openresty / set-misc-nginx-module

Various set_xxx directives added to nginx's rewrite module (md5/sha1, sql/json quoting, and many more)
http://wiki.nginx.org/NginxHttpSetMiscModule

207 commits 2 branches 42 releases 8 contributors

branch: master set-misc-nginx-module / +

doc: bumped version to 0.28.

agentzh authored 27 days ago latest commit f72abf16db

doc	doc: bumped version to 0.28.	27 days ago
src	bugfix: set_quote_sql_str: we incorrectly escaped 0x1a to \z instead …	a month ago
t	bugfix: set_quote_sql_str: we incorrectly escaped 0x1a to \z instead …	a month ago
util	bugfix: fixed build failure when --with-mail_ssl_module is specified …	4 months ago
.gitignore	updated .gitignore.	6 months ago
README.markdown	doc: bumped version to 0.28.	27 days ago
config	bugfix: fixed build failure when --with-mail_ssl_module is specified …	4 months ago
valgrind.suppress	suppressed a valgrind false positive in libdl.	11 months ago

README.markdown

Name

ngx_set_misc - Various set_xxx directives added to nginx's rewrite module (md5/sha1, sql/json quoting, and many more)

This module is not distributed with the Nginx source. See the installation instructions.

Table of Contents

Version

This document describes ngx_set_misc v0.28 released on 21 January 2015.

Synopsis

```
location /foo {
```

```
    set $a $arg_a;
    set_if_empty $a 56;

    # GET /foo?a=32 will yield $a == 32
    # while GET /foo and GET /foo?a= will
    # yeild $a == 56 here.
}

location /bar {
    set $foo "hello\n\n'\"\\";
    set_quote_sql_str $foo $foo; # for mysql

    # OR in-place editing:
    #   set_quote_sql_str $foo;

    # now $foo is: 'hello\n\n\'\"\\'
}

location /bar {
    set $foo "hello\n\n'\"\\";
    set_quote_pgsql_str $foo;  # for PostgreSQL

    # now $foo is: E'hello\n\n\'\"\\'
}

location /json {
    set $foo "hello\n\n'\"\\";
    set_quote_json_str $foo $foo;

    # OR in-place editing:
    #   set_quote_json_str $foo;

    # now $foo is: "hello\n\n'\"\\"
}

location /baz {
    set $foo "hello%20world";
    set_unescape_uri $foo $foo;
```

```
    # OR in-place editing:
    #   set_unescape_uri $foo;

    # now $foo is: hello world
}

upstream_list universe moon sun earth;
upstream moon { ... }
upstream sun { ... }
upstream earth { ... }
location /foo {
    set_hashed_upstream $backend universe $arg_id;
    drizzle_pass $backend; # used with ngx_drizzle
}

location /base32 {
    set $a 'abcde';
    set_encode_base32 $a;
    set_decode_base32 $b $a;

    # now $a == 'c5h66p35' and
    # $b == 'abcde'
}

location /base64 {
    set $a 'abcde';
    set_encode_base64 $a;
    set_decode_base64 $b $a;

    # now $a == 'YWJjZGU=' and
    # $b == 'abcde'
}

location /hex {
    set $a 'abcde';
    set_encode_hex $a;
    set_decode_hex $b $a;
```

```
    # now $a == '6162636465' and
    # $b == 'abcde'
}

# GET /sha1 yields the output
#   aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
location /sha1 {
    set_sha1 $a hello;
    echo $a;
}

# ditto
location /sha1 {
    set $a hello;
    set_sha1 $a;
    echo $a;
}

# GET /today yields the date of today in local time using format 'yyyy-mm-dd'
location /today {
    set_local_today $today;
    echo $today;
}

# GET /signature yields the hmac-sha-1 signature
# given a secret and a string to sign
# this example yields the base64 encoded singature which is
# "HkADYytcoQQzqbjQX33k/ZBB/DQ="
location /signature {
    set $secret_key 'secret-key';
    set $string_to_sign "some-string-to-sign";
    set_hmac_sha1 $signature $secret_key $string_to_sign;
    set_encode_base64 $signature $signature;
    echo $signature;
}

location = /rand {
```

```
    set $from 3;
    set $to 15;
    set_random $rand $from $to;

    # or write directly
    #   set_random $rand 3 15;

    echo $rand;  # will print a random integer in the range [3, 15]
}
```

Description

This module extends the standard HttpRewriteModule's directive set to provide more functionalities like URI escaping and unescaping, JSON quoting, Hexadecimal/MD5/SHA1/Base32/Base64 digest encoding and decoding, random number generator, and more!

Every directive provided by this module can be mixed freely with other ngx_http_rewrite_module's directives, like if and set. (Thanks to the Nginx Devel Kit!)

Back to TOC

Directives

Back to TOC

set_if_empty

syntax: *set_if_empty $dst <src>*

default: *no*

context: *location, location if*

phase: *rewrite*

Assign the value of the argument `<src>` if and only if variable `$dst` is empty (i.e., not found or has an empty string value).

In the following example,

```
set $a 32;
set_if_empty $a 56;
```

the variable `$dst` will take the value 32 at last. But in the sample

```
set $a '';
set $value "hello, world"
set_if_empty $a $value;
```

`$a` will take the value `"hello, world"` at last.

Back to TOC

set_quote_sql_str

syntax: *set_quote_sql_str $dst <src>*

syntax: *set_quote_sql_str $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

When taking two arguments, this directive will quote the value of the second argument `<src>` by MySQL's string value quoting rule and assign the result into the first argument, variable `$dst`. For example,

```
location /test {
    set $value "hello\n\r'\"\\";
    set_quote_sql_str $quoted $value;

    echo $quoted;
}
```

Then request `GET /test` will yield the following output

```
'hello\n\r\'\"\\'
```

Please note that we're using echo-nginx-module's echo directive here to output values of nginx variables directly.

When taking a single argument, this directive will do in-place modification of the argument variable. For example,

```
location /test {
    set $value "hello\n\r'\"\\";
    set_quote_sql_str $value;

    echo $value;
```

```
    }
```

then request `GET /test` will give exactly the same output as the previous example.

This directive is usually used to prevent SQL injection.

This directive can be invoked by lua-nginx-module's ndk.set_var.DIRECTIVE interface and array-var-nginx-module's array_map_op directive.

Back to TOC

set_quote_pgsql_str

syntax: *set_quote_pgsql_str $dst <src>*

syntax: *set_quote_pgsql_str $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

Very much like set_quote_sql_str, but with PostgreSQL quoting rules for SQL string literals.

Back to TOC

set_quote_json_str

syntax: *set_quote_json_str $dst <src>*

syntax: *set_quote_json_str $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

When taking two arguments, this directive will quote the value of the second argument `<src>` by JSON string value quoting rule and assign the result into the first argument, variable `$dst`. For example,

```
location /test {
    set $value "hello\n\r'\"\\";
    set_quote_json_str $quoted $value;

    echo $quoted;
}
```

Then request `GET /test` will yield the following output

```
"hello\n\r'\"\\"
```

Please note that we're using echo-nginx-module's echo directive here to output values of nginx variables directly.

When taking a single argument, this directive will do in-place modification of the argument variable. For example,

```
location /test {
```

```
        set $value "hello\n\r'\"\\";
        set_quote_json_str $value;

        echo $value;
    }
```

then request `GET /test` will give exactly the same output as the previous example.

This directive can be invoked by lua-nginx-module's ndk.set_var.DIRECTIVE interface and array-var-nginx-module's array_map_op directive.

Back to TOC

set_unescape_uri

syntax: *set_unescape_uri $dst <src>*

syntax: *set_unescape_uri $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

When taking two arguments, this directive will unescape the value of the second argument `<src>` as a URI component and assign the result into the first argument, variable `$dst`. For example,

```
location /test {
    set_unescape_uri $key $arg_key;
    echo $key;
```

```
}
```

Then request `GET /test?key=hello+world%21` will yield the following output

```
hello world!
```

The nginx standard $arg_PARAMETER variable holds the raw (escaped) value of the URI parameter. So we need the `set_unescape_uri` directive to unescape it first.

Please note that we're using echo-nginx-module's echo directive here to output values of nginx variables directly.

When taking a single argument, this directive will do in-place modification of the argument variable. For example,

```
location /test {
    set $key $arg_key;
    set_unescape_uri $key;

    echo $key;
}
```

then request `GET /test?key=hello+world%21` will give exactly the same output as the previous example.

This directive can be invoked by lua-nginx-module's ndk.set_var.DIRECTIVE interface and array-var-nginx-module's array_map_op directive.

Back to TOC

set_escape_uri

syntax: *set_escape_uri $dst <src>*

syntax: *set_escape_uri $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

Very much like the set_unescape_uri directive, but does the conversion the other way around, i.e., URL component escaping.

Back to TOC

set_hashed_upstream

syntax: *set_hashed_upstream $dst <upstream_list_name> <src>*

default: *no*

context: *location, location if*

phase: *rewrite*

Hashes the string argument `<src>` into one of the upstream name included in the upstream list named `<upstream_list_name>`. The hash function being used is simple modulo.

Here's an example,

```
upstream moon { ... }
```

```
upstream sun { ... }
upstream earth { ... }

upstream_list universe moon sun earth;

location /test {
    set_unescape_uri $key $arg_key;
    set $list_name universe;
    set_hashed_upstream $backend $list_name $key;

    echo $backend;
}
```

Then `GET /test?key=blah` will output either "moon", "sun", or "earth", depending on the actual value of the `key` query argument.

This directive is usually used to compute an nginx variable to be passed to memc-nginx-module's memc_pass directive, redis2-nginx-module's [[HttpRedis2Module#redis2_pass]] directive, and ngx_http_proxy_module's proxy_pass directive, among others.

Back to TOC

set_encode_base32

syntax: *set_encode_base32 $dst <src>*

syntax: *set_encode_base32 $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

When taking two arguments, this directive will encode the value of the second argument `<src>` to its base32(hex) digest and assign the result into the first argument, variable `$dst`. For example,

```
location /test {
    set $raw "abcde";
    set_encode_base32 $digest $raw;

    echo $digest;
}
```

Then request `GET /test` will yield the following output

```
c5h66p35
```

Please note that we're using echo-nginx-module's echo directive here to output values of nginx variables directly.

RFC forces the `[A-Z2-7]` RFC-3548 compliant encoding, but we are using the "base32hex" encoding (`[0-9a-v]`) by default. The set_base32_alphabet directive (first introduced in `v0.28`) allows you to change the alphabet used for encoding/decoding so RFC-3548 compliant encoding is still possible by custom configurations.

By default, the `=` character is used to pad the left-over bytes due to alignment. But the padding behavior can be completely disabled by setting set_base32_padding `off`.

When taking a single argument, this directive will do in-place modification of the argument variable. For example,

```
location /test {
```

```
    set $value "abcde";
    set_encode_base32 $value;

    echo $value;
}
```

then request `GET /test` will give exactly the same output as the previous example.

This directive can be invoked by lua-nginx-module's ndk.set_var.DIRECTIVE interface and array-var-nginx-module's array_map_op directive.

Back to TOC

set_base32_padding

syntax: *set_base32_padding on|off*

default: *on*

context: *http, server, server if, location, location if*

phase: *no*

This directive can control whether to pad left-over bytes with the "=" character when encoding a base32 digest by the set_encode_base32 directive.

This directive was first introduced in `v0.28`. If you use earlier versions of this module, then you should use set_misc_base32_padding instead.

Back to TOC

set_misc_base32_padding

syntax: *set_misc_base32_padding on|off*

default: *on*

context: *http, server, server if, location, location if*

phase: *no*

This directive has been deprecated since `v0.28`. Use set_base32_padding instead if you are using `v0.28+`.

Back to TOC

set_base32_alphabet

syntax: *set_base32_alphabet <alphabet>*

default: *"0123456789abcdefghijklmnopqrstuv"*

context: *http, server, server if, location, location if*

phase: *no*

This directive controls the alphabet used for encoding/decoding a base32 digest. It accepts a string containing the desired alphabet like "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567" for standard alphabet.

Extended (base32hex) alphabet is used by default.

This directive was first introduced in `v0.28`.

Back to TOC

set_decode_base32

syntax: *set_decode_base32 $dst <src>*

syntax: *set_decode_base32 $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

Similar to the set_encode_base32 directive, but does exactly the the opposite operation, .i.e, decoding a base32(hex) digest into its original form.

Back to TOC

set_encode_base64

syntax: *set_encode_base64 $dst <src>*

syntax: *set_encode_base64 $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

When taking two arguments, this directive will encode the value of the second argument `<src>` to its base64 digest and assign the result into the first argument, variable `$dst`. For example,

```
location /test {
    set $raw "abcde";
    set_encode_base64 $digest $raw;

    echo $digest;
}
```

Then request `GET /test` will yield the following output

```
YWJjZGU=
```

Please note that we're using echo-nginx-module's echo directive here to output values of nginx variables directly.

When taking a single argument, this directive will do in-place modification of the argument variable. For example,

```
location /test {
    set $value "abcde";
    set_encode_base64 $value;

    echo $value;
}
```

then request `GET /test` will give exactly the same output as the previous example.

This directive can be invoked by lua-nginx-module's ndk.set_var.DIRECTIVE interface and array-var-nginx-module's array_map_op directive.

Back to TOC

set_decode_base64

syntax: *set_decode_base64 $dst <src>*

syntax: *set_decode_base64 $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

Similar to the set_encode_base64 directive, but does exactly the the opposite operation, .i.e, decoding a base64 digest into its original form.

Back to TOC

set_encode_hex

syntax: *set_encode_hex $dst <src>*

syntax: *set_encode_hex $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

When taking two arguments, this directive will encode the value of the second argument `<src>` to its hexadecimal digest and assign the result into the first argument, variable `$dst`. For example,

```
location /test {
    set $raw "章亦春";
    set_encode_hex $digest $raw;

    echo $digest;
}
```

Then request `GET /test` will yield the following output

```
e7aba0e4baa6e698a5
```

Please note that we're using echo-nginx-module's echo directive here to output values of nginx variables directly.

When taking a single argument, this directive will do in-place modification of the argument variable. For example,

```
location /test {
    set $value "章亦春";
    set_encode_hex $value;

    echo $value;
}
```

then request `GET /test` will give exactly the same output as the previous example.

This directive can be invoked by lua-nginx-module's ndk.set_var.DIRECTIVE interface and array-

var-nginx-module's array_map_op directive.

Back to TOC

set_decode_hex

syntax: *set_decode_hex $dst <src>*

syntax: *set_decode_hex $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

Similar to the set_encode_hex directive, but does exactly the the opposite operation, .i.e, decoding a hexadecimal digest into its original form.

Back to TOC

set_sha1

syntax: *set_sha1 $dst <src>*

syntax: *set_sha1 $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

When taking two arguments, this directive will encode the value of the second argument `<src>` to its SHA-1 digest and assign the result into the first argument, variable `$dst`. The hexadecimal form of the `SHA-1` digest will be generated automatically, use set_decode_hex to decode the result if you want the binary form of the `SHA-1` digest.

For example,

```
location /test {
    set $raw "hello";
    set_sha1 $digest $raw;

    echo $digest;
}
```

Then request `GET /test` will yield the following output

```
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
```

Please note that we're using echo-nginx-module's echo directive here to output values of nginx variables directly.

When taking a single argument, this directive will do in-place modification of the argument variable. For example,

```
location /test {
    set $value "hello";
    set_sha1 $value;
```

```
    echo $value;
}
```

then request `GET /test` will give exactly the same output as the previous example.

This directive can be invoked by lua-nginx-module's ndk.set_var.DIRECTIVE interface and array-var-nginx-module's array_map_op directive.

Back to TOC

set_md5

syntax: *set_md5 $dst <src>*

syntax: *set_md5 $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

category: *ndk_set_var_value*

When taking two arguments, this directive will encode the value of the second argument `<src>` to its MD5 digest and assign the result into the first argument, variable `$dst`. The hexadecimal form of the `MD5` digest will be generated automatically, use set_decode_hex to decode the result if you want the binary form of the `MD5` digest.

For example,

```
location /test {
```

```
    set $raw "hello";
    set_md5 $digest $raw;

    echo $digest;
}
```

Then request `GET /test` will yield the following output

```
5d41402abc4b2a76b9719d911017c592
```

Please note that we're using echo-nginx-module's echo directive here to output values of nginx variables directly.

When taking a single argument, this directive will do in-place modification of the argument variable. For example,

```
location /test {
    set $value "hello";
    set_md5 $value;

    echo $value;
}
```

then request `GET /test` will give exactly the same output as the previous example.

This directive can be invoked by lua-nginx-module's ndk.set_var.DIRECTIVE interface and array-var-nginx-module's array_map_op directive.

Back to TOC

set_hmac_sha1

syntax: *set_hmac_sha1 $dst <secret_key> <src>*

syntax: *set_hmac_sha1 $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

Computes the HMAC-SHA1 digest of the argument `<src>` and assigns the result into the argument variable `$dst` with the secret key `<secret_key>`.

The raw binary form of the `HMAC-SHA1` digest will be generated, use set_encode_base64, for example, to encode the result to a textual representation if desired.

For example,

```
location /test {
    set $secret 'thisisverysecretstuff';
    set $string_to_sign 'some string we want to sign';
    set_hmac_sha1 $signature $secret $string_to_sign;
    set_encode_base64 $signature $signature;
    echo $signature;
}
```

Then request `GET /test` will yield the following output

```
R/pvxzHC4NLtj7S+kXFg/NePTmk=
```

Please note that we're using echo-nginx-module's echo directive here to output values of nginx variables directly.

This directive requires the OpenSSL library enabled in your Nignx build (usually by passing the `--with-http_ssl_module` option to the `./configure` script).

Back to TOC

set_random

syntax: *set_random $res <from> <to>*

default: *no*

context: *location, location if*

phase: *rewrite*

Generates a (pseudo) random number (in textual form) within the range `[<$from>, <$to>]` (inclusive).

Only non-negative numbers are allowed for the `<from>` and `<to>` arguments.

When `<from>` is greater than `<to>`, their values will be exchanged accordingly.

For instance,

```
location /test {
    set $from 5;
    set $to 7;
    set_random $res $from $to;

    echo $res;
}
```

then request `GET /test` will output a number between 5 and 7 (i.e., among 5, 6, 7).

For now, there's no way to configure a custom random generator seed.

Behind the scene, it makes use of the standard C function `rand()`.

This directive was first introduced in the `v0.22rc1` release.

See also set_secure_random_alphanum and set_secure_random_lcalpha.

Back to TOC

set_secure_random_alphanum

syntax: *set_secure_random_alphanum $res <length>*

default: *no*

context: *location, location if*

phase: *rewrite*

Generates a cryptographically-strong random string `<length>` characters long with the alphabet `[a-zA-Z0-9]`.

`<length>` may be between 1 and 64, inclusive.

For instance,

```
location /test {
    set_secure_random_alphanum $res 32;

    echo $res;
}
```

then request `GET /test` will output a string like `ivVVRP2DGaAqDmdf3Rv4ZDJ7k0gOfASz`.

This functionality depends on the presence of the `/dev/urandom` device, available on most UNIX-like systems.

See also set_secure_random_lcalpha and set_random.

This directive was first introduced in the `v0.22rc8` release.

Back to TOC

set_secure_random_lcalpha

syntax: *set_secure_random_lcalpha $res <length>*

default: *no*

context: *location, location if*

phase: *rewrite*

Generates a cryptographically-strong random string `<length>` characters long with the alphabet `[a-z]`.

`<length>` may be between 1 and 64, inclusive.

For instance,

```
location /test {
    set_secure_random_lcalpha $res 32;

    echo $res;
}
```

then request `GET /test` will output a string like `kcuxcddktffsippuekhshdaclaquiusj`.

This functionality depends on the presence of the `/dev/urandom` device, available on most UNIX-like systems.

This directive was first introduced in the `v0.22rc8` release.

See also set_secure_random_alphanum and set_random.

Back to TOC

set_rotate

syntax: *set_rotate $value <from> <to>*

default: *no*

context: *location, location if*

phase: *rewrite*

Increments `$value` but keeps it in range from `<from>` to `<to>`. If `$value` is greater than `<to>` or less than `<from>` is will be set to `<from>` value.

The current value after running this directive will always be saved on a per-location basis. And the this saved value will be used for incrementation when the `$value` is not initialized or has a bad value.

Only non-negative numbers are allowed for the `<from>` and `<to>` arguments.

When `<from>` is greater than `<to>`, their values will be exchanged accordingly.

For instance,

```
location /rotate {
    default_type text/plain;
```

```
    set $counter $cookie_counter;
    set_rotate $counter 1 5;
    echo $counter;
    add_header Set-Cookie counter=$counter;
}
```

then request `GET /rotate` will output next number between 1 and 5 (i.e., 1, 2, 3, 4, 5) on each refresh of the page. This directive may be userful for banner rotation purposes.

Another example is to use server-side value persistence to do simple round-robin:

```
location /rotate {
    default_type text/plain;
    set_rotate $counter 0 3;
    echo $counter;
}
```

And accessing `/rotate` will also output integer sequence 0, 1, 2, 3, 0, 1, 2, 3, and so on.

This directive was first introduced in the `v0.22rc7` release.

Back to TOC

set_local_today

syntax: *set_local_today $dst*

default: *no*

context: *location, location if*

phase: *rewrite*

Set today's date ("yyyy-mm-dd") in localtime to the argument variable `$dst`.

Here's an example,

```
location /today {
    set_local_today $today;
    echo $today;
}
```

then request `GET /today` will output something like

```
2011-08-16
```

and year, the actual date you get here will vary every day ;)

Behind the scene, this directive utilizes the `ngx_time` API in the Nginx core, so usually no syscall is involved due to the time caching mechanism in the Nginx core.

Back to TOC

set_formatted_gmt_time

syntax: *set_formatted_gmt_time $res <time-format>*

default: *no*

context: *location, location if*

phase: *rewrite*

Set a formatted GMT time to variable `$res` (as the first argument) using the format string in the second argument.

All the conversion specification notations in the standard C function `strftime` are supported, like `%Y` (for 4-digit years) and `%M` (for minutes in decimal). See http://linux.die.net/man/3/strftime for a complete list of conversion specification symbols.

Below is an example:

```
location = /t {
    set_formatted_gmt_time $timestr "%a %b %e %H:%M:%S %Y GMT";
    echo $timestr;
}
```

Accessing `/t` yields the output

```
Fri Dec 13 15:34:37 2013 GMT
```

This directive was first added in the `0.23` release.

See also set_formatted_local_time.

Back to TOC

set_formatted_local_time

syntax: *set_formatted_local_time $res <time-format>*

default: *no*

context: *location, location if*

phase: *rewrite*

Set a formatted local time to variable `$res` (as the first argument) using the format string in the

second argument.

All the conversion specification notations in the standard C function `strftime` are supported, like `%Y` (for 4-digit years) and `%M` (for minutes in decimal). See http://linux.die.net/man/3/strftime for a complete list of conversion specification symbols.

Below is an example:

```
location = /t {
    set_formatted_local_time $timestr "%a %b %e %H:%M:%S %Y %Z";
    echo $timestr;
}
```

Accessing `/t` yields the output

```
Fri Dec 13 15:42:15 2013 PST
```

This directive was first added in the `0.23` release.

See also set_formatted_gmt_time.

Back to TOC

Caveats

Do not use $arg_PARAMETER, $cookie_COOKIE, $http_HEADER or other special variables defined in the Nginx core module as the target variable in this module's directives. For instance,

```
set_if_empty $arg_user 'foo';  # DO NOT USE THIS!
```

may lead to segmentation faults.

Back to TOC

Installation

This module is included and enabled by default in the ngx_openresty bundle. If you want to install this module manually with your own Nginx source tarball, then follow the steps below:

Grab the nginx source code from nginx.org, for example, the version 1.7.7 (see nginx compatibility), and then build the source with this module:

```
wget 'http://nginx.org/download/nginx-1.7.7.tar.gz'
tar -xzvf nginx-1.7.7.tar.gz
cd nginx-1.7.7/

# Here we assume you would install you nginx under /opt/nginx/.
./configure --prefix=/opt/nginx \
    --with-http_ssl_module \
    --add-module=/path/to/ngx_devel_kit \
    --add-module=/path/to/set-misc-nginx-module

make -j2
make install
```

Download the latest version of the release tarball of this module from set-misc-nginx-module file list, and the latest tarball for ngx_devel_kit from its file list.

Also, this module is included and enabled by default in the ngx_openresty bundle.

Back to TOC

Compatibility

The following versions of Nginx should work with this module:

- **1.7.x** (last tested: 1.7.7)
- **1.6.x**
- **1.5.x** (last tested: 1.5.8)
- **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: 1.0.15)
- **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)

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

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. send a bug report or even patches to the openresty-en mailing list,
2. or create a ticket on the issue tracking interface provided by GitHub.

Back to TOC

Source Repository

Available on github at openresty/set-misc-nginx-module.

Back to TOC

Changes

The change logs for 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::Nginx module in the Perl world.

To run it on your side:

```
$ PATH=/path/to/your/nginx-with-set-misc-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.

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.

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 Zhang (agentzh) <*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

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.

Back to TOC

See Also

- Nginx Devel Kit
- The ngx_openresty bundle