Memcached 通信协议(中文版)

memcache 通信协议

英文水平很烂，做梦都想着能把英语学习，可以使用一口流利的英文和 洋鬼子交流，顺便忽悠下自己的同胞。没有地方学习英语，看还可以，网上有很多关于计算机的英文文献，写还行，说就完全不可能了。在以后的工作中慢慢的锻炼 自己的英语水平吧，先翻译一下一些计算机的英文文献，锻炼下^_^，能读则读不能读则一笑置之。

协议

memcached 的客户端通过 TCP 连接与服务器通信（UDP 协议的接口也可以使用，详细说明请参考”UDP 协议”部分）。一个给定的运行中的 memcached 服务器在某个（可配置的）端口上监听连接；客户端连接该端口，发送命令给服务器，读取反馈，最后关闭连接。

没有必要发送一个专门的命令去结束会话。客户端可以在不需要该连接的时候就关闭它。注意：我们鼓励客户端缓存它们与服务器的连接，而不是每次要存储或读取数据的时候再次重新建立与服务器的连接。memcache 同时打开很多连接不会对性能造成到大的影响，这是因为 memcache 在设计之处，就被设计成即使打开了很多连接（数百或者需要时上千个连接）也可以高效的运行。缓存连接可以节省与服务器建立 TCP 连接的时间开销（于此相比，在服务器段为建立一个新的连接所做准备的开销可以忽略不计）。

memcache 通信协议有两种类型的数据：文本行和非结构化数 据。文本行用来发送从客户端到服务器的命令以及从服务器回送的反馈信息。非结构化的数据用在客户端希望存储或者读取数据时。服务器会以字符流的形式严格准 确的返回相应数据在存储时存储的数据。服务器不关注字节序，它也不知道字节序的存在。memcahce 对非结构化数据中的字符没有任何限制，可以是任意的字符，读取数据时，客户端可以在前次返回的文本行中确切的知道接下来的数据块的长度。

文本行通常以“"r"n”结束。非结构化数据通常也是以“"r"n”结束，尽管"r、"n 或者其他任何 8 位字符可以出现在数据块中。所以当客户端从服务器读取数据时，必须使用前面提供的数据块的长度，来确定数据流的结束，二不是依据跟随在字符流尾部的“"r"n”来确定数据流的结束，尽管实际上数据流格式如此。

关键字　Keys

memcached 使用关键字来区分存储不同的数据。关键字是一个字符串，可以唯一标识一条数据。当前关键字的长度限制是 250 个字符（当然目前客户端似乎没有需求用这么长的关键字）；关键字一定不能包含控制字符和空格。

命令 Commands

memcahe 有 3 种类型的命令：

- 存储命令—（3 个命令：set、add 和 replace）要求服务器安装关键字存储数据。客户端发送一个命令行，然后一个数据块；命令执行后客户端等待一行反馈，用来表示命令执行成功与否。
- 读取命令--（只有 1 个命令：get） 要求服务器根据一组关键字读取数据（在一个请求总可以包含一个或多个关键字）。客户端发送一个包含请求关键字的命令行；命令传递到服务器后，服务器就查找 每一个关键字下的数据，然户将数据以“一个关键字数据，一个反馈信息行跟着一个数据块”的格式回送数据，直到服务器发送“END”的反馈行。
- 其他命令，如 flush_all，version 等。这些命令不使用非结构化的数据。对于这些命令，客户端发送一个文本的命令行，根据命令的特性等待一行数据或者在最后一行以“END“结尾的几行反馈信息。

所有的命令行总是以命令的名字开始，紧接着是以空格分割的参数。命令名称都是小写，并且是大小写敏感的。

超时时间　Expiration times

一些发送到服务器的命令包含超时时间（该超时时间对应于：数据项保存时间；客户端操作限时）。在这些例子中，被发送的真实时间要么是 UNIX 时间戳（自 1970 年 1 月 1 日零时起的秒数数值），或者从当前时间开始算起的秒数。对于后一种情况，秒数的数值不能超过 60*60*24*30（30 天的秒数）；如果秒数的数值大于了这个数值，服务器会认为该数值是 UNIX 时间戳，而不是自当前时间开始的秒数偏移值。

错误信息　Error strings

每个命令都有可能被反馈以一个错误消息。这些错误消息有以下三个类型：

- “ERROR"r"n”

  意味着客户端发送了一个在协议中不存在的命令。

- "CLIENT_ERROR <error>"r"n"

  表示客户端输入的命令行上存在某种错误，输入不符合协议规定。<error>是一个人工可读（human-readable）的错误注释。

- "SERVER_ERROR <error>"r"n"

  表示服务器在执行命令时发生了某些错误，致使服务器无法执行下去。<error>也是一个人工可读（human-readable）的错误注释。在一些情况下，错误导致服务器不能再为客户端服务（这样的情况很少发生），服务器就会在发生错误消息后主动关闭连接。这也是服务器主动关闭到客户端连接的唯一情况。

后续秒数各种命令的时候，我们不再赘述错误消息的情况，当我们要清楚错误是存在的，不可忽略。

存储命令　Storage commands

首先，客户端发生如下这样的命令：

```
<command name> <key> <flags> <exptime> <bytes>"r"n

<data block>"r"n
```

其中：

<command name> 是 set、add 或者 replace。set 表示存储该数据；add 表示如果服务器没有保存该关键字的情况下，存储该数据；replace 表示在服务器已经拥有该关键字的情况下，替换原有内容。

<key>是客户端要求服务器存储数据的关键字。

<flags>是一个 16 位的无符号整数，服务器将它和数据一起存储并且当该数据被检索时一起返回。客户端可能使用该数值作为一个位图来存储特殊数据信息；这个字段对服务器不是透明的。

<exptime>是超时时间。如果值为 0 表示该数据项永远不超时（但有时候该数据项可能被删除以

为其他数据腾出空间）；如果值不为 0，可能是绝对的 UNIX 时间，也可能是自现在开始的偏移值，它保证客户段在这个超时时间到达后，客户端将取不到该数据项。

<bytes>是随后数据的字节数，不包括终结符”"r"n”。<bytes>有可能是 0，它后面将是一个空的数据块。

<data block>是真正要存储数据流。

发送命令行和数据后，客户端等待反馈，可以是如下几种情况：

- "STORED"r"n" 表示存储数据成功。
- "NOT_STORED"r"n" 表示发送的数据没有存储，但这不因为错误，而是发生在 add 或者 replace 命令不能满足条件时，或者数据项正处于要删除的队列中。
- 错误消息

读取命令 Retrieval command:

读取命令如下所示：

```
get <key>*"r"n
```

<key>*表示一个或多个使用空格分割的关键字字符串。

发送命令后，客户端等待返回一个或多个数据项，每个数据项的格式是一个文本行，后跟着一个数据块。当所有的数据项发送完毕后，服务器发送字符串”END"r"n”表示服务器反馈数据的结束。

返回数据项的格式如下：

```
VALUE <key> <flags> <bytes>"r"n
<data block>"r"n
```

<key>是发生数据项的关键字。

<flags>是存储该数据项时，客户端命令中的标志字段。

<bytes>是紧跟文本行后数据块的长度，不包括终结符”"r"n”。

<data block>是数据项的数据部分。

如果请求命令行中的有些关键字对应的数据项没有被返回，这意味着服务器没有该关键字标示下的数据项（有可能是从来没有被存储过，或者存储过但被删除掉以腾出内存空间，或者数据项超时了，再或者它被某个客户端删除了）。

删除　Deletion

删除命令允许直接删除数据项，命令格式如下：

```
delete <key> <time>"r"n
```

<key>是客户端希望服务器删除数据项的关键字

<time>是客户端希望服务器阻止 add 和 replace 命令使用该关键字数据项的秒数，可以是相对时间也可以是 UNIX 的绝对时间。在这段时间内，数据项被放入一个删除队列，它不能被 get 命令读取，在其上使用 add 和 replace 也会失败，但使用 set 命令可以成功。当这个时间过去后，数据项从服务器的内存中真正的删除。该参数是可选参数，如果不存在默认为 0，这意味着立即从服务器上删除。

服务器返回信息：

- "DELETED"r"n"　表示数据项删除成功
- "NOT_FOUND"r"n"　表示该关键字指定的数据项在服务器上没有找到
- 其他错误消息

下面的 flush_all 命令使得所有存在的数据项立即失效。

增加/减少　Increment/Decrement

“incr”和”decr”命令用来修改以及存在的数据项的内容，增加或者减少它。该数据被当作 32 位无符号整数处理。如果当前数据非此类数据，则经将该内容当作 0 来处理。另外在其上施加 incr/decr 命令的数据项必须是业已存在的；对于不存在的数据项不会将它作为 0 对待，而是以错误结束。

客户端发送命令行如下格式：

```
incr <key> <value>"r"n

或者

decr <key> <value>"r"n
```

<key>是客户端要修改数据项的关键字

<value>是对该数据行进行增加或者减少的操作数。它是一个 32 位的无符号整数。

反馈信息有如下几种：

l "NOT_FOUND"r"n"

表明在服务器上没有找到该数据项。

l "<value>"r"n "

value 是执行完增加/减少命令后，该数据项新的数值。

l 错误信息。

注意到“decr”命令的下溢问题，如果客户端尝试减少的数量小于 0，其结果是 0。“incr”命令的溢出问题没有检查。另外减少一个数据而使它减少了长度，但不保证减少它返回时的长度。该数字可能是附加空格的数字，但这只是实现的优化，所以你不能相信它。

统计 Statistics

“stats”命令用来查询服务器的运行情况和其他内部数据。它有两种情况，以有无参数来区分：

```
stats"r"n

或者

stats <args>"r"n
```

第一种情况它导致服务器输出一般统计信息以及设置信息和文档化内容。

第二种情况根据<args>具体的参数，服务器发送各种内部数据。这部分没有在协议中文档化，因为与 memcache 的开发者有关其可能是随时变化的。

多用途统计 General-purpose statistics

当接收到没有带参数的“stats”命令后，服务器发送许多类似与如下格式的文本行：

```
STAT <name> <value>"r"n
```

当类似的文本行全部发送完毕后，服务器发送如下的文本行结束反馈信息：

```
END"r"n
```

在所有 STAT 文本行中，<name>是该统计项目的名称，<value>是其数据。下面是一份 stats 命令反馈的所有统计项目的列表，后面跟着其值的数据类型。在数据类型列中，”32u”表示一个 32 位无符号整数，”64u”表示一个 64 位无符号整数，”32u:32u”表示是两个用冒号分割的 32 位无

符号整数。

名称	值类型	含义
pid	32u	服务器进程的进程号
uptime	32u	服务器自运行以来的秒数
time	32u	当前服务器上的 UNIX 时间
version	string	服务器的版本字符串
rusage_user	32u:32u	服务器进程积累的用户时间（秒:微妙）
rusage_system	32u:32u	服务器进程积累的系统时间（秒:微妙）
curr_items	32u	当前在服务器上存储的数据项的个数
total_items	32u	在服务器上曾经保存过的数据项的个数
bytes	64u	当前服务器上保存数据的字节数
curr_connections	32u	处于打开状态的连接数目
total_connections	32u	曾经打开过的所有连接的数目
connection_structures	32u	服务器分配的连接结构体的个数
cmd_get	64u	get 命令请求的次数
cmd_set	64u	存储命令请求的次数
get_hits	64u	关键字获取命中的次数

get_misses	64u	关键字获取没有命中的次数
evictions	64u	所有因超时而被替换出内存的数据项的个数
bytes_read	64u	服务器从网络上读取到的字节数
bytes_write	64u	服务器向网络上写的字节数
limit_maxbytes	64u	服务器允许存储数据的最大值

其他命令　Other commands

"flush_all"是一个带有可选数字参数的命令，它的执行总是成功的，服务器总是响应以"OK"r"n"字符串。它的作用是使得所有的数据项立即（默认）或者经过一个指定的超时时间后全部失效。在置数据项失效后，对于读取命令将不会返回任何内容，除非在失效后这些数据再次被存储。flush_all 并没有真正的释放这些存在过的数据项占用的内存空间；数据空间真实被占用的情况发生在使用新的数据项覆盖老的数据项时。该命令作用最准确的定义是：它导致所有更新时间早于该命令设定的时间点的数据项，在被检索时被忽略，其表现就像已被删除了一样。

使用带有延时 flush_all 命令的目的是，当你有个 memcached 服务器池，需要刷新所有的内容时，但不能在同一时间刷洗所有的服务器，这样就可能因为所有的服务器突然都要重新建立数据内容，而导致数据库压力的颠簸。延时选项允许你设置他们隔 10 秒失效（设置第一个延时为 0，第二个 10 秒，第三个 20 秒等等）。

"version"是一个没有参数的命令，命令格式如下：

```
version"r"n
```

服务器发回的反馈信息如下：

l　　"VERSION <version>"r"n"

<version>是从服务器返回的版本字符串。

l　　错误消息。

"**verbosity**"是一个带有数字参数的命令。它的执行总是成功的，服务器反馈以"OK"r"n"表示执行完成。它用来设置日志输出的详细等级。

"**quit**"是一个没有参数的命令。其格式如下：

```
quit"r"n
```

当服务器接受到此命令后，就关闭与该客户的连接。不管怎样，客户端可以在任意不需要该连接的时刻关闭它，而不需要发送该命令。

UDP 协议　　UDP protocol

当基于 TCP 协议的连接数超过 TCP 连接的上限时，我们可以使用 UDP 协议来替代。但是 UDP 协议接口不提供可靠的传输，所以多用在不严格要求成功的操作上；典型的 get 请求会因为缓存的问题，引起丢失或者不完整的传输。

每个 UDP 数据包包含一个简单的帧头，接着就是如 TCP 协议描述的数据格式的数据流。在当前的实现中，请求必须包含在一个单独的 UDP 数据包中，但返回可能分散在多个数据包中。（唯一的可以拆分请求数据包的是大的多关键字 get 请求和 set 请求，鉴于可靠性相比而言他们更适合用 TCP 传输。）

帧头有 8 字节长，如下是其格式（所有的数字都是 16 位网络字节序整形，高位在前）：

0 - 1	请求 ID
2 - 3	序列号
4 - 5	在当前的消息中含有的数据包的个数
6-7	保留以后使用，当前必须为 0

请求 ID 由客户端提供。它的典型值是一个从随机种子开始递增值，实际上客户端可以使用任意的请求 ID。服务器的反馈信息中包含了和请求命令中一样的请求 ID。客户端凭借这个请求 ID 区分来自于同一服务器的反馈。每一个包含未知请求 ID 的数据包，可能是由于延时反馈造成，这些数据包都应该抛弃不用。

序列号从 0 到 n-1，n 是消息中总的数据包的个数。客户端按照序列号排序重组数据包；结果序列中包含了一个完整的如 TCP 协议一样格式的反馈信息（包含了“"r"n”总结字符串）。