

 openresty / memc-nginx-module

Watch9

Star120

Fork27

An extended version of the standard memcached module that supports set, add, delete, and many more memcached commands. <http://wiki.nginx.org/NginxHttpMemcModule>

227 commits











2 branches


20 releases

3 contributors

 branch: master memc-nginx-module / +



tests: skipped a test for conditional GETs. ...		
	agentzh authored on Jul 15	latest commit 4ff1453bae 
 doc	doc: bumped version to 0.15 and updated other parts to reflect recent...	2 months ago
 src	added copyright notices to source files.	2 months ago
 t	tests: skipped a test for conditional GETs.	2 months ago
 util	removed the luajit suppression rules because we no longer need it. al...	2 months ago
 .gitignore	updated .gitignore a bit.	a year ago
 README.markdown	doc: bumped version to 0.15 and updated other parts to reflect recent...	2 months ago
 config	further refactoring by moving some parts of ngx_http_memc_module.c in...	5 years ago
 valgrind.suppress	removed the luajit suppression rules because we no longer need it. al...	2 months ago

 README.markdown

Name

ngx_memc - An extended version of the standard memcached module that supports set, add, delete, and many more memcached commands.

This module is not distributed with the Nginx source. See [the installation instructions](#).

Table of Contents

- Version
- Synopsis
- Description
 - Keep-alive connections to memcached servers
 - How it works
- Memcached commands supported
 - get \$memc_key
 - set \$memc_key \$memc_flags \$memc_exptime \$memc_value
 - add \$memc_key \$memc_flags \$memc_exptime \$memc_value
 - replace \$memc_key \$memc_flags \$memc_exptime \$memc_value
 - append \$memc_key \$memc_flags \$memc_exptime \$memc_value
 - prepend \$memc_key \$memc_flags \$memc_exptime \$memc_value
 - delete \$memc_key
 - delete \$memc_key \$memc_exptime
 - incr \$memc_key \$memc_value

<> Code

Issues5


Pull Requests0


Pulse


Graphs


HTTPS clone URL

https://github.com



You can clone with HTTPS, SSH, or Subversion. 

 Clone in Desktop

 Download ZIP

- `decr $memc_key $memc_value`
- `flush_all`
- `flush_all $memc_exptime`
- `stats`
- `version`
- Directives
 - `memc_pass`
 - `memc_cmds_allowed`
 - `memc_flags_to_last_modified`
 - `memc_connect_timeout`
 - `memc_send_timeout`
 - `memc_read_timeout`
 - `memc_buffer_size`
 - `memc_ignore_client_abort`
- Installation
 - For Developers
- Compatibility
- Community
 - English Mailing List
 - Chinese Mailing List
- Report Bugs
- Source Repository
- Changes
- Test Suite
- TODO
- Getting involved
- Author
- Copyright & License
- See Also

Version

This document describes ngx_memc v0.15 released on 8 July 2014.

Synopsis

```
# GET /foo?key=dog
#
# POST /foo?key=cat
# Cat's value...
#
# PUT /foo?key=bird
# Bird's value...
#
# DELETE /foo?key=Tiger
location /foo {
    set $memc_key $arg_key;

    # $memc_cmd defaults to get for GET,
    # add for POST, set for PUT, and
    # delete for the DELETE request method.

    memc_pass 127.0.0.1:11211;
}
```



```

# GET /bar?cmd=get&key=cat
#
# POST /bar?cmd=set&key=dog
# My value for the "dog" key...
#
# DELETE /bar?cmd=delete&key=dog
# GET /bar?cmd=delete&key=dog
location /bar {
    set $memc_cmd $arg_cmd;
    set $memc_key $arg_key;
    set $memc_flags $arg_flags; # defaults to 0
    set $memc_exptime $arg_exptime; # defaults to 0

    memc_pass 127.0.0.1:11211;
}

# GET /bar?cmd=get&key=cat
# GET /bar?cmd=set&key=dog&val=animal&flags=1234&exptime=2
# GET /bar?cmd=delete&key=dog
# GET /bar?cmd=flush_all
location /bar {
    set $memc_cmd $arg_cmd;
    set $memc_key $arg_key;
    set $memc_value $arg_val;
    set $memc_flags $arg_flags; # defaults to 0
    set $memc_exptime $arg_exptime; # defaults to 0

    memc_cmds_allowed get set add delete flush_all;

    memc_pass 127.0.0.1:11211;
}

http {
    ...
    upstream backend {
        server 127.0.0.1:11984;
        server 127.0.0.1:11985;
    }
    server {
        location /stats {
            set $memc_cmd stats;
            memc_pass backend;
        }
        ...
    }
}
...

# read the memcached flags into the Last-Modified header
# to respond 304 to conditional GET
location /memc {
    set $memc_key $arg_key;

    memc_pass 127.0.0.1:11984;

    memc_flags_to_last_modified on;
}

location /memc {

```



```
set $memc_key foo;
set $memc_cmd get;

# access the unix domain socket listend by memcached
memc_pass unix:/tmp/memcached.sock;
}
```

Description

This module extends the standard [memcached module](#) to support almost the whole [memcached ascii protocol](#).

It allows you to define a custom [REST](#) interface to your memcached servers or access memcached in a very efficient way from within the nginx server by means of subrequests or [independent fake requests](#).

This module is not supposed to be merged into the Nginx core because I've used [Ragel](#) to generate the memcached response parsers (in C) for joy :)

If you are going to use this module to cache location responses out of the box, try [srcache-nginx-module](#) with this module to achieve that.

When used in conjunction with [lua-nginx-module](#), it is recommended to use the [lua-resty-memcached](#) library instead of this module though, because the former is much more flexible and memory-efficient.

[Back to TOC](#)

Keep-alive connections to memcached servers

You need the (now standard) [HttpUpstreamKeepaliveModule](#) together with this module for keep-alive TCP connections to your backend memcached servers.

Here's a sample configuration:

```
http {
    upstream backend {
        server 127.0.0.1:11211;

        # a pool with at most 1024 connections
        # and do not distinguish the servers:
        keepalive 1024;
    }

    server {
        ...
        location /memc {
            set $memc_cmd get;
            set $memc_key $arg_key;
            memc_pass backend;
        }
    }
}
```

[Back to TOC](#)

How it works

It implements the memcached TCP protocol all by itself, based upon the `upstream` mechanism. Everything involving I/O is non-blocking.

The module itself does not keep TCP connections to the upstream memcached servers across requests, just like other upstream modules. For a working solution, see section [Keep-alive connections to memcached servers](#).

[Back to TOC](#)

Memcached commands supported

The memcached storage commands `set`, `add`, `replace`, `prepend`, and `append` uses the `$memc_key` as the key, `$memc_exptime` as the expiration time (or delay) (defaults to 0), `$memc_flags` as the flags (defaults to 0), to build the corresponding memcached queries.

If `$memc_value` is not defined at all, then the request body will be used as the value of the `$memc_value` except for the `incr` and `decr` commands. Note that if `$memc_value` is defined as an empty string (`""`), that empty string will still be used as the value as is.

The following memcached commands have been implemented and tested (with their parameters marked by corresponding nginx variables defined by this module):

[Back to TOC](#)

get \$memc_key

Retrieves the value using a key.

```
location /foo {
    set $memc_cmd 'get';
    set $memc_key 'my_key';

    memc_pass 127.0.0.1:11211;

    add_header X-Memc-Flags $memc_flags;
}
```

Returns `200 OK` with the value put into the response body if the key is found, or `404 Not Found` otherwise. The `flags` number will be set into the `$memc_flags` variable so it's often desired to put that info into the response headers by means of the standard `add_header` directive.

It returns `502` for `ERROR`, `CLIENT_ERROR`, or `SERVER_ERROR`.

[Back to TOC](#)

set \$memc_key \$memc_flags \$memc_exptime \$memc_value

To use the request body as the memcached value, just avoid setting the `$memc_value` variable:

```
# POST /foo
# my value...
location /foo {
    set $memc_cmd 'set';
    set $memc_key 'my_key';
```



```

set $memc_flags 12345;
set $memc_exptime 24;

memc_pass 127.0.0.1:11211;
}

```

Or let the `$memc_value` hold the value:

```

location /foo {
    set $memc_cmd 'set';
    set $memc_key 'my_key';
    set $memc_flags 12345;
    set $memc_exptime 24;
    set $memc_value 'my_value';

    memc_pass 127.0.0.1:11211;
}

```

Returns 201 Created if the upstream memcached server replies STORED , 200 for NOT_STORED , 404 for NOT_FOUND , 502 for ERROR , CLIENT_ERROR , or SERVER_ERROR .

The original memcached responses are returned as the response body except for 404 NOT FOUND .

[Back to TOC](#)

add \$memc_key \$memc_flags \$memc_exptime \$memc_value

Similar to the `set` command.

[Back to TOC](#)

replace \$memc_key \$memc_flags \$memc_exptime \$memc_value

Similar to the `set` command.

[Back to TOC](#)

append \$memc_key \$memc_flags \$memc_exptime \$memc_value

Similar to the `set` command.

Note that at least memcached version 1.2.2 does not support the "append" and "prepend" commands. At least 1.2.4 and later versions seem to supports these two commands.

[Back to TOC](#)

prepend \$memc_key \$memc_flags \$memc_exptime \$memc_value

Similar to the `append` command.

[Back to TOC](#)

delete \$memc_key

Deletes the memcached entry using a key.

```
location /foo
    set $memc_cmd delete;
    set $memc_key my_key;

    memc_pass 127.0.0.1:11211;
}
```

Returns 200 OK if deleted successfully, 404 Not Found for NOT_FOUND, or 502 for ERROR, CLIENT_ERROR, or SERVER_ERROR.

The original memcached responses are returned as the response body except for 404 NOT FOUND.

[Back to TOC](#)

delete \$memc_key \$memc_exptime

Similar to the `delete $memc_key` command except it accepts an optional `expiration` time specified by the `$memc_exptime` variable.

This command is no longer available in the latest memcached version 1.4.4.

[Back to TOC](#)

incr \$memc_key \$memc_value

Increments the existing value of `$memc_key` by the amount specified by `$memc_value`:

```
location /foo {
    set $memc_key my_key;
    set $memc_value 2;
    memc_pass 127.0.0.1:11211;
}
```

In the preceding example, every time we access `/foo` will cause the value of `my_key` increments by 2.

Returns 200 OK with the new value associated with that key as the response body if successful, or 404 Not Found if the key is not found.

It returns 502 for ERROR, CLIENT_ERROR, or SERVER_ERROR.

[Back to TOC](#)

decr \$memc_key \$memc_value

Similar to `incr $memc_key $memc_value`.

[Back to TOC](#)

flush_all

Mark all the keys on the memcached server as expired:

```
location /foo {
    set $memc_cmd flush_all;
    memc_pass 127.0.0.1:11211;
}
```

[Back to TOC](#)

flush_all \$memc_exptime

Just like `flush_all` but also accepts an expiration time specified by the `$memc_exptime` variable.

[Back to TOC](#)

stats

Causes the memcached server to output general-purpose statistics and settings

```
location /foo {
    set $memc_cmd stats;
    memc_pass 127.0.0.1:11211;
}
```

Returns `200 OK` if the request succeeds, or `502` for `ERROR`, `CLIENT_ERROR`, or `SERVER_ERROR`.

The raw `stats` command output from the upstream memcached server will be put into the response body.

[Back to TOC](#)

version

Queries the memcached server's version number:

```
location /foo {
    set $memc_cmd version;
    memc_pass 127.0.0.1:11211;
}
```

Returns `200 OK` if the request succeeds, or `502` for `ERROR`, `CLIENT_ERROR`, or `SERVER_ERROR`.

The raw `version` command output from the upstream memcached server will be put into the response body.

[Back to TOC](#)

Directives

All the standard `memcached module` directives in nginx 0.8.28 are directly inherited, with the `memcached_` prefixes replaced by `memc_`. For example, the `memcached_pass` directive is spelled `memc_pass`.

Here we only document the most important two directives (the latter is a new directive introduced by

this module).

[Back to TOC](#)

memc_pass

syntax: *memc_pass* <memcached server IP address>:<memcached server port>

syntax: *memc_pass* <memcached server hostname>:<memcached server port>

syntax: *memc_pass* <upstream_backend_name>

syntax: *memc_pass* unix:<path_to_unix_domain_socket>

default: *none*

context: *http, server, location, if*

phase: *content*

Specify the memcached server backend.

[Back to TOC](#)

memc_cmds_allowed

syntax: *memc_cmds_allowed* <cmd>...

default: *none*

context: *http, server, location, if*

Lists memcached commands that are allowed to access. By default, all the memcached commands supported by this module are accessible. An example is

```
location /foo {
    set $memc_cmd $arg_cmd;
    set $memc_key $arg_key;
    set $memc_value $arg_val;

    memc_pass 127.0.0.1:11211;

    memc_cmds_allowed get;
}
```

[Back to TOC](#)

memc_flags_to_last_modified

syntax: *memc_flags_to_last_modified* on|off

default: *off*

context: *http, server, location, if*

Read the memcached flags as epoch seconds and set it as the value of the `Last-Modified` header. For conditional GET, it will signal nginx to return `304 Not Modified` response to save bandwidth.

[Back to TOC](#)

memc_connect_timeout

syntax: *memc_connect_timeout <time>*

default: 60s

context: *http, server, location*

The timeout for connecting to the memcached server, in seconds by default.

It's wise to always explicitly specify the time unit to avoid confusion. Time units supported are "s" (seconds), "ms"(milliseconds), "y"(years), "M"(months), "w"(weeks), "d"(days), "h"(hours), and "m" (minutes).

This time must be less than 597 hours.

[Back to TOC](#)

memc_send_timeout

syntax: *memc_send_timeout <time>*

default: 60s

context: *http, server, location*

The timeout for sending TCP requests to the memcached server, in seconds by default.

It's wise to always explicitly specify the time unit to avoid confusion. Time units supported are "s" (seconds), "ms"(milliseconds), "y"(years), "M"(months), "w"(weeks), "d"(days), "h"(hours), and "m" (minutes).

This time must be less than 597 hours.

[Back to TOC](#)

memc_read_timeout

syntax: *memc_read_timeout <time>*

default: 60s

context: *http, server, location*

The timeout for reading TCP responses from the memcached server, in seconds by default.

It's wise to always explicitly specify the time unit to avoid confusion. Time units supported are "s" (seconds), "ms"(milliseconds), "y"(years), "M"(months), "w"(weeks), "d"(days), "h"(hours), and "m" (minutes).

This time must be less than 597 hours.

[Back to TOC](#)

memc_buffer_size

syntax: *memc_buffer_size* <size>

default: 4k/8k

context: *http, server, location*

This buffer size is used for the memory buffer to hold

- the complete response for memcached commands other than `get`,
- the complete response header (i.e., the first line of the response) for the `get` memcached command.

This default size is the page size, may be 4k or 8k .

[Back to TOC](#)

memc_ignore_client_abort

syntax: *memc_ignore_client_abort* on|off

default: off

context: *location*

Determines whether the connection with a memcache server should be closed when a client closes a connection without waiting for a response.

This directive was first added in the `v0.14` release.

[Back to TOC](#)

Installation

You're recommended to install this module (as well as the Nginx core and many other goodies) via the [ngx_openresty bundle](#). See the [installation steps](#) for `ngx_openresty` .

Alternatively, you can compile this module into the standard Nginx source distribution by hand:

Grab the nginx source code from [nginx.org](#), for example, the version 1.7.2 (see [nginx compatibility](#)), and then build the source with this module:

```
wget 'http://nginx.org/download/nginx-1.7.2.tar.gz'
tar -xzvf nginx-1.7.2.tar.gz
cd nginx-1.7.2/

# Here we assume you would install you nginx under /opt/nginx/.
./configure --prefix=/opt/nginx \
    --add-module=/path/to/memc-nginx-module

make -j2
make install
```

Download the latest version of the release tarball of this module from [memc-nginx-module file list](#).

[Back to TOC](#)

For Developers

The memcached response parsers were generated by [Ragel](#). If you want to regenerate the parser's C file, i.e., `src/nginx_http_memc_response.c`, use the following command from the root of the memc module's source tree:

```
$ ragel -G2 src/nginx_http_memc_response.rl
```

[Back to TOC](#)

Compatibility

The following versions of Nginx should work with this module:

- **1.7.x** (last tested: 1.7.2)
- **1.5.x** (last tested: 1.5.12)
- **1.4.x** (last tested: 1.4.4)
- **1.2.x** (last tested: 1.2.9)
- **1.1.x** (last tested: 1.1.5)
- **1.0.x** (last tested: 1.0.10)
- **0.9.x** (last tested: 0.9.4)
- **0.8.x** (last tested: 0.8.54)
- **0.7.x >= 0.7.46** (last tested: 0.7.68)

It's worth mentioning that some 0.7.x versions older than 0.7.46 might also work, but I can't easily test them because the test suite makes extensive use of the [echo module's echo_location directive](#), which requires at least nginx 0.7.46 :)

Earlier versions of Nginx like 0.6.x and 0.5.x will *not* work.

If you find that any particular version of Nginx above 0.7.46 does not work with this module, please consider [reporting a bug](#).

[Back to TOC](#)

Community

[Back to TOC](#)

English Mailing List

The [openresty-en](#) mailing list is for English speakers.

[Back to TOC](#)

Chinese Mailing List

The [openresty](#) mailing list is for Chinese speakers.

[Back to TOC](#)

Report Bugs

Although a lot of effort has been put into testing and code tuning, there must be some serious bugs lurking somewhere in this module. So whenever you are bitten by any quirks, please don't hesitate to

1. create a ticket on the [issue tracking interface](#) provided by GitHub,
2. or send a bug report or even patches to the [nginx mailing list](#).

[Back to TOC](#)

Source Repository

Available on github at [openresty/memc-nginx-module](#).

[Back to TOC](#)

Changes

The changes of every release of this module can be obtained from the ngx_openresty bundle's change logs:

<http://openresty.org/#Changes>

[Back to TOC](#)

Test Suite

This module comes with a Perl-driven test suite. The [test cases](#) are [declarative](#) too. Thanks to the [Test::Base](#) module in the Perl world.

To run it on your side:

```
$ PATH=/path/to/your/nginx-with-memc-module:$PATH prove -r t
```

You need to terminate any Nginx processes before running the test suite if you have changed the Nginx server binary.

Either [LWP::UserAgent](#) or [IO::Socket](#) is used by the [test scaffold](#).

Because a single nginx server (by default, `localhost:1984`) is used across all the test scripts (`.t` files), it's meaningless to run the test suite in parallel by specifying `-jN` when invoking the `prove` utility.

You should also keep a memcached server listening on the `11211` port at localhost before running the test suite.

Some parts of the test suite requires modules [rewrite](#) and [echo](#) to be enabled as well when building Nginx.

[Back to TOC](#)

TODO

- add support for the memcached commands `cas`, `gets` and `stats $memc_value`.

- add support for the `noreply` option.

[Back to TOC](#)

Getting involved

You'll be very welcomed to submit patches to the [author](#) or just ask for a commit bit to the [source repository](#) on GitHub.

[Back to TOC](#)

Author

Yichun "agentzh" Zhang (章亦春) <agentzh@gmail.com>, CloudFlare Inc.

This wiki page is also maintained by the author himself, and everybody is encouraged to improve this page as well.

[Back to TOC](#)

Copyright & License

The code base is borrowed directly from the standard [memcached module](#) in the Nginx core. This part of code is copyrighted by Igor Sysoev and Nginx Inc.

Copyright (c) 2009-2013, Yichun "agentzh" Zhang (章亦春) agentzh@gmail.com, CloudFlare Inc.

This module is licensed under the terms of the BSD license.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

[Back to TOC](#)

See Also

- The original announcement email on the nginx mailing list: [ngx_memc](#): "an extended version of [ngx_memcached](#) that supports set, add, delete, and many more commands"
- My slides demonstrating various ngx_memc usage: <http://agentzh.org/misc/slides/nginx-conf-scripting/nginx-conf-scripting.html#34> (use the arrow or pageup/pagedown keys on the keyboard to swith pages)
- The latest [memcached TCP protocol](#).
- The [ngx_srcache](#) module
- The [lua-resty-memcached](#) library based on the [lua-nginx-module](#) cosocket API.
- The standard [memcached](#) module.
- The [echo module](#) for Nginx module's automated testing.
- The standard [headers](#) module and the 3rd-parth [headers-more](#) module.

