This repository Search Pull requests Issues Gist

openresty / stream-echo-nginx-module

Watch 7 Star 29 Fork 2

TCP/stream echo module for NGINX (a port of ngx_http_echo_module)

45 commits **1** branch **0** releases **1** contributor

Branch: **master** stream-echo-nginx-module / +

agentzh fixed a Microsoft C compiler warning C4702 (unreachable code). thanks… ··· Latest commit `951e2cd` a day ago

src	fixed a Microsoft C compiler warning C4702 (unreachable code). thanks…	a day ago
t	feature: implemented the "echo_discard_request" directive.	5 days ago
util	util/build.sh: added the --with-threads configure option to ensure ma…	2 days ago
.gitignore	feature: added new directives echo_read_bytes, echo_read_buffer_size,…	7 days ago
README.md	doc: added one more caveat regarding mixing with other stream modules.	2 days ago
config	initial check in.	14 days ago

Code

Issues 0

Pull requests 0

Wiki

Pulse

Graphs

HTTPS clone URL

`https://github.com`

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

README.md

NAME

ngx_stream_echo - TCP/stream echo module for NGINX (a port of the ngx_http_echo module)

Table of Contents

Version

This module is still under early development.

Synopsis

Example 1

```
# nginx.conf

stream {
    server {
        listen 1234;

        echo_send_timeout   10s;    # default to 60s
        echo "Hello, world!";
        echo I really like doing downstream TCP;
    }
}
```

```
# on the terminal

$ telnet 127.0.0.1 1234
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Hello, world!
I really like doing downstream TCP
Connection closed by foreign host.
```

Back to TOC

Example 2

```
stream {
    server {
        listen 1234;

        echo "before sleep...";
        echo_flush_wait;    # ensure that any pending output is flushed

        echo_sleep 3.1;     # sleep for 3.1 sec

        echo "after sleep...";
        echo_duplicate 3 " hello";  # repeat " hello" for 3 times
        echo;   # just to ouput a new line

    }
}
```

```
$ time telnet 127.0.0.1 1234
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
before sleep...
after sleep...
 hello hello hello
 Connection closed by foreign host.

 real   0m3.106s
 user   0m0.000s
 sys    0m0.002s
```

Back to TOC

Example 3

```
stream {
    server {
        listen 1234;

        echo_read_buffer_size 2k;
        echo_read_timeout 60s;

        echo_read_bytes 2;
        echo -n 'Got prompt: ';
        echo_request_data;
        echo;
        echo_read_line;
        echo -n "Got command: ";
        echo_request_data;
    }
}
```

```
# we type the line ">>print("hello, world")" in the telnet session below.

$ telnet 127.0.0.1 1234
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
>>print("hello, world!")
Got prompt: >>
Got command: print("hello, world!")
Connection closed by foreign host.
```

Back to TOC

Example 4

```
stream {
    server {
        listen 1999;

        # emulate a blackhole that swallows any incoming TCP
        # messages greedily like a logging service.
        # this can be used to mock up a logging service like
        # syslog-ng (TCP), which is much more efficient than
        # a typical netcat (nc) server.

        echo_discard_request;
```

```
            echo_sleep 3600;    # in sec
        }
    }
```

Back to TOC

Description

This module is a port of the handy ngx_http_echo module over the shiny new "stream" subsystem of NGINX. With this module, you can do simple custom output from constant strings directly from memory in your generic TCP (or stream-typed unix domain socket) server.

This module is particularly handy for mocking silly TCP endpoints during unit testing (like mocking a buggy and evil memcached server).

Also, this module can serve as a useful simple demo for writing NGINX stream-typed 3rd-party modules. Well, it is just a little bit more complex than a "hello world" module anyway.

Back to TOC

Directives

echo

syntax: *echo [options] <string>...*

default: *no*

context: *server*

phase: *content*

Sends string arguments joined by spaces, along with a trailing newline, out to the client.

For example,

```
stream {
    server {
        listen 1234;

        echo "Hello, world!";
        echo foo bar baz;
    }
}
```

Then connecting to the server port 1234 will immediately receive the response data

```
Hello, world!
foo bar baz
```

and then the server closes the connection right away.

When no argument is specified, *echo* emits the trailing newline alone, just like the *echo* command in shell.

one can suppress the trailing newline character in the output by using the `-n` option, as in

```
echo -n "hello, ";
echo "world";
```

Connecting to the server will receive the response data

```
hello, world
```

where the first `echo` command generates no trailing new-line due to the use of the `-n` option.

To output string values prefixed with a dash (`-`), you can specify the special `--` option to disambiguate such arguments from options. For instance,

```
echo -n -- -32+5;
```

The response is

```
-32+5
```

This command sends the data *asynchronously* to the main execution flow, that is, this command will return immediately without waiting for the output to be actually flushed into the system socket send buffers.

For slow connections the sending timeout protection is subject to the configuration of the echo_send_timeout configuration directive.

This command can be mixed with other `echo_*` commands (like echo_duplicate) freely in the same server. The module handler will run them sequentially in the same order of their appearance in the NGINX configuration file.

Back to TOC

echo_duplicate

syntax: *echo_duplicate <count> <string>*

default: *no*

context: *server*

phase: *content*

Outputs duplication of a string indicated by the second argument, using the count specified in the first argument.

For instance,

```
echo_duplicate 3 "abc";
```

will lead to the output of `"abcabcabc"`.

Underscores are allowed in the count number, just like in Perl. For example, to emit 1000,000,000 instances of `"hello, world"`:

```
echo_duplicate 1000_000_000 "hello, world";
```

The `count` argument could be zero, but not negative. The second `string` argument could be an empty string ("") likewise.

Unlike the echo directive, no trailing newline is appended to the result.

Like the echo command, this command sends the data *asynchronously* to the main execution flow, that is, this command will return immediately without waiting for the output to be actually flushed into the system socket send buffers.

For slow connections the sending timeout protection is subject to the configuration of the echo_send_timeout configuration directive.

This command can be mixed with other `echo*` commands (like echo and echo_sleep) freely in the same server. The module handler will run them sequentially in the same order of their appearance in the NGINX configuration file.

Back to TOC

echo_flush_wait

syntax: *echo_flush_wait;*

default: *no*

context: *server*

phase: *content*

Synchronously waits for all the pending output to be flushed out into the system socket send buffers. When the downstream connection is fast enough and there is no pending data, then this directive completes immediately without waiting.

The wait is a nonblocking operation. That is, it never blocks the NGINX event loop or any operating system threads.

The maximum waiting time is subject to the echo_send_timeout setting.

This command can be mixed with other `echo*` commands (like echo and echo_sleep) freely in the same server. The module handler will run them sequentially in the same order of their appearance in the NGINX configuration file.

Back to TOC

echo_sleep

syntax: *echo_sleep <seconds>*

default: *no*

context: *server*

phase: *content*

Sleeps for the time period specified by the argument, which is in seconds.

This operation is nonblocking, that is, it never blocks the NGINX event loop or any operating system threads.

The period might takes three digits after the decimal point and must be greater than 0.001 (i.e., 1ms).

An example is

```
echo_sleep 1.234;   # sleep for 1.234 sec
echo "resumed!";
```

Behind the scene, it sets up a per-request "sleep" event object, and adds a timer using that custom event to the Nginx event model and just waits for a period of time on that event.

This command can be mixed with other `echo*` commands (like echo and echo_duplicate) freely in the same server. The module handler will run them sequentially in the same order of their appearance in the NGINX configuration file.

Back to TOC

echo_send_timeout

syntax: *echo_send_timeout <time>*

default: *echo_send_timeout 60s*

context: *stream, server*

Sets the sending timeout for the downstream socket, in seconds by default.

It is 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.

If this directive is not specified, this module will use `60s` as the default.

Back to TOC

echo_read_bytes

syntax: *echo_read_bytes <size>*

default: *no*

context: *server*

Reads the request data of the specified size and append it into the "reading buffer". The size of the buffer is controlled by the echo_read_buffer_size directive. The length of data dictated in this command cannot exceed the echo_read_buffer_size setting.

For example,

```
echo_read_bytes 5;
```

reads 5 bytes of request data from the downstream connection.

On the other hand,

```
echo_read_bytes 4k;
```

reads 4KB of data.

This command would not return (until timeout) until exactly the acount of data has been read as specified.

The timeout threshold is subject to the echo_read_timeout directive.

The data read (in the "reading buffer") can later be output by the echo_request_data directive.

This command can be mixed with other `echo*` commands (like echo and echo_duplicate) freely in the same server. The module handler will run them sequentially in the same order of their appearance in the NGINX configuration file.

Back to TOC

echo_read_line

syntax: *echo_read_line;*

default: *no*

context: *server*

Reads the request data of the specified size and append it into the "reading buffer". The size of the buffer is controlled by the echo_read_buffer_size directive. The length of data read by this command cannot exceed the echo_read_buffer_size setting.

The timeout threshold is subject to the echo_read_timeout directive.

The data read (in the "reading buffer") can later be output by the echo_request_data directive.

This command can be mixed with other `echo*` commands (like echo and echo_duplicate) freely in the same server. The module handler will run them sequentially in the same order of their appearance in the NGINX configuration file.

Back to TOC

echo_request_data

syntax: *echo_request_data <size>*

default: *no*

context: *server*

Sends all the data accumulated in the "reading buffer" to the downstream connection and clears all the data in the "reading buffer".

Unlike echo or echo_duplicate, this command does not return until all the data is actually flushed into the system socket send buffer. Or in other words, this command is a synchronous operation (but still doing nonblocking I/O, of course).

This command can be mixed with other `echo*` commands (like echo and echo_duplicate) freely in the same server. The module handler will run them sequentially in the same order of their appearance in the NGINX configuration file.

Back to TOC

echo_discard_request

syntax: *echo_discard_request*

default: *no*

context: *server*

Discards any request data already pre-read in the "reading buffer" or any future incoming request data.

This command is an asynchronous operation which returns immediately without waiting for all the incoming request.

Once this command is executed, any subsequent request reading commands like echo_read_line are disallowed.

This command can be mixed with other `echo*` commands (like echo and echo_duplicate) freely in the same server. The module handler will run them sequentially in the same order of their appearance in the NGINX configuration file.

Back to TOC

echo_read_buffer_size

syntax: *echo_read_buffer_size <size>*

default: *echo_read_buffer_size 1k*

context: *stream, server*

Controls the size of the "reading buffer" used to receive downstream data via commands like echo_read_bytes.

Back to TOC

echo_read_timeout

syntax: *echo_read_timeout <time>*

default: *echo_read_timeout 60s*

context: *stream, server*

Sets the reading timeout for the downstream socket, in seconds by default. Affecting reading directives like echo_read_bytes.

It is 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.

If this directive is not specified, this module will use `60s` as the default.

Back to TOC

echo_client_error_log_level

syntax: *echo_client_error_log_level info* | *notice* | *warn* | *error*

default: *echo_client_error_log_level info*

context: *stream, server*

Specifies the error log level for client side errors (like the error that the client closes the connection prematurely). Default to `info` to avoid real-world clients from flooding the server error log files (which can be quite expensive).

Back to TOC

echo_lingering_close

syntax: *echo_lingering_close off* | *on* | *always*

default: *echo_lingering_close on*

context: *stream, server*

Controls how nginx closes client connections.

The default value `on` instructs nginx to wait for and process (read and discard) additional data from a client before fully closing a connection, but only if heuristics suggests that a client may be sending more data (like there is unprocessed pre-read data in the "reading buffer" or the socket is still ready for reading).

The value `always` will cause nginx to unconditionally wait for and process additional client data.

The value `off` tells nginx to never wait for more data and close the connection immediately. This behavior breaks the protocol and may result in interrupting RST packets sent. Thus this configuration value should not be used under normal circumstances.

How long nginx should wait is controlled by both the echo_lingering_time and echo_lingering_timeout directives.

Back to TOC

echo_lingering_time

syntax: *echo_lingering_time <time>*

default: *echo_lingering_time 30s*

context: *stream, server*

When lingering_close is in effect, this directive specifies the maximum time during which nginx will process (read and ignore) additional data coming from a client. After that, the connection will be closed, even if there will be more data.

Back to TOC

echo_lingering_timeout

syntax: *echo_lingering_timeout <time>*

default: *echo_lingering_timeout 5s*

context: *stream, server*

When lingering_close is in effect, this directive specifies the maximum waiting time between successive arrivals of client data. If data is not received during this time, the connection is closed. Otherwise, the data are read and ignored, and nginx starts waiting for more data again. The “wait-read-ignore” cycle is repeated, but no longer than specified by the lingering_time directive.

Back to TOC

Caveats

- Unlike the ngx_http_echo module, this module has no NGINX variable support since NGINX variables are not supported in the "stream" subsystem of NGINX (yet).
- The commands of this module cannot be mixed with other response-generating modules like the standard ngx_stream_proxy module in the same `server {}` block, for obvious reasons.

Back to TOC

TODO

- Implement the `echo_test_request_data` directive to test existing data read in the "reading buffer" with a literal string specified as the directive argument. This is very useful for mock-up testing.
- Implement the `echo_read_literal` directive for the combination of the command sequence `echo_read_bytes n; echo_test_request_data str;` where `n` is the length of the `str` literal string. This could be very useful for mock-up testing.
- Implement the `echo_read_until` directive to allow reading request data until seeing a terminator string literal specified as the directive argument.
- Implement the `echo_read_regex` directive to allow reading request data according to a user-supplied Perl-compatible regular expression (maybe we could use libsregex to ensure efficient streaming reading).
- Port over the "postpone_output" feature of the "http" subsystem. This could improve performance by reducing syscalls (and potentially underlying data packets too).

Back to TOC

Installation

Grab the nginx source code from nginx.org, for example, the version 1.9.3 (see nginx compatibility), and then build the source with this module:

```
wget 'http://nginx.org/download/nginx-1.9.3.tar.gz'
tar -xzvf nginx-1.9.3.tar.gz
cd nginx-1.9.3/

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

make -j2
sudo make install
```

Back to TOC

Compatibility

The following versions of Nginx should work with this module:

- **1.9.x** (last tested: 1.9.3)

NGINX versions older than 1.9.0 will *not* work due to the lack of the "stream" subsystem.

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, questions, or even patches to the OpenResty Community.

Back to TOC

Source Repository

Available on github at openresty/stream-echo-nginx-module.

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

Back to TOC

See Also

- ngx_http_echo_module
- NGINX's stream subsystem
- OpenResty

Back to TOC