分布式缓存系统之memcache

分布式缓存之memcache

说明书
版本 0.1
项目名称：

修订历史

版本号	作者	修订章节	修订原因	修订日期
0.1	甘建新	初稿	初稿	2017-04-26

目 录

概述

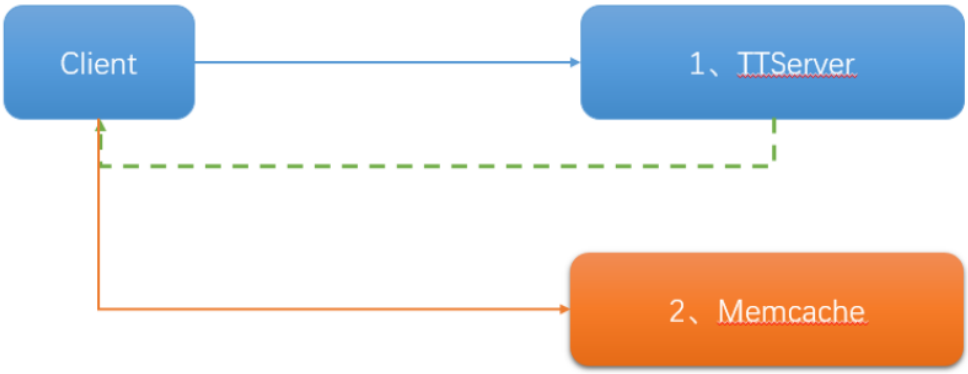
术语

需求背景

- 为提高系统性能;
- 减轻数据库的负载;

架构与组件

架构



ttserver

ttserver是一款 DBM 数据库，该数据库读写非常快，哈希模式写入100万条数据只需0.643秒，读取100万条数据只需0.773秒，是 Berkeley DB 等 DBM 的几倍。利用Tokyo
Tyrant构建兼容Memcached协议、支持故障转移、高并发的分布式key-value持久存储系统。key-value分布式存储系统查询速度快、存放数据量大、支持高并发，非常适合通过主键进行查询，但不能进行复杂的条件查询。
服务器启动进程：
/usr/local/tokyotyrant/bin/ttserver -port 1978 -dmn -pid /root/ttserver.pid /root/ttserver_db/index.tch

ttserver相关参数
ttserver [-host name] [-port num] [-thnum num] [-tout num] [-dmn] [-pid path] [-log path] [-ld|-le] [-ulog path] [-ulim num] [-uas] [-sid num] [-mhost name] [-mport num] [-rts path] [-ext path] [-mask expr] [-unmask expr] [dbname]
-host name : 指定需要绑定的服务器域名或IP地址。默认绑定这台服务器上的所有IP地址。
-port num : 指定需要绑定的端口号。默认端口号为1978
-thnum num : 指定线程数。默认为8个线程。
-tout num : 指定每个会话的超时时间（单位为秒）。默认永不超时。
-dmn : 以守护进程方式运行。
-pid path : 输出进程ID到指定文件（这里指定文件名）。
-log path : 输出日志信息到指定文件（这里指定文件名）。
-ld : 在日志文件中还记录DEBUG调试信息。
-le : 在日志文件中仅记录错误信息。
-ulog path : 指定同步日志文件存放路径（这里指定目录名）。
-ulim num : 指定每个同步日志文件的大小（例如128m）。
-uas :
使用异步IO记录更新日志（使用此项会减少磁盘IO消耗，但是数据会先放在内存中，不会立即写入磁盘，如果重启服务器或ttserver进程被kill掉，

将导致部分数据丢失。一般情况下不建议使用）。
-sid num : 指定服务器ID号（当使用主辅模式时，每台ttserver需要不同的ID号）
-mhost name : 指定主辅同步模式下，主服务器的域名或IP地址。
-mport num : 指定主辅同步模式下，主服务器的端口号。
-rts path : 指定用来存放同步时间戳的文件名。
-ext path : 扩展的脚本文件
-mask expr : 需要禁止的命令，多个命名用"，"隔开
-unmaks expr : 允许的命令

memcache

介绍

memcached 是由 Danga Interactive 开发并使用 BSD 许可的一种通用的分布式内存缓存系统。
它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提高了网站访问的速度。 MemCaChe是一个存储键值对的HashMap，在内存中对任意的数据（比如字符串、对象等）所使用的key-value存储，数据可以来自数据库调用、API调用，或者页面渲染的结果。MemCache设计理念就是小而强大，它简单的设计促进了快速部署、易于开发并解决面对大规模的数据缓存的许多难题，而所开放的API使得MemCache能用于Java、C/C++/C#、Perl、Python、PHP、Ruby等大部分流行的程序语言。

特性和限制

- MemCache中可以保存的item数据量是没有限制的，只要内存足够；
- MemCache单进程在32位机中最大使用内存为2G，64位机则没有限制；
- Key最大为250个字节，超过该长度无法存储；
- 单个item最大数据是1MB，超过1MB的数据不予存储；
- MemCache服务端是不安全的，比如已知某个MemCache节点，可以直接telnet过去，并通过flush_all让已经存在的键值对立即失效；
- 不能够遍历MemCache中所有的item，因为这个操作的速度相对缓慢且会阻塞其他的操作；
- MemCache的高性能源自于两阶段哈希结构：第一阶段在客户端，通过Hash算法根据Key值算出一个节点；第二阶段在服务端，通过一个内部的Hash算法，查找真正的item并返回给客户端。从实现的角度看，MemCache是一个非阻塞的、基于事件的服务器程序；
- MemCache设置添加某一个Key值的时候，传入expiry为0表示这个Key值永久有效，这个Key值也会在30天之后失效。

指令汇总

全部指令

已知MemCache的某个节点，直接telnet过去，就可以使用各种命令操作MemCache了，下面看下MemCache有哪几种命令：

命　令	作　用
get	返回Key对应的Value值
add	添加一个Key值，没有则添加成功并提示STORED，有则失败并提示NOT_STORED
set	无条件地设置一个Key值，没有就增加，有就覆盖，操作成功提示STORED
replace	按照相应的Key值替换数据，如果Key值不存在则会操作失败
stats	返回MemCache通用统计信息（下面有详细解读）
stats items	返回各个slab中item的数目和最老的item的年龄（最后一次访问距离现在的秒数）
stats slabs	返回MemCache运行期间创建的每个slab的信息（下面有详细解读）
version	返回当前MemCache版本号
flush_all	清空所有键值，但不会删除items，所以此时MemCache依旧占用内存
quit	关闭连接

stats指令解读

stats是一个比较重要的指令，用于列出当前MemCache服务器的状态，拿一组数据举个例子：
[js] view plaincopy

1. STAT pid 1023

2. STAT uptime 21069937
3. STAT time 1447235954
4. STAT version 1.4.5
5. STAT pointer_size 64
6. STAT rusage_user 1167.020934
7. STAT rusage_system 3346.933170
8. STAT curr_connections 29
9. STAT total_connections 21
10. STAT connection_structures 49
11. STAT cmd_get 49
12. STAT cmd_set 7458
13. STAT cmd_flush 0
14. STAT get_hits 7401
15. STAT get_misses 57
16. .. (delete、incr、decr、cas的hits和misses数，cas还多一个badval)
17. STAT auth_cmds 0
18. STAT auth_errors 0
19. STAT bytes_read 22026555
20. STAT bytes_written 8930466
21. STAT limit_maxbytes 4134304000
22. STAT accepting_conns 1
23. STAT listen_disabled_num 0
24. STAT threads 4
25. STAT bytes 151255336
26. STAT current_items 57146
27. STAT total_items 580656
28. STAT evicitions 0

这些参数反映着MemCache服务器的基本信息，它们的意思是：

参 数 名	作　　用
pid	MemCache服务器的进程id
uptime	服务器已经运行的秒数
time	服务器当前的UNIX时间戳
version	MemCache版本
pointer_size	当前操作系统指针大小，反映了操作系统的位数，64意味着MemCache服务器是64位的
rusage_user	进程的累计用户时间
rusage_system	进程的累计系统时间
curr_connections	当前打开着的连接数
total_connections	当服务器启动以后曾经打开过的连接数
connection_structures	服务器分配的连接构造数
cmd_get	get命令总请求次数
cmd_set	set命令总请求次数
cmd_flush	flush_all命令总请求次数
get_hits	总命中次数，重要，缓存最重要的参数就是缓存命中率，以get_hits / (get_hits + get_misses)表示，比如这个缓存命中率就是99.2%
get_misses	总未命中次数
auth_cmds	认证命令的处理次数
auth_errors	认证失败的处理次数
bytes_read	总读取的字节数
bytes_written	总发送的字节数
limit_maxbytes	分配给MemCache的内存大小（单位为字节）

accepting_conns	是否已经达到连接的最大值，1表示达到，0表示未达到
listen_disabled_num	统计当前服务器连接数曾经达到最大连接的次数，这个次数应该为0或者接近于0，如果这个数字不断增长， 就要小心我们的服务了
threads	当前MemCache总线程数，由于MemCache的线程是基于事件驱动机制的，因此不会一个线程对应一个用户请求
bytes	当前服务器存储的items总字节数
current_items	当前服务器存储的items总数量
total_items	自服务器启动以后存储的items总数量

stats slab指令解读

如果对上面的MemCache存储机制比较理解了，那么我们来看一下各个slab中的信息，还是拿一组数据举个例子：
[js] view plaincopy

```
1 STAT1:chunk_size 96
2 ...
3 STAT 2:chunk_size 144
4 STAT 2:chunks_per_page 7281
5 STAT 2:total_pages 7
6 STAT 2:total_chunks 50967
7 STAT 2:used_chunks 45197
8 STAT 2:free_chunks 1
9 STAT 2:free_chunks_end 5769
10 STAT 2:mem_requested 6084638
11 STAT 2:get_hits 48084
12 STAT 2:cmd_set 59588271
13 STAT 2:delete_hits 0
14 STAT 2:incr_hits 0
15 STAT 2:decr_hits 0
16 STAT 2:cas_hits 0
17 STAT 2:cas_badval 0
18 ...
19 STAT 3:chunk_size 216
20 ...
```

首先看到，第二个slab的chunk_size（144）/第一个slab的chunk_size（96）=1.5，第三个slab的chunk_size（216）/第二个slab的chunk_size（144）=1.5，可以确定这个MemCache的增长因子是1.5，chunk_size以1.5倍增长。然后解释下字段的含义：

参 数 名	作　　用
chunk_size	当前slab每个chunk的大小，单位为字节
chunks_per_page	每个page可以存放的chunk数目，由于每个page固定为1M即1024*1024字节，所以这个值就是（1024*1024/chunk_size）
total_pages	分配给当前slab的page总数
total_chunks	当前slab最多能够存放的chunk数，这个值是total_pages*chunks_per_page
used_chunks	已经被分配给存储对象的chunks数目
free_chunks	曾经被使用过但是因为过期而被回收的chunk数
free_chunks_end	新分配但还没有被使用的chunk数，这个值不为0则说明当前slab从来没有出现过容量不够的时候
mem_requested	当前slab中被请求用来存储数据的内存空间字节总数，（total_chunks*chunk_size）-mem_requested表示有多少内存在当
get_hits	当前slab中命中的get请求数
cmd_set	当前slab中接收的所有set命令请求数
delete_hits	当前slab中命中的delete请求数
incr_hits	当前slab中命中的incr请求数
decr_hits	当前slab中命中的decr请求数
cas_hits	当前slab中命中的cas请求数

cas_badval	当前slab中命中但是更新失败的cas请求数

看到这个命令的输出量很大，所有信息都很有作用。举个例子吧，比如第一个slab中使用的chunks很少，第二个slab中使用的chunks很多，这时就可以考虑适当增大MemCache的增长因子了，让一部分数据落到第一个slab中去，适当平衡两个slab中的内存，避免空间浪费。

开发部分

maven引用

```
<dependency>
<groupId>com.netfinworks.ucs</groupId>
<artifactId>ucs-support-annotation</artifactId>
<version>1.0.0</version>
</dependency>
```

系统中使用的版本是1.0.0；
1.1.1版本修复了在泛型方法上获取不到注解的问题；

配置properties

```
#本地缓存数量
ucs.support.annotation.cache.localCacheMaxSize = 10000
#cache名称，对应ttserver选择不同的memcache实际地址
ucs.support.annotation.cache.name = index.com.netfinworks.cache.cc
#ttserver地址
ucs.support.annotation.cache.name.nameServerAddress = tcp://192.168.180.35:1978
#超时时间
ucs.support.cache.conf.defaultExpireSecond=864000
#memcache连接数
ucs.support.annotation.cache.memcachedConnectionPoolSize = 2
#连接超时时间
ucs.support.annotation.cache.connectTimeoutMs = 60000
#缓存操作超时时间
ucs.support.annotation.cache.opTimeoutMs = 1000
#ucs listener
ucs.support.annotation.cache.listenerBeanNames = ucs.support.annotation.commonListener
#MQ描述
ucs.support.annotation.cache.commonListener.description = ues.ws.ucs
#MQ队列名称
ucs.support.annotation.cache.commonListener.queueName = ucs_warn
#MQ初始化工厂
netfinworksmq.java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory
#MQ连接地址，在删除缓存失败会发送MQ
netfinworksmq.java.naming.provider.url=failover:tcp://192.168.180.35:61616?soTimeout=30000&connectionTimeout=30000)?jms.useAsyncSend=true&timeout=30000
netfinworksmq.java.naming.security.principal=netfinworks
netfinworksmq.java.naming.security.credentials=netfinworks
```

加载配置文件

applicationContext.xml中加入

```
<aop:aspectj-autoproxy />
<import resource="classpath:ucs-support-annotatiion.xml" />
<import resource="classpath:ucs-support-annotatiion-commonlistener.xml" />
```

注解介绍

@InvalidateCache

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface InvalidateCache{
/**
```

```
* 缓存命名空间
* @return
*/
public abstract String namespace();
/**
* 指定的缓存key，如果方法签名中指定了KeyProvider标签，则优先使用KeyProvider，忽略assignedKey.
* @return
*/
public abstract String assignedKey() default "";
/**
* <ul>
* 清空模式：
* <li><code>InvalidateCache.MODE_SINGE</code> 将多个keyGenerator生成的key合并为一个key清除;
如果没有keyGenerator则清除assignedKey</li>
* <li><code>InvalidateCache.MODE_MULTI</code> 将多个keyGenerator生成的key当成一个独立的key清除;
如果没有keyGenerator则清除assignedKey</li>
* </ul>
* @return
*/
public abstract InvalidateCacheMode mode() default InvalidateCacheMode.MODE_SINGE;
/**
* <ul>
* 切面执行顺序
* <li><code>InvalidateExecuteOrder.BEFORE</code>在原方法之前执行，默认</li>
* <li><code>InvalidateExecuteOrder.AFTER</code>在原方法之后执行</li>
* </ul>
* @see InvalidateExecuteOrder
* @return
*/
public abstract InvalidateExecuteOrder executeOrder() default InvalidateExecuteOrder.BEFORE;
}
```

功能：删除缓存，一般添加在增加，修改，删除的方法上，key值需使用@KeyGenerator生成；
范围：放在方法上；
参数注释：namespace一般为每个业务设置个对应的值

@Retention(RetentionPolicy.RUNTIME)

```
@Target(ElementType.PARAMETER)
public @interface KeyGenerator {
/**
* 生成key的方法名，默认为toString方法
*
* @return
*/
public abstract String[] keyMethod() default "toString";
/**
* 是否允许Key为null，默认不允许
*
* @return
*/
public abstract boolean couldBeNull() default false;
}@KeyGenerator
```

功能：获取指定方法上的值作为key值；
范围：放在参数上；

@CacheResult

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface CacheResult {
/**
* 缓存命名空间
* @return
*/
public abstract String namespace();
```

```
/**
* 缓存过期时间,默认采用系统配置的时间
* @return
*/
public abstract int expireSecond() default -1;
/**
* 指定的缓存key，如果方法签名中指定了KeyProvider标签，则优先使用KeyProvider，忽略assignedKey.
* @return
*/
public abstract String assignedKey() default "";
}
```

功能：获取缓存结果，一般添加查询方法上，key值需使用@KeyGenerator生成；
范围：放在方法上；

index规则

index是作为查找memcache实际地址的条件。
一般需要在basis后台增加刷新缓存功能。
对于index使用规则，可以使用一个业务一个index，也可以一个业务系统一个index，建议采用后者，可根据实际业务场景再判断。
以下是生产环境index与memcache对应关系：
index.com.netfinworks.cache.cc,10.228.6.158:30001
index.com.netfinworks.cache.pbs,10.228.6.158:30022
index.com.netfinworks.cache.second-merchant-service,10.228.6.158:30023
index.com.netfinworks.captcha,10.228.6.158:30016
index.com.netfinworks.cashier.service,10.228.6.158:30013
index.com.netfinworks.cmf.fundChannel,10.228.6.158:30004
index.com.netfinworks.cmf.unityCode,10.228.6.158:30005
index.com.netfinworks.deposit.service,10.228.6.158:30014
index.com.netfinworks.dpm,10.228.6.158:30003
index.com.netfinworks.enterprisesite.cache,10.228.6.158:30015
index.com.netfinworks.guardian.login,10.228.6.158:30008
index.com.netfinworks.guardian.roles,10.228.6.158:30009
index.com.netfinworks.lflt,10.228.6.158:30007
index.com.netfinworks.ma.authorize,10.228.6.158:30017
index.com.netfinworks.ma.cache,10.228.6.158:30002
index.com.netfinworks.pfs.basis,10.228.6.158:30011
index.com.netfinworks.pfs.payment,10.228.6.158:30010
index.com.netfinworks.rms.rules,10.228.6.158:30006
index.com.netfinworks.site.cache,10.228.6.158:30012
index.com.netfinworks.site.login,10.228.6.158:30020
index.com.netfinworks.cache.tsp,10.255.6.158:30025

运维部分

启动ttserver

/usr/local/tokyotyrant/bin/ttserver -port 1978 -dmn -pid /root/ttserver.pid /root/ttserver_db/index.tch

启动memcache

以开发环境为例：
/opt/commonCache1.4.20/bin/memcached -d -u nobody -m 100 -p 30024 -A -c 10240
其中30024为端口号；

缓存管理平台地址

开发环境地址： http://192.168.180.35:8121/cache-mgmt/ui/;jsessionid=4C71B8920CCB6D6096F94D3F50D61BEE?4
用户名/密码：****

添加category

示例：index.com.netfinworks.pns.service

添加address

到此可正常使用memcache。

参考资料

http://memcached.org/
http://fallabs.com/tokyocabinet/
http://jaist.dl.sourceforge.net/project/tokyocabinet/tokyotyrant/