C实现，采用了单进程，单线程，异步I/O，基于事件 (event\_based) 的服务方式.使用libevent 作为事件通知实现。多个 Server 可以协同工作，但这些 Server 之间是没有任何通讯联系的，每个 Server 只是对自己的数据进行管理。Client 端通过指定 Server 端的 ip 地址(通过域名应该也可以)。需要缓存的对象或数据是以 key->value对的形式保存在Server端。key 的值通过 hash 进行转换，根据 hash 值把 value 传递到对应的具体的某个 Server 上。当需要获取对象数据时，也根据 key 进行。首先对 key 进行 hash，通过获得的值可以确定它被保存在了哪台 Server 上，然后再向该 Server 发出请求。Client 端只需要知道保存 hash(key) 的值在哪台服务器上就可以了。

        其实说到底，memcache 的工作就是在专门的机器的内存里维护一张巨大的 hash 表，来存储经常被读写的一些数组与文件，从而极大的提高网站的运行效率。

# 3       如何使用

# 建立Manager类

java代码：

[查看复制到剪贴板打印](http://sishuok.com/forum/blogPost/list/311.html)

* 1. **package** com.alisoft.sme.memcached;
  3. **import** java.util.Date;
  5. **import** com.danga.MemCached.MemCachedClient;
  6. **import** com.danga.MemCached.SockIOPool;
  8. **public** **class** MemCachedManager {
  10. // 创建全局的唯一实例
  11. **protected** **static** MemCachedClient mcc = **new** MemCachedClient();
  13. **protected** **static** MemCachedManager memCachedManager = **new** MemCachedManager();
  15. // 设置与缓存服务器的连接池
  16. **static** {
  17. // 服务器列表和其权重
  18. String[] servers = { "127.0.0.1:11211" };
  19. Integer[] weights = { 3 };
  21. // 获取socke连接池的实例对象
  22. SockIOPool pool = SockIOPool.getInstance();
  24. // 设置服务器信息
  25. pool.setServers(servers);
  26. pool.setWeights(weights);
  28. // 设置初始连接数、最小和最大连接数以及最大处理时间
  29. pool.setInitConn(5);
  30. pool.setMinConn(5);
  31. pool.setMaxConn(250);
  32. pool.setMaxIdle(1000 \* 60 \* 60 \* 6);
  34. // 设置主线程的睡眠时间
  35. pool.setMaintSleep(30);
  37. // 设置TCP的参数，连接超时等
  38. pool.setNagle(**false**);
  39. pool.setSocketTO(3000);
  40. pool.setSocketConnectTO(0);
  42. // 初始化连接池
  43. pool.initialize();
  45. // 压缩设置，超过指定大小（单位为K）的数据都会被压缩
  46. mcc.setCompressEnable(**true**);
  47. mcc.setCompressThreshold(64 \* 1024);
  48. }
  50. /\*\*
  51. \* 保护型构造方法，不允许实例化！
  52. \*
  53. \*/
  54. **protected** MemCachedManager() {
  56. }
  58. /\*\*
  59. \* 获取唯一实例.
  60. \*
  61. \* @return
  62. \*/
  63. **public** **static** MemCachedManager getInstance() {
  64. **return** memCachedManager;
  65. }
  67. /\*\*
  68. \* 添加一个指定的值到缓存中.
  69. \*
  70. \* @param key
  71. \* @param value
  72. \* @return
  73. \*/
  74. **public** **boolean** add(String key, Object value) {
  75. **return** mcc.add(key, value);
  76. }
  78. **public** **boolean** add(String key, Object value, Date expiry) {
  79. **return** mcc.add(key, value, expiry);
  80. }
  82. **public** **boolean** replace(String key, Object value) {
  83. **return** mcc.replace(key, value);
  84. }
  86. **public** **boolean** replace(String key, Object value, Date expiry) {
  87. **return** mcc.replace(key, value, expiry);
  88. }
  90. /\*\*
  91. \* 根据指定的关键字获取对象.
  92. \*
  93. \* @param key
  94. \* @return
  95. \*/
  96. **public** Object get(String key) {
  97. **return** mcc.get(key);
  98. }
  100. **public** **static** **void** main(String[] args) {
  101. MemCachedManager cache = MemCachedManager.getInstance();
  102. cache.add("hello", 234);
  103. System.out.print("get value : " + cache.get("hello"));
  104. }
  105. }

**[java]** [view plain](http://csdnimg.cn/release/phoenix/) [copy](http://csdnimg.cn/release/phoenix/)

* 1. **package** com.alisoft.sme.memcached;
  3. **import** java.util.Date;
  5. **import** com.danga.MemCached.MemCachedClient;
  6. **import** com.danga.MemCached.SockIOPool;
  8. **public** **class** MemCachedManager {
  10. // 创建全局的唯一实例
  11. **protected** **static** MemCachedClient mcc = **new** MemCachedClient();
  13. **protected** **static** MemCachedManager memCachedManager = **new** MemCachedManager();
  15. // 设置与缓存服务器的连接池
  16. **static** {
  17. // 服务器列表和其权重
  18. String[] servers = { "127.0.0.1:11211" };
  19. Integer[] weights = { 3 };
  21. // 获取socke连接池的实例对象
  22. SockIOPool pool = SockIOPool.getInstance();
  24. // 设置服务器信息
  25. pool.setServers(servers);
  26. pool.setWeights(weights);
  28. // 设置初始连接数、最小和最大连接数以及最大处理时间
  29. pool.setInitConn(5);
  30. pool.setMinConn(5);
  31. pool.setMaxConn(250);
  32. pool.setMaxIdle(1000 \* 60 \* 60 \* 6);
  34. // 设置主线程的睡眠时间
  35. pool.setMaintSleep(30);
  37. // 设置TCP的参数，连接超时等
  38. pool.setNagle(**false**);
  39. pool.setSocketTO(3000);
  40. pool.setSocketConnectTO(0);
  42. // 初始化连接池
  43. pool.initialize();
  45. // 压缩设置，超过指定大小（单位为K）的数据都会被压缩
  46. mcc.setCompressEnable(**true**);
  47. mcc.setCompressThreshold(64 \* 1024);
  48. }
  50. /\*\*
  51. \* 保护型构造方法，不允许实例化！
  52. \*
  53. \*/
  54. **protected** MemCachedManager() {
  56. }
  58. /\*\*
  59. \* 获取唯一实例.
  60. \*
  61. \* @return
  62. \*/
  63. **public** **static** MemCachedManager getInstance() {
  64. **return** memCachedManager;
  65. }
  67. /\*\*
  68. \* 添加一个指定的值到缓存中.
  69. \*
  70. \* @param key
  71. \* @param value
  72. \* @return
  73. \*/
  74. **public** **boolean** add(String key, Object value) {
  75. **return** mcc.add(key, value);
  76. }
  78. **public** **boolean** add(String key, Object value, Date expiry) {
  79. **return** mcc.add(key, value, expiry);
  80. }
  82. **public** **boolean** replace(String key, Object value) {
  83. **return** mcc.replace(key, value);
  84. }
  86. **public** **boolean** replace(String key, Object value, Date expiry) {
  87. **return** mcc.replace(key, value, expiry);
  88. }
  90. /\*\*
  91. \* 根据指定的关键字获取对象.
  92. \*
  93. \* @param key
  94. \* @return
  95. \*/
  96. **public** Object get(String key) {
  97. **return** mcc.get(key);
  98. }
  100. **public** **static** **void** main(String[] args) {
  101. MemCachedManager cache = MemCachedManager.getInstance();
  102. cache.add("hello", 234);
  103. System.out.print("get value : " + cache.get("hello"));
  104. }
  105. }

# 建立数据对象

### 

java代码：

[查看复制到剪贴板打印](http://sishuok.com/forum/blogPost/list/311.html)

1. **package** com.alisoft.sme.memcached;
3. **import** java.io.Serializable;
5. **public** **class** TBean **implements** Serializable {
7. **private** **static** **final** **long** serialVersionUID = 1945562032261336919L;
9. **private** String name;
11. **public** String getName() {
12. **return** name;
13. }
15. **public** **void** setName(String name) {
16. **this**.name = name;
17. }
18. }

**[java]** [view plain](http://csdnimg.cn/release/phoenix/) [copy](http://csdnimg.cn/release/phoenix/)

1. **package** com.alisoft.sme.memcached;
3. **import** java.io.Serializable;
5. **public** **class** TBean **implements** Serializable {
7. **private** **static** **final** **long** serialVersionUID = 1945562032261336919L;
9. **private** String name;
11. **public** String getName() {
12. **return** name;
13. }
15. **public** **void** setName(String name) {
16. **this**.name = name;
17. }
18. }

## 创建测试用例

### 

java代码：

[查看复制到剪贴板打印](http://sishuok.com/forum/blogPost/list/311.html)

1. **package** com.alisoft.sme.memcached.test;
3. **import** junit.framework.TestCase;
5. **import** org.junit.Test;
7. **import** com.alisoft.sme.memcached.MemCachedManager;
8. **import** com.alisoft.sme.memcached.TBean;
10. **public** **class** TestMemcached **extends** TestCase {
12. **private** **static** MemCachedManager cache;
14. @Test
15. **public** **void** testCache() {
17. TBean tb = **new** TBean();
18. tb.setName("E网打进");
19. cache.add("bean", tb);
21. TBean tb1 = (TBean) cache.get("bean");
22. System.out.println("name=" + tb1.getName());
23. tb1.setName("E网打进\_修改的");
25. tb1 = (TBean) cache.get("bean");
26. System.out.println("name=" + tb1.getName());
27. }
29. @Override
30. **protected** **void** setUp() **throws** Exception {
31. **super**.setUp();
32. cache = MemCachedManager.getInstance();
33. }
35. @Override
36. **protected** **void** tearDown() **throws** Exception {
37. **super**.tearDown();
38. cache = **null**;
39. }
41. }

**[java]** [view plain](http://csdnimg.cn/release/phoenix/) [copy](http://csdnimg.cn/release/phoenix/)

1. **package** com.alisoft.sme.memcached.test;
3. **import** junit.framework.TestCase;
5. **import** org.junit.Test;
7. **import** com.alisoft.sme.memcached.MemCachedManager;
8. **import** com.alisoft.sme.memcached.TBean;
10. **public** **class** TestMemcached **extends** TestCase {
12. **private** **static** MemCachedManager cache;
14. @Test
15. **public** **void** testCache() {
17. TBean tb = **new** TBean();
18. tb.setName("E网打进");
19. cache.add("bean", tb);
21. TBean tb1 = (TBean) cache.get("bean");
22. System.out.println("name=" + tb1.getName());
23. tb1.setName("E网打进\_修改的");
25. tb1 = (TBean) cache.get("bean");
26. System.out.println("name=" + tb1.getName());
27. }
29. @Override
30. **protected** **void** setUp() **throws** Exception {
31. **super**.setUp();
32. cache = MemCachedManager.getInstance();
33. }
35. @Override
36. **protected** **void** tearDown() **throws** Exception {
37. **super**.tearDown();
38. cache = **null**;
39. }
41. }

## 测试结果

java代码：

[查看复制到剪贴板打印](http://sishuok.com/forum/blogPost/list/311.html)

1. [INFO] ++++ serializing **for** key: bean **for** **class**: com.alisoft.sme.memcached.TBean
2. [INFO] ++++ memcache cmd (result code): add bean 8 0 93 (NOT\_STORED)
3. [INFO] ++++ data not stored in cache **for** key: bean
4. [INFO] ++++ deserializing **class** com.alisoft.sme.memcached.TBean
5. name=E网打进
6. [INFO] ++++ deserializing **class** com.alisoft.sme.memcached.TBean
7. name=E网打进

* 上一篇 [Java开发中的Memcache原理及实现](http://blog.csdn.net/xzknet/article/details/44066141)
* 下一篇 [mybatis generator生成带有分页的Mybatis代码](http://blog.csdn.net/xzknet/article/details/44158009)