



Jekyll theme for documentation — mydoc product

version 6.0

Last generated: April 13, 2021



Company
logo

© 2021 OpenResty 中文. This is a boilerplate copyright statement... All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

Table of Contents

Overview

Get started	3
Introduction	202
Supported features	203
About the theme author	208
Support.....	209

Release Notes

6.0 Release notes	210
5.0 Release notes	212

Installation

About Ruby, Gems, Bundler, etc.	214
Install Jekyll on Mac	222
Install Jekyll on Windows	227

Authoring

Pages	230
Posts	237
Lists	239
Conditional logic.....	243
Content reuse.....	248
Collections.....	250
WebStorm editor tips	252
Atom editor tips.....	256

Navigation

Sidebar navigation.....	257
YAML tutorial in the context of Jekyll.....	260
Tags.....	271
Series.....	277

Formatting

Tooltips.....	280
Alerts	281

Icons.....	289
Images.....	296
Code samples	301
Labels	302
Links	303
Navtabs	304
Tables.....	308
Syntax highlighting	312
Workflow maps	315

Handling reviews

Commenting on files	319
---------------------------	-----

Publishing

Build arguments	322
Themes.....	325
Generating PDFs	326
Help APIs and UI tooltips	339
Search configuration	351
iTerm profiles.....	355
Pushing builds to server	357
Publishing on Github Pages.....	358

Special layouts

Knowledge-base layout.....	361
Glossary layout.....	364
FAQ layout.....	367
Shuffle layout.....	368

Troubleshooting

Troubleshooting	371
-----------------------	-----

nginx-lua-module-zh-wiki

Summary: nginx-lua-module-zh-wiki

Name

ngx_http_lua_module - 嵌入强有力的 Lua 到 Nginx HTTP 服务中。

该模块不是随着 *Nginx* 源码发行。更多请看 [安装说明 \(page 11\)](#)。

Table of Contents

- [Name \(page 3\)](#)
- [Status \(page 4\)](#)
- [Version \(page 4\)](#)
- [Synopsis \(page 5\)](#)
- [Description \(page 8\)](#)
- [Typical Uses \(page 9\)](#)
- [Nginx Compatibility \(page 10\)](#)
- [Installation \(page 11\)](#)
 - [Building as a dynamic module \(page 12\)](#)
 - [C Macro Configurations \(page 13\)](#)
 - [Installation on Ubuntu 11.10 \(page 13\)](#)
- [Community \(page 14\)](#)
 - [English Mailing List \(page 0\)](#)
 - [Chinese Mailing List \(page 0\)](#)
- [Code Repository \(page 14\)](#)
- [Bugs and Patches \(page 14\)](#)
- [Lua/LuaJIT 字节码 support \(page 0\)](#)
- [System Environment Variable Support \(page 16\)](#)
- [HTTP 1.0 support \(page 16\)](#)

- [Statically Linking Pure Lua Modules \(page 16\)](#)
- [Data Sharing within an Nginx Worker \(page 18\)](#)
- [Known Issues \(page 20\)](#)
 - [TCP socket connect operation issues \(page 0\)](#)
 - [Lua Coroutine Yielding/Resuming \(page 0\)](#)
 - [Lua Variable Scope \(page 20\)](#)
 - [Locations Configured by Subrequest Directives of Other Modules \(page 22\)](#)
 - [Cosockets Not Available Everywhere \(page 22\)](#)
 - [Special Escaping Sequences \(page 23\)](#)
 - [Mixing with SSI Not Supported \(page 25\)](#)
 - [SPDY Mode Not Fully Supported \(page 25\)](#)
 - [Missing data on short circuited requests \(page 25\)](#)
- [TODO \(page 26\)](#)
- [Changes \(page 27\)](#)
- [Test Suite \(page 27\)](#)
- [Copyright and License \(page 28\)](#)
- [See Also \(page 29\)](#)
- [Directives \(page 30\)](#)
- [Nginx API for Lua \(page 77\)](#)
- [Obsolete Sections \(page 201\)](#)
 - [Special PCRE Sequences \(page 201\)](#)

Status

生产版本可用

Version

该文档描述的 `ngx_lua v0.10.7` 是2016年11月4号发布。

Synopsis

```

# 设置纯 Lua 扩展库的搜寻路径(';;' 是默认路径):
lua_package_path '/foo/bar/?.lua;/blah/?.lua;;';

# 设置 C 编写的 Lua 扩展模块的搜寻路径(也可以用 ';;'):
lua_package_cpath '/bar/baz/?.so;/blah/blah/?.so;;';

server {
    location /lua_content {
        # 通过 default_type 设置默认的 MIME 类型:
        default_type 'text/plain';

        content_by_lua_block {
            ngx.say('Hello,world!')
        }
    }

    location /nginx_var {
        # 通过 default_type 设置默认的 MIME 类型:
        default_type 'text/plain';

        # 试试访问 /nginx_var?a=hello,world
        content_by_lua_block {
            ngx.say(ngx.var.arg_a)
        }
    }

    location /request_body {
        client_max_body_size 50k;
        client_body_buffer_size 50k;

        content_by_lua_block {
            ngx.req.read_body() -- explicitly read the req bo
dy

            local data = ngx.req.get_body_data()
            if data then
                ngx.say("body data:")
                ngx.print(data)
                return
            end

            -- body may get buffered in a temp file:
            local file = ngx.req.get_body_file()
            if file then
                ngx.say("body is in file ", file)
            else

```

```

        ngx.say("no body found")
    end
}

# 在子请求中直接发起 Lua 非阻塞 I/O 调用
# (其实, 更好的方式是使用 cosockets)
location /lua {
    # 通过 default_type 设置默认的 MIME 类型:
    default_type 'text/plain';

    content_by_lua_block {
        local res = ngx.location.capture("/some_other_location")

        if res then
            ngx.say("status: ", res.status)
            ngx.say("body:")
            ngx.print(res.body)
        end
    }
}

location = /foo {
    rewrite_by_lua_block {
        res = ngx.location.capture("/memc",
            { args = { cmd = "incr", key = ngx.var.uri } }
        )
    }

    proxy_pass http://blah.blah.com;
}

location = /mixed {
    rewrite_by_lua_file /path/to/rewrite.lua;
    access_by_lua_file /path/to/access.lua;
    content_by_lua_file /path/to/content.lua;
}

# 在代码中使用 Nginx 变量
# 注意: Nginx 变量的内容一定要做仔细的过滤, 否则会有很大的安全风险
location ~ ^/app/([-_a-zA-Z0-9/]+) {
    set $path $1;
    content_by_lua_file /path/to/lua/app/root/$path.lua;
}

```

```
location / {
    lua_need_request_body on;

    client_max_body_size 100k;
    client_body_buffer_size 100k;

    access_by_lua_block {
        -- 检测客户端 IP 地址是否在我们的黑名单中
        if ngx.var.remote_addr == "132.5.72.3" then
            ngx.exit(ngx.HTTP_FORBIDDEN)
        end

        -- 检测客户端 URI 数据是否包含禁用词汇
        if ngx.var.uri and
            string.match(ngx.var.request_body, "evil")
        then
            return ngx.redirect("/terms_of_use.html")
        end

        -- tests passed
    }

    # proxy_pass/fastcgi_pass/etc settings
}
}
```

[返回目录 \(page 3\)](#)

Description

该模块通过标准 Lua5.1 解释器或 [LuaJIT 2.0/2.1](#)，把 Lua 嵌入到 Nginx 里面，并利用 Nginx 子请求，把强大的 Lua 线程（Lua 协程）混合到 Nginx 的事件模型中。

与 [Apache's mod_lua](#)、[Lighttpd's mod_magnet](#) 不同的是，只要使用这个模块提供的 [Nginx API for Lua \(page 77\)](#) 来处理请求上游服务，该模块的 Lua 代码被执行在网络上 100% 非阻塞的。其中上游请求服务有：MySQL、PostgreSQL、Memcached、Redis 或 upstream HTTP web 服务等。

至少下面这些 Lua 库、Nginx 模块是可以与 ngx_lua 模块配合使用的：

- [lua-resty-memcached](#)
- [lua-resty-mysql](#)
- [lua-resty-redis](#)

- [lua-resty-dns](#)
- [lua-resty-upload](#)
- [lua-resty-websocket](#)
- [lua-resty-lock](#)
- [lua-resty-logger-socket](#)
- [lua-resty-lrucache](#)
- [lua-resty-string](#)
- [ngx_memc](#)
- [ngx_postgres](#)
- [ngx_redis2](#)
- [ngx_redis](#)
- [ngx_proxy](#)
- [ngx_fastcgi](#)

几乎所有的 Nginx 模块都可以通过 [ngx.location.capture](#) (page 96) 或 [ngx.location.capture_multi](#) (page 103) 与 ngx_lua 模块完成调用，但推荐使用类似 `lua-resty-*` 库，而不是子请求访问 Nginx 上游模块，因为前者更加灵活并且内存效率更高。

在单个 Nginx worker 中，标准 Lua 或 LuaJIT 的实例在所有请求中是共享使用的，但每个请求上下文是通过轻量的 Lua 协程做到隔离的。

在 Nginx worker 进程中加载 Lua 模块，坚持小内存的使用，即使在重负载下依然如此。

该模块是 Nginx 的 HTTP 子系统插件，所以它只能对 HTTP 环境的下游进行对话（例如：HTTP 0.9/1.0/1.1/2.0, WebSockets等）。

如果你想获得通用的 TCP 下游客户端对话能力，这时应使用 [ngx_stream_lua](#) 模块，同样它也是兼容 Lua API 的。

[返回目录](#) (page 3)

Typical Uses

列举部分：

- 在 Lua 中揉和处理各种不同的 Nginx 上游输出（proxy, drizzle, postgres, redis, memcached等）
- 在请求真正到达上游服务之前，Lua 可以随心所欲的做复杂访问控制和安全

检测

- 随心所欲的操控响应头里面的信息（通过 Lua）
- 从外部存储服务（比如 redis, memcached, mysql, postgresql）中获取后端信息，并用这些信息来实时选择哪一个后端来完成业务访问
- 在内容 handler 中随意编写复杂的 web 应用，使用同步非阻塞的方式，访问后端数据库和其他存储
- 在 rewrite 阶段，通过 Lua 完成非常复杂的 URL dispatch
- 用 Lua 可以为 Nginx 子请求和任意 location，实现高级缓存机制

本模块会把你带入一个拥有无限可能的服务端开发新世界，你可以把 Nginx 的各种功能进行自由拼接，更重要的是，开发门槛并不高，这一切都是用强大轻巧的 Lua 语言来操控。

本模块的脚本有充分的灵活性，并且性能和原生 C 语言编程相比毫不逊色，无论是 CPU 时间还是内存占用方面。当然这里需要启用 LuaJIT 2.x。

其他脚本语言的实现通常很难达到类似性能。

Lua state（Lua VM instance）会被共享给单个 nginx worker 内所有的请求，从而达到最小化内存消耗。

[返回目录 \(page 3\)](#)

Nginx Compatibility

最新模块版本和 Nginx 的兼容列表：

- 1.11.x (最后测试: 1.11.2)
- 1.10.x
- 1.9.x (最后测试: 1.9.15)
- 1.8.x
- 1.7.x (最后测试: 1.7.10)
- 1.6.x

比 Nginx 1.6.0 更老的版本 不再提供支持。

[返回目录 \(page 3\)](#)

Installation

强烈推荐使用 [OpenResty](#) 安装包，它包含了 Nginx, ngx_lua, LuaJIT 2.0/2.1 (或者可选的标准 Lua 5.1解释器)，还包含很多强劲、好用的 Nginx 模块。使用一个简单的命令就可以完成基础安装：`./configure --with-luajit && make && make install`。

当然，ngx_lua 也可以手动的编译到 Nginx 中：

1. 安装LuaJIT 2.0 或 2.1 (推荐) 或 Lua 5.1 (Lua 5.2 暂时还 不支持)。LuaJIT 可从 [The LuaJIT project 站点](#) 获取，Lua 5.1可从 [Lua project 站点](#) 获取。
2. 下载最新版本的 ngx_devel_kit (NDK)开发模块 [这里](#) 。
3. 下载最新版本的 ngx_lua [这里](#) 。
4. 下载最新版本的 Nginx [这里](#) (查看 [Nginx 兼容列表 \(page 10\)](#))。

源码编译本模块：

```
wget 'http://nginx.org/download/nginx-1.11.2.tar.gz'
tar -xzf nginx-1.11.2.tar.gz
cd nginx-1.11.2/

# tell nginx's build system where to find LuaJIT 2.0:
export LUAJIT_LIB=/path/to/luajit/lib
export LUAJIT_INC=/path/to/luajit/include/luajit-2.0

# tell nginx's build system where to find LuaJIT 2.1:
export LUAJIT_LIB=/path/to/luajit/lib
export LUAJIT_INC=/path/to/luajit/include/luajit-2.1

# or tell where to find Lua if using Lua instead:
#export LUA_LIB=/path/to/lua/lib
#export LUA_INC=/path/to/lua/include

# Here we assume Nginx is to be installed under /opt/nginx/.
./configure --prefix=/opt/nginx \
    --with-ld-opt="-Wl,-rpath,/path/to/luajit-or-lua/lib" \
    --add-module=/path/to/nginx_devel_kit \
    --add-module=/path/to/lua-nginx-module

make -j2
make install
```

[返回目录 \(page 3\)](#)

Building as a dynamic module

从 NGINX 1.9.11 开始，你也能编译动态模块了，使用 `--add-dynamic-module=PATH` 选项替代 `./configure` 命令行的 `--add-module=PATH`。然后就能在 `nginx.conf` 配置中通过 `load_module` 指令完成模块加载，例如：

```
load_module /path/to/modules/ndk_http_module.so; # assuming ND
K is built as a dynamic module too
load_module /path/to/modules/nginx_http_lua_module.so;
```

[返回目录 \(page 3\)](#)

C Macro Configurations

通过 OpenResty 或者 Nginx 内核方式构建该模块，你可以定义下面的 C 宏定义作为可选项提供给 C 编译器：

- **NGX_LUA_USE_ASSERT** 声明后，将在 ngx_lua C 代码中开启断言。推荐用在调试或者测试版本中。启用后，它会引入额外一些（小的）运行时开销。在 **v0.9.10** 版本中首次引入此选项。
- **NGX_LUA_ABORT_AT_PANIC** 当 Lua/LuaJIT 虚拟机出现 panic 错误时，ngx_lua 默认会让当前的工作进程优雅退出。通过指定这个宏定义，ngx_lua 将立即终止当前的 Nginx 工作进程（通常会生成一个 core dump 文件）。这个选项主要用来调试虚拟机的 panic 错误。在 **v0.9.8** 版本中首次引入此选项。
- **NGX_LUA_NO_FFI_API** 去除 Nginx 中 FFI-based Lua API 需要的纯 C 函数（例如 **lua-resty-core** 所需要的）。开启这个宏可以让 Nginx 二进制代码更小。

在 Nginx 或者 OpenResty 启用一个或多个宏定义，只用传给 **./configure** 脚本几个额外的 C 编译选项。例如：

```
# ./configure --with-cc-opt="-DNGX_LUA_USE_ASSERT -DNGX_LUA_ABORT_AT_PANIC"
```

[返回目录 \(page 3\)](#)

Installation on Ubuntu 11.10

注意：这里推荐使用 LuaJIT 2.1 或 LuaJIT 2.0 替换掉标准 Lua 5.1 解释器。

如果不得不使用标准的 Lua 5.1 解释器，在 Ubuntu 上使用这个命令完成安装：

```
apt-get install -y lua5.1 liblua5.1-0 liblua5.1-0-dev
```

应该可以正确被安装，除了一个小“麻烦”：

liblua.so 库在 **liblua5.1** 包中已经发生改变，只能使用 **liblua5.1.so**，并且需要被链接到 **/usr/lib**，这样才可以在 configure 执行阶段被找到。

```
ln -s /usr/lib/x86_64-linux-gnu/liblua5.1.so /usr/lib/liblua.so
```

[返回目录 \(page 3\)](#)

Community

英文邮件列表

英文邮件列表：[openresty-en](#)。

[返回目录 \(page 3\)](#)

中文邮件列表

中文邮件列表：[openresty](#)。

[返回目录 \(page 3\)](#)

Code Repository

本项目代码放在github上 [openresty/lua-nginx-module](#)。

[返回目录 \(page 3\)](#)

Bugs and Patches

提交bug报告、想法或补丁，可通过下面方式：

1. 创建一个ticket [GitHub Issue Tracker](#)
2. 或者发到这里 [OpenResty 社区 \(page 14\)](#).

[返回目录 \(page 3\)](#)

Lua/LuaJIT 字节码 support

从 [v0.5.0rc32](#) release 开始，所有 [*_by_lua_file](#) 的配置指令（比如 [content_by_lua_file \(page 46\)](#)）都支持直接加载 Lua 5.1 和 LuaJIT 2.0/2.1 的二进制字节码文件。

请注意，LuaJIT 2.0/2.1 生成的二进制格式与标准 Lua 5.1 解析器是不兼容的。所以如果在 ngx_lua 下使用 LuaJIT 2.0/2.1，那么 LuaJIT 兼容的二进制文件必须是下面这样生成的：

```
/path/to/luajit/bin/luajit -b /path/to/input_file.lua /path/to/output_file.luac
```

-bg 选项是在 LuaJIT 字节码文件中包含调试信息。

```
/path/to/luajit/bin/luajit -bg /path/to/input_file.lua /path/to/output_file.luac
```

对于 **-b** 选项，请参考官方 LuaJIT 文档获取更多细节：

http://luajit.org/running.html#opt_b

同样的，由 LuaJIT 2.1 生成的字节码文件对于 LuaJIT 2.0 也是不兼容的，反之亦然。第一次对于 LuaJIT 2.1 版本的字节支持，是在 v0.9.3 上完成的。

近似的，如果使用标准 Lua 5.1 解释器，Lua 的兼容字节码文件必须用 **luac** 命令行来生成：

```
luac -o /path/to/output_file.luac /path/to/input_file.lua
```

与 LuaJIT 不太一样，Lua 5.1 的字节码文件是默认包含调试信息的。这里可以使用 **-s** 选项去掉调试信息：

```
luac -s -o /path/to/output_file.luac /path/to/input_file.lua
```

在使用 LuaJIT 2.0/2.1 的 ngx_lua 实例中试图加载标准 Lua 5.1 字节码文件（反之亦然），将会在 error.log 中记录一条类似的错误信息：

```
[error] 13909#0: *1 failed to load Lua inlined code: bad byte-code header in /path/to/test_file.luac
```

使用 Lua **require** 和 **dofile** 这类原语加载字节码文件，应该总能按照预期工作。

[返回目录 \(page 3\)](#)

System Environment Variable Support

如果你想在 Lua 中通过标准 Lua API `os.getenv` 来访问系统环境变量，例如 `foo`，那么你需要在你的 `nginx.conf` 中，通过 `env` 指令，把这个环境变量列出来。例如：

```
env foo;
```

[返回目录 \(page 3\)](#)

HTTP 1.0 support

HTTP 1.0 协议不支持分块输出，当响应体不为空时，需要在响应头中明确指定 `Content-Length`，以支持 HTTP 1.0 长连接。所以当有一个 HTTP 1.0 请求发生，同时把 `lua_http10_buffering` (page 74) 指令设置为 `on` 时，`ngx_lua` 将缓存 `ngx.say` (page 131) 和 `ngx.print` (page 130) 的所有输出，同时延迟发送响应头直到接收到所有输出内容。这时 `ngx_lua` 可以计算响应体的总长度且构建一个正确的 `Content-Length` 响应头返回给 HTTP 1.0 客户端。即使已经将 `lua_http10_buffering` (page 74) 指令设置为 `on`，但如果正在执行的 Lua 代码中设置了 `Content-Length` 响应头，这种缓冲模式也将被禁用。

对于大型流式响应输出，禁用 `lua_http10_buffering` (page 74) 以最小化内存占用非常重要。

请注意，一些常见的 HTTP 性能测试工具，例如 `ab` 和 `http_load` 默认发送 HTTP 1.0 请求。要强制 `curl` 发送 HTTP 1.0 请求，使用 `-0` 选项。

[返回目录 \(page 3\)](#)

Statically Linking Pure Lua Modules

当使用 LuaJIT 2.x 时，可以把一个纯 Lua 字节码模块，静态链接到可运行的 Nginx 中。

首先你要使用 `LuaJIT` 的可执行程序，把 `.lua` 的 Lua 模块编译成 `.o` 的目标文件（包含导出的字节码数据），然后链接这些 `.o` 文件到你的 Nginx 构造环境。

用下面这个小例子来印证一下。这里我们的 `.lua` 文件使用 `foo.lua`：

```
-- foo.lua
local _M = {}

function _M.go()
    print("Hello from foo")
end

return _M
```

我们把 `.lua` 文件编译成 `foo.o` 文件：

```
/path/to/luajit/bin/luajit -bg foo.lua foo.o
```

这里重要的是 `.lua` 文件名，它决定了这个模块在业务 Lua 中是如何使用的。文件名 `foo.o` 除了扩展名是 `.o` 以外其他都不重要（只是用来告诉 `LuaJIT` 使用什么格式输出）。如果你想从输出的字节码中去掉 Lua 调试信息，你可以用 `-b` 选项替代原有的 `-bg`。

然后在构建 Nginx 或者 OpenResty 时，传给 `./configure` 脚本 `--with-lua-ld-opt="foo.o"` 选项：

```
./configure --with-lua-ld-opt="/path/to/foo.o" ...
```

最后，你可以在运行在 `ngx_lua` 中的任意 Lua 代码中调用：

```
local foo = require "foo"
foo.go()
```

并且，这段代码再也不会依赖外部的 `foo.lua` 文件，因为它已经被编译到了 `nginx` 程序中。

在调用 `require` 时，如果你想 Lua 模块名中使用点号，如下所示：

```
local foo = require "resty.foo"
```

那么在你使用 `LuaJIT` 命令行工具把他编译成 `.o` 文件之前，你需要把 `foo.lua` 文件重命名为 `resty_foo.lua`。

把 `.lua` 文件编译成 `.o` 文件，和构建 Nginx + ngx_lua，这两个过程中，使用完全相同版本的 LuaJIT 是至关重要的。

这是因为 LuaJIT 字节码格式在不同版本之间可能是不兼容的。当字节码文件出现了不兼容情况，你将看到一行 Lua 运行时错误信息：没找到 Lua 模块。

当你拥有多个 `.lua` 文件需要链接，你可以一次指明所有的 `.o` 文件，并赋给 `--with-luajit` 选项，参考：

```
./configure --with-luajit="/path/to/foo.o /path/to/bar.o" ...
```

如果你有非常多的 `.o` 文件，把这些文件的名称都写到命令行中不太可行，这种情况下，对你的 `.o` 文件可以构建一个静态库（或者归档），参考：

```
ar rcus libmyluafiles.a *.o
```

然后你就可以把 `myluafiles` 链接到你的 nginx 可执行程序中：

```
./configure \
    --with-luajit="-L/path/to/lib -Wl,--whole-archive -lmyluafiles -Wl,--no-whole-archive"
```

`/path/to/lib` 目录中应包含 `libmyluafiles.a` 文件。应当指出的是，这里要添加链接选项 `--whole-archive`，否则我们的归档将被跳过，因为在我们的归档没有导出 nginx 执行需要的函数符号。

[返回目录 \(page 3\)](#)

Data Sharing within an Nginx Worker

在同一个 nginx worker 进程处理的所有请求中共享数据，需要将共享数据封装进一个 Lua 模块中，并使用 Lua 语言内置的 `require` 方法加载该模块，之后就可以在 Lua 中操作共享数据了。这种方法之所以可行，是因为(在同一个nginx worker中)加载模块的操作仅被执行一次，所有的协程都会共享同一份拷贝(包括代码和数据)。但是请注意，Lua的全局变量(注意，不是模块级变量)将因为“每个请求一个协程”的隔离要求而不被保持。

下面是一个完整的例子：


```
-- mydata.lua
local _M = {}

local data = {
    dog = 3,
    cat = 4,
    pig = 5,
}

function _M.get_age(name)
    return data[name]
end

return _M
```

然后通过 nginx.conf 访问：

```
location /lua {
    content_by_lua_block {
        local mydata = require "mydata"
        ngx.say(mydata.get_age("dog"))
    }
}
```

例子中的 `mydata` 模块将只在第一个请求到达 `/lua` 的时候被加载运行，之后同一个nginx worker进程处理的所有请求，都将使用此模块已经加载的实例，并共享实例中的数据，直到 nginx 主进程接到 `HUP` 信号强制重新进行加载。这种数据共享技术是基于本模块(`ngx_lua`)的高性能 Lua 应用的基础。

注意，这种数据共享方式是基于worker而不是基于服务器的。也就是说，当 Nginx 主进程下面有多个 worker 进程时，数据共享不能跨越这些 worker 之间的进程边界。

一般来说，仅推荐使用这种方式共享只读数据。当计算过程中没有非阻塞性 I/O 操作时(包括 `ngx.sleep` (page 134))，你也可以在 nginx worker 进程内所有并发请求中共享可改变的数据。只要你不把控制权交还给 nginx 事件循环以及 `ngx_lua` 的轻量级线程调度器(包括隐含的)，它们之间就不会有任何竞争。因此，当你决定在 worker 中共享可变数据时，一定要非常小心。错误的优化经常会导致在高负载时产生竞争，这种 bug 非常难以发现。

如果需要在服务器级别共享数据，请使用以下方法：

1. 使用本模块提供的 `ngx.shared.DICT` (page 155) API
2. 使用单服务器单 nginx worker 进程(当使用多CPU或者多核CPU的服务器时)

不推荐)

3. 使用类似 `memcached` , `redis` , `MySQL` 或 `PostgreSQL` 等数据共享机制。与本模块相关的 `OpenResty` 软件包 包含了一系列相关的 Nginx 模块以及 Lua 库, 提供与这些数据存储机制的交互界面。

[返回目录 \(page 3\)](#)

Known Issues

[返回目录 \(page 3\)](#)

TCP socket 连接操作遗留问题

`tcpsock:connect` (page 171)方法, 返回 `success` 但实际上连接故障, 例如出现 `Connection Refused` 错误。

然而, 后面尝试对 `cosocket` 对象的任何操作都将失败, 并返回由失效连接操作所产生实际的错误状态消息。

这个问题是由于在 Nginx 的事件模型的局限性, 似乎只影响 Mac OS X 系统。

[返回目录 \(page 3\)](#)

Lua 协程 Yielding/Resuming

无论 Lua 5.1 and LuaJIT 2.0/2.1, 内建的 `dofile` 和 `require` 当前都是通过绑定 C 函数的方式, 如果 Lua 文件的加载是 `dofile` 或 `require` 并调用 `[ngx.location.capture]/(#ngxlocationcapture)`, `ngx.exec` (page 126), `ngx.exit` (page 132), 或者 Lua 中其他 API 函数的 `*top-level` 范围调用 `yielding`, 均会得到 “attempt to yield across C-call boundary” 的错误信息。为了避免这个情况, 把需要调用 `yielding` 部分放到你自己的 Lua 函数中, 这样在当前文件就不再是 `top-level` 范围。

对于标准 Lua 5.1 解析器的虚拟机唤醒支持是不完善的, `ngx.location.capture` (page 96), `ngx.location.capture_multi` (page 103), `ngx.redirect` (page 128), `ngx.exec` (page 126) 和 `ngx.exit` (page 132) 方法, 在 Lua `pcall()` 或 `xpcall()` 中是不能使用的。甚至 `for ... in ...` 小节的第一行在标准 Lua 5.1 解析器中都会报错 `attempt to yield across metamethod/C-call boundary`。请使用 LuaJIT 2.x, 它可以完美支持虚拟机唤醒, 避免这些问题。

[返回目录 \(page 3\)](#)

Lua Variable Scope

在代码中导入模块时应注意一些细节, 推介使用如下格式:

```
local xxx = require('xxx')
```

而非：

```
require('xxx')
```

理由如下：从设计上讲，全局环境的生命周期和一个 Nginx 的请求的生命周期是相同的。为了做到请求隔离，每个请求都有自己的 Lua 全局变量环境。Lua 模块在第一次请求打到服务器上的时候被加载起来，通过 `package.loaded` 表内建的 `require()` 完成缓存，为后续代码复用。并且一些 Lua 模块内的 `module()` 存在边缘问题，对加载完成的模块设置成全局表变量，但是这个全局变量在请求处理最后将被清空，并且每个后续请求都拥有自己（干净）的全局空间。所以它将因为访问 `nil` 值收到 Lua 异常。

一般来说，在 `ngx_lua` 的上下文中使用 Lua 全局变量真的不是什么好主意：

1. 滥用全局变量的副作用会对并发场景产生副作用，比如当使用者把这些变量看作是本地变量的时候；
2. Lua 的全局变量需要向上查找一个全局环境（只是一个 Lua 表），代价比较高；
3. 一些 Lua 的全局变量引用只是拼写错误，这会导致出错很难排查。

所以，我们极力推介在使用变量的时候总是使用 `local` 来定义以限定起生效范围是有理由的。

为了在你的 Lua 代码中找出所有使用 Lua 全局变量的地方，你可以运行 `lua-releng tool` 把所有 .lua 源文件检测一遍：

```
$ lua-releng
Checking use of Lua global variables in file lib/foo/bar.lua
...
   1      [1489]  SETGLOBAL      7 -1    ; contains
  55      [1506]  GETGLOBAL      7 -3    ; setvar
   3      [1545]  GETGLOBAL      3 -4    ; varexpend
```

上述输出说明文件 `lib/foo/bar.lua` 的 1489 行写入一个名为 `contains` 的全局变量，1506 行读取一个名为 `setvar` 的全局变量，1545 行读取一个名为 `varexpend` 的全局变量，

这个工具能保证 Lua 模块中的局部变量全部是用 `local` 关键字定义过的，否则将会抛出一个运行时异常。这样能阻止类似变量这样的资源的竞争。理由请参考 [Data Sharing within an Nginx Worker](#)

[返回目录 \(page 3\)](#)

Locations Configured by Subrequest Directives of Other Modules

`ngx.location.capture` (page 96) 和 `ngx.location.capture_multi` (page 103) 指令无法抓取包含以下指令的 location：`add_before_body`，`add_after_body`，`auth_request`，`echo_location`，`echo_location_async`，`echo_subrequest` 或 `echo_subrequest_async`。

```
location /foo {
    content_by_lua_block {
        res = ngx.location.capture("/bar")
    }
}
location /bar {
    echo_location /blah;
}
location /blah {
    echo "Success!";
}
```

```
$ curl -i http://example.com/foo
```

将不会按照预期工作。

[返回目录 \(page 77\)](#)

Cosockets Not Available Everywhere

归咎于 `nginx` 内核的各种限制规则，`cosocket` API 在这些环境中是被禁的：
`set_by_lua` (`#set_by_lua`)，`[log_by_lua` (page 58)，
`header_filter_by_lua` (`#header_filter_by_lua`) 和 `[body_filter_by_lua` (page 55)。

`cosocket` 在 `init_by_lua` (`#init_by_lua`) 和 `[init_worker_by_lua` (page 40) 小节中也是被禁的，但我们后面将会添加这些环境的支持，因为在 `nginx` 内核上是没有这个限制的（或者这个限制是可以被绕过的）。

这里有个绕路方法，前提是原始场景 不需要等待cosocket结果。就是说，通过 [ngx.timer.at \(page 190\)](#) API 创建一个零延迟的 `timer`，在 `timer` 中完成 cosocket 的处理结果，用这种同步的方式进行协作。

[返回目录 \(page 3\)](#)

Special Escaping Sequences

注意 自引入 `*_by_lua_block {}` 配置指令，我们将不再被该问题折磨。

PCRE 的转义符号例如 `\d`，`\s` 以及 `\w` 等需要特别注意，因为在字符串语义中，反斜线字符 `\` 会被 Lua 语言解析器和 Nginx 配置文件解析器在执行前同时处理掉，所以以下代码片段将无法按预期运行：

```
# nginx.conf
? location /test {
?     content_by_lua_block {
?         local regex = "\d+" -- 这里是错的!!
?         local m = ngx.re.match("hello, 1234", regex)
?         if m then ngx.say(m[0]) else ngx.say("not matched!") end
?     }
? }
# 结果为 "not matched!"
```

为避免这个问题，需要双重转义反斜线符号：

```
# nginx.conf
location /test {
    content_by_lua_block {
        local regex = "\\d+"
        local m = ngx.re.match("hello, 1234", regex)
        if m then ngx.say(m[0]) else ngx.say("not matched!") end
    }
}
# 结果为 "1234"
```

这里的 `\\d+`，先被 Nginx 配置文件解析器处理成 `\d+`，再被 Lua 语言解析器处理成 `\d+`，之后才被执行。

或者，正则表达式模板可以使用 Lua 字符串“长括号”语义写出，其语法形式为 `[...]`，在这种情况下，反斜线仅需为 Nginx 配置文件解析器转义一次。

```
# nginx.conf
location /test {
    content_by_lua_block {
        local regex = [[\\d+]]
        local m = ngx.re.match("hello, 1234", regex)
        if m then ngx.say(m[0]) else ngx.say("not matched!") e
nd
    }
}
# 结果为 to "1234"
```

这里，`[[\\d+]]` 被 Nginx 配置文件解析器处理成 `[[\\d+]]`，符合预期。

注意，当正则表达式模板中包括 `[...]` 序列时，Lua 语言中“更长的长括号”形式 `[=[...]=]` 是必要的。如果需要，可以将 `[=[...]=]` 作为默认形式。

```
# nginx.conf
location /test {
    content_by_lua_block {
        local regex = [=[0-9]+=]
        local m = ngx.re.match("hello, 1234", regex)
        if m then ngx.say(m[0]) else ngx.say("not matched!") e
nd
    }
}
# 结果为 "1234"
```

还有一种转义 PCRE 序列的方法是把 Lua 代码放到外部脚本文件中，通过各种 `*_by_lua_file` 指令执行。在这种方法中，反斜线仅被 Lua 语言解析器处理，因此只需要转义一次。

```
-- test.lua
local regex = "\\d+"
local m = ngx.re.match("hello, 1234", regex)
if m then ngx.say(m[0]) else ngx.say("not matched!") end
-- 结果为 "1234"
```

在外部脚本文件中，PCRE 序列如果使用“长括号”形式 Lua 字符串，则无需修改。

```
-- test.lua
local regex = [[\d+]]
local m = ngx.re.match("hello, 1234", regex)
if m then ngx.say(m[0]) else ngx.say("not matched!") end
-- 结果为 "1234"
```

[返回目录 \(page 3\)](#)

Mixing with SSI Not Supported

在同样的 Nginx 请求混合 SSI 与 ngx_lua 是完全不被支持的，只使用 ngx_lua 即可。使用 SSI 你可以做的任何事情，使用 ngx_lua 可以更好的完成，并且效能更棒。

[返回目录 \(page 3\)](#)

SPDY Mode Not Fully Supported

一些 ngx_lua 提供的 Lua APIs 在 Nginx 的 **SPDY** 模式下确定不能工作：
[ngx.location.capture \(page 96\)](#)，[ngx.location.capture_multi \(page 103\)](#) 和
[ngx.req.socket \(page 125\)](#)。

[返回目录 \(page 3\)](#)

Missing data on short circuited requests

Nginx提前销毁一个请求的可能（至少）：

- 400 (Bad Request) – 错误请求
- 405 (Not Allowed) – 不允许
- 408 (Request Timeout) – 请求超时
- 414 (Request URI Too Large) – 请求URI过大
- 494 (Request Headers Too Large) – 请求Headers过大
- 499 (Client Closed Request) – 客户端关闭请求
- 500 (Internal Server Error) – 内部服务错误
- 501 (Not Implemented) – 未实现

这意味着正常执行阶段被跳过，如重写或访问阶段。这也意味着，后面的所有阶段，例如 [log_by_lua \(page 58\)](#)，将无法获得在这个阶段存放的普通信息。

[返回目录 \(page 3\)](#)

TODO

- cosocket : 实现 LuaSocket 非连接的 UDP API。
- 实现普通的 UDP 服务替代 HTTP 服务，并支持 Lua 代码。例如：

```
datagram {
    server {
        listen 1953;
        handler_by_lua_block {
            -- custom Lua code implementing the special UDP se
rver...
        }
    }
}
```

- shm : 实现一个 “shared queue API”，对 [shared dict \(page 69\)](#) 补充 API。
- cosocket : 在 [init_by_lua* \(page 37\)](#) 添加支持。
- cosocket : 对于流式 cosocket，实现 bind() 方法。
- cosocket : 基于池子的后端并发连接控制，当后端并发超过它的连接池限制，实现自动排队的 [connect](#)。
- cosocket : 查看合并 aviramc's 的 [patch](#)，添加 [bsdrecv](#) 方法。
- 添加新的API函数完成标准 [add_header](#) 配置功能。
- 查看、合并 vadim-pavlov 的 补丁，给 [ngx.location.capture \(page 96\)](#)添加 [extra_headers](#) 选项。
- 使用 [ngx_hash_t](#) 去优化内建的 header 查找，涉及[ngx.req.set_header \(page 119\)](#), [ngx.header HEADER \(page 105\)](#) 等。
- cosocket 连接池溢出，支持配置选项定义不同策略。
- 添加新的小节，当 nginx 关闭时执行一段代码。
- 添加 [ignore_resp_headers](#)，[ignore_resp_body](#) 和 [ignore_resp](#) 选项给 [ngx.location.capture \(page 96\)](#)、[ngx.location.capture_multi \(page 103\)](#)，对于用户提升微小性能。
- 增加抢占式的协程调度，用来自动 yielding 或 resuming Lua 虚拟机。
- 添加 [stat](#) 类似 [mod_lua](#)。

- cosocket: 添加 SSL certificate 客户端支持。

[返回目录 \(page 3\)](#)

Changes

该模块每个发行版本的变更，都记录在OpenResty绑定的变更日志中：

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

[返回目录 \(page 3\)](#)

Test Suite

为了运行测试套件，依赖下面这些条件：

- Nginx version $\geq$ 1.4.2
- Perl modules:
 - Test::Nginx: <https://github.com/openresty/test-nginx>
- Nginx 模块:
 - [ngx_devel_kit](#)
 - [ngx_set_misc](#)
 - [ngx_auth_request](#) (如果你使用 1.5.4.+，这不是必须的)
 - [ngx_echo](#)
 - [ngx_memc](#)
 - [ngx_srcache](#)
 - [ngx_lua](#) (该模块)
 - [ngx_lua_upstream](#)
 - [ngx_headers_more](#)
 - [ngx_drizzle](#)
 - [ngx_rds_json](#)
 - [ngx_coolkit](#)
 - [ngx_redis2](#)

configure 时添加的这些模块顺序是非常重要的。因为在 filter 链中不同的过滤模块位置决定最终输出。正确的添加顺序如上所示。

- 第三方 Lua 库:

- [lua-cjson](#)
- 应用:
 - mysql: 创建数据库 'ngx_test', 对用户 'ngx_test' 赋予所有权限, 密码也是 'ngx_test'。
 - memcached: 监听默认端口, 11211.
 - redis: 监听默认端口, 6379.

查看 [developer build script](#) 内容, 在搭建测试环境时确定更多细节。

在默认的测试模式下启动测试套件:

```
cd /path/to/lua-nginx-module
export PATH=/path/to/your/nginx/sbin:$PATH
prove -I/path/to/test-nginx/lib -r t
```

运行指定的测试文件:

```
cd /path/to/lua-nginx-module
export PATH=/path/to/your/nginx/sbin:$PATH
prove -I/path/to/test-nginx/lib t/002-content.t t/003-errors.t
```

在一个特别的测试文件中, 运行指定的测试块, 对你需要进行块测试部分添加一行 `--- ONLY` 信息, 并使用 `prove` 工具运行这个 `.t` 文件。

此外, 还有其他各种测试方式, 基于 `mockeagain`, `valgrind` 等。参考 [Test::Nginx documentation](#), 有更多不同高级测试方式的资料。也可以看看在 Amazon EC2 的 Nginx 集群测试报告 <http://qa.openresty.org>。

[返回目录 \(page 3\)](#)

Copyright and License

该模块是根据BSD许可证授权。

Copyright (C) 2009-2016, by Xiaozhe Wang (chaoslawful) chaoslawful@gmail.com.

Copyright (C) 2009-2016, by Yichun “agentzh” Zhang (章亦春)
agentzh@gmail.com, CloudFlare Inc.

版权所有。

在源代码和二进制形式的二次发行和使用，无论修改与否，允许的前提是满足以下条件：

- 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.

[返回目录 \(page 3\)](#)

See Also

- [ngx_stream_lua_module](#) Nginx “stream” 子系统的官方模块版本（通用的下游 TCP 对话）。
- [lua-resty-memcached](#) 基于 ngx_lua cosocket 的库。
- [lua-resty-redis](#) 基于 ngx_lua cosocket 的库。
- [lua-resty-mysql](#) 基于 ngx_lua cosocket 的库。
- [lua-resty-upload](#) 基于 ngx_lua cosocket 的库。

- [lua-resty-dns](#) 基于 ngx_lua cosocket的库。
- [lua-resty-websocket](#) 提供 WebSocket 的客户端、服务端，基于 ngx_lua cosocket 的库。
- [lua-resty-string](#) 基于 LuaJIT FFI 的库。
- [lua-resty-lock](#) 一个简单非阻塞 lock API 库。
- [lua-resty-cookie](#) HTTP cookie 的库。
- [基于 URI 参数，路由到发起不同 MySQL 查询](#)
- [基于 Redis 和 Lua 的动态路由](#)
- [使用 LuaRocks 与 ngx_lua](#)
- [ngx_lua 的介绍信息](#)
- [ngx_devel_kit](#)
- [echo-nginx-module](#)
- [drizzle-nginx-module](#)
- [postgres-nginx-module](#)
- [memc-nginx-module](#)
- [OpenResty 捆绑包](#)
- [Nginx Systemtap 工具箱](#)

[返回目录 \(page 3\)](#)

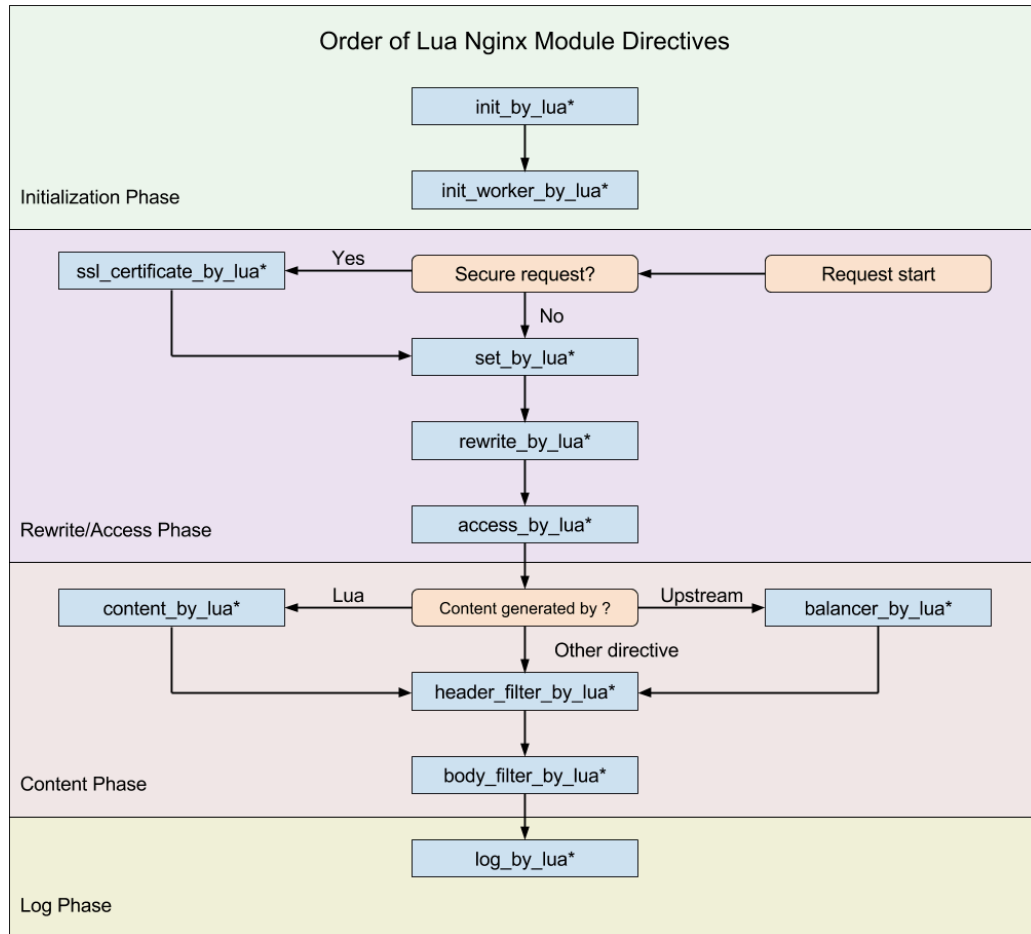
Directives

- [lua_use_default_type \(page 33\)](#)
- [lua_malloc_trim \(page 34\)](#)
- [lua_code_cache \(page 34\)](#)
- [lua_regex_cache_max_entries \(page 35\)](#)
- [lua_regex_match_limit \(page 36\)](#)
- [lua_package_path \(page 36\)](#)
- [lua_package_cpath \(page 37\)](#)
- [init_by_lua \(page 37\)](#)
- [init_by_lua_block \(page 39\)](#)
- [init_by_lua_file \(page 39\)](#)

- [init_worker_by_lua](#) (page 40)
- [init_worker_by_lua_block](#) (page 41)
- [init_worker_by_lua_file](#) (page 42)
- [set_by_lua](#) (page 42)
- [set_by_lua_block](#) (page 44)
- [set_by_lua_file](#) (page 44)
- [content_by_lua](#) (page 45)
- [content_by_lua_block](#) (page 45)
- [content_by_lua_file](#) (page 46)
- [rewrite_by_lua](#) (page 46)
- [rewrite_by_lua_block](#) (page 50)
- [rewrite_by_lua_file](#) (page 50)
- [access_by_lua](#) (page 51)
- [access_by_lua_block](#) (page 52)
- [access_by_lua_file](#) (page 53)
- [header_filter_by_lua](#) (page 53)
- [header_filter_by_lua_block](#) (page 54)
- [header_filter_by_lua_file](#) (page 55)
- [body_filter_by_lua](#) (page 55)
- [body_filter_by_lua_block](#) (page 57)
- [body_filter_by_lua_file](#) (page 58)
- [log_by_lua](#) (page 58)
- [log_by_lua_block](#) (page 61)
- [log_by_lua_file](#) (page 61)
- [balancer_by_lua_block](#) (page 61)
- [balancer_by_lua_file](#) (page 63)
- [lua_need_request_body](#) (page 63)
- [ssl_certificate_by_lua_block](#) (page 64)
- [ssl_certificate_by_lua_file](#) (page 65)

- [ssl_session_fetch_by_lua_block](#) (page 65)
- [ssl_session_fetch_by_lua_file](#) (page 67)
- [ssl_session_store_by_lua_block](#) (page 67)
- [ssl_session_store_by_lua_file](#) (page 68)
- [lua_shared_dict](#) (page 69)
- [lua_socket_connect_timeout](#) (page 69)
- [lua_socket_send_timeout](#) (page 70)
- [lua_socket_send_lowat](#) (page 70)
- [lua_socket_read_timeout](#) (page 70)
- [lua_socket_buffer_size](#) (page 71)
- [lua_socket_pool_size](#) (page 71)
- [lua_socket_keepalive_timeout](#) (page 71)
- [lua_socket_log_errors](#) (page 72)
- [lua_ssl_ciphers](#) (page 72)
- [lua_ssl_crl](#) (page 73)
- [lua_ssl_protocols](#) (page 73)
- [lua_ssl_trusted_certificate](#) (page 73)
- [lua_ssl_verify_depth](#) (page 74)
- [lua_http10_buffering](#) (page 74)
- [rewrite_by_lua_no_postpone](#) (page 75)
- [access_by_lua_no_postpone](#) (page 75)
- [lua_transform_underscores_in_response_headers](#) (page 75)
- [lua_check_client_abort](#) (page 76)
- [lua_max_pending_timers](#) (page 77)
- [lua_max_running_timers](#) (page 77)

构建基本的 Nginx Lua 脚本均是通过指令完成。当用户 Lua 代码执行时，这些指令用来指名 如何使用脚本的返回结果。下面的流程图给我们演示这些指令的执行顺序。



[返回目录 \(page 3\)](#)

lua_use_default_type

语法: `lua_use_default_type on | off`

默认: `lua_use_default_type on`

环境: `http, server, location, location if`

指定响应头中 Content-Type 的默认值，是否使用 default_type 指令中指定的 MIME 类型。如果你的 Lua 请求处理程序不想要默认的 Content-Type 响应头，可以关闭这个指令。

默认情况下该指令被打开。

这个指令在 0.9.1 版本中首次引入。

[返回目录 \(page 3\)](#)

lua_malloc_trim

语法: `lua_malloc_trim <request-count>`

默认: `lua_malloc_trim 1000`

环境: `http`

当 NGINX 核心每执行 **N** 次请求，告诉底层 **libc** 运行库，释放已缓存空闲内存还给操作系统。**N** 的默认值是 1000。可以配置新的数值来控制“请求数”，小的数字意味着更频繁的释放，这可能会引入比较高的 CPU 时间消耗和较少内存占用。而大的数字通常则占用较少的 CPU 时间消耗和较大的内存占用。所以这里需要根据自己的使用场景来调整。

配置参数为 **0**，代表关闭周期性的内存整理。

```
lua_malloc_trim 0; # 完全关闭 trim
```

为了完成请求计数，目前是在 NGINX 的 log 阶段实现的。当有子请求存在并且 `nginx.conf` 中出现 `log_subrequest on` 指令，可以更快获得计数增长。默认情况只统计“主请求”计数。

注意：该指令不影响 LuaJIT 基于 `mmap` 系统调用的内存分配。

该指令在 **v0.10.7** 版本首次引入。

[返回目录 \(page 30\)](#)

lua_code_cache

语法: `lua_code_cache on | off`

默认: `lua_code_cache on`

环境: `http, server, location, location if`

打开或者关闭 `*_by_lua_file` 指令（类似 `set_by_lua_file` 和 `content_by_lua_file`）中指定的 Lua 代码，以及 Lua 模块的 Lua 代码缓存。

当缓存关闭时，每个 `ngx_lua` 处理的请求都将会运行在一个单独的 Lua 虚拟机实例中，从 0.9.3 版本开始。所以在 `set_by_lua_file`, `content_by_lua_file`, `access_by_lua_file` 中引用的 Lua 文件不会被缓存，并且所有使用的 Lua 模块都会从头开始加载。有了这个选项，开发者很容易通过编辑文件并重新请求的方法进行测试。

然而需要注意的是，当你编辑写在 `nginx.conf` 中内嵌的 Lua 代码时，比如通过 `set_by_lua`, `content_by_lua`, `access_by_lua`, `rewrite_by_lua` 这些指令 写在 `nginx.conf` 里面的 Lua 代码，缓存不会被更新。因为只有 Nginx 的配置文件解释器才能正确解析 `nginx.conf`，所以重新加载配置文件的唯一办法是发送 HUP 信号或者重启 Nginx。

```
kill -HUP pid
nginx -s reload
```

即使代码缓存打开了，在 `*_by_lua_file` 中使用 `dofile` 或 `loadfile` 函数时内容不会被缓存（除非你自己缓存结果）。通常你可以在 `init_by_lua` 或 `init_by_lua_file` 指令中加载所有这些文件，或者让这些 Lua 文件变成真正的 Lua 模块，通过 `require` 来加载。

现在 `ngx_lua` 模块还不支持 Apache `mod_lua` 模块中可用的 `stat` 模式（已经列入待做任务）。

不推荐在生产环境中关闭 lua 代码缓存，请确保它只在开发环境中使用，他对整体性能有非常明显的影响。举个例子，输出“你好世界”在没有开启 lua 代码缓存时可以降低一个量级。

[返回目录 \(page 30\)](#)

lua_regex_cache_max_entries

语法: `lua_regex_cache_max_entries <num>`

默认: `lua_regex_cache_max_entries 1024`

环境: `http`

在工作进程级别，指定正则表达式编译缓存允许的最大数目。

正则表达式被用于 [ngx.re.match \(page 145\)](#)，[ngx.re.gmatch \(page 150\)](#)，[ngx.re.sub \(page 152\)](#)，和 [ngx.re.gsub \(page 154\)](#)，如果使用 `o`（既，编译一次的标识）正则选项，将会被缓存。

允许的默认数量为 1024，当达到此限制，新的正则表达式将不会被缓存（就像没指定 `o` 选项一样），将会有且仅只有一个告警信息在 `error.log` 文件中：

```
2011/08/27 23:18:26 [warn] 31997#0: *1 lua exceeding regex cache max entries (1024), ...
```

如果你是通过加载 `resty.core.regex` 模块（或者仅仅是 `resty.core` 模块）来使用 `lua-resty-core` 实现的 `ngx.re.*`，一个基于 LRU 的缓存将被用到这里的正则表达式缓存。

对于部分正则表达式（字符串的各种替换，如 `ngx.re.sub` (page 152) 和 `ngx.re.gsub` (page 154)），不要使用 `o` 选项，这类正则每次都不一样，缓存无法被利用。这样我们可以避免撞上最大数的限制。

[返回目录 \(page 30\)](#)

lua_regex_match_limit

语法: `lua_regex_match_limit <num>`

默认: `lua_regex_match_limit 0`

环境: `http`

指定执行 `ngx.re API` (page 145) 时使用 PCRE 库的“匹配限制”。引述 PCRE 手册，“the limit ... has the effect of limiting the amount of backtracking that can take place”。

当触发了这个限制，在 Lua 代码的 `ngx.re API` (page 145) 函数，将返回错误信息“`pcre_exec() failed: -8`”。

当设置了限制为 0，将使用编译 PCRE 库的默认“match limit”。这也是这个配置的默认值。

这个指令是在 `v0.8.5` 发行版被首次引入的。

[返回目录 \(page 30\)](#)

lua_package_path

语法: `lua_package_path <lua-style-path-str>`

默认: 当前环境 `LUA_PATH` 的环境变量或编译指定的默认值

环境: `http`

设置 `set_by_lua`(#`set_by_lua`) , `[content_by_lua` (page 45) 和其他脚本对 Lua 模块的查找路径。路径字符串是标准 Lua 路径格式，特殊标识 `::` 可被用来代表原始搜索路径。

从 `v0.5.0rc29` 发行版开始，特殊符号 `$prefix` 或 `${prefix}` 可用于搜索路径字符串中。`server prefix` 的值，通常是由 Nginx 服务启动时的 `-p PATH` 命令行决定的。

[返回目录 \(page 30\)](#)

lua_package_cpath

语法: `lua_package_cpath <lua-style-cpath-str>`

默认: 当前环境 `LUA_CPATH` 的环境变量或编译指定的默认值

环境: `http`

设置 `set_by_lua`(#`set_by_lua`) , `[content_by_lua` (page 45) 和其他脚本对 Lua C 模块的查找路径。cpath 路径字符串是标准 Luacpath 路径格式, 特殊标识 `;;` 可被用来代表原始 cpath 路径。

从 `v0.5.0rc29` 发行版开始, 特殊符号 `$prefix` 或 `${prefix}` 可用于搜索路径字符串中。 `server prefix` 的值, 通常是由 Nginx 服务启动时的 `-p PATH` 命令行决定的。

[返回目录 \(page 30\)](#)

init_by_lua

语法: `init_by_lua <lua-script-str>`

环境: `http`

阶段: `loading-config`

注意 自从 `v0.9.17` 版本, 不鼓励使用该指令, 应使用新的 `init_by_lua_block` (page 39) 指令进行替代。

当 Nginx master 进程 (如果有) 加载 Nginx 配置文件时, 在全局的 Lua 虚拟机上运行 `<lua-script-str>` 指定的 Lua 代码。

当 Nginx 收到 `HUP` 信号并开始重新加载配置文件, Lua 虚拟机将重新创建并且 `init_by_lua` 在新的 Lua 虚拟机中再次执行。为防止 `lua_code_cache` (page 34) 指令是关闭的 (默认打开), 对于这个场景 `init_by_lua` 将在每个请求之上运行, 因为在这个场景中, 每个请求都会创建新的 Lua 虚拟机, 他们都是独立存在。

通常, 你可以在服务启动时注册 Lua 全局变量或预加载 Lua 模块。这是个预加载 Lua 模块的示例代码:

```
init_by_lua 'cjson = require "cjson";

server {
    location = /api {
        content_by_lua_block {
            ngx.say(cjson.encode({dog = 5, cat = 6}))
        }
    }
}
```

你也可以在这个阶段初始化[lua_shared_dict](#) (page 69)共享内存内容。这里是示例代码：

```
lua_shared_dict dogs 1m;

init_by_lua '
    local dogs = ngx.shared.dogs;
    dogs:set("Tom", 56)
';

server {
    location = /api {
        content_by_lua_block {
            local dogs = ngx.shared.dogs;
            ngx.say(dogs:get("Tom"))
        }
    }
}
```

需要注意，当配置重载（例如 [HUP](#) 信号）时 [lua_shared_dict](#) (page 69)的共享数据是不会被清空的。这种情况下，如果你不想在 [init_by_lua](#) 再次初始化你的共享数据，你需要设置一个个性标识并且在你的 [init_by_lua](#) 代码中每次都做检查。

因为在这个上下文中的Lua代码是在 Nginx fork 工作进程之前（如果有）执行，加载的数据和代码将被友好 [Copy-on-write \(COW\)](#) 特性提供给其他所有工作进程，从而节省了大量内存。

不要在这个上下文中初始化你自己的 Lua 全局变量，因为全局变量的使用有性能损失并会带来全局命名污染（可以查看 [Lua 变量范围](#) (page 20)获取更多细节）。推荐的方式是正确使用 [Lua模块](#) 文件（不要使用标准 Lua 函数 [module\(\)](#) 来定义 Lua 模块，因为它同样对全局命名空间有污染），在 [init_by_lua](#) 或其他上下文中调用

`require()` 来加载你自己的模块文件。`require()` 会在全局 Lua 注册的 `package.loaded` 表中缓存 Lua 模块，所以在整个 Lua 虚拟机实例中你的模块将只会加载一次。

在这个上下文中，只有一小部分的 [Nginx Lua API \(page 77\)](#) 是被支持的：

- 记录日志的 APIs：[ngx.log \(page 131\)](#) 和 [print \(page 92\)](#)
- 共享内存字典 APIs：[ngx.shared.DICT \(page 155\)](#)

在这个上下文中，根据用户的后续需要，将会支持更多的 Nginx Lua APIs。

基本上，在这个上下文中，你可以保守使用 Lua 库完成阻塞 I/O 调用，因为在 master 进程的阻塞调用在服务的启动过程中是完全没问题的。进一步说在配置加载阶段，Nginx 核心就是阻塞 I/O 方式处理的（至少在解析上游主机名称时）。

你应该非常小心，在这种情况下注册的 Lua 代码潜在的安全漏洞，因为 Nginx 的主进程经常是 'root' 帐户下运行。

这个指令是 `v0.5.5` 版本中第一次引入的。

[返回目录 \(page 30\)](#)

init_by_lua_block

语法: `init_by_lua_block { lua-script }`

环境: `http`

阶段: `loading-config`

与 [init_by_lua \(page 37\)](#) 指令相似，只不过该指令在一对括号 (`{ }`) 中直接内嵌 Lua 代码，替代之前 Nginx 的字符串（需要特殊字符转义）。

例如：

```
init_by_lua_block {  
    print("I need no extra escaping here, for example: \r\nbla  
h")  
}
```

该指令在 `v0.9.17` 版本首次引入。

[返回目录 \(page 30\)](#)

init_by_lua_file

语法: `init_by_lua_file <path-to-lua-script-file>`

环境: *http*

阶段: *loading-config*

与 [init_by_lua \(page 37\)](#) 等价，通过 `<path-to-lua-script-file>` 指定文件的 Lua 代码 或 [Lua/LuaJIT 字节码 \(page 0\)](#) 来执行。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

该指令在 `v0.5.5` 发行版第一次被引入。

[返回目录 \(page 30\)](#)

init_worker_by_lua

语法: `init_worker_by_lua <lua-script-str>`

环境: *http*

阶段: *starting-worker*

注意 自从 `v0.9.17` 版本，使用该指令是 *不优雅* 的，应使用新的 [init_worker_by_lua_block \(page 41\)](#) 指令进行替代。

开启 master 进程模式，Nginx 工作进程启动时执行指定的 Lua 代码。关闭 master 模式，将在 [init_by_lua* \(page 37\)](#) 后直接运行。

这个指令经常被用来创建单进程的反复执行定时器（通过 [ngx.timer.at \(page 190\)](#) Lua API 创建），可以是后端服务健康检查，也可以是其他定时的日常工作。下面是个例子：

```
init_worker_by_lua '  
    local delay = 3 -- in seconds  
    local new_timer = ngx.timer.at  
    local log = ngx.log  
    local ERR = ngx.ERR  
    local check  
  
    check = function(premature)  
        if not premature then  
            -- do the health check or other routine work  
            local ok, err = new_timer(delay, check)  
            if not ok then  
                log(ERR, "failed to create timer: ", err)  
                return  
            end  
        end  
    end  
end  
  
local ok, err = new_timer(delay, check)  
if not ok then  
    log(ERR, "failed to create timer: ", err)  
    return  
end  
';
```

这个指令是在 **v0.9.5** 发行版第一次引入。

[返回目录 \(page 30\)](#)

init_worker_by_lua_block

语法: `init_worker_by_lua_block { lua-script }`

环境: `http`

阶段: `starting-worker`

与 [init_worker_by_lua \(page 40\)](#) 指令相似，只不过该指令在一对括号 (`{ }`) 中直接内嵌 Lua 代码，替代之前 Nginx 的字符串（需要特殊字符转义）。

例如：

```
init_worker_by_lua_block {  
    print("I need no extra escaping here, for example: \r\nbla  
h")  
}
```

该指令在 **v0.9.17** 版本首次引入。

[返回目录 \(page 30\)](#)

init_worker_by_lua_file

语法: `init_worker_by_lua_file <lua-file-path>`

环境: *http*

阶段: *starting-worker*

与 `init_worker_by_lua` (page 40) 等价, 通过 `<lua-file-path>` 指定的 Lua 或 LuaJIT 字节码 (page 0) 文件来执行。

该指令是在 **v0.9.5** 发行版第一次引入。

[返回目录 \(page 30\)](#)

set_by_lua

语法: `set_by_lua $res <lua-script-str> [$arg1 $arg2 ...]`

环境: *server, server if, location, location if*

阶段: *rewrite*

注意 自从 **v0.9.17** 版本, 使用该指令是 不优雅 的, 应使用新的 `set_by_lua_block` (page 44) 指令进行替代。

使用可选的输入参数 `$arg1 $arg2 ...`, 执行指定的代码 `<lua-script-str>`, 并返回字符串结果到 `$res`。

`<lua-script-str>` 的代码可以做 [API 调用 \(page 77\)](#), 并能从 `ngx.arg` 表中获取输入参数 (下标起始值是 **1** 并顺序增长)。

该指令被设计为执行短小、快速的代码块, 因为代码执行时Nginx的事件循环是被阻塞的。因此应避免耗时的代码处理。

这个指令是通过挂载自定义命令到标准 `ngx_http_rewrite_module` 模块列表来实现。因为模块 `ngx_http_rewrite_module` 中是不支持非阻塞 I/O, 所以在本指令中, 是无法 `yield` 当前 Lua 的“轻线程”。

在 `set_by_lua` 的上下文中，至少下列 API 函数目前是被禁止的：

- 输出 API 函数 (例如 [ngx.say \(page 131\)](#) 和 [ngx.send_headers \(page 129\)](#))
- 控制 API 函数 (例如 [ngx.exit \(page 132\)](#))
- 子请求 API 函数 (例如 [ngx.location.capture \(page 96\)](#) 和 [ngx.location.capture_multi \(page 103\)](#))
- Cosocket API 函数 (例如 [ngx.socket.tcp \(page 170\)](#) 和 [ngx.req.socket \(page 125\)](#))
- 休眠 API 函数 [ngx.sleep \(page 134\)](#)

额外注意的，本指令一次只能写回一个值到一个 Nginx 变量。尽管如此，可以使用 [ngx.var.VARIABLE \(page 88\)](#) 接口绕过这个限制。

```
location /foo {
    set $diff ''; # we have to predefine the $diff variable here

    set_by_lua $sum '
        local a = 32
        local b = 56

        ngx.var.diff = a - b; -- write to $diff directly
        return a + b;         -- return the $sum value normally
    ';

    echo "sum = $sum, diff = $diff";
}
```

这个指令可以自由的与其他指令模块混合使用，如 [ngx_http_rewrite_module](#)，[set-misc-nginx-module](#) 和 [array-var-nginx-module](#)。所有这些指令的执行顺序，将和他们配置文件中出现的顺序一致。

```
set $foo 32;
set_by_lua $bar 'return tonumber(ngx.var.foo) + 1';
set $baz "bar: $bar"; # $baz == "bar: 33"
```

自 `v0.5.0rc29` 版本开始，本指令的 `<lua-script-str>` 参数中不再支持内联 Nginx 变量，所以可以直接使用 `$` 字符作为其字面值。

这个指令需要 [ngx_devel_kit](#) 模块。

[返回目录 \(page 30\)](#)

set_by_lua_block

语法: `set_by_lua_block $res { lua-script }`

环境: `server, server if, location, location if`

阶段: `rewrite`

与 `set_by_lua*` (page 42) 指令相似，以下情况除外：

1. 该指令在一对括号 (`{ }`) 中直接内嵌 Lua 代码，替代之前 Nginx 的字符串（需要特殊字符转义）
2. 该指令和 `set_by_lua*` (page 42) 一样，在 Lua 脚本的后面不支持额外参数

例如：

```
set_by_lua_block $res { return 32 + math.cos(32) }  
# $res now has the value "32.834223360507" or alike.
```

在 Lua 代码块中无需任何的特殊转义。

该指令在 `v0.9.17` 版本首次引入。

[返回目录 \(page 30\)](#)

set_by_lua_file

语法: `set_by_lua_file $res <path-to-lua-script-file> [$arg1 $arg2 ...]`

环境: `server, server if, location, location if`

阶段: `rewrite`

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 `set_by_lua*` (page 42) 是等价的，该指令从 `v0.5.0rc32` 开始支持 Lua/LuaJIT 字节码 (page 0) 的执行。

对于该指令，对 `<path-to-lua-script-file>` 的字符串参数支持内联 Nginx 变量。但必须要额外注意注入攻击。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

当 Lua 代码缓存开启（默认），用户代码在第一次请求时完成加载（只有一次）并缓存，当 Lua 文件被修改时，每次都要对 Nginx 配置进行重新加载。Lua 代码缓存是可以暂时被禁用，通过开关 `lua_code_cache` (page 34) 在 `nginx.conf` 中设置为 `off`，这样就可以避免反复重新加载 Nginx。

该指令需要 `ngx_devel_kit` 模块。

[返回目录 \(page 30\)](#)

content_by_lua

语法: `content_by_lua <lua-script-str>`

环境: `location`, `location if`

阶段: `content`

注意 自从 `v0.9.17` 版本, 使用该指令是 不优雅 的, 应使用新的 `content_by_lua_block` (page 45) 指令进行替代。

作为“内容处理程序”，为每一个请求执行 `<lua-script-str>` 中指定的 Lua 代码。

这些 Lua 代码可以调用 [全部 API \(page 77\)](#)，并作为一个新的协程，在一个独立的全局环境中执行（就像一个沙盒）。

不要将本指令和其他内容处理程序指令放到同一个 `location` 中。比如，本指令和 `proxy_pass` 指令就不能在同一个 `location` 中使用。

[返回目录 \(page 30\)](#)

content_by_lua_block

语法: `content_by_lua_block { lua-script }`

环境: `location`, `location if`

阶段: `content`

与 `content_by_lua` (page 45) 指令相似，只不过该指令在一对括号 (`{ }`) 中直接内嵌 Lua 代码，替代之前 Nginx 的字符串（需要特殊字符转义）。

例如：

```
content_by_lua_block {
    ngx.say("I need no extra escaping here, for example: \r\nb
lah")
}
```

该指令在 **v0.9.17** 版本首次引入。

[返回目录 \(page 30\)](#)

content_by_lua_file

语法: `content_by_lua_file <path-to-lua-script-file>`

环境: `location, location if`

阶段: `content`

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 [content_by_lua \(page 45\)](#) 是等价的，该指令从 **v0.5.0rc32** 开始支持 Lua/LuaJIT 字节码 ([page 0](#)) 的执行。

在 `<path-to-lua-script-file>` 中可以使用 Nginx 的内置变量来提高灵活性，然而这带有一定的风险，通常并不推荐使用。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

当 Lua 代码缓存开启（默认），用户代码在第一次请求时完成加载（只有一次）并缓存，当 Lua 文件被修改时，每次都要对 Nginx 配置进行重新加载。Lua 代码缓存是可以暂时被禁用，通过开关 [lua_code_cache \(page 34\)](#) 在 `nginx.conf` 中设置为 `off`，这样就可以避免反复重新加载 Nginx。

支持通过 Nginx 变量完成动态调度文件路径，例如：

```
# 注意: nginx 变量必须要小心过滤, 否则它将带来严重的安全风险!
location ~ ^/app/([-_a-zA-Z0-9/]+) {
    set $path $1;
    content_by_lua_file /path/to/lua/app/root/$path.lua;
}
```

一定要非常小心恶意用户的输入，并始终仔细验证或过滤掉用户提供的路径项。

[返回目录 \(page 30\)](#)

rewrite_by_lua

语法: `rewrite_by_lua <lua-script-str>`

环境: `http, server, location, location if`

阶段: `rewrite tail`

注意 自从 **v0.9.17** 版本, 使用该指令是 **不优雅** 的, 应使用新的 [rewrite_by_lua_block \(page 50\)](#) 指令进行替代。

作为一个重写阶段的处理程序, 为每个请求执行由 **<lua-script-str>** 指定的 Lua 代码。

这些 Lua 代码可以调用 [全部 API \(page 77\)](#), 并作为一个新的协程, 在一个独立的全局环境中执行 (就像一个沙盒)。

注意这个处理过程总是在标准 [ngx_http_rewrite_module](#) 的 **后面**。所以下面的示例可以按照预期执行：

```
location /foo {
    set $a 12; # create and initialize $a
    set $b ""; # create and initialize $b
    rewrite_by_lua 'ngx.var.b = tonumber(ngx.var.a) + 1';
    echo "res = $b";
}
```

因为 **set \$a 12** 和 **set \$b ""** 的执行都在 [rewrite_by_lua \(page 46\)](#) 的前面。

另一方面, 下面的示例是不能按照预期执行：

```
? location /foo {
?     set $a 12; # create and initialize $a
?     set $b ''; # create and initialize $b
?     rewrite_by_lua 'ngx.var.b = tonumber(ngx.var.a) + 1';
?     if ($b = '13') {
?         rewrite ^ /bar redirect;
?         break;
?     }
?
?     echo "res = $b";
? }
```

因为 **if** 是 [rewrite_by_lua \(page 46\)](#) 之 **前** 执行的, 尽管在配置中它被放到 [rewrite_by_lua \(page 46\)](#) 后面。

正确的处理方式应该是这样：

```
location /foo {
    set $a 12; # create and initialize $a
    set $b ''; # create and initialize $b
    rewrite_by_lua '
        ngx.var.b = tonumber(ngx.var.a) + 1
        if tonumber(ngx.var.b) == 13 then
            return ngx.redirect("/bar");
        end
    ';

    echo "res = $b";
}
```

注意, `ngx_eval` 模块可以近似的使用 `rewrite_by_lua` (page 46)。例如：

```
location / {
    eval $res {
        proxy_pass http://foo.com/check-spam;
    }

    if ($res = 'spam') {
        rewrite ^ /terms-of-use.html redirect;
    }

    fastcgi_pass ...;
}
```

在 `ngx_lua` 中可以这样实施：

```
location = /check-spam {
    internal;
    proxy_pass http://foo.com/check-spam;
}

location / {
    rewrite_by_lua '
        local res = ngx.location.capture("/check-spam")
        if res.body == "spam" then
            return ngx.redirect("/terms-of-use.html")
        end
    ';

    fastcgi_pass ...;
}
```

如同其他重写阶段处理，[rewrite_by_lua \(page 46\)](#) 在子请求中也执行的。

注意，在 [rewrite_by_lua \(page 46\)](#) 处理内部，当调用 `ngx.exit(ngx.OK)` 时，nginx 请求将继续下一阶段的内容处理。要在 [rewrite_by_lua \(page 46\)](#) 处理中终结当前请求，调用 `ngx.exit (page 132)`，成功的请求设定 `status >= 200 ( ngx.HTTP_OK )` 并 `status < 300 ( ngx.HTTP_SPECIAL_RESPONSE )`，失败的请求设定 `ngx.exit(ngx.HTTP_INTERNAL_SERVER_ERROR)` (或其他相关的)。

如果使用了 `ngx_http_rewrite_module` 的 `rewrite` 指令来改变 URI 并发起内部重定向，那么在当前 location 内任何有序的 [rewrite_by_lua \(page 46\)](#) 或 [rewrite_by_lua_file \(page 50\)](#) 代码都将不被执行。例如：

```
location /foo {
    rewrite ^ /bar;
    rewrite_by_lua 'ngx.exit(503)';
}
location /bar {
    ...
}
```

如果 `rewrite ^ /bar last` 被用做一个类似内部重定向使用，它将被忽略，这里的 Lua 代码 `ngx.exit(503)` 是永远不能执行的。如果使用了 `break` 标识，这里将没有内部重定向，并且执行 `rewrite_by_lua` 中的 Lua 代码。

`rewrite_by_lua` 代码将永远在 `rewrite` 请求处理阶段后面，除非 [rewrite_by_lua_no_postpone \(page 75\)](#) 配置开启。

[返回目录 \(page 30\)](#)

rewrite_by_lua_block

语法: `rewrite_by_lua_block { lua-script }`

环境: `http, server, location, location if`

阶段: `rewrite tail`

与 [rewrite_by_lua \(page 46\)](#) 指令相似，只不过该指令在一对括号 (`{ }`) 中直接内嵌 Lua 代码，替代之前 Nginx 的字符串（需要特殊字符转义）。

例如：

```
rewrite_by_lua_block {  
    do_something("hello, world!\nhiya\n")  
}
```

该指令在 `v0.9.17` 版本首次引入。

[返回目录 \(page 30\)](#)

rewrite_by_lua_file

语法: `rewrite_by_lua_file <path-to-lua-script-file>`

环境: `http, server, location, location if`

阶段: `rewrite tail`

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 [rewrite_by_lua \(page 46\)](#) 是等价的，该指令从 `v0.5.0rc32` 开始支持 [Lua/LuaJIT 字节码 \(page 0\)](#) 的执行。

在 `<path-to-lua-script-file>` 中可以使用 Nginx 的内置变量用来提高灵活性，然而这带有一定的风险，通常并不推荐使用。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

当 Lua 代码缓存开启（默认），用户代码在第一次请求时完成加载（只有一次）并缓存，当 Lua 文件被修改时，每次都要对 Nginx 配置进行重新加载。Lua 代码缓存是可以暂时被禁用，通过开关 [lua_code_cache \(page 34\)](#) 在 `nginx.conf` 中设置为 `off`，这样就可以避免反复重新加载 Nginx。

`rewrite_by_lua_file` 代码将永远在 `rewrite` 请求处理阶段后面，除非 `rewrite_by_lua_no_postpone` (page 75) 配置开启。

支持通过 Nginx 变量完成动态调度文件路径，就像 `content_by_lua_file` (page 46) 一样。

[返回目录](#) (page 30)

access_by_lua

语法: `access_by_lua <lua-script-str>`

环境: `http, server, location, location if`

阶段: `access tail`

注意 自从 `v0.9.17` 版本, 使用该指令是 *不优雅* 的, 应使用新的 `access_by_lua_block` (page 52) 指令进行替代。

扮演 `access` 阶段处理, 对每次请求执行在 `<lua-script-str>` 中指名的 Lua 代码。

这些 Lua 代码可以调用 [全部 API](#) (page 77), 并作为一个新的协程, 在一个独立的全局环境中执行 (就像一个沙盒)。

注意: 本指令的处理总是在标准 `ngx_http_access_module` 的后面。所以下面的示例可以按照预期工作:

```
location / {
    deny    192.168.1.1;
    allow   192.168.1.0/24;
    allow   10.1.1.0/16;
    deny    all;

    access_by_lua '
        local res = ngx.location.capture("/mysql", { ... })
        ...
    ';

    # proxy_pass/fastcgi_pass/...
}
```

换句话说, 如果一个客户端 IP 地址在黑名单中, 它将在 `access_by_lua` (page 51) 中的 Mysql 复杂认证请求之前被拒绝。

注意, `ngx_auth_request` 模块可以近似的被 `access_by_lua` (page 51) 实现:

```
location / {
    auth_request /auth;

    # proxy_pass/fastcgi_pass/postgres_pass/...
}
```

使用 `ngx_lua` 是这样：

```
location / {
    access_by_lua '
        local res = ngx.location.capture("/auth")

        if res.status == ngx.HTTP_OK then
            return
        end

        if res.status == ngx.HTTP_FORBIDDEN then
            ngx.exit(res.status)
        end

        ngx.exit(ngx.HTTP_INTERNAL_SERVER_ERROR)
    ';

    # proxy_pass/fastcgi_pass/postgres_pass/...
}
```

和其他 `access` 阶段处理实现一样，[access_by_lua \(page 51\)](#) 将不能运行在子请求中。

注意，在 [access_by_lua \(page 51\)](#) 处理内部，当调用 `ngx.exit(ngx.OK)` 时，nginx 请求将继续下一阶段的内容处理。要在 [access_by_lua \(page 51\)](#) 处理中终结当前请求，调用 `ngx.exit (page 132)`，成功的请求设定 `status >= 200` (`ngx.HTTP_OK`) 并 `status < 300` (`ngx.HTTP_SPECIAL_RESPONSE`)，失败的请求设定 `ngx.exit(ngx.HTTP_INTERNAL_SERVER_ERROR)` (或其他相关的)。

从 `v0.9.20` 版本开始，你可以使用 [access_by_lua_no_postpone \(page 75\)](#) 指令来控制该 handler 在 Nginx 的“access”请求处理中的执行时机。

[返回目录 \(page 30\)](#)

access_by_lua_block

语法: `access_by_lua_block { lua-script }`

环境: *http, server, location, location if*

阶段: *access tail*

与 [access_by_lua \(page 51\)](#) 指令相似，只不过该指令在一对括号 (`{ }`) 中直接内嵌 Lua 代码，替代之前 Nginx 的字符串（需要特殊字符转义）。

For instance,

```
access_by_lua_block {  
    do_something("hello, world!\nhiya\n")  
}
```

该指令在 `v0.9.17` 版本首次引入。

access_by_lua_file

语法: *access_by_lua_file <path-to-lua-script-file>*

环境: *http, server, location, location if*

阶段: *access tail*

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 [access_by_lua \(page 51\)](#) 是等价的，该指令从 `v0.5.0rc32` 开始支持 Lua/LuaJIT 字节码 (page 0) 的执行。

在 `<path-to-lua-script-file>` 中可以使用 Nginx 的内置变量用来提高灵活性，然而这带有一定的风险，通常并不推荐使用。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

当 Lua 代码缓存开启（默认），用户代码在第一次请求时完成加载（只有一次）并缓存，当 Lua 文件被修改时，每次都要对 Nginx 配置进行重新加载。Lua 代码缓存是可以暂时被禁用，通过开关 [lua_code_cache \(page 34\)](#) 在 `nginx.conf` 中设置为 `off`，这样就可以避免反复重新加载 Nginx。

支持通过 Nginx 变量完成动态调度文件路径，就像 [content_by_lua_file \(page 46\)](#) 一样。

[返回目录 \(page 30\)](#)

header_filter_by_lua

语法: *header_filter_by_lua <lua-script-str>*

环境： *http, server, location, location if*

阶段： *output-header-filter*

注意 自从 **v0.9.17** 版本, 使用该指令是 *不优雅* 的, 应使用新的 [header_filter_by_lua_block \(page 54\)](#) 指令进行替代。

用 **<lua-script-str>** 中指定的lua代码, 来完成应答消息头部的过滤。

注意, 下列的接口函数在这个执行阶段是被禁用的:

- 输出类函数 (例: ngx.say 和 ngx.send_headers)
- 控制类函数 (例: ngx.redirect 和 ngx.exec)
- 子请求相关函数 (例: ngx.location.capture和ngx.location.capture_multi)
- cosocket 类函数 (例: ngx.socket.tcp 和 ngx.req.socket)

这里有个使用 Lua 过滤完成覆盖应答头的例子 (如果没有则添加):

```
location / {  
    proxy_pass http://mybackend;  
    header_filter_by_lua 'ngx.header.Foo = "blah";'  
}
```

该指令在版本 **v0.2.1rc20** 中第一次引入。

[返回目录 \(page 30\)](#)

header_filter_by_lua_block

语法: *header_filter_by_lua_block { lua-script }*

环境: *http, server, location, location if*

阶段: *output-header-filter*

与 [header_filter_by_lua \(page 53\)](#) 指令相似, 只不过该指令在一对括号 (**{ }**) 中直接内嵌 Lua 代码, 替代之前 Nginx 的字符串 (需要特殊字符转义)。

例如:

```
header_filter_by_lua_block {  
    ngx.header["content-length"] = nil  
}
```

该指令在 **v0.9.17** 版本首次引入。

[返回目录 \(page 30\)](#)

header_filter_by_lua_file

语法: `header_filter_by_lua_file <path-to-lua-script-file>`

环境: `http, server, location, location if`

阶段: `output-header-filter`

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 [header_filter_by_lua \(page 53\)](#) 是等价的，该指令从 **v0.5.0rc32** 开始支持 Lua/LuaJIT 字节码 (page 0) 的执行。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

该指令是在 **v0.2.1rc20** 版本第一次引入。

[返回目录 \(page 30\)](#)

body_filter_by_lua

语法: `body_filter_by_lua <lua-script-str>`

环境: `http, server, location, location if`

阶段: `output-body-filter`

注意 自从 **v0.9.17** 版本，使用该指令是 *不优雅* 的，应使用新的 [body_filter_by_lua_block \(page 57\)](#) 指令进行替代。

使用 `<lua-script-str>` 指定的 Lua 代码定义一个输出应答体过滤器。

输入数据块是 `ngx.arg (page 87)[1]` (Lua 的字符串形式)，结束标识“eof”是应答体数据最后一位 `ngx.arg (page 87)[2]` (Lua 的布尔值形式)。

在这个场景下，结束标识“eof”仅仅是 Nginx chain 缓冲区的 `last_buf` (主请求) 或 `last_in_chain` (子请求)。(在 **v0.7.14** 版本之前，结束标识“eof”在子请求中是完全不能使用的。)

使用下面 Lua 代码，可以对输出数据流立即终止：

```
return ngx.ERROR
```

这样截断响应体，通常导致结果不完整的，也是无效的响应。

本指令的 Lua 代码可以使用 Lua 字符串或字符串的表重写 [ngx.arg \(page 87\)](#)[1] 输入数据块内容，从而完成 Nginx 输出体下游过滤数据修改。例如，在输出体转换所有的小写字母，我们可以这样用：

```
location / {
    proxy_pass http://mybackend;
    body_filter_by_lua 'ngx.arg[1] = string.upper(ngx.ar
g[1])';
}
```

当设置 `nil` 或一个空的 Lua 字符串值给 `ngx.arg[1]`，将没有任何数据块下发到 Nginx 下游。

同样，新的结束标识“eof”也可以通过对 [ngx.arg \(page 87\)](#)[2] 设定一个布尔值。例如：

```
location /t {
    echo hello world;
    echo hiya globe;

    body_filter_by_lua '
        local chunk = ngx.arg[1]
        if string.match(chunk, "hello") then
            ngx.arg[2] = true -- new eof
            return
        end

        -- just throw away any remaining chunk data
        ngx.arg[1] = nil
    ';
}
```

然后 `GET /t` 将返回下面的结果：

```
hello world
```

就是说，当应答体过滤发现一个块包含关键字“hello”，它将立即设置结束标识“eof”为 true，应答内容被截断尽管后面还有有效数据。

当 Lua 代码可能改变应答体的长度时，我们必须总是清空响应头中的 `Content-Length`（如果有），强制使用流式输出，如：

```
location /foo {  
    # fastcgi_pass/proxy_pass/...  
  
    header_filter_by_lua_block { ngx.header.content_length = n  
il }  
    body_filter_by_lua_block { ngx.arg[1] = string.len(ngx.ar  
g[1]) .. "\\n" };  
}
```

注意：下面这些 API 函数在这个环境中是禁用的，这受制于当前 Nginx 输出过滤器的实现：

- 输出 API 函数类（例如：[ngx.say \(page 131\)](#) 和 [ngx.send_headers \(page 129\)](#)）
- 控制 API 函数类（例如：[ngx.redirect \(page 128\)](#) 和 [ngx.exec \(page 126\)](#)）
- 子请求函数类（例如：[ngx.location.capture \(page 96\)](#) 和 [ngx.location.capture_multi \(page 103\)](#)）
- cosocket 函数类（例如：[ngx.socket.tcp \(page 170\)](#) 和 [ngx.req.socket \(page 125\)](#)）

Nginx 输出过滤器在一个单独请求中可能被调用多次，因为应答体可能使用块的方式进行投递。所以，本指令中的 Lua 代码在这个单独的 HTTP 请求生命周期内，同样会执行多次。

该指令在 [v0.5.0rc32](#) 版本中首次引入。

[返回目录 \(page 30\)](#)

body_filter_by_lua_block

语法: `body_filter_by_lua_block { lua-script-str }`

环境: `http, server, location, location if`

阶段: `output-body-filter`

与 [body_filter_by_lua* \(page 55\)](#) 指令相似，只不过该指令在一对括号（`{ }`）中直接内嵌 Lua 代码，替代之前 Nginx 的字符串（需要特殊字符转义）。

例如：

```
body_filter_by_lua_block {  
    local data, eof = ngx.arg[1], ngx.arg[2]  
}
```

该指令在 **v0.9.17** 版本首次引入。

[返回目录 \(page 30\)](#)

body_filter_by_lua_file

语法: `body_filter_by_lua_file <path-to-lua-script-file>`

环境: `http, server, location, location if`

阶段: `output-body-filter`

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 `body_filter_by_lua*` (page 55) 是等价的，该指令从 **v0.5.0rc32** 开始支持 Lua/LuaJIT 字节码 (page 0) 的执行。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

该指令是在 **v0.5.0rc32** 版本第一次引入。

[返回目录 \(page 30\)](#)

log_by_lua

语法: `log_by_lua <lua-script-str>`

环境: `http, server, location, location if`

阶段: `log`

注意 自从 **v0.9.17** 版本，使用该指令是 *不优雅* 的，应使用新的 `log_by_lua_block` (page 61) 指令进行替代。

在 `log` 请求处理阶段执行内嵌在 `<lua-script-str>` 的 Lua 代码。它不替代当前 `access` 的日志，而是在其前面执行。

注意，当前环境中以下 API 函数当前是被禁用的：

- 输出API函数类（例如：`ngx.say` (page 131) 和 `ngx.send_headers` (page 129)）
- 控制API函数类（例如：`ngx.exit` (page 132) 和 `ngx.exec` (page 126)）
- 子请求函数类（例如：`ngx.location.capture` (page 96) 和

[ngx.location.capture_multi \(page 103\)](#))

- cosocket 函数类 (例如 : [ngx.socket.tcp \(page 170\)](#) 和 [ngx.req.socket \(page 125\)](#))

这是一个收集 [\\$upstream_response_time](#) 平均处理的例子 :

```

lua_shared_dict log_dict 5M;

server {
    location / {
        proxy_pass http://mybackend;

        log_by_lua '
            local log_dict = ngx.shared.log_dict
            local upstream_time = tonumber(ngx.var.upstream_re
sponse_time)

            local sum = log_dict:get("upstream_time-sum") or 0
            sum = sum + upstream_time
            log_dict:set("upstream_time-sum", sum)

            local newval, err = log_dict:incr("upstream_time-n
b", 1)
            if not newval and err == "not found" then
                log_dict:add("upstream_time-nb", 0)
                log_dict:incr("upstream_time-nb", 1)
            end
        ';
    }

    location = /status {
        content_by_lua_block {
            local log_dict = ngx.shared.log_dict
            local sum = log_dict:get("upstream_time-sum")
            local nb = log_dict:get("upstream_time-nb")

            if nb and sum then
                ngx.say("average upstream response time: ", su
m / nb,
                        " (" , nb, " reqs)")
            else
                ngx.say("no data yet")
            end
        }
    }
}

```

该指令在 `v0.5.0rc31` 版本被首次引入。

[返回目录 \(page 30\)](#)

log_by_lua_block

语法: `log_by_lua_block { lua-script }`

内容: `http, server, location, location if`

阶段: `log`

与 [log_by_lua \(page 58\)](#) 指令相似，只不过该指令在一对括号 (`{ }`) 中直接内嵌 Lua 代码，替代之前 Nginx 的字符串（需要特殊字符转义）。

例如：

```
log_by_lua_block {  
    print("I need no extra escaping here, for example: \r\nbl  
h")  
}
```

该指令在 `v0.9.17` 版本首次引入。

[返回目录 \(page 30\)](#)

log_by_lua_file

语法: `log_by_lua_file <path-to-lua-script-file>`

环境: `http, server, location, location if`

阶段: `log`

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 [log_by_lua \(page 58\)](#) 是等价的，该指令从 `v0.5.0rc32` 开始支持 Lua/LuaJIT 字节码 ([page 0](#)) 的执行。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

该指令是在 `v0.5.0rc31` 版本第一次引入。

[返回目录 \(page 30\)](#)

balancer_by_lua_block

语法: `balancer_by_lua_block { lua-script }`

环境: *upstream*

阶段: *content*

该指令执行上游的负载均衡 Lua 代码（任何上游实体），代码配置在 `upstream {}` 小节中。

举例：

```
upstream foo {
    server 127.0.0.1;
    balancer_by_lua_block {
        -- 使用 Lua 作为一个动态均衡器完成一些有趣的事情
    }
}

server {
    location / {
        proxy_pass http://foo;
    }
}
```

Lua 的负载均衡可以和任何已经存在的 nginx 上游模块一起工作，例如：[ngx_proxy](#) 和 [ngx_fastcgi](#)。

同样，Lua 负载均衡可以和标准的上游连接池机制一起工作，例如标准的 [keepalive](#) 指令。只要确保在单个 `upstream {}` 配置小节中 [keepalive](#) 指令要放在 `balancer_by_lua_block` 小节的后面。

Lua 负载均衡能完全忽略配置在 `upstream {}` 小节中定义的服务列表，并且从一个完全动态的服务列表中挑选一个节点（甚至每次请求都在变），所有这些均是通过 [lua-resty-core](#) 库的 [ngx.balancer](#) 完成。

该指令配置的 Lua 代码在单个下游请求中可能被调用多次，例如使用 [proxy_next_upstream](#) 配置小节，这是 nginx 自身上游尝试请求机制。

这里 Lua 代码的执行环境不支持 `yield` 操作，所以可能 `yield` 的 Lua API（例如 [cosockets](#) 和“轻线程”），在这个环境中是被禁用的。一个可以使用并绕过这个限制的玩法，是可以在更早的阶段处理，比如 [access_by_lua*](#) ([page 51](#)))，并传递结果到环境的 [ngx.ctx](#) ([page 93](#)) 表。

该指令在 [v0.10.0](#) 版本首次引入。

[返回目录 \(page 30\)](#)

balancer_by_lua_file

语法: `balancer_by_lua_file <path-to-lua-script-file>`

环境: *upstream*

阶段: *content*

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 [balancer_by_lua_block](#) (page 61) 是等价的，该指令从 `v0.5.0rc32` 开始支持 Lua/LuaJIT 字节码 (page 0) 的执行。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

该指令在 `v0.10.0` 版本首次引入。

[返回目录](#) (page 30)

lua_need_request_body

语法: `*lua_need_request_body <on off>*`

默认: *off*

环境: *http, server, location, location if*

阶段: *depends on usage*

在运行 `rewrite/access/access_by_lua*` 之前决定是否强制获取请求体数据。Nginx 内部默认不读取客户端请求体，如果需要读取请求体数据，需要使用该指令设置为 `on` 或者在 Lua 代码中调用 [ngx.req.read_body](#) (page 120) 函数。

为了读取请求体数据到 `$request_body` 变量，`client_body_buffer_size` 必须要与 `client_max_body_size` 有同样的大小。因为内容大小超过 `client_body_buffer_size` 但是小于 `client_max_body_size` 时，Nginx 将把缓冲内存数据存到一个磁盘的临时文件上，这将导致 `$request_body` 变量是一个空值。

如果当前 location 包含 `rewrite_by_lua[/#rewrite_by_lua]` 指令，请求体将在 [\[rewrite_by_lua](#) (page 46) 代码运行之前（还是在 `rewrite` 阶段）被读取。如果只有 [content_by_lua](#) (page 45) 指令，请求体直到内容生成的 Lua 代码执行时才会读取（既，请求体在处理生成返回数据阶段才回被读取）。

无论如何都非常推荐，使用 [ngx.req.read_body](#) (page 120) 和 [ngx.req.discard_body](#) (page 121) 函数，可以更好的控制请求体的读取过程。

这个规则也适用于 [access_by_lua*](#) (page 51)。

[返回目录](#) (page 30)

ssl_certificate_by_lua_block

语法: `ssl_certificate_by_lua_block { lua-script }`

环境: `server`

阶段: `right-before-SSL-handshake`

当 Nginx 开始对下游进行 SSL (https) 握手连接时, 该指令执行用户 Lua 代码。

特别是基于每个请求, 设置 SSL 证书链与相应的私有密钥, 这种情况特别有用。通过非阻塞 IO 操作, 从远程 (例如, 使用 [cosocket](#) (page 170) API) 加载 SSL 握手配置也是很有用的。并且在每请求中使用纯 Lua 完成 OCSP stapling 处理也是可以的。

另一个典型应用场景是在当前环境中非阻塞的方式完成 SSL 握手信号控制, 例如在 [lua-resty-limit-traffic](#) 库的辅助下。

我们也可以针对来自客户端的 SSL 握手请求做一些有趣的处理, 比如可以有选择地拒绝使用了 SSL v3 甚至更低版本协议的老客户端。

[ngx.ssl](#) 和 [ngx.ocsp](#) Lua 模块是由 [lua-resty-core](#) 库提供, 并且在该环境中特别有用。你可以使用这两个模块提供的 Lua API, 处理当前 SSL 连接初始化的 SSL 证书链和私有密钥。

不管怎样, 对于当前 SSL 连接, 在 Nginx/OpenSSL 通过 SSL session IDs 或 TLS session tickets 成功唤醒之前, 该 Lua 是不会运行的。换句话说, 这个 Lua 只有当 Nginx 已经发起了完整的 SSL 握手才执行。

下面是个简陋的与 [ngx.ssl](#) 模块一起使用的例子:

```
server {
    listen 443 ssl;
    server_name test.com;

    ssl_certificate_by_lua_block {
        print("About to initiate a new SSL handshake!")
    }

    location / {
        root html;
    }
}
```

更多信息，可以参考 [ngx.ssl](#) 的更多复杂例子 和 [ngx.ocsp](#) Lua 模块的官方文档。

在用户 Lua 代码中未捕获的 Lua 异常将立即终止当前 SSL 请求，就如同使用 `ngx.ERROR` 错误码调用 [ngx.exit \(page 132\)](#)。

该环境下的 Lua 代码执行 支持 yielding，所以可能 yield 的 Lua API 在这个环境中是启用的（例如 cosockets，sleeping，和“轻线程”）。

注意，无论如何，你仍然需要配置 [ssl_certificate](#) 和 [ssl_certificate_key](#) 指令，尽管你将完全不再使用这个静态的证书和私有密钥。这是因为 Nginx 内核要求它们出现否则启动 Nginx 时你将看到下面错误：

```
nginx: [emerg] no ssl configured for the server
```

该指令需要能工作的 Nginx 补丁版本地址：

<http://mailman.nginx.org/pipermail/nginx-devel/2016-January/007748.html>

对于 OpenResty 1.9.7.2（或更高）绑定的 Nginx 版本，已经默认打上了补丁。

此外，该指令需要至少 OpenSSL [1.0.2e](#) 版本才能工作。

该指令是在 [v0.10.0](#) 版本首次引入。

[返回目录 \(page 30\)](#)

ssl_certificate_by_lua_file

语法: `ssl_certificate_by_lua_file <path-to-lua-script-file>`

环境: `server`

阶段: `right-before-SSL-handshake`

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 [ssl_certificate_by_lua_block \(page 64\)](#) 是等价的，该指令从 [v0.5.0rc32](#) 开始支持 Lua/LuaJIT 字节码 [\(page 0\)](#) 的执行。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

该指令是在 [v0.10.0](#) 版本首次引入。

[返回目录 \(page 30\)](#)

ssl_session_fetch_by_lua_block

语法: `ssl_session_fetch_by_lua_block { lua-script }`

环境: *http*

阶段: *right-before-SSL-handshake*

该指令执行的代码，根据当前下游的 SSL 握手请求中的会话 ID，查找并加载 SSL 会话（如果有）。

由 [lua-resty-core](#) Lua 模块库内置的 `ngx.ssl.session` API，可以获取当前会话 ID 并加载一个已缓存的 SSL 缓存数据。

Lua API 可能会挂起，比如 [ngx.sleep \(page 134\)](#) 和 [cosockets \(page 170\)](#)，在这个环境中是启用的。

该钩子可以与 [ssl_session_store_by_lua*](#) ([page 67](#)) 一起使用，实现纯 Lua 的分布式缓存模型（例如基于 [cosocket \(page 170\)](#) API）。如果找到一个已缓存 SSL 会话，将会加载到当前 SSL 会话环境中，SSL 会话将立即启动恢复，绕过昂贵的完整 SSL 握手过程（这里有非常昂贵 CPU 计算代价）。

请注意，TLS 会话票证是非常不同的，当使用会话票证时它是客户端完成 SSL 会话状态缓存。SSL 会话恢复是基于 TLS 会话票证自动完成，不需要该钩子参与（也不需要 [ssl_session_store_by_lua_block \(page 0\)](#) 钩子）。该钩子主要是给老版本或缺少 SSL 客户端能力（只能通过会话 ID 方式完成 SSL 会话）。

当同时指定了 [ssl_certificate_by_lua\]\(#ssl_certificate_by_lua_block\)](#)，该钩子通常在 [\[ssl_certificate_by_lua \(page 64\)](#) 之前运行。找到 SSL 会话并成功对当前 SSL 连接加载后，SSL 会话将会恢复，从而绕过 [ssl_certificate_by_lua* \(page 64\)](#) 钩子。这种情况下，NGINX 也将直接绕过 [ssl_session_store_by_lua_block \(page 0\)](#) 钩子，不需要了嘛。

借助现代网络浏览器，在本地是比较容易测试这个钩子的。你可以暂时把下面这行配置放到 `https server` 小节，禁用 TLS 会话票证。

```
ssl_session_tickets off;
```

但是在你把网站放到外网之前，不要忘记注释掉这行配置。

如果你使用 [OpenResty 1.11.2.1](#) 或后续版本绑定的 [官方的预编译包](#)，那么一切都应只欠东风。

如果你正在使用的不是 [OpenResty](#) 提供的 OpenSSL 库，你需要对 OpenSSL 1.0.2h 或后续版本打个补丁：

https://github.com/openresty/openresty/blob/master/patches/openssl-1.0.2h-sess_set_get_cb_yield.patch

如果你没有使用 [OpenResty 1.11.2.1](#) 或后续版本绑定的 Nginx，那么你需要对标准 Nginx 1.11.2 或后续版本打个补丁：

http://openresty.org/download/nginx-1.11.2-nonblocking_ssl_handshake_hooks.patch

该小节在 **v0.10.6** 首次引入。

请注意: 从 **v0.10.7** 版本开始, 该指令只允许在 **http context** 环境中使用 (因为 SSL 会话唤醒发生在服务名生效之前)。

[返回目录 \(page 30\)](#)

ssl_session_fetch_by_lua_file

语法: `ssl_session_fetch_by_lua_file <path-to-lua-script-file>`

环境: *http*

阶段: *right-before-SSL-handshake*

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外, 该指令与 [ssl_session_fetch_by_lua_block \(page 65\)](#) 是等价的, 该指令支持 [Lua/LuaJIT 字节码 \(page 0\)](#) 的执行。

当给定了一个相对路径如 `foo/bar.lua`, 它将会被转换成绝对路径, 前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

该指令在 **v0.10.6** 版本首次引入。

请注意: 从 **v0.10.7** 版本开始, 该指令只允许在 **http context** 环境中使用 (因为 SSL 会话唤醒发生在服务名生效之前)。

[返回目录 \(page 30\)](#)

ssl_session_store_by_lua_block

语法: `ssl_session_store_by_lua_block { lua-script }`

环境: *http*

阶段: *right-after-SSL-handshake*

该指令执行的代码, 根据当前下游的 SSL 握手请求中的会话 ID, 获取并保存 SSL 会话 (如果有)。

This directive runs Lua code to fetch and save the SSL session (if any) according to the session ID provided by the current SSL handshake request for the downstream.

被保存或缓存的 SSL 会话数据能被用到将来的 SSL 连接, 恢复 SSL 会话却不需要历经完整 SSL 握手过程 (这里有非常昂贵 CPU 计算代价)。

The saved or cached SSL session data can be used for future SSL connections to resume SSL sessions without going through the full SSL handshake process (which is very expensive in terms of CPU time).

Lua API 可能会挂起，比如 [ngx.sleep \(page 134\)](#) 和 [cosockets \(page 170\)](#)，在这个环境中被禁用了。尽管如此，你仍然可以通过 [ngx.timer.at \(page 190\)](#) API 来创建一个零延迟的 timer 用来异步方式保存 SSL 会话数据到外部服务中（比如 `redis` 或 `memcached`）。

由 [lua-resty-core](#) Lua 模块库提供的 `ngx.ssl.session` API，可以获取当前会话 ID 并关联到会话状态数据。

借助现代网络浏览器，在本地是更容易测试这个钩子的。你可以暂时把下面这行配置放到 `https server` 小节，禁用 TLS 回话票证。

```
ssl_session_tickets off;
```

但是在你把网站放到外网之前，不要忘记注释掉这行配置。

该指令在 `v0.10.6` 版本首次引入。

请注意：从 `v0.10.7` 版本开始，该指令只允许在 **http context** 环境中使用（因为 SSL 会话唤醒发生在服务名生效之前）。

[返回目录 \(page 30\)](#)

ssl_session_store_by_lua_file

语法: `ssl_session_store_by_lua_file <path-to-lua-script-file>`

环境: `http`

阶段: `right-before-SSL-handshake`

除了通过文件 `<path-to-lua-script-file>` 的内容指定 Lua 代码外，该指令与 [ssl_session_store_by_lua_block \(page 67\)](#) 是等价的，该指令支持 [Lua/LuaJIT 字节码 \(page 0\)](#) 的执行。

当给定了一个相对路径如 `foo/bar.lua`，它将会被转换成绝对路径，前面增加的部分路径是 Nginx 服务启动时通过命令行选项 `-p PATH` 决定的 `server prefix`。

该指令在 `v0.10.6` 版本首次引入。

请注意：从 `v0.10.7` 版本开始，该指令只允许在 **http context** 环境中使用（因为 SSL 会话唤醒发生在服务名生效之前）。

[返回目录 \(page 30\)](#)

lua_shared_dict

语法: `lua_shared_dict <name> <size>`

默认: `no`

环境: `http`

阶段: `depends on usage`

声明一个共享内存区块 `<name>`，用来存储基于共享内存的 Lua 字典 `ngx.shared.<name>`。

在当前 Nginx 服务器实例中，共享内存区块被所有 nginx worker 进程共享。

`<size>` 参数可以通过类似 `k` 和 `m` 的大小单位来设置。

```
http {
    lua_shared_dict dogs 10m;
    ...
}
```

硬编码限制最小大小是 8KB，而实际的最小大小取决于实际中用户数据集（有些人是从 12KB 开始）。

更多细节请参考 [ngx.shared.DICT \(page 155\)](#)。

这个指令最早出现在版本 `v0.3.1rc22` 中。

[返回目录 \(page 30\)](#)

lua_socket_connect_timeout

语法: `lua_socket_connect_timeout <time>`

默认: `lua_socket_connect_timeout 60s`

环境: `http, server, location`

该指令控制 TCP/unix-domain socket 对象的 [connect \(page 171\)](#) 方法默认超时时间，这个值可以被 [settimeout \(page 179\)](#) 或 [settimeouts \(page 179\)](#) 方法覆盖。

`<time>` 参数可以是整数，后面可以跟着像 `s` (秒)，`ms` (毫秒)，`m` (分钟) 的单位可选项。默认的时间单位是 `s`，也就是“秒”。默认值是 `60s`。

这个指令最早出现在版本 `v0.5.0rc1` 中。

[返回目录 \(page 30\)](#)

lua_socket_send_timeout

语法: `lua_socket_send_timeout <time>`

默认: `lua_socket_send_timeout 60s`

环境: `http, server, location`

该指令控制 TCP/unix-domain socket 对象的 [send \(page 174\)](#) 方法默认超时时间，这个值可以被 [settimeout \(page 179\)](#) 或 [settimeouts \(page 179\)](#) 方法覆盖。

`<time>` 参数可以是整数，后面可以跟着像 `s` (秒), `ms` (毫秒), `m` (分钟) 的单位可选项。默认的时间单位是 `s`，也就是“秒”。默认值是 `60s`。

该指令是在 `v0.5.0rc1` 版本第一次引入。

[返回目录 \(page 30\)](#)

lua_socket_send_lowat

语法: `lua_socket_send_lowat <size>`

默认: `lua_socket_send_lowat 0`

环境: `http, server, location`

控制 cosocket 发送缓冲区 `lowat`（低水位）的值。

[返回目录 \(page 30\)](#)

lua_socket_read_timeout

语法: `lua_socket_read_timeout <time>`

默认: `lua_socket_read_timeout 60s`

环境: `http, server, location`

阶段: 依赖于使用环境

该指令控制 TCP/unix-domain socket 对象的 [receive \(page 175\)](#) 方法、[receiveuntil \(page 176\)](#) 方法返回迭代函数的默认超时时间。这个值可以被 [settimeout \(page 179\)](#) 或 [settimeouts \(page 179\)](#) 方法覆盖。

`<time>` 参数可以是整数，后面可以跟着像 `s` (秒), `ms` (毫秒), `m` (分钟) 的单位可选项。默认的时间单位是 `s`，也就是“秒”。默认值是 `60s`。

该指令是在 **v0.5.0rc1** 版本第一次引入。

[返回目录 \(page 30\)](#)

lua_socket_buffer_size

语法: *lua_socket_buffer_size <size>*

默认: *lua_socket_buffer_size 4k/8k*

环境: *http, server, location*

指定使用 **cosocket** 进行读取操作时的缓冲区大小。

这个缓冲区不必为了同时解决所有事情而设置的太大，因为 **cosocket** 支持 100% 的非缓存读取和解析。所以即使是 **1** 字节的缓冲区大小依旧可以在任何地方正常工作，只不过效率比较糟糕。

该指令是在 **v0.5.0rc1** 版本首次引入。

[返回目录 \(page 30\)](#)

lua_socket_pool_size

语法: *lua_socket_pool_size <size>*

默认: *lua_socket_pool_size 30*

环境: *http, server, location*

指定每个 **cosocket** 通过远程服务(例如，使用主机+端口配对或 **unix socket** 文件路径作为标识)关联的连接池的大小限制（每个地址中的连接数）。

每个连接池默认是 30 个连接。

当连接池中连接数超过限制大小，在连接池中最近最少使用的（空闲）连接将被关闭，给当前连接腾挪空间。

注意，**cosocket** 连接池是每个 **Nginx** 工作进程使用的，而不是每个 **Nginx** 服务实例，所以这里指定的限制也只能在每个独立的 **nginx** 工作进程上生效。

该指令在 **v0.5.0rc1** 版本首次引入。

[返回目录 \(page 30\)](#)

lua_socket_keepalive_timeout

语法: *lua_socket_keepalive_timeout <time>*

默认: *lua_socket_keepalive_timeout 60s*

环境: *http, server, location*

该指令控制在 `cosocket` 连接池中连接的默认最大空闲时间。当这个时间到达，空闲的连接将被关闭并从连接池中移除。这个值可以使用 `cosocket` 对象的 [setkeepalive \(page 180\)](#) 方法覆盖。

<time> 参数可以是整数，后面可以跟着像 `s` (秒)，`ms` (毫秒)，`m` (分钟) 的单位可选项。默认的时间单位是 `s`，也就是“秒”。默认值是 `60s`。

这个指令最早出现在版本 `v0.5.0rc1`。

[返回目录 \(page 30\)](#)

lua_socket_log_errors

语法: `*lua_socket_log_errors on off*`

默认: `lua_socket_log_errors on`

环境: *http, server, location*

当 TCP 或 UDP `cosockets` 出现失败时，该指令可被用来切换错误日志输出。如果你已经正确处理了你的 Lua 代码错误日志，这里就推荐设置当前指令的开关为 `off`，防止数据刷写到你的 nginx 错误日志文件（通常这个代价是比较昂贵的）。

这个指令最早出现在版本 `v0.5.13` 中。

[返回目录 \(page 30\)](#)

lua_ssl_ciphers

语法: `lua_ssl_ciphers <ciphers>`

默认: `lua_ssl_ciphers DEFAULT`

环境: *http, server, location*

指定在 [tcpsock:sslhandshake \(page 173\)](#) 方法中请求 SSL/TLS 服务的加密方式。其中参数 `ciphers` 是 OpenSSL 库里面指定的格式。

可以使用“`openssl ciphers`”来查看完整的加密方式列表。

该指令是在 `v0.9.11` 版本首次引入的。

[返回目录 \(page 30\)](#)

lua_ssl_crl

语法: `lua_ssl_crl <file>`

默认: `no`

环境: `http, server, location`

指定一个 PEM 格式吊销证书文件，在 [tcpsock:sslhandshake \(page 173\)](#) 方法里验证 SSL/TLS 服务的证书。

该指令是在 **v0.9.11** 版本首次引入的。

[返回目录 \(page 30\)](#)

lua_ssl_protocols

语法: `lua_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2]`

默认: `lua_ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2`

环境: `http, server, location`

在 [tcpsock:sslhandshake \(page 173\)](#) 方法中开启请求 SSL/TLS 服务的传输协议列表。

该指令是在 **v0.9.11** 版本首次引入的。

[返回目录 \(page 30\)](#)

lua_ssl_trusted_certificate

语法: `lua_ssl_trusted_certificate <file>`

默认: `no`

环境: `http, server, location`

指定一个 PEM 格式信任 CA 证书文件，在 [tcpsock:sslhandshake \(page 173\)](#) 方法里验证 SSL/TLS 服务的证书。

该指令是在 **v0.9.11** 版本首次引入的。

也可以看看 [lua_ssl_verify_depth \(page 74\)](#).

[返回目录 \(page 30\)](#)

lua_ssl_verify_depth

语法: `lua_ssl_verify_depth <number>`

默认: `lua_ssl_verify_depth 1`

环境: `http, server, location`

设置服务端证书链的验证深度。

该指令是在 `v0.9.11` 版本首次引入的。

也可以看看 [lua_ssl_trusted_certificate \(page 73\)](#)。

[返回目录 \(page 30\)](#)

lua_http10_buffering

语法: `*lua_http10_buffering on off*`

默认: `lua_http10_buffering on`

环境: `http, server, location, location-if`

对 HTTP 1.0 (或更老) 请求, 启用或禁用自动应答缓冲区。这个缓冲机制主要用于应答头包含合适 `Content-Length` 长度的 HTTP 1.0 长连接。

如果 Lua 代码在发送应答头之前明确设置了应答头的 `Content-Length` (调用 [ngx.send_headers \(page 129\)](#) 或 隐式首次调用 [ngx.say \(page 131\)](#) 或 [ngx.print \(page 130\)](#) 其中任何一个), HTTP 1.0 应答缓冲区都将被禁用, 即使这个指令是打开的。

流式输出 (例如, 调用 [ngx.flush \(page 132\)](#)) 非常大的应答体, 为了占用内存最小, 该指令必须设置为 `off`。

该指令默认值是 `on`。

该指令是在 `v0.5.0rc19` 版本首次引入的。

[返回目录 \(page 30\)](#)

rewrite_by_lua_no_postpone

语法: `*rewrite_by_lua_no_postpone on off*`

默认: `rewrite_by_lua_no_postpone off`

环境: `http`

控制是否禁用 `rewrite_by_lua*` (page 46) 指令在 `rewrite` 阶段的延迟执行。该指令的默认值是 `off`，在 `rewrite` 阶段的 Lua 代码将被延迟到最后执行。

该指令是在 `v0.5.0rc29` 版本首次引入的。

[返回目录 \(page 30\)](#)

access_by_lua_no_postpone

语法: `*access_by_lua_no_postpone on off*`

默认: `access_by_lua_no_postpone off`

环境: `http`

控制是否禁用 `access_by_lua*` (page 51) 指令在 `access` 请求处理阶段末尾的推迟执行。默认的，该指令是 `off` 并且 `access` 阶段的 Lua 代码是被延迟到末尾执行。

该指令在 `v0.9.20` 版本首次引入。

[返回目录 \(page 30\)](#)

lua_transform_underscores_in_response_headers

语法: `*lua_transform_underscores_in_response_headers on off*`

默认: `lua_transform_underscores_in_response_headers on`

环境: `http, server, location, location-if`

对于 `ngx.header.HEADER` (page 105) API 中指定响应头，该指令指定是否将下划线 (`_`) 转化为连接线 (`-`)。

该指令是在 `v0.5.0rc32` 版本首次引入的。

[返回目录 \(page 30\)](#)

lua_check_client_abort

语法: `*lua_check_client_abort on off*`

默认: `lua_check_client_abort off`

环境: `http, server, location, location-if`

该指令控制是否探测客户端连接的过早终止。

当启用该指令，`ngx_lua` 模块将会在下游连接上监控连接过早关闭事件。当有这样的
事件时，它将调用用户指定 Lua 的回调函数（通过 [ngx.on_abort \(page 189\)](#) 注册），当这里没有用户回调函数注册时，将停止当前请求并清理所有当前请求中运行的
Lua “轻线程”。

根据目前实现，无论如何，如果请求正在通过 [ngx.req.socket \(page 125\)](#) 读取请求
体，在它之前客户连接发生关闭，`ngx_lua` 将不会停止任何正在执行的“轻线程”也不会
调用用户的回调（尽管已经调用 [ngx.on_abort \(page 189\)](#)）。作为替代，使用
[ngx.req.socket \(page 125\)](#) 的读操作第二个参数将直接返回错误信息 “client
aborted” 作为返回值（第一个返回值确定是 `nil`）。

当 TCP 长连接被禁用，它依靠客户端 socket 关闭的优雅实现（通过发送一个 `FIN`
包或类似的东西）。

对与（软）实时 Web 应用，强烈推荐使用系统 TCP 协议栈支持的选项对 [TCP
keepalive](#) 进行配置，以便及时发现“半开” TCP 连接。

例如，在 Linux，在你的 `nginx.conf` 文件中你能使用标准 `listen` 指令配置，像这
样：

```
listen 80 so_keepalive=2s:2s:8;
```

对于 FreeBSD，你可以只调整 TCP 长连接的系统范围的配置，例如：

```
# sysctl net.inet.tcp.keepintvl=2000  
# sysctl net.inet.tcp.keeptimeout=2000
```

该指令是在 `v0.7.4` 版本首次引入的。

也可以看看 [ngx.on_abort \(page 189\)](#)。

[返回目录 \(page 30\)](#)

lua_max_pending_timers

语法: `lua_max_pending_timers <count>`

默认: `lua_max_pending_timers 1024`

环境: `http`

控制允许使用的 `pending timers` 最大数量。

`pending timers` 指的是还没有过期的 `timers`。

当超过这个限制，`ngx.timer.at` (page 190) 调用将立即返回 `nil` 和 错误信息 “too many pending timers”。

该指令是在 `v0.8.0` 版本首次引入的。

[返回目录 \(page 30\)](#)

lua_max_running_timers

语法: `lua_max_running_timers <count>`

默认: `lua_max_running_timers 256`

环境: `http`

控制允许的 `running timers` 最大数量。

`running timers` 指的是那些正在执行用户回调函数的 `timers`。

当超过这个限制，Nginx 将停止执行新近过期的 `timers` 回调，并记录一个错误日志 “N lua_max_running_timers are not enough”，这里的 “N” 是这个指令的当前值。

该指令是在 `v0.8.0` 版本首次引入的。

[返回目录 \(page 30\)](#)

Nginx API for Lua

- [Name \(page 3\)](#)
- [Table of Contents \(page 3\)](#)
- [Status \(page 4\)](#)
- [Version \(page 4\)](#)

- [Synopsis \(page 5\)](#)
- [Description \(page 8\)](#)
- [Typical Uses \(page 9\)](#)
- [Nginx Compatibility \(page 10\)](#)
- [Installation \(page 11\)](#)
 - [Building as a dynamic module \(page 12\)](#)
 - [C Macro Configurations \(page 13\)](#)
 - [Installation on Ubuntu 11.10 \(page 13\)](#)
- [Community \(page 14\)](#)
 - [英文邮件列表 \(page 14\)](#)
 - [中文邮件列表 \(page 14\)](#)
- [Code Repository \(page 14\)](#)
- [Bugs and Patches \(page 14\)](#)
- [Lua/LuaJIT 字节码 support \(page 14\)](#)
- [System Environment Variable Support \(page 16\)](#)
- [HTTP 1.0 support \(page 16\)](#)
- [Statically Linking Pure Lua Modules \(page 16\)](#)
- [Data Sharing within an Nginx Worker \(page 18\)](#)
- [Known Issues \(page 20\)](#)
 - [TCP socket 连接操作遗留问题 \(page 20\)](#)
 - [Lua 协程 Yielding/Resuming \(page 20\)](#)
 - [Lua Variable Scope \(page 20\)](#)
 - [Locations Configured by Subrequest Directives of Other Modules \(page 22\)](#)
 - [Cosockets Not Available Everywhere \(page 22\)](#)
 - [Special Escaping Sequences \(page 23\)](#)
 - [Mixing with SSI Not Supported \(page 25\)](#)
 - [SPDY Mode Not Fully Supported \(page 25\)](#)
 - [Missing data on short circuited requests \(page 25\)](#)
- [TODO \(page 26\)](#)

- [Changes \(page 27\)](#)
- [Test Suite \(page 27\)](#)
- [Copyright and License \(page 28\)](#)
- [See Also \(page 29\)](#)
- [Directives \(page 30\)](#)
 - [lua_use_default_type \(page 33\)](#)
 - [lua_malloc_trim \(page 34\)](#)
 - [lua_code_cache \(page 34\)](#)
 - [lua_regex_cache_max_entries \(page 35\)](#)
 - [lua_regex_match_limit \(page 36\)](#)
 - [lua_package_path \(page 36\)](#)
 - [lua_package_cpath \(page 37\)](#)
 - [init_by_lua \(page 37\)](#)
 - [init_by_lua_block \(page 39\)](#)
 - [init_by_lua_file \(page 39\)](#)
 - [init_worker_by_lua \(page 40\)](#)
 - [init_worker_by_lua_block \(page 41\)](#)
 - [init_worker_by_lua_file \(page 42\)](#)
 - [set_by_lua \(page 42\)](#)
 - [set_by_lua_block \(page 44\)](#)
 - [set_by_lua_file \(page 44\)](#)
 - [content_by_lua \(page 45\)](#)
 - [content_by_lua_block \(page 45\)](#)
 - [content_by_lua_file \(page 46\)](#)
 - [rewrite_by_lua \(page 46\)](#)
 - [rewrite_by_lua_block \(page 50\)](#)
 - [rewrite_by_lua_file \(page 50\)](#)
 - [access_by_lua \(page 51\)](#)
 - [access_by_lua_block \(page 52\)](#)

- [access_by_lua_file](#) (page 53)
- [header_filter_by_lua](#) (page 53)
- [header_filter_by_lua_block](#) (page 54)
- [header_filter_by_lua_file](#) (page 55)
- [body_filter_by_lua](#) (page 55)
- [body_filter_by_lua_block](#) (page 57)
- [body_filter_by_lua_file](#) (page 58)
- [log_by_lua](#) (page 58)
- [log_by_lua_block](#) (page 61)
- [log_by_lua_file](#) (page 61)
- [balancer_by_lua_block](#) (page 61)
- [balancer_by_lua_file](#) (page 63)
- [lua_need_request_body](#) (page 63)
- [ssl_certificate_by_lua_block](#) (page 64)
- [ssl_certificate_by_lua_file](#) (page 65)
- [ssl_session_fetch_by_lua_block](#) (page 65)
- [ssl_session_fetch_by_lua_file](#) (page 67)
- [ssl_session_store_by_lua_block](#) (page 67)
- [ssl_session_store_by_lua_file](#) (page 68)
- [lua_shared_dict](#) (page 69)
- [lua_socket_connect_timeout](#) (page 69)
- [lua_socket_send_timeout](#) (page 70)
- [lua_socket_send_lowat](#) (page 70)
- [lua_socket_read_timeout](#) (page 70)
- [lua_socket_buffer_size](#) (page 71)
- [lua_socket_pool_size](#) (page 71)
- [lua_socket_keepalive_timeout](#) (page 71)
- [lua_socket_log_errors](#) (page 72)
- [lua_ssl_ciphers](#) (page 72)

- [lua_ssl_crl \(page 73\)](#)
- [lua_ssl_protocols \(page 73\)](#)
- [lua_ssl_trusted_certificate \(page 73\)](#)
- [lua_ssl_verify_depth \(page 74\)](#)
- [lua_http10_buffering \(page 74\)](#)
- [rewrite_by_lua_no_postpone \(page 75\)](#)
- [access_by_lua_no_postpone \(page 75\)](#)
- [lua_transform_underscores_in_response_headers \(page 75\)](#)
- [lua_check_client_abort \(page 76\)](#)
- [lua_max_pending_timers \(page 77\)](#)
- [lua_max_running_timers \(page 77\)](#)
- [Nginx API for Lua \(page 77\)](#)
 - [Introduction \(page 86\)](#)
 - [ngx.arg \(page 87\)](#)
 - [ngx.var.VARIABLE \(page 88\)](#)
 - [Core constants \(page 89\)](#)
 - [HTTP method constants \(page 90\)](#)
 - [HTTP status constants \(page 90\)](#)
 - [Nginx log level constants \(page 92\)](#)
 - [print \(page 92\)](#)
 - [ngx.ctx \(page 93\)](#)
 - [ngx.location.capture \(page 96\)](#)
 - [ngx.location.capture_multi \(page 103\)](#)
 - [ngx.status \(page 105\)](#)
 - [ngx.header.HEADER \(page 105\)](#)
 - [ngx.resp.get_headers \(page 107\)](#)
 - [ngx.req.is_internal \(page 108\)](#)
 - [ngx.req.start_time \(page 108\)](#)
 - [ngx.req.http_version \(page 109\)](#)

- [ngx.req.raw_header](#) (page 109)
- [ngx.req.get_method](#) (page 110)
- [ngx.req.set_method](#) (page 110)
- [ngx.req.set_uri](#) (page 111)
- [ngx.req.set_uri_args](#) (page 113)
- [ngx.req.get_uri_args](#) (page 113)
- [ngx.req.get_post_args](#) (page 115)
- [ngx.req.get_headers](#) (page 118)
- [ngx.req.set_header](#) (page 119)
- [ngx.req.clear_header](#) (page 120)
- [ngx.req.read_body](#) (page 120)
- [ngx.req.discard_body](#) (page 121)
- [ngx.req.get_body_data](#) (page 121)
- [ngx.req.get_body_file](#) (page 122)
- [ngx.req.set_body_data](#) (page 123)
- [ngx.req.set_body_file](#) (page 123)
- [ngx.req.init_body](#) (page 123)
- [ngx.req.append_body](#) (page 124)
- [ngx.req.finish_body](#) (page 125)
- [ngx.req.socket](#) (page 125)
- [ngx.exec](#) (page 126)
- [ngx.redirect](#) (page 128)
- [ngx.send_headers](#) (page 129)
- [ngx.headers_sent](#) (page 130)
- [ngx.print](#) (page 130)
- [ngx.say](#) (page 131)
- [ngx.log](#) (page 131)
- [ngx.flush](#) (page 132)
- [ngx.exit](#) (page 132)

- [ngx.eof](#) (page 133)
- [ngx.sleep](#) (page 134)
- [ngx.escape_uri](#) (page 135)
- [ngx.unescape_uri](#) (page 135)
- [ngx.encode_args](#) (page 135)
- [ngx.decode_args](#) (page 137)
- [ngx.encode_base64](#) (page 137)
- [ngx.decode_base64](#) (page 138)
- [ngx.crc32_short](#) (page 138)
- [ngx.crc32_long](#) (page 138)
- [ngx.hmac_sha1](#) (page 139)
- [ngx.md5](#) (page 139)
- [ngx.md5_bin](#) (page 140)
- [ngx.sha1_bin](#) (page 140)
- [ngx.quote_sql_str](#) (page 141)
- [ngx.today](#) (page 141)
- [ngx.time](#) (page 141)
- [ngx.now](#) (page 142)
- [ngx.update_time](#) (page 142)
- [ngx.localtime](#) (page 142)
- [ngx.utctime](#) (page 143)
- [ngx.cookie_time](#) (page 143)
- [ngx.http_time](#) (page 144)
- [ngx.parse_http_time](#) (page 144)
- [ngx.is_subrequest](#) (page 145)
- [ngx.re.match](#) (page 145)
- [ngx.re.find](#) (page 149)
- [ngx.re.gmatch](#) (page 150)
- [ngx.re.sub](#) (page 152)

- [ngx.re.gsub \(page 154\)](#)
- [ngx.shared.DICT \(page 155\)](#)
- [ngx.shared.DICT.get \(page 157\)](#)
- [ngx.shared.DICT.get_stale \(page 158\)](#)
- [ngx.shared.DICT.set \(page 158\)](#)
- [ngx.shared.DICT.safe_set \(page 160\)](#)
- [ngx.shared.DICT.add \(page 160\)](#)
- [ngx.shared.DICT.safe_add \(page 160\)](#)
- [ngx.shared.DICT.replace \(page 161\)](#)
- [ngx.shared.DICT.delete \(page 161\)](#)
- [ngx.shared.DICT.incr \(page 162\)](#)
- [ngx.shared.DICT.lpush \(page 163\)](#)
- [ngx.shared.DICT.rpush \(page 163\)](#)
- [ngx.shared.DICT.lpop \(page 163\)](#)
- [ngx.shared.DICT.rpop \(page 164\)](#)
- [ngx.shared.DICT.llen \(page 164\)](#)
- [ngx.shared.DICT.flush_all \(page 165\)](#)
- [ngx.shared.DICT.flush_expired \(page 165\)](#)
- [ngx.shared.DICT.get_keys \(page 166\)](#)
- [ngx.socket.udp \(page 166\)](#)
- [udpsock:setpeername \(page 167\)](#)
- [udpsock:send \(page 168\)](#)
- [udpsock:receive \(page 168\)](#)
- [udpsock:close \(page 169\)](#)
- [udpsock:settimeout \(page 170\)](#)
- [ngx.socket.stream \(page 170\)](#)
- [ngx.socket.tcp \(page 170\)](#)
- [tcpsock:connect \(page 171\)](#)
- [tcpsock:sslhandshake \(page 173\)](#)

- [tcpsock:send \(page 174\)](#)
- [tcpsock:receive \(page 175\)](#)
- [tcpsock:receiveuntil \(page 176\)](#)
- [tcpsock:close \(page 178\)](#)
- [tcpsock:settimeout \(page 179\)](#)
- [tcpsock:settimeouts \(page 179\)](#)
- [tcpsock:setoption \(page 180\)](#)
- [tcpsock:setkeepalive \(page 180\)](#)
- [tcpsock:getreusedtimes \(page 181\)](#)
- [ngx.socket.connect \(page 181\)](#)
- [ngx.get_phase \(page 182\)](#)
- [ngx.thread.spawn \(page 183\)](#)
- [ngx.thread.wait \(page 186\)](#)
- [ngx.thread.kill \(page 189\)](#)
- [ngx.on_abort \(page 189\)](#)
- [ngx.timer.at \(page 190\)](#)
- [ngx.timer.running_count \(page 193\)](#)
- [ngx.timer.pending_count \(page 193\)](#)
- [ngx.config.subsystem \(page 194\)](#)
- [ngx.config.debug \(page 194\)](#)
- [ngx.config.prefix \(page 194\)](#)
- [ngx.config.nginx_version \(page 195\)](#)
- [ngx.config.nginx_configure \(page 195\)](#)
- [ngx.config.ngx_lua_version \(page 195\)](#)
- [ngx.worker.exiting \(page 195\)](#)
- [ngx.worker.pid \(page 196\)](#)
- [ngx.worker.count \(page 196\)](#)
- [ngx.worker.id \(page 196\)](#)
- [ngx.semaphore \(page 197\)](#)

- [ngx.balancer \(page 197\)](#)
- [ngx.ssl \(page 197\)](#)
- [ngx.ocsp \(page 198\)](#)
- [ndk.set_var.DIRECTIVE \(page 198\)](#)
- [coroutine.create \(page 199\)](#)
- [coroutine.resume \(page 200\)](#)
- [coroutine.yield \(page 200\)](#)
- [coroutine.wrap \(page 200\)](#)
- [coroutine.running \(page 201\)](#)
- [coroutine.status \(page 201\)](#)
- [Obsolete Sections \(page 201\)](#)
 - [Special PCRE Sequences \(page 201\)](#)

[返回目录 \(page 3\)](#)

Introduction

`nginx.conf` 文件中各种 `*_by_lua`、`*_by_lua_block` 和 `*_by_lua_file` 配置指令的作用是提供 Lua API 的网关。下面介绍的这些 Lua API 指令，只能在上述配置指令的环境中，通过用户 Lua 代码调用。

Lua 中使用的 API 以两个标准模块的形式封装：`ngx` 和 `ndk`。这两个模块在 `ngx_lua` 默认的全局作用域中，在 `ngx_lua` 指令中总是可用。

这两个模块可以被用在外部 Lua 模块中，例如：

```
local say = ngx.say

local _M = {}

function _M.foo(a)
    say(a)
end

return _M
```

强烈不推荐使用 `package.seeall` 标记，因为它有很多不好的副作用。

在外部 Lua 模块中也可以直接 `require` 这两个模块：

```
local ngx = require "ngx"
local ndk = require "ndk"
```

自 **v0.2.1rc19** 版开始可以 `require` 这两个模块。

用户代码中的网络 I/O 操作应该使用这些 Nginx Lua API 实现，否则 Nginx 的事件循环可能被阻塞，从而严重影响性能。相对小数据量的磁盘操作可以通过标准的 Lua **io** 库来实现，但大规模的文件读写如果可能应该避免，因为可能会严重阻塞 Nginx 进程。为获得最好性能，强烈建议将所有网络和磁盘 I/O 操作发送到 Nginx 的子请求中 (通过类似 [ngx.location.capture \(page 96\)](#) 的方法) 处理。

[返回目录 \(page 77\)](#)

ngx.arg

语法: `val = ngx.arg[index]`

环境: `set_by_lua*`, `body_filter_by_lua*`

当被用在 [set_by_lua* \(page 42\)](#) 指令环境中时，本表是一个只读表，包含输入参数供配置命令使用：

```
value = ngx.arg[n]
```

例如，

```
location /foo {
    set $a 32;
    set $b 56;

    set_by_lua $sum
        'return tonumber(ngx.arg[1]) + tonumber(ngx.arg[2])'
        $a $b;

    echo $sum;
}
```

将输出 **88**，是 **32** 和 **56** 的和。

当被用在 [body_filter_by_lua* \(page 55\)](#) 指令中时，本表第一个元素是送给输出过滤器的输入数据块，第二个元素是“eof”布尔标记，用以标识整个输出数据流是否结束。

可以通过直接给相应的表元素赋值，设置送给下游 Nginx 输出过滤器的数据块和“eof”标记。当给 `ngx.arg[1]` 赋值 `nil` 或 Lua 空字符串时，将不发送任何数据给下游的 Nginx 输出过滤器。

[返回目录 \(page 77\)](#)

ngx.var.VARIABLE

语法: `ngx.var.VAR_NAME`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`

读写 Nginx 变量值。

```
value = ngx.var.some_nginx_variable_name
ngx.var.some_nginx_variable_name = value
```

请注意，只有已经定义的 nginx 变量可以被写入。例如，

```
location /foo {
    set $my_var ''; # 需要在设置时创建 $my_var 变量
    content_by_lua_block {
        ngx.var.my_var = 123;
        ...
    }
}
```

也就是说，nginx 变量无法“随用随创建”。

一些特殊的 nginx 变量，比如 `$args` 和 `$limit_rate`，可以被赋值，但许多其他的变量不能，包括 `$query_string`，`$arg_PARAMETER`，和 `$http_NAME` 等。

Nginx 正则表达式捕获组变量 `$1`、`$2`、`$3` 等，也可以通过这个界面读取，方式为通过 `ngx.var[1]`、`ngx.var[2]`、`ngx.var[3]` 等。

设置 `ngx.var.Foo` 为 `nil` 值将删除 Nginx 变量 `$Foo`。

```
ngx.var.args = nil
```

注意 当从 Nginx 变量中读取值时，Nginx 将从基于请求的内存池中分配内存，只有在请求中止时才释放。所以如果用户的 Lua 代码中需要反复读取 Nginx 变量，请在用户程序的 Lua 变量中缓存，例如，

```
local val = ngx.var.some_var
--- 在后面反复使用变量 val
```

以避免在当前请求周期内的 (临时) 内存泄露。另外一个缓存结果的方法是使用 [ngx.ctx \(page 93\)](#) 表。

未定义的 Nginx 变量会被认定为 `nil`，而未初始化（但已定义）的 Nginx 变量会被认定为空 Lua 字符串。

这个 API 需要进行相对“昂贵”的元方法调用，所以请避免高频使用。

[返回目录 \(page 77\)](#)

Core constants

环境: `init_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `*log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`

```
ngx.OK (0)
ngx.ERROR (-1)
ngx.AGAIN (-2)
ngx.DONE (-4)
ngx.DECLINED (-5)
```

请注意，这些常量中只有三个可以被 [Nginx API for Lua \(page 77\)](#) 使用 (即 [ngx.exit \(page 132\)](#) 只接受 `NGX_OK`，`NGX_ERROR`，和 `NGX_DECLINED` 作为输入)。

```
ngx.null
```

`ngx.null` 常量是一个 `NULL` 的 [轻量用户数据](#)，一般被用来表达 Lua table 等里面的 `nil` (空) 值，类似于 [lua-cjson](#) 库中的 `cjson.null` 常量。在 `v0.5.0rc5` 版本中首次引入这个常量。

`ngx.DECLINED` 这个常量在 `v0.5.0rc19` 版本中首次引入。

[返回目录 \(page 77\)](#)

HTTP method constants

环境: *init_by_lua**, *set_by_lua**, *rewrite_by_lua**, *access_by_lua**, *content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**, *ngx.timer.**, *balancer_by_lua**, *ssl_certificate_by_lua**, *ssl_session_fetch_by_lua**, *ssl_session_store_by_lua**

```
ngx.HTTP_GET
ngx.HTTP_HEAD
ngx.HTTP_PUT
ngx.HTTP_POST
ngx.HTTP_DELETE
ngx.HTTP_OPTIONS    (v0.5.0rc24 版本加入)
ngx.HTTP_MKCOL      (v0.8.2 版本加入)
ngx.HTTP_COPY        (v0.8.2 版本加入)
ngx.HTTP_MOVE        (v0.8.2 版本加入)
ngx.HTTP_PROPFIND    (v0.8.2 版本加入)
ngx.HTTP_PROPPATCH   (v0.8.2 版本加入)
ngx.HTTP_LOCK        (v0.8.2 版本加入)
ngx.HTTP_UNLOCK      (v0.8.2 版本加入)
ngx.HTTP_PATCH        (v0.8.2 版本加入)
ngx.HTTP_TRACE        (v0.8.2 版本加入)
```

这些常量一般被用在 [ngx.location.capture \(page 96\)](#) 和 [ngx.location.capture_multi \(page 103\)](#) 方法中。

[返回目录 \(page 77\)](#)

HTTP status constants

环境: *init_by_lua**, *set_by_lua**, *rewrite_by_lua**, *access_by_lua**, *content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**, *ngx.timer.**, *balancer_by_lua**, *ssl_certificate_by_lua**, *ssl_session_fetch_by_lua**, *ssl_session_store_by_lua**


```
value = ngx.HTTP_CONTINUE (100) (first added in the v0.9.20
release)
  value = ngx.HTTP_SWITCHING_PROTOCOLS (101) (first added in t
he v0.9.20 release)
  value = ngx.HTTP_OK (200)
  value = ngx.HTTP_CREATED (201)
  value = ngx.HTTP_ACCEPTED (202) (first added in the v0.9.20
release)
  value = ngx.HTTP_NO_CONTENT (204) (first added in the v0.9.2
0 release)
  value = ngx.HTTP_PARTIAL_CONTENT (206) (first added in the v
0.9.20 release)
  value = ngx.HTTP_SPECIAL_RESPONSE (300)
  value = ngx.HTTP_MOVED_PERMANENTLY (301)
  value = ngx.HTTP_MOVED_TEMPORARILY (302)
  value = ngx.HTTP_SEE_OTHER (303)
  value = ngx.HTTP_NOT_MODIFIED (304)
  value = ngx.HTTP_TEMPORARY_REDIRECT (307) (first added in th
e v0.9.20 release)
  value = ngx.HTTP_BAD_REQUEST (400)
  value = ngx.HTTP_UNAUTHORIZED (401)
  value = ngx.HTTP_PAYMENT_REQUIRED (402) (first added in the
v0.9.20 release)
  value = ngx.HTTP_FORBIDDEN (403)
  value = ngx.HTTP_NOT_FOUND (404)
  value = ngx.HTTP_NOT_ALLOWED (405)
  value = ngx.HTTP_NOT_ACCEPTABLE (406) (first added in the v
0.9.20 release)
  value = ngx.HTTP_REQUEST_TIMEOUT (408) (first added in the v
0.9.20 release)
  value = ngx.HTTP_CONFLICT (409) (first added in the v0.9.20
release)
  value = ngx.HTTP_GONE (410)
  value = ngx.HTTP_UPGRADE_REQUIRED (426) (first added in the
v0.9.20 release)
  value = ngx.HTTP_TOO_MANY_REQUESTS (429) (first added in th
e v0.9.20 release)
  value = ngx.HTTP_CLOSE (444) (first added in the v0.9.20 rel
ease)
  value = ngx.HTTP_ILLEGAL (451) (first added in the v0.9.20 r
elease)
  value = ngx.HTTP_INTERNAL_SERVER_ERROR (500)
  value = ngx.HTTP_METHOD_NOT_IMPLEMENTED (501)
  value = ngx.HTTP_BAD_GATEWAY (502) (first added in the v
0.9.20 release)
```

```

value = ngx.HTTP_SERVICE_UNAVAILABLE (503)
value = ngx.HTTP_GATEWAY_TIMEOUT (504) (first added in the v
0.3.1rc38 release)
value = ngx.HTTP_VERSION_NOT_SUPPORTED (505) (first added i
n the v0.9.20 release)
value = ngx.HTTP_INSUFFICIENT_STORAGE (507) (first added in
the v0.9.20 release)

```

[返回目录 \(page 77\)](#)

Nginx log level constants

环境: *init_by_lua**, *init_worker_by_lua**, *set_by_lua**, *rewrite_by_lua**,
*access_by_lua**, *content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**,
*log_by_lua**, *ngx.timer.**, *balancer_by_lua**, *ssl_certificate_by_lua**,
*ssl_session_fetch_by_lua**, *ssl_session_store_by_lua**

```

ngx.STDERR
ngx.EMERG
ngx.ALERT
ngx.CRIT
ngx.ERR
ngx.WARN
ngx.NOTICE
ngx.INFO
ngx.DEBUG

```

这些常量一般用于 [ngx.log \(page 131\)](#) 方法。

[返回目录 \(page 77\)](#)

print

语法: *print(...)*

环境: *init_by_lua**, *init_worker_by_lua**, *set_by_lua**, *rewrite_by_lua**,
*access_by_lua**, *content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**,
*log_by_lua**, *ngx.timer.**, *balancer_by_lua**, *ssl_certificate_by_lua**,
*ssl_session_fetch_by_lua**, *ssl_session_store_by_lua**

将参数值以 `ngx.NOTICE` 日志级别写入 nginx 的 `error.log` 文件。Writes argument values into the nginx `error.log` file with the `ngx.NOTICE` log level.

等同于

```
ngx.log(ngx.NOTICE, ...)
```

Lua 的 `nil` 值输出 `"nil"` 字符串，Lua 的布尔值输出 `"true"` 或 `"false"` 字符串。`ngx.null` 常量输出为 `"null"` 字符串。

在 Nginx 内核中硬编码限制了单条错误信息最长为 `2048` 字节。这个长度包含了最后的换行符和开始的时间戳。如果信息长度超过这个限制，Nginx 将把信息文本截断。这个限制可以通过修改 Nginx 源码中 `src/core/nginx_log.h` 文件中的 `NGX_MAX_ERROR_STR` 宏定义调整。

[返回目录 \(page 77\)](#)

ngx.ctx

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`

这个 Lua 表可以用来存储基于请求的 Lua 环境数据，其生存周期与当前请求相同 (类似 Nginx 变量)。

参考下面例子，

```
location /test {
    rewrite_by_lua_block {
        ngx.ctx.foo = 76
    }
    access_by_lua_block {
        ngx.ctx.foo = ngx.ctx.foo + 3
    }
    content_by_lua_block {
        ngx.say(ngx.ctx.foo)
    }
}
```

访问 `GET /test` 输出

```
79
```

也就是说, `ngx.ctx.foo` 条目跨越一个请求的 `rewrite` (重写), `access` (访问), 和 `content` (内容) 各处理阶段保持一致。

每个请求, 包括子请求, 都有一份自己的 `ngx.ctx` 表。例如:

```
location /sub {
    content_by_lua_block {
        ngx.say("sub pre: ", ngx.ctx.blah)
        ngx.ctx.blah = 32
        ngx.say("sub post: ", ngx.ctx.blah)
    }
}

location /main {
    content_by_lua_block {
        ngx.ctx.blah = 73
        ngx.say("main pre: ", ngx.ctx.blah)
        local res = ngx.location.capture("/sub")
        ngx.print(res.body)
        ngx.say("main post: ", ngx.ctx.blah)
    }
}
```

访问 `GET /main` 输出

```
main pre: 73
sub pre: nil
sub post: 32
main post: 73
```

这里, 在子请求中修改 `ngx.ctx.blah` 条目并不影响父请求中的同名条目, 因为它们各自维护不同版本的 `ngx.ctx.blah`。

内部重定向将摧毁原始请求中的 `ngx.ctx` 数据 (如果有), 新请求将会有有一个空白的 `ngx.ctx` 表。例如,

```
location /new {
    content_by_lua_block {
        ngx.say(ngx.ctx.foo)
    }
}

location /orig {
    content_by_lua_block {
        ngx.ctx.foo = "hello"
        ngx.exec("/new")
    }
}
```

访问 `GET /orig` 将输出

```
nil
```

而不是原始的 `"hello"` 值。

任意数据值，包括 Lua 闭包与嵌套表，都可以被插入这个“魔法”表，也允许注册自定义元方法。

也可以将 `ngx.ctx` 覆盖为一个新 Lua 表，例如，

```
ngx.ctx = { foo = 32, bar = 54 }
```

当用在 [init_worker_by_lua*](#) (page 40) 环境中，这个表与当前 Lua 句柄生命周期相同。

`ngx.ctx` 表查询需要相对昂贵的元方法调用，这比通过用户自己的函数参数直接传递基于请求的数据要慢得多。所以不要为了节约用户函数参数而滥用此 API，因为它可能对性能有明显影响。

而且由于元方法“魔法”，不要在 lua 模块级别试图使用“local”级别的 `ngx.ctx`，例如 [worker-level data sharing](#) (page 18)。下面示例是糟糕的：

```

-- mymodule.lua
local _M = {}

-- 下面一行的 ngx.ctx 是属于单个请求的，但 `ctx` 变量是在 Lua 模块级别
-- 并且属于单个 worker 的。
local ctx = ngx.ctx

function _M.main()
    ctx.foo = "bar"
end

return _M

```

应使用下面方式替代：

```

-- mymodule.lua
local _M = {}

function _M.main(ctx)
    ctx.foo = "bar"
end

return _M

```

就是说，调用者对 `ctx` 表调用应通过函数传参方式完成。

[返回目录 \(page 77\)](#)

ngx.location.capture

语法: `res = ngx.location.capture(uri, options?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

向 `uri` 发起一个同步非阻塞 Nginx 子请求。

Nginx 子请求是一种非常强有力的方式，它可以发起非阻塞的内部请求访问目标 location。目标 location 可以是配置文件中其他文件目录，或任何其他 nginx C 模块，包括 `ngx_proxy`、`ngx_fastcgi`、`ngx_memc`、`ngx_postgres`、`ngx_drizzle`，甚至 `ngx_lua` 自身等等。

需要注意的是，子请求只是模拟 HTTP 接口的形式，没有额外的 HTTP/TCP 流量，也没有 IPC (进程间通信) 调用。所有工作在内部高效地在 C 语言级别完成。

子请求与 HTTP 301/302 重定向指令 (通过 [ngx.redirect \(page 128\)](#)) 完全不同，也与内部重定向 (通过 [ngx.exec \(page 126\)](#)) 完全不同。

在发起子请求前，用户程序应总是读取完整的 HTTP 请求体 (通过调用 [ngx.req.read_body \(page 120\)](#) 或设置 [lua_need_request_body \(page 63\)](#) 指令为 on)。

该 API 方法 ([ngx.location.capture_multi \(page 103\)](#) 也一样) 总是缓冲整个请求体到内存中。因此，当需要处理一个大的子请求响应，用户程序应使用 [cosockets \(page 170\)](#) 进行流式处理，

下面是一个简单例子：

```
res = ngx.location.capture(uri)
```

返回一个包含四个元素的 Lua 表 ([res.status](#) , [res.header](#) , [res.body](#) , 和 [res.truncated](#))。

[res.status](#) (状态) 保存子请求的响应状态码。

[res.header](#) (头) 用一个标准 Lua 表储子请求响应的所有头信息。如果是“多值”响应头，这些值将使用 Lua (数组) 表顺序存储。例如，如果子请求响应头包含下面的行：

```
Set-Cookie: a=3
Set-Cookie: foo=bar
Set-Cookie: baz=blah
```

则 [res.header\["Set-Cookie"\]](#) 将存储 Lua 表 [{"a=3", "foo=bar", "baz=blah"}](#)。

[res.body](#) (体) 保存子请求的响应体数据，它可能被截断。用户需要检测 [res.truncated](#) (截断) 布尔值标记来判断 [res.body](#) 是否包含截断的数据。这种数据截断的原因只可能是因为子请求发生了不可恢复的错误，例如远端在发送响应体时过早中断了连接，或子请求在接收远端响应体时超时。

URI 请求串可以与 URI 本身连在一起，例如，

```
res = ngx.location.capture('/foo/bar?a=3&b=4')
```

因为 Nginx 内核限制，子请求不允许类似 [@foo](#) 命名 location。请使用标准 location，并设置 [internal](#) 指令，仅服务内部请求。

可选的选项表可以作为第二个参数传入，支持以下选项：

- `method` 指定子请求的请求方法，只接受类似 `ngx.HTTP_POST` 的常量。
- `body` 指定子请求的请求体 (仅接受字符串值)。
- `args` 指定子请求的 URI 请求参数 (可以是字符串或者 Lua 表)。
- `ctx` 指定一个 Lua 表作为子请求的 `ngx.ctx` (page 93) 表，可以是当前请求的 `ngx.ctx` (page 93) 表。这种方式可以让父请求和子请求共享完全相同的上下文环境。此选项最早出现在版本 `v0.3.1rc25` 中。
- `vars` 用一个 Lua 表设置子请求中的 Nginx 变量值。此选项最早出现在版本 `v0.3.1rc31` 中。
- `copy_all_vars` 设置是否复制所有当前请求的 Nginx 变量值到子请求中，修改子请求的 nginx 变量值不影响当前 (父) 请求。此选项最早出现在版本 `v0.3.1rc31` 中。
- `share_all_vars` 设置是否共享所有当前 (父) 请求的 Nginx 变量值到子请求中，修改子请求的 nginx 变量值将影响当前 (父) 请求。应用此选项将可能导致非常难以发现的错误，这种副作用是非常有害的。所以只有当完全明确知道自己在做什么时才打开此选项。
- `always_forward_body` 当设置为 `true` 时，如果没有设置 `body` 选项，当前 (父) 请求的请求体将被转发给子请求。被 `ngx.req.read_body()` (page 120) 或 `lua_need_request_body on` (page 63) 指令读取的请求体将直接转发给子请求，而不是在创建子请求时再复制整个请求体 (无论此请求体是缓存在内存中还是临时文件中)。默认情况下，此选项值为 `false`，当 `body` 选项没有设置，且当子请求的请求方法是 `PUT` 和 `POST` 时，当前 (父) 请求的请求体才被转发。

例如，发送一个 POST 子请求，可以这样做：

```
res = ngx.location.capture(  
    '/foo/bar',  
    { method = ngx.HTTP_POST, body = 'hello, world' }  
)
```

除了 POST 的其他 HTTP 请求方法请参考 [HTTP method constants](#) (page 90)。

`method` 选项默认值是 `ngx.HTTP_GET`。

`args` 选项可以设置附加的 URI 参数，例如：


```
ngx.location.capture('/foo?a=1',  
    { args = { b = 3, c = ':' } }  
)
```

等同于

```
ngx.location.capture('/foo?a=1&b=3&c=%3a')
```

也就是说，这个方法将根据 URI 规则转义参数键和值，并将它们拼接在一起组成一个完整的请求串。`args` 选项要求的 Lua 表的格式与 [ngx.encode_args \(page 135\)](#) 方法中使用的完全相同。

`args` 选项也可以直接包含 (转义过的) 请求串：

```
ngx.location.capture('/foo?a=1',  
    { args = 'b=3&c=%3a' } }  
)
```

这个例子与上个例子的功能相同。

`share_all_vars` 选项控制是否让当前请求与其子请求共享 nginx 变量。如果设为 `true`，则当前请求与所有子请求共享所有 nginx 变量作用域。因此，在子请求中修改 nginx 变量将影响当前请求。

使用此选项时需要非常小心，因为共享变量作用域容易导致难以预测的副作用。一般来说，推荐使用 `args`、`vars`、或 `copy_all_vars` 来代替。

此选项默认值是 `false`。

```
location /other {
    set $dog "$dog world";
    echo "$uri dog: $dog";
}

location /lua {
    set $dog 'hello';
    content_by_lua_block {
        res = ngx.location.capture("/other",
            { share_all_vars = true });

        ngx.print(res.body)
        ngx.say(ngx.var.uri, ": ", ngx.var.dog)
    }
}
```

请求 location `/lua` 将输出

```
/other dog: hello world
/lua: hello world
```

`copy_all_vars` 在子请求被创建时，提供给它一份父请求的 Nginx 变量拷贝。在子请求中对这些变量的修改将不会影响父请求和其他任何共享父请求变量的子请求。

```
location /other {
    set $dog "$dog world";
    echo "$uri dog: $dog";
}

location /lua {
    set $dog 'hello';
    content_by_lua_block {
        res = ngx.location.capture("/other",
            { copy_all_vars = true });

        ngx.print(res.body)
        ngx.say(ngx.var.uri, ": ", ngx.var.dog)
    }
}
```

请求 `GET /lua` 会输出

```
/other dog: hello world
/lu: hello
```

请注意，当 `share_all_vars` 和 `copy_all_vars` 都被设置为 `true` 时，`share_all_vars` 优先。

除了上面提到的两个设置方法外，也可以通过配置 `vars` 选项来设置子请求的变量值。在共享或复制父请求的 nginx 变量值后，子请求被设置这些值。和把特定参数通过 URL 参数编码传入子请求，再在 Nginx 配置文件中解码相比，这个方法效率更高。

```
location /other {
    content_by_lua_block {
        ngx.say("dog = ", ngx.var.dog)
        ngx.say("cat = ", ngx.var.cat)
    }
}

location /lua {
    set $dog '';
    set $cat '';
    content_by_lua_block {
        res = ngx.location.capture("/other",
            { vars = { dog = "hello", cat = 32 } });

        ngx.print(res.body)
    }
}
```

访问 `/lua` 将输出

```
dog = hello
cat = 32
```

选项 `ctx` 被用于给子请求设定一个自定义的 Lua 表作为 `ngx.ctx` (page 93) 表。

```
location /sub {
    content_by_lua_block {
        ngx.ctx.foo = "bar";
    }
}
location /lua {
    content_by_lua_block {
        local ctx = {}
        res = ngx.location.capture("/sub", { ctx = ctx })

        ngx.say(ctx.foo);
        ngx.say(ngx.ctx.foo);
    }
}
```

请求 `GET /lua` 输出

```
bar
nil
```

也可以通过 `ctx` 选项设置当前 (父) 请求与子请求共享同一个 [ngx.ctx \(page 93\)](#) 表。

```
location /sub {
    content_by_lua_block {
        ngx.ctx.foo = "bar";
    }
}
location /lua {
    content_by_lua_block {
        res = ngx.location.capture("/sub", { ctx = ngx.ctx })
        ngx.say(ngx.ctx.foo);
    }
}
```

请求 `GET /lua` 输出

```
bar
```

请注意，通过 [ngx.location.capture \(page 96\)](#) 创建的子请求默认继承当前请求的所有请求头信息，这有可能导致子请求响应中不可预测的副作用。例如，当使用标准的 `ngx_proxy` 模块服务子请求时，如果主请求头中包含 “Accept-Encoding: gzip”，可能导致子请求返回 Lua 代码无法正确处理的 gzip 压缩过的结果。通过设置 `proxy_pass_request_headers` 为 `off`，在子请求 location 中忽略原始请求头。

当没有设置 `body` 选项，且 `always_forward_body` 选项为 `false` (默认值) 时，`POST` 和 `PUT` 子请求将继承父请求的请求体 (如果有的话)。

Nginx 代码中有一个硬编码的数字，来控制每个主请求最多可以有多少个并发的子请求。在旧版 Nginx 中，这个数字是 `50`，自 Nginx `1.1.x` 以后，并发子请求数被提高到了 `200`。当超过这个限制时，`error.log` 文件中将出现下面的信息：

```
[error] 13983#0: *1 subrequests cycle while processing "/uri"
```

如果需要修改这个限制，可以手工在 Nginx 源代码树的 `nginx/src/http/ngx_http_request.h` 文件中修改 `NGX_HTTP_MAX_SUBREQUESTS` 宏定义。

请参考 [subrequest directives of other modules \(page 22\)](#) 了解目标 location 的配置限制。

[返回目录 \(page 77\)](#)

ngx.location.capture_multi

语法: `res1, res2, ... = ngx.location.capture_multi({ {uri, options?}, {uri, options?}, ... })`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

与 [ngx.location.capture \(page 96\)](#) 类似，并允许多个子请求并行访问。

此函数根据输入列表并行发起多个子请求，并按原顺序返回结果。例如，

```

res1, res2, res3 = ngx.location.capture_multi{
    { "/foo", { args = "a=3&b=4" } },
    { "/bar" },
    { "/baz", { method = ngx.HTTP_POST, body = "hello" } },
}

if res1.status == ngx.HTTP_OK then
    ...
end

if res2.body == "BLAH" then
    ...
end

```

此函数只有在所有子请求中止后才返回。总等待时间是耗时最长一个子请求的时间，而不是所有子请求等待时间的总和。

当子请求数量预先未知时，可以通过 Lua 表来存储所有请求和返回。

```

-- 创建请求表
local reqs = {}
table.insert(reqs, { "/mysql" })
table.insert(reqs, { "/postgres" })
table.insert(reqs, { "/redis" })
table.insert(reqs, { "/memcached" })

-- 同时执行所有请求，并等待所有返回
local resps = { ngx.location.capture_multi(reqs) }

-- 循环返回表
for i, resp in ipairs(resps) do
    -- 处理返回表 "resp"
end

```

[ngx.location.capture \(page 96\)](#) 函数是本函数的特殊形式。逻辑上说，[ngx.location.capture \(page 96\)](#) 函数可以按下面方法实现：

```

ngx.location.capture =
function (uri, args)
    return ngx.location.capture_multi({ {uri, args} })
end

```

请参考 [subrequest directives of other modules \(page 22\)](#) 了解目标 location 的配置限制。

[返回目录 \(page 77\)](#)

ngx.status

环境: *set_by_lua**, *rewrite_by_lua**, *access_by_lua**, *content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**

读写当前请求的响应状态码。这个方法需要在发送响应头前调用。

```
ngx.status = ngx.HTTP_CREATED
status = ngx.status
```

在发送响应头之后设置 `ngx.status` 不会生效，且 nginx 的错误日志中会有下面一条记录：

```
attempt to set ngx.status after sending out response headers
```

[返回目录 \(page 77\)](#)

ngx.header.HEADER

语法: *ngx.header.HEADER* = *VALUE*

语法: *value* = *ngx.header.HEADER*

环境: *rewrite_by_lua**, *access_by_lua**, *content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**

修改、添加、或清除当前请求待发送的 `HEADER` 响应头信息。

头名称中的下划线 (`_`) 将被默认替换为连字符 (`-`)。可以通过 [lua_transform_underscores_in_response_headers \(page 75\)](#) 指令关闭这个替换。

头名称大小写不敏感。

```
-- 与 ngx.header["Content-Type"] = 'text/plain' 相同
ngx.header.content_type = 'text/plain';

ngx.header["X-My-Header"] = 'blah blah';
```

多值头信息可以按以下方法设置：

```
ngx.header['Set-Cookie'] = {'a=32; path=/', 'b=4; path='}
```

在响应头中将输出：

```
Set-Cookie: a=32; path=/
Set-Cookie: b=4; path=/
```

只接受 Lua 表输入 (对于类似 `Content-Type` 之类的只接受一个值的标准头信息，只有 Lua 表中的最后一个元素有效)。

```
ngx.header.content_type = {'a', 'b'}
```

等同于

```
ngx.header.content_type = 'b'
```

将一个头信息的值设为 `nil` 将从响应头中移除该输出：

```
ngx.header["X-My-Header"] = nil;
```

赋值一个空 Lua 表效果相同：

```
ngx.header["X-My-Header"] = {};
```

在输出响应头 (不管是显示的通过 `ngx.send_headers` (page 129) 或隐式的通过类似 `ngx.print` (page 130) 指令) 以后设置 `ngx.header.HEADER` 将抛出 Lua 异常。

读取 `ngx.header.HEADER` 将返回响应头中名为 `HEADER` 的头信息的值。

读取时，头名称中的下划线 (`_`) 也会被替换成连字符 (`-`)，并且大小写不敏感。如果该头信息不存在，将返回 `nil`。

这个 API 在 [header_filter_by_lua*](#) (page 53) 环境中非常有用，例如：

```
location /test {
    set $footer '';

    proxy_pass http://some-backend;

    header_filter_by_lua '
        if ngx.header["X-My-Header"] == "blah" then
            ngx.var.footer = "some value"
        end
    ';

    echo_after_body $footer;
}
```

对于多值头信息，所有值将被按顺序放入一个 Lua 表中，例如，响应头

```
Foo: bar
Foo: baz
```

在读取 `ngx.header.Foo` 时返回的结果为

```
{"bar", "baz"}
```

需要注意的是，`ngx.header` 不是一个标准 Lua 表，不能通过 Lua 的 `ipairs` 函数进行迭代查询。

读取 请求 头信息，请使用 [ngx.req.get_headers](#) (page 118) 函数。

[返回目录](#) (page 77)

ngx.resp.get_headers

语法: `headers = ngx.resp.get_headers(max_headers?, raw?)`

环境: *set_by_lua**, *rewrite_by_lua**, *access_by_lua**, *content_by_lua**,
*header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**, *balancer_by_lua**

返回一个 Lua 表，包含当前请求的所有响应头信息。

```
local h = ngx.resp.get_headers()  
for k, v in pairs(h) do  
    ...  
end
```

此函数与 [ngx.req.get_headers \(page 118\)](#) 有相似之处，唯一区别是获取的是响应头信息而不是请求头信息。

这个 API 最早出现在 **v0.9.5** 版中。

[返回目录 \(page 77\)](#)

ngx.req.is_internal

语法: *is_internal* = *ngx.req.is_internal()*

环境: *set_by_lua**, *rewrite_by_lua**, *access_by_lua**, *content_by_lua**,
*header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**

返回一个布尔值，说明当前请求是否是一个“内部请求”，既：一个请求的初始化是在当前 nginx 服务端完成初始化，不是在客户端。

子请求都是内部请求，并且都是内部重定向后的请求。

该 API 在 **v0.9.20** 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.req.start_time

语法: *secs* = *ngx.req.start_time()*

环境: *set_by_lua**, *rewrite_by_lua**, *access_by_lua**, *content_by_lua**,
*header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**

返回当前请求创建时的时间戳，格式为浮点数，其中小数部分代表毫秒值。

以下用 Lua 代码模拟计算了 **\$request_time** 变量值 (由 [ngx_http_log_module](#) 模块生成)

```
local request_time = ngx.now() - ngx.req.start_time()
```

这个函数在 **v0.7.7** 版本中首次引入。

更多使用方法请参考 [ngx.now \(page 142\)](#) 和 [ngx.update_time \(page 142\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.http_version

语法: *num* = *ngx.req.http_version()*

环境: *set_by_lua**, *rewrite_by_lua**, *access_by_lua**, *content_by_lua**, *header_filter_by_lua**

返回一个 Lua 数字代表当前请求的 HTTP 版本号。

当前的可能结果值为 2.0, 1.0, 1.1 和 0.9。无法识别时值时返回 **nil**。

这个方法在 **v0.7.17** 版本中首次引入。

[返回目录 \(page 77\)](#)

ngx.req.raw_header

语法: *str* = *ngx.req.raw_header(no_request_line?)*

环境: *set_by_lua**, *rewrite_by_lua**, *access_by_lua**, *content_by_lua**, *header_filter_by_lua**

返回 Nginx 服务器接收到的原始 HTTP 协议头。

默认时，请求行和末尾的 **CR LF** 结束符也被包括在内。例如，

```
ngx.print(ngx.req.raw_header())
```

输出结果类似：

```
GET /t HTTP/1.1
Host: localhost
Connection: close
Foo: bar
```

可以通过指定可选的 `no_request_line` 参数为 `true` 来去除结果中的请求行。例如，

```
ngx.print(ngx.req.raw_header(true))
```

输出结果类似：

```
Host: localhost
Connection: close
Foo: bar
```

这个方法在 `v0.7.17` 版本中首次引入。

该方法还不能在 HTTP/2 请求中工作。

[返回目录 \(page 77\)](#)

ngx.req.get_method

语法: `method_name = ngx.req.get_method()`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `balancer_by_lua*`

获取当前请求的 HTTP 请求方法名称。结果为类似 `"GET"` 和 `"POST"` 的字符串，而不是 [HTTP 方法常量 \(page 90\)](#) 中定义的数值。

如果当前请求为 Nginx 子请求，将返回子请求的 HTTP 请求方法名称。

这个方法在 `v0.5.6` 版本中首次引入。

更多用法请参考 [ngx.req.set_method \(page 110\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.set_method

语法: `ngx.req.set_method(method_id)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`

用 `method_id` 参数的值改写当前请求的 HTTP 请求方法。当前仅支持 [HTTP 请求方法 \(page 90\)](#) 中定义的数值常量，例如 `ngx.HTTP_POST` 和 `ngx.HTTP_GET`。

如果当前请求是 Nginx 子请求，子请求的 HTTP 请求方法将被改写。

这个方法在 `v0.5.6` 版本中首次引入。

更多用法请参考 [ngx.req.get_method \(page 110\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.set_uri

语法: `ngx.req.set_uri(uri, jump?)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`

用 `uri` 参数重写当前请求 (已解析过的) URI。该 `uri` 参数必须是 Lua 字符串，并且长度不能是 0，否则将抛出 Lua 异常。

可选的布尔值参数 `jump` 会触发类似 [ngx_http_rewrite_module](#) 中 `rewrite` 指令的 location 重匹配 (或 location 跳转)。换句话说，当 `jump` 参数是 `true` (默认值 `false`) 时，此函数将不会返回，它会让 Nginx 在之后的 `post-rewrite` 执行阶段，根据新的 URI 重新搜索 location，并跳转到新 location。

默认值时，location 跳转不会被触发，只有当前请求的 URI 被改写。当 `jump` 参数值为 `false` 或不存在时，此函数将正常返回，但没有返回值。

例如，下面 nginx 配置片段

```
rewrite ^ /foo last;
```

可以通过 Lua 代码写成下面这样：

```
ngx.req.set_uri("/foo", true)
```

类似的，Nginx 配置

```
rewrite ^ /foo break;
```

可以通过 Lua 代码写成：

```
ngx.req.set_uri("/foo", false)
```

等同于写成：

```
ngx.req.set_uri("/foo")
```

`jump` 参数只可以在 `rewrite_by_lua*` (page 46) 指令中被设置为 `true`。不能在其他环境中使用 `jump`，否则将抛出 Lua 异常。

下面的示例复杂一些，包含正则表达式替换：

```
location /test {
    rewrite_by_lua '
        local uri = ngx.re.sub(ngx.var.uri, "^/test/(.*)",
"/$1", "o")
        ngx.req.set_uri(uri)
    ';
    proxy_pass http://my_backend;
}
```

功能上等同于：

```
location /test {
    rewrite ^/test/(.*) /$1 break;
    proxy_pass http://my_backend;
}
```

请注意，不能使用这个函数重写 URI 参数，应该使用 `ngx.req.set_uri_args` (page 113) 代替。例如，Nginx 配置

```
rewrite ^ /foo?a=3? last;
```

可以被写成

```
ngx.req.set_uri_args("a=3")
ngx.req.set_uri("/foo", true)
```

或

```
ngx.req.set_uri_args({a = 3})  
ngx.req.set_uri("/foo", true)
```

这个方法在 `v0.3.1rc14` 版本中首次引入。

[返回目录 \(page 77\)](#)

ngx.req.set_uri_args

语法: `ngx.req.set_uri_args(args)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`

用 `args` 参数重写当前请求的 URI 请求参数。`args` 参数可以是一个 Lua 字符串，
比如

```
ngx.req.set_uri_args("a=3&b=hello%20world")
```

或一个包含请求参数 key-value 对的 Lua table，例如

```
ngx.req.set_uri_args({ a = 3, b = "hello world" })
```

在第二种情况下，本方法将根据 URI 转义规则转义参数的 key 和 value。

本方法也支持多值参数：

```
ngx.req.set_uri_args({ a = 3, b = {5, 6} })
```

此时请求参数字符串为 `a=3&b=5&b=6`。

这个方法在 `v0.3.1rc13` 版本中首次引入。

更多用法请参考 [ngx.req.set_uri \(page 111\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.get_uri_args

语法: `args = ngx.req.get_uri_args(max_args?)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`

返回一个 Lua table , 包含当前请求的所有 URL 查询参数。

```
location = /test {
    content_by_lua_block {
        local args = ngx.req.get_uri_args()
        for key, val in pairs(args) do
            if type(val) == "table" then
                ngx.say(key, ": ", table.concat(val, ", "))
            else
                ngx.say(key, ": ", val)
            end
        end
    }
}
```

访问 `GET /test?foo=bar&bar=baz&bar=blah` 将输出 :

```
foo: bar
bar: baz, blah
```

多次出现同一个参数 key 时, 将生成一个 Lua table , 按顺序保存其所有 value。

key 和 value 将根据 URI 编码规则进行解码。访问上面的配置文件, `GET /test?a%20b=1%61+2` 将输出 :

```
a b: 1a 2
```

不包含 `=<value>` 部分的参数被视为布尔值参数。 `POST /test` , 请求体是 `foo&bar` 则输出 :

```
foo: true
bar: true
```

换句话说, 它们将被赋值 Lua 布尔值 `true`。但是, 它们与空字符串值参数不同, 如 `GET /test?foo=&bar=` 将输出 :


```
foo:  
bar:
```

没有 key 的参数将被忽略。例如 `GET /test?=hello&=world` 将没有任何输出。

支持通过 nginx 变量 `$args` (或在 Lua 中访问 `ngx.var.args`) 动态更新查询参数：

```
ngx.var.args = "a=3&b=42"  
local args = ngx.req.get_uri_args()
```

这个例子中的 `args` table 将一直是：

```
{a = 3, b = 42}
```

而无论实际请求查询串是什么内容。

请注意，为防止拒绝服务式攻击 (denial of service attacks)，默认最多解析前 100 个请求参数 (包括同名的)，更多的参数将直接忽略。

可选的 `max_args` 函数参数可以用来修改这个限制：

```
local args = ngx.req.get_uri_args(10)
```

这个参数可以被设置为 0 以移除此限制，此时将解析所有接收到的请求参数：

```
local args = ngx.req.get_uri_args(0)
```

强烈不推荐移除 `max_args` 限制。

[返回目录 \(page 77\)](#)

ngx.req.get_post_args

语法: `args, err = ngx.req.get_post_args(max_args?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`

返回一个 Lua table，包含当前请求的所有 POST 查询参数 (MIME type 是 `application/x-www-form-urlencoded`)。使用前需要调用 `ngx.req.read_body` (page 120) 读取完整请求体，或通过设置 `lua_need_request_body` (page 63) 指令为 `on` 以避免报错。

```
location = /test {
    content_by_lua_block {
        ngx.req.read_body()
        local args, err = ngx.req.get_post_args()
        if not args then
            ngx.say("failed to get post args: ", err)
            return
        end
        for key, val in pairs(args) do
            if type(val) == "table" then
                ngx.say(key, ": ", table.concat(val, ", "))
            else
                ngx.say(key, ": ", val)
            end
        end
    }
}
```

请求

```
# Post request with the body 'foo=bar&bar=baz&bar=blah'
$ curl --data 'foo=bar&bar=baz&bar=blah' localhost/test
```

将输出：

```
foo: bar
bar: baz, blah
```

多次出现同一个参数 key 时，将生成一个 Lua table，按顺序保存其所有 value。

key 和 value 将根据 URI 编码规则进行解码。

访问上面的配置文件，

```
# POST request with body 'a%20b=1%61+2'
$ curl -d 'a%20b=1%61+2' localhost/test
```

将输出：

```
a b: 1a 2
```

不包含 `=<value>` 部分的参数被视为布尔值参数。例如 `POST /test` 的请求体是 `foo&bar` 时输出：

```
foo: true
bar: true
```

换句话说，它们将被赋值 Lua 布尔值 `true`。但是，它们与空字符串值参数不同，如 `POST /test` 的请求体是 `foo=&bar=` 将输出：

```
foo:
bar:
```

没有 key 的参数将被忽略。例如 `POST /test` 的请求体是 `=hello&=world` 时将没有任何输出。

请注意，为防止拒绝服务式攻击 (denial of service attacks)，默认最多解析前 100 个请求参数 (包括同名的)，更多的参数将直接忽略。

可选的 `max_args` 函数参数可以用来修改这个限制：

```
local args = ngx.req.get_post_args(10)
```

这个参数可以被设置为 0 以移除此限制，此时将解析所有接收到的请求参数：

```
local args = ngx.req.get_post_args(0)
```

强烈不推荐移除 `max_args` 限制。

[返回目录 \(page 77\)](#)

ngx.req.get_headers

语法: `headers = ngx.req.get_headers(max_headers?, raw?)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`

返回一个 Lua table , 包含当前请求的所有请求头信息。

```
local h = ngx.req.get_headers()
for k, v in pairs(h) do
    ...
end
```

读某一个头信息：

```
ngx.say("Host: ", ngx.req.get_headers()["Host"])
```

请注意, [ngx.var.HEADER \(page 88\)](#) API 使用 nginx 内核 `$http_HEADER` 变量, 在读取单个请求头信息时更加适用。

如果请求头中多次出现某一字段, 例如：

```
Foo: foo
Foo: bar
Foo: baz
```

`ngx.req.get_headers()["Foo"]` 的值会是一个 Lua (数组) table , 类似：

```
{"foo", "bar", "baz"}
```

请注意, 为防止拒绝服务式攻击 (denial of service attacks), 默认最多解析前 100 个请求头信息 (包括同名的), 更多的头信息将直接忽略。

可选的 `max_headers` 函数参数可以用来修改这个限制：

```
local headers = ngx.req.get_headers(10)
```

这个参数可以被设置为 0 以移除此限制，此时将解析所有接收到的请求头信息：

```
local headers = ngx.req.get_headers(0)
```

强烈不推荐移除 `max_headers` 限制。

自 0.6.9 版本开始，默认情况下，返回的 Lua table 中的所有头名称被转换成纯小写字母形式，除非设置 `raw` 参数为 `true` (默认是 `false`)。

同时，默认情况下，Lua table 中将被加入 `__index` 元方法 (metamethod)，用来在查询失败时，将 key 转换为全小写字母、且下划线变为连字符后，重新搜索。例如，如果请求头信息中有 `My-Foo-Header` 字段，下面的调用都将正确取出值：

```
ngx.say(headers.my_foo_header)
ngx.say(headers["My-Foo-Header"])
ngx.say(headers["my-foo-header"])
```

当 `raw` 参数设置为 `true` 时，`__index` 元方法不会被加入。

[返回目录 \(page 77\)](#)

ngx.req.set_header

语法: `ngx.req.set_header(header_name, header_value)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`

将当前请求的名为 `header_name` 的头信息值设置为 `header_value`，如果此头信息名称已经存在，修改其值。

默认时，之后通过 [ngx.location.capture \(page 96\)](#) 和 [ngx.location.capture_multi \(page 103\)](#) 发起的所有子请求都将继承新的头信息。

下面的例子中将设置 `Content-Type` 头信息：

```
ngx.req.set_header("Content-Type", "text/css")
```

`header_value` 可以是一个值数组，例如：

```
ngx.req.set_header("Foo", {"a", "abc"})
```

将生成两个新的请求头信息：

```
Foo: a  
Foo: abc
```

如果原来已经有名为 `Foo` 的头信息，将被覆盖。

当 `header_value` 参数是 `nil` 时，此请求头将被移除。所以，

```
ngx.req.set_header("X-Foo", nil)
```

等同于

```
ngx.req.clear_header("X-Foo")
```

[返回目录 \(page 77\)](#)

ngx.req.clear_header

语法: `ngx.req.clear_header(header_name)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`

清除当前请求的名为 `header_name` 的请求头信息。已经存在的子请求不受影响，此命令之后发起的子请求将默认继承修改后的头信息。

[返回目录 \(page 77\)](#)

ngx.req.read_body

语法: `ngx.req.read_body()`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

同步读取客户端请求体，不阻塞 Nginx 事件循环。

```
ngx.req.read_body()  
local args = ngx.req.get_post_args()
```

如果已经通过打开 [lua_need_request_body](#) (page 63) 选项或其他模块读取请求体，此函数将不会执行，立即返回。

如果已经通过 [ngx.req.discard_body](#) (page 121) 函数或其他模块明确丢弃请求体，此函数将不会执行，立即返回。

当出错时，例如读取数据时连接出错，此方法将立即抛出 Lua 异常 或以 500 状态码中断当前请求。

通过此函数读取的请求体，之后可以通过 [ngx.req.get_body_data](#) (page 121) 获得，或者，通过 [ngx.req.get_body_file](#) (page 122) 得到请求体数据缓存在磁盘上的临时文件名。这取决于：

1. 是否当前读求体已经大于 [client_body_buffer_size](#) ，
2. 是否 [client_body_in_file_only](#) 选项被打开。

在当前请求中包含请求体，但不需要时，必须使用 [ngx.req.discard_body](#) (page 121) 明确丢弃请求体，以避免影响 HTTP 1.1 长连接或 HTTP 1.1 流水线 (pipelining)。

这个函数在 [v0.3.1rc17](#) 版本中首次引入。

[返回目录](#) (page 77)

ngx.req.discard_body

语法: *ngx.req.discard_body()*

环境: *rewrite_by_lua*, access_by_lua*, content_by_lua**

明确丢弃请求体，也就是说，读取连接中的数据后立即丢弃（不以任何形式使用请求体）。

这个函数是异步调用，将立即返回。

如果请求体已经被读取，此函数将不会执行，立即返回。

这个函数在 [v0.3.1rc17](#) 版本中首次引入。

更多用法请参考 [ngx.req.read_body](#) (page 120)。

[返回目录](#) (page 77)

ngx.req.get_body_data

语法: *data = ngx.req.get_body_data()*

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `log_by_lua*`

取回内存中的请求体数据。本函数返回 Lua 字符串而不是包含解析过参数的 Lua table。如果想要返回 Lua table，请使用 [ngx.req.get_post_args \(page 115\)](#) 函数。

当以下情况时，此函数返回 `nil`，

1. 请求体尚未被读取，
2. 请求体已经被存入磁盘上的临时文件，
3. 或请求体大小是 0。

如果请求体尚未被读取，请先调用 [ngx.req.read_body \(page 120\)](#) (或打开 [lua_need_request_body \(page 63\)](#) 选项强制本模块读取请求体，此方法不推荐)。

如果请求体已经被存入临时文件，请使用 [ngx.req.get_body_file \(page 122\)](#) 函数代替。

如需要强制在内存中保存请求体，请设置 `client_body_buffer_size` 和 `client_max_body_size` 为同样大小。

请注意，调用此函数比使用 `ngx.var.request_body` 或 `ngx.var.echo_request_body` 更有效率，因为本函数能够节省一次内存分配与数据复制。

这个函数在 `v0.3.1rc17` 版本中首次引入。

更多用法请参考 [ngx.req.get_body_file \(page 122\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.get_body_file

语法: `file_name = ngx.req.get_body_file()`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

获取存储请求体数据的临时文件名。如果请求体尚未被读取或已被读取到内存中，此函数将返回 `nil`。

本函数返回的文件是只读的，通常情况下会被 Nginx 的内存池清理机制清理。不应该手工修改、更名或删除这个文件。

如果请求体尚未被读取，请先调用 [ngx.req.read_body \(page 120\)](#) (或打开 [lua_need_request_body \(page 63\)](#) 选项强制本模块读取请求体。此方法不推荐)。

如果请求体已经被读入内存，请使用 [ngx.req.get_body_data \(page 121\)](#) 函数代替。

如需要强制在临时文件中保存请求体，请打开 `client_body_in_file_only` 选项。

这个函数在 `v0.3.1rc17` 版本中首次引入。

更多用法请参考 [ngx.req.get_body_data \(page 121\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.set_body_data

语法: `ngx.req.set_body_data(data)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

使用 `data` 参数指定的内存数据设置当前请求的请求体。

如果当前请求的请求体尚未被读取，它将被安全地丢弃。当请求体已经被读进内存或缓存在磁盘文件中时，相应的内存或磁盘文件将被立即清理回收。

这个函数在 `v0.3.1rc18` 版本中首次引入。

更多用法请参考 [ngx.req.set_body_file \(page 123\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.set_body_file

语法: `ngx.req.set_body_file(file_name, auto_clean?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

使用 `file_name` 参数指定的数据文件设置当前请求的请求体。

当可选参数 `auto_clean` 设置为 `true` 时，在本次请求完成，或本次请求内再次调用本函数或 [ngx.req.set_body_data \(page 123\)](#) 时，`file_name` 文件将被删除。

`auto_clean` 默认值是 `false`。

请确保 `file_name` 参数指定的文件存在，并设置合适的操作权限对 Nginx worker 可读，以避免抛出 Lua 异常。

如果当前请求的请求体尚未被读取，它将被安全地丢弃。当请求体已经被读进内存或缓存在磁盘文件中时，相应的内存或磁盘文件将被立即清理回收。

这个函数在 `v0.3.1rc18` 版本中首次引入。

更多用法请参考 [ngx.req.set_body_data \(page 123\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.init_body

语法: `ngx.req.init_body(buffer_size?)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

为当前请求创建一个新的空请求体并初始化缓冲区，为后续通过 [ngx.req.append_body \(page 124\)](#) 和 [ngx.req.finish_body \(page 125\)](#) API 写请求体数据做好准备。

如果设置了 `buffer_size` 参数，将设置该大小的内存缓冲区，用于后续的 [ngx.req.append_body \(page 124\)](#) 写请求体数据。如果省略此参数，将使用 Nginx 标准指令 `client_body_buffer_size` 中设置的值作为缓冲区大小。

当请求体数据过大，不再能保存在内存缓冲区中时，数据将被写入一个临时文件，类似 Nginx 内核中的标准请求体处理方式。

需要强调的是，在当前请求的所有请求体被写入完成后，必须调用 [ngx.req.finish_body \(page 125\)](#) 以结束写入。另外，当此函数与 [ngx.req.socket \(page 125\)](#) 一起使用时，需要在执行此函数之前调用 [ngx.req.socket \(page 125\)](#)，否则将会报 “request body already exists” (请求体已经存在) 错。

此函数典型用法如下：

```
ngx.req.init_body(128 * 1024) -- 缓冲区 128KB
for chunk in next_data_chunk() do
    ngx.req.append_body(chunk) -- 每块可以是 4KB
end
ngx.req.finish_body()
```

此函数可以与 [ngx.req.append_body \(page 124\)](#)，[ngx.req.finish_body \(page 125\)](#) 和 [ngx.req.socket \(page 125\)](#) 一起，使用纯 Lua 语言实现高效的输入过滤器 (在 [rewrite_by_lua/#rewrite_by_lua](#) 或 [\[access_by_lua \(page 51\)\]](#) 环境中)，与其他 Nginx 内容处理 handler 或上游模块例如 [ngx_http_proxy_module](#) 和 [ngx_http_fastcgi_module](#) 配合使用。

这个函数在 `v0.5.11` 版本中首次引入。

[返回目录 \(page 77\)](#)

ngx.req.append_body

语法: `ngx.req.append_body(data_chunk)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

向已存在的请求体中追加写入 `data_chunk` 参数指定的新数据块，此请求体最初使用 [ngx.req.init_body \(page 123\)](#) 创建。

当请求体数据过大，不再能保存在内存缓冲区中时，数据将被写入一个临时文件，类似 Nginx 内核中的标准请求体处理方式。

需要强调的是，在当前请求的所有请求体被写入完成后，必须调用 [ngx.req.finish_body \(page 125\)](#) 以结束写入。

此函数可以与 [ngx.req.init_body \(page 123\)](#)，[ngx.req.finish_body \(page 125\)](#)，和 [ngx.req.socket \(page 125\)](#) 一起，使用纯 Lua 语言实现高效的输入过滤器 (在 [rewrite_by_lua/#rewrite_by_lua](#) 或 [access_by_lua \(page 51\)](#) 环境中)，与其他 Nginx 内容处理程序或上游模块例如 [ngx_http_proxy_module](#) 和 [ngx_http_fastcgi_module](#) 配合使用。

这个函数在 **v0.5.11** 版本中首次引入。

更多用法请参考 [ngx.req.init_body \(page 123\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.finish_body

语法: `ngx.req.finish_body()`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

结束新请求体构造过程，该请求体由 [ngx.req.init_body \(page 123\)](#) 和 [ngx.req.append_body \(page 124\)](#) 创建。

此函数可以与 [ngx.req.init_body \(page 123\)](#)，[ngx.req.append_body \(page 124\)](#)，和 [ngx.req.socket \(page 125\)](#) 一起，使用纯 Lua 语言实现高效的输入过滤器 (在 [rewrite_by_lua/#rewrite_by_lua](#) 或 [access_by_lua \(page 51\)](#) 环境中)，与其他 Nginx 内容处理程序或上游模块例如 [ngx_http_proxy_module](#) 和 [ngx_http_fastcgi_module](#) 配合使用。

这个函数在 **v0.5.11** 版本中首次引入。

更多用法请参考 [ngx.req.init_body \(page 123\)](#)。

[返回目录 \(page 77\)](#)

ngx.req.socket

语法: `tcpsock, err = ngx.req.socket()`

语法: `tcpsock, err = ngx.req.socket(raw)`

环境: `rewrite_by_lua`, `access_by_lua*`, `content_by_lua**`

返回一个包含下游连接的只读 `cosocket` 对象。只有 [receive \(page 175\)](#) 和 [receiveuntil \(page 176\)](#) 方法在该对象上是支持的。

错误情况，将返回 **nil** 和错误字符描述信息。

通过该方法返回的 `socket` 对象，通常是用流式格式读取当前请求体。不要开启 `lua_need_request_body` (page 63) 指令，并且不要混合调用 `ngx.req.read_body` (page 120) 和 `ngx.req.discard_body` (page 121)。

如果任何的请求体数据已经被预读到 Nginx 内核请求缓冲区，得到的 `cosocket` 对象需要小心对待，应避免由这种预读导致的潜在数据丢失。

从 `v0.9.0` 版本开始，该函数接受一个可选的布尔值参数 `raw`。当该参数为 `true` 时，该方法将返回一个包含原生下游连接的全双工 `cosocket` 对象，你能对它调用 `receive` (page 175)，`receiveuntil` (page 176) 和 `send` (page 174)。

当指定 `raw` 参数为 `true`，这里需要没有任何来自 `ngx.say` (page 131)、`ngx.print` (page 130) 或 `ngx.send_headers` (page 129) 方法调用的待处理数据。所以如果你有下游输出调用，你应当在调用 `ngx.req.socket(true)` 之前调用 `ngx.flush(true)` (page 132) 确保这里没有任何待处理数据。如果请求体还没有读取，那么这个“原生 socket”也能用来读取请求体。

你可以使用通过 `ngx.req.socket(true)` 返回的“原生请求 socket”来实现各种样式协议如 `WebSocket`，或仅发出自己的 HTTP 请求头或体数据。真实世界，你可以参考 `lua-resty-websocket` 库。

该函数是在 `v0.5.0rc1` 版本首次引入的。

[返回目录 \(page 77\)](#)

ngx.exec

语法: `ngx.exec(uri, args?)`

环境: `rewrite_by_lua`, `access_by_lua*`, `content_by_lua**`

使用 `uri`、`args` 参数执行一个内部跳转，与 `echo-nginx-module` 的 `echo_exec` 指令有些相似。

```
ngx.exec('/some-location');
ngx.exec('/some-location', 'a=3&b=5&c=6');
ngx.exec('/some-location?a=3&b=5', 'c=6');
```

可选第二个参数 `args` 可被用来指名额外的 URI 查询参数，例如：

```
ngx.exec("/foo", "a=3&b=hello%20world")
```

另外，对于 `args` 参数可使用一个 Lua 表，内部通过 `ngx_lua` 完成 URI 转义和字符串的连接。

```
ngx.exec("/foo", { a = 3, b = "hello world" })
```

该结果和上一个示例是一样的。

作为 `args` 参数的 Lua 表格式化结果，与使用 `ngx.encode_args` (page 135) 方法格式化结果是相同的。

命名 location 也是支持的，但是第二个 `args` 参数将被忽略（如果存在的话），并且对新目标地址的查询字符串会从引用 location（如果有）中继承。

下面的例子中，`GET /foo/file.php?a=hello` 将返回“hello”，而不是“goodbye”：

```
location /foo {
    content_by_lua_block {
        ngx.exec("@bar", "a=goodbye");
    }
}

location @bar {
    content_by_lua_block {
        local args = ngx.req.get_uri_args()
        for key, val in pairs(args) do
            if key == "a" then
                ngx.say(val)
            end
        end
    }
}
```

注意，`ngx.exec` 方法与 `ngx.redirect` (page 128) 是完全不同的，前者是个纯粹的内部跳转并且没有引入任何额外 HTTP 信号。

注意，此方法的调用终止当前请求的处理，并且它 必须 在 `ngx.send_headers` (page 129) 或明确有响应体应答（比如 `ngx.print` (page 130) 或 `ngx.say` (page 131) 之一）之前调用。

该方法调用与 `return` 语句联合使用，是推荐的代码样式，例如，通过 `return ngx.exec(...)`，可使用在 `header_filter_by_lua*` (page 53) 之外的环境中加强处理该请求被终止。

[返回目录 \(page 77\)](#)

ngx.redirect

语法: `ngx.redirect(uri, status?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

发出一个 HTTP `301` 或 `302` 重定向到 `uri`。

可选项 `status` 参数指定使用什么 HTTP 状态码。目前支持下面几个状态码：

- `301`
- `302` (默认)
- `303`
- `307`

默认使用 `302` (`ngx.HTTP_MOVED_TEMPORARILY`)。

假设当前服务名是 `localhost` 并且监听端口是 1984，这里有个例子：

```
return ngx.redirect("/foo")
```

和下面例子是等价的：

```
return ngx.redirect("/foo", ngx.HTTP_MOVED_TEMPORARILY)
```

重定向到任意外部 URL 也是支持的，例如：

```
return ngx.redirect("http://www.google.com")
```

我们也可以使用数值类型的值当做第二个 `status` 参数：

```
return ngx.redirect("/foo", 301)
```

该方法与使用 `redirect` 修饰的 `rewrite` 标准指令 `ngx_http_rewrite_module` 有些相似，例如，该 `nginx.conf` 片段：

```
rewrite ^ /foo? redirect; # nginx config
```

与下面 Lua 代码是等价的：

```
return ngx.redirect('/foo'); -- Lua code
```

当这样

```
rewrite ^ /foo? permanent; # nginx config
```

等同于：

```
return ngx.redirect('/foo', ngx.HTTP_MOVED_PERMANENTLY) -- Lua code
```

也可以指定具体 URI 参数，例如：

```
return ngx.redirect('/foo?a=3&b=4')
```

注意，此方法的调用终止当前请求的处理，并且它 必须 在 [ngx.send_headers \(page 129\)](#) 或明确有响应体应答（比如 [ngx.print \(page 130\)](#) 或 [ngx.say \(page 131\)](#) 之一）之前调用。

该方法调用与 `return` 语句联合使用，是推荐的代码样式，例如，通过 `return ngx.redirect(...)`，可使用在 [header_filter_by_lua* \(page 53\)](#) 之外的环境中加强处理该请求被终止。

[返回目录 \(page 77\)](#)

ngx.send_headers

语法: `ok, err = ngx.send_headers()`

环境: `rewrite_by_lua`, `access_by_lua*`, `content_by_lua**`

指名把应答头发送出去。

自从 `v0.8.3` 版本开始，成功情况该函数返回 `1`，否则返回 `nil` 和错误字符描述信息。

注意，在内容通过 `ngx.say` (page 131)、`ngx.print` (page 130) 输出或当 `content_by_lua*` (page 45) 存在时，`ngx_lua` 将自动发送头部内容，所以通常情况不需要手动发送应答头。

[返回目录](#) (page 77)

ngx.headers_sent

语法: `value = ngx.headers_sent`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

如果应答头部已经被发送（通过 `ngx_lua`）返回 `true`，否则返回 `false`。

该 API 是在 `v0.3.1rc6` 版本首次引入的。

[返回目录](#) (page 77)

ngx.print

语法: `ok, err = ngx.print(...)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

将输入参数合并发送给 HTTP 客户端 (作为 HTTP 响应体)。如果此时还没有发送响应头信息，本函数将先发送 HTTP 响应头，再输出响应体。

自版本 `v0.8.3` 起，本函数当成功时返回 `1`，失败时返回 `nil` 以及一个描述错误的字符串。

Lua 的 `nil` 值输出 `"nil"` 字符串，Lua 的布尔值输出 `"true"` 或 `"false"` 字符串。

输入允许字符串嵌套数组，数组中所有元素相按顺序输出。

```
local table = {
    "hello, ",
    {"world: ", true, " or ", false,
     {"": ", nil"}}
}
ngx.print(table)
```

将输出


```
hello, world: true or false: nil
```

非数组表(哈希表)参数将导致抛出 Lua 异常。

`ngx.null` 常量输出为 `"null"` 字符串。

本函数为异步调用，将立即返回，不会等待所有数据被写入系统发送缓冲区。要以同步模式运行，请在调用 `ngx.print` 之后调用 `ngx.flush(true)`。这种方式在流式输出时非常有用。更多细节请参考 [ngx.flush \(page 132\)](#)。

请注意，`ngx.print` 和 [ngx.say \(page 131\)](#) 都会调用 Nginx body 输出过滤器，这种操作非常“昂贵”。所以，在“热”循环中使用这两个函数要非常小心；可以通过 Lua 进行缓存以节约调用。

[返回目录 \(page 77\)](#)

ngx.say

语法: `ok, err = ngx.say(...)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

与 [ngx.print \(page 130\)](#) 相同,同时末尾添加一个回车符。

[返回目录 \(page 77\)](#)

ngx.log

语法: `ngx.log(log_level, ...)`

环境: `init_by_lua*`, `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`,
`access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`,
`log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`,
`ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

将参数拼接起来，按照设定的日志级别记入 `error.log`。

Lua `nil` 参数将输出 `"nil"` 字符串；Lua 布尔参数将输出 `"true"` 或 `"false"` 字符串；`ngx.null` 常量将输出 `"null"` 字符串。

`log_level` 参数可以使用类似 `ngx.ERR` 和 `ngx.WARN` 的常量。更多信息请参考 [Nginx log level constants \(page 92\)](#)。

在 Nginx 内核中硬编码限制了单条错误信息最长为 `2048` 字节。这个长度包含了最后的换行符和开始的时间戳。如果信息长度超过这个限制，Nginx 将把信息文本截断。这个限制可以通过修改 Nginx 源码中 `src/core/ngx_log.h` 文件中的 `NGX_MAX_ERROR_STR` 宏定义调整。

[返回目录 \(page 77\)](#)

ngx.flush

语法: `ok, err = ngx.flush(wait?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

向客户端刷新响应输出。

自 `v0.3.1rc34` 版本开始, `ngx.flush` 接受一个布尔型可选参数 `wait` (默认值 `false`)。当通过默认参数调用时,本函数发起一个异步调用(将直接返回,不等待输出数据被写入系统发送缓冲区)。当把 `wait` 参数设置为 `true` 时,本函数将以同步模式执行。

在同步模式下,本函数不会立即返回,一直到所有输出数据被写入系统输出缓冲区,或者到达发送超时 `send_timeout` 时间。请注意,因为使用了 Lua 协程机制,本函数即使在同步模式下也不会阻塞 Nginx 事件循环。

当 `ngx.flush(true)` 在 `ngx.print (page 130)` 或 `ngx.say (page 131)` 之后被立刻调用时,它将使这两个函数以同步模式执行。这在流式输出时非常有用。

请注意, `ngx.flush` 在 HTTP 1.0 缓冲输出模式下不起作用。详情请参考 [HTTP 1.0 support \(page 16\)](#)。

自 `v0.8.3` 版本开始,本函数执行成功是返回 `1`, 否则返回 `nil` 和错误信息串。

[返回目录 \(page 77\)](#)

ngx.exit

语法: `ngx.exit(status)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

当 `status >= 200` (即 `ngx.HTTP_OK` 及以上) 时,本函数中断当前请求执行并返回状态值给 nginx。

当 `status == 0` (即 `ngx.OK`) 时,本函数退出当前的“处理阶段句柄”(或当使用 `content_by_lua* (page 45)` 指令时的“内容句柄”),继续执行当前请求的下一个阶段(如果有)。

`status` 参数可以是 `status` argument can be `ngx.OK`, `ngx.ERROR`, `ngx.HTTP_NOT_FOUND`, `ngx.HTTP_MOVED_TEMPORARILY` 或其它 [HTTP status constants \(page 90\)](#)。

要返回一个自定义内容的错误页,使用类似下面的代码:

```

ngx.status = ngx.HTTP_GONE
ngx.say("This is our own content")
-- 退出整个请求而不是当前处理阶段
ngx.exit(ngx.HTTP_OK)

```

实际效果:

```

$ curl -i http://localhost/test
HTTP/1.1 410 Gone
Server: nginx/1.0.6
Date: Thu, 15 Sep 2011 00:51:48 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Connection: keep-alive

This is our own content

```

可以直接使用数字作为参数, 例如:

```

ngx.exit(501)

```

请注意, 虽然此方法接受所有 [HTTP status constants \(page 90\)](#) 作为输入, 但在 [core constants \(page 89\)](#) 中仅支持 `NGX_OK` 和 `NGX_ERROR` 作为输入。

同时请注意, 因为调用此方法会中断当前请求处理, 所以建议代码风格是与 `return` 语句一起调用此方法, 即, 使用 `return ngx.exit(...)` 来强调请求处理过程已经中断。

当使用在 [header_filter_by_lua \(page 53\)](#) 环境中时, `ngx.exit()` 是一个异步操作, 会立即返回。这个行为在未来版本中可能会改变, 所以建议用户一直使用上述与 `return` 同时使用的代码风格。

[返回目录 \(page 77\)](#)

ngx.eof

语法: `ok, err = ngx.eof()`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`

明确指定响应输出流的末尾。在 HTTP 1.1 分块编码输出模式下, 它会触发 Nginx 内核发送“最后一块”。

当禁用下游连接的 HTTP 1.1 保持连接功能后，用户程序可以通过调用此方法，使下游 HTTP 客户端主动关闭连接。这招可以用来执行后台任务，而无需 HTTP 客户端等待连接关闭，例如：

```
location = /async {
    keepalive_timeout 0;
    content_by_lua_block {
        ngx.say("got the task!")
        ngx.eof() -- 下游 HTTP 客户端将在这里断开连接
        -- 在这里访问 MySQL, PostgreSQL, Redis, Memcached 等 ...
    }
}
```

但是，如果用户程序创建子请求通过 Nginx 上游模块访问其他 location 时，需要配置上游模块忽略客户端连接中断 (如果不是默认)。例如，默认时，基本模块 `ngx_http_proxy_module` 在客户端关闭连接后，立刻中断主请求和子请求，所以在 `ngx_http_proxy_module` 配置的 location 块中打开 `proxy_ignore_client_abort` 开关非常重要：

```
proxy_ignore_client_abort on;
```

一个执行后台任务的方法是使用 `ngx.timer.at` (page 190) API。

自版本 `v0.8.3` 开始，此函数成功时返回 `1`，失败时返回 `nil` 和错误信息串。

[返回目录 \(page 77\)](#)

ngx.sleep

语法: `ngx.sleep(seconds)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

无阻塞地休眠特定秒。时间可以精确到 0.001 秒 (毫秒)。

在后台，此方法使用 Nginx 的定时器。

自版本 `0.7.20` 开始，`0` 也可以作为时间参数被指定。

这个方法最早在版本 `0.5.0rc30` 中出现。

[返回目录 \(page 77\)](#)

ngx.escape_uri

语法: `newstr = ngx.escape_uri(str)`

环境: `init_by_lua*`, `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`,
`access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`,
`log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`,
`ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

对 `str` 进行 URI 编码。

[返回目录 \(page 77\)](#)

ngx.unescape_uri

语法: `newstr = ngx.unescape_uri(str)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

将转义过的 URI 内容 `str` 解码。

例如,

```
ngx.say(ngx.unescape_uri("b%20r56+7"))
```

输出

```
b r56 7
```

[返回目录 \(page 77\)](#)

ngx.encode_args

语法: `str = ngx.encode_args(table)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`

根据 URI 编码规则，将 Lua 表编码成一个查询参数字符串。

例如，

```
ngx.encode_args({foo = 3, ["b r"] = "hello world"})
```

生成

```
foo=3&b%20r=hello%20world
```

Lua 表的 key 必须是 Lua 字符串。

支持多值参数。可以使用 Lua 表存储参数值，例如：

```
ngx.encode_args({baz = {32, "hello"}})
```

输出

```
baz=32&baz=hello
```

如果 value 表是空的，效果等同于 `nil` 值。

支持布尔值参数，例如，

```
ngx.encode_args({a = true, b = 1})
```

输出

```
a&b=1
```

如果参数值是 `false`，效果等同于 `nil` 值。

这个方法最早出现在版本 `v0.3.1rc27` 中。

[返回目录 \(page 77\)](#)

ngx.decode_args

语法: `table = ngx.decode_args(str, max_args?)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`

将 URI 编码的查询字符串解码为 Lua 表。本函数是 [ngx.encode_args \(page 135\)](#) 的逆函数。

可选的参数 `max_args` 可以用来指定从 `str` 中最多解析的参数个数。默认时，最多解析 100 个请求参数 (包括同名的)。为避免潜在的拒绝服务式攻击 (denial of services, DOS)，超过 `max_args` 数量上限的 URI 参数被丢弃，

这个参数可以被设成 0 以去掉解析参数数量上限：

```
local args = ngx.decode_args(str, 0)
```

强烈不推荐移除 `max_args` 限制。

这个方法最早出现在版本 `v0.5.0rc29` 中。

[返回目录 \(page 77\)](#)

ngx.encode_base64

语法: `newstr = ngx.encode_base64(str, no_padding?)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

通过 `base64` 对 `str` 字符串编码。

自 `0.9.16` 版本后，引入了一个布尔值参数 `no_padding` 用来控制是否需要编码数据填充 `等号` 字符串（默认为 `false`，代表需要填充）。

[返回目录 \(page 77\)](#)

ngx.decode_base64

语法: `newstr = ngx.decode_base64(str)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

通过 `base64` 解码 `str` 字符串得到未编码过的字符串。如果 `str` 字符串没有被正常解码将会返回 `nil`。

[返回目录 \(page 77\)](#)

ngx.crc32_short

语法: `intval = ngx.crc32_short(str)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

通过一个字符串计算循环冗余校验码。

这个方法最好在字符串较少时调用（比如少于30-60字节），他的结果和 `ngx.crc32_long` 是一样的。

本质上，它只是 Nginx 内核函数 `ngx_crc32_short` 的简单封装。

这个方法最早出现在版本 `v0.3.1rc8` 中。

[返回目录 \(page 77\)](#)

ngx.crc32_long

语法: `intval = ngx.crc32_long(str)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

通过一个字符串计算循环冗余校验码。

这个方法最好在字符串较多时调用（比如大于30-60字节），他的结果和 `ngx.crc32_short` 是一样的。

本质上，它只是 Nginx 内核函数 `ngx_crc32_long` 的简单封装。

这个方法最早出现在版本 `v0.3.1rc8` 中。

[返回目录 \(page 77\)](#)

ngx.hmac_sha1

语法: `digest = ngx.hmac_sha1(secret_key, str)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

通过 `str` 待运算数据和 `secret_key` 密钥串生成结果。关于 [HMAC-SHA1](#)。

通过 [HMAC-SHA1](#) 的运算会得到二进制数据，如果你想要把结果转为文本形式，你可以使用 [ngx.encode_base64 \(page 0\)](#) 函数。

举一个例子，

```
local key = "thisisverysecretstuff"
local src = "some string we want to sign"
local digest = ngx.hmac_sha1(key, src)
ngx.say(ngx.encode_base64(digest))
```

将会输出

```
R/pvxzHC4NLtj7S+kXfg/NePTmk=
```

这个 [API](#) 需要在安装 [Nginx](#) 时启用 [OpenSSL](#) 库（通常通过 `./configure` 脚本的 `--with-http_ssl_module` 选项来控制）

这个方法最早出现在版本 `v0.3.1rc29` 中。

[返回目录 \(page 77\)](#)

ngx.md5

语法: `digest = ngx.md5(str)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

通过 `MD5` 计算 `str` 字符串返回十六进制的数据。

举一个例子,

```
location = /md5 {  
    content_by_lua_block { ngx.say(ngx.md5("hello")) }  
}
```

将会输出

```
5d41402abc4b2a76b9719d911017c592
```

如果需要返回二进制数据请看 `ngx.md5_bin` (page 0) 方法。

[返回目录 \(page 77\)](#)

ngx.md5_bin

语法: `digest = ngx.md5_bin(str)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

通过 `MD5` 计算 `str` 字符串返回二进制的的数据。

如果需要返回纯文本数据请看 `ngx.md5` (page 0) 方法。

[返回目录 \(page 77\)](#)

ngx.sha1_bin

语法: `digest = ngx.sha1_bin(str)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

通过 `SHA-1` 计算 `str` 字符串返回二进制的的数据。

在安装 `Nginx` 时 这个函数需要 `SHA-1` 的支持。(这通常说明应该在安装 `Nginx` 时一起安装 `OpenSSL` 库)。

这个方法在 `v0.5.0rc6` 版本中首次引入。

[返回目录 \(page 77\)](#)

ngx.quote_sql_str

语法: `quoted_value = ngx.quote_sql_str(raw_value)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

根据 `MySQL` 转义规则返回一个转义后字符串。

[返回目录 \(page 77\)](#)

ngx.today

语法: `str = ngx.today()`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`,
`content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`,
`ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

从 `nginx` 的时间缓存(不像 `Lua` 的日期库, 该时间不涉及系统调用)返回当前的日期(格式: `yyyy-mm-dd`)。

这是个本地时间。

[返回目录 \(page 77\)](#)

ngx.time

语法: `secs = ngx.time()`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`,
`content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`,
`ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

返回从新纪元到从 nginx 时间缓存(不像 Lua 的日期库, 该时间不涉及系统调用)获取的当前时间戳所经过的秒数。

通过先调用 [ngx.update_time \(page 142\)](#) 会强制更新 nginx 的时间缓存。

[返回目录 \(page 77\)](#)

ngx.now

语法: `secs = ngx.now()`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

返回一个浮点型的数字, 该数字是从新纪元到从 nginx 时间缓存(不像 Lua 的日期库, 该时间不涉及系统调用)获取的当前时间戳所经过的时间(以秒为单位, 小数部分是毫秒)。

通过先调用 [ngx.update_time \(page 142\)](#), 你可以强制更新 nginx 时间缓存。

这个API最早出现在 `v0.3.1rc32` 版本中。

[返回目录 \(page 77\)](#)

ngx.update_time

语法: `ngx.update_time()`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

强行更新 Nginx 当前时间缓存。此调用会涉及到一个系统调用, 因此会有一些系统开销, 所以不要滥用。

这个API最早出现在 `v0.3.1rc32` 版本中。

[返回目录 \(page 77\)](#)

ngx.localtime

语法: `str = ngx.localtime()`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

返回 nginx 时间缓存(不像 Lua 的 `os.date` 函数, 该时间不涉及系统调用)的当前时间戳(格式: `yyyy-mm-dd hh:mm:ss`)。

这是个本地时间。

[返回目录 \(page 77\)](#)

ngx.utctime

语法: `str = ngx.utctime()`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

返回 nginx 时间缓存(不像 Lua 的 `os.date` 函数, 该时间不涉及系统调用)的当前时间戳(格式: `yyyy-mm-dd hh:mm:ss`)。

这是个UTC时间。

[返回目录 \(page 77\)](#)

ngx.cookie_time

语法: `str = ngx.cookie_time(sec)`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

返回一个可以用做 cookie 过期时间的格式化字符串。参数 `sec` 是以秒为单位的时间戳(比如 `ngx.time` (page 141) 的返回)。

```
ngx.say(ngx.cookie_time(1290079655))
-- yields "Thu, 18-Nov-10 11:27:35 GMT"
```

[返回目录 \(page 77\)](#)

ngx.http_time

语法: `str = ngx.http_time(sec)`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

返回一个可以用在 http 头部时间的格式化字符串（例如，在 `Last-Modified` 头的使用）。参数 `sec` 是以秒为单位的时间戳（比如 `ngx.time` (page 141) 的返回）。

```
ngx.say(ngx.http_time(1290079655))
-- yields "Thu, 18 Nov 2010 11:27:35 GMT"
```

[返回目录 \(page 77\)](#)

ngx.parse_http_time

语法: `sec = ngx.parse_http_time(str)`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

解析 http 时间字符串（比如从 `ngx.http_time` (page 144) 返回内容）。成功情况下返回秒数，错误的输入字符格式返回 `nil`。

```
local time = ngx.parse_http_time("Thu, 18 Nov 2010 11:27:35 GMT")
if time == nil then
    ...
end
```

[返回目录 \(page 77\)](#)

ngx.is_subrequest

语法: `value = ngx.is_subrequest`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`

如果当前请求是 nginx 子请求返回 `true` , 否则返回 `false` 。

[返回目录 \(page 77\)](#)

ngx.re.match

语法: `captures, err = ngx.re.match(subject, regex, options?, ctx?, res_table?)`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`,
`content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`,
`ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

使用 Perl 兼容正则表达式 `regex` 匹配字符串 `subject` , 并使用可选的参数 `options` 作为正则表达式选项。

仅返回第一个匹配结果, 无结果时返回 `nil` 。当出错时, 例如正则表达式出错或者超出 PCRE 堆栈限制, 将返回 `nil` 以及一个描述错误的字符串。

当匹配成功时, 返回一个 Lua 表 `captures` , 其中 `captures[0]` 存储(整个模板)匹配出的完整子字符串, `captures[1]` 存储第一个括号内的子模板匹配结果, `captures[2]` 存储第二个, 以此类推。

```
local m, err = ngx.re.match("hello, 1234", "[0-9]+")
if m then
    -- m[0] == "1234"

else
    if err then
        ngx.log(ngx.ERR, "error: ", err)
        return
    end

    ngx.say("match not found")
end
```

```
local m, err = ngx.re.match("hello, 1234", "([0-9])[0-9]+")
-- m[0] == "1234"
-- m[1] == "1"
```

自 **v0.7.14** 版本后, 本模块支持正则表达式命名捕获(Named capture), 结果以键值对的方式与数字编号的结果在同一个 Lua 表中返回。

```
local m, err = ngx.re.match("hello, 1234", "([0-9])(?<remaining>[0-9]+)")
-- m[0] == "1234"
-- m[1] == "1"
-- m[2] == "234"
-- m["remaining"] == "234"
```

在 **captures** 表中, 不匹配的子模板将返回 **false** 值。

```
local m, err = ngx.re.match("hello, world", "(world)|(hello)|(?<named>howdy)")
-- m[0] == "hello"
-- m[1] == false
-- m[2] == "hello"
-- m[3] == false
-- m["named"] == false
```

通过指定 **options** (选项)来控制匹配操作的执行方式。支持以下选项字符。

a	锚定模式（仅从目标字符串开始位置匹配）
d	启用 DFA 模式（又名最长令牌匹配语义）。 此选项需要 PCRE 6.0 以上版本，否则将抛出 Lua 异常。 此选项最早出现在 ngx_lua v0.3.1rc30 版本中。
D	启用重复命名模板支持。子模板命名可以重复，在结果中以数组方式返回。例如：
d	<pre> local m = ngx.re.match("hello, world", "(?<named>\w+), (?<name d>\w+)", "D") -- m["named"] == {"hello", "world"} </pre> 此选项最早出现在 v0.7.14 版本中，需要 PCRE 8.12 以上版本支持。
i	大小写不敏感模式（类似 Perl 的 /i 修饰符）
j	启用 PCRE JIT 编译，此功能需要 PCRE 8.21 以上版本以 --enable-jit 选项编译。 为达到最佳性能，此选项应与 'o' 选项同时使用。 此选项最早出现在 ngx_lua v0.3.1rc30 版本中。
J	启用 PCRE Javascript 兼容模式。 此选项最早出现在 v0.7.14 版本中，需要 PCRE 8.12 以上版本支持。
m	多行模式（类似 Perl 的 /m 修饰符）
o	仅编译一次模式（类似 Perl 的 /o 修饰符） 启用 worker 进程级正则表达式编译缓存。
s	单行模式（类似 Perl 的 /s 修饰符）
u	UTF-8 模式。此选项需要 PCRE 以 --enable-utf8 选项编译，否则将抛出 Lua 异常。
U	类似 "u" 模式，但禁用了 PCRE 对目标字符串的 UTF-8 合法性检查。 此选项最早出现在 ngx_lua v0.8.1 版本中。
x	扩展模式（类似 Perl 的 /x 修饰符）

这些选项可以组合使用：

```
local m, err = ngx.re.match("hello, world", "HEL L0", "ix")
-- m[0] == "hello"
```

```
local m, err = ngx.re.match("hello, 美好生活", "HELLO,
(.{2})", "iu")
-- m[0] == "hello, 美好"
-- m[1] == "美好"
```

在优化性能时，`o` 选项非常有用，因为正则表达式模板将仅仅被编译一次，之后缓存在 worker 级的缓存中，并被此 nginx worker 处理的所有请求共享。缓存数量上限可以通过 `lua_regex_cache_max_entries` (page 35) 指令调整。

可选的第四个参数 `ctx` 是一个 Lua 表，包含可选的 `pos` 域。当 `ctx` 表的 `pos` 域有值时，`ngx.re.match` 将从该位置起执行匹配(位置下标从 1 开始)。不论 `ctx` 表中是否已经有 `pos` 域，`ngx.re.match` 将在正则表达式被成功匹配后，设置 `pos` 域值为完整匹配子字符串 之后 的位置。当匹配失败时，`ctx` 表将保持不变。

```
local ctx = {}
local m, err = ngx.re.match("1234, hello", "[0-9]+", "", ctx)
-- m[0] = "1234"
-- ctx.pos == 5
```

```
local ctx = { pos = 2 }
local m, err = ngx.re.match("1234, hello", "[0-9]+", "", ctx)
-- m[0] = "34"
-- ctx.pos == 5
```

参数 `ctx` 表与正则表达式修饰符 `a` 组合使用，可以用来建立一个基于 `ngx.re.match` 的词法分析器。

注意，当指定参数 `ctx` 时，参数 `options` 不能空缺，当不需要使用 `options` 来指定正则表达式选项时，必须使用 Lua 空字符串 ("") 作为占位符。

这个方法需要在 Nginx 中启用 PCRE 库。([Known Issue With Special Escaping Sequences](#) (page 23)).

要想确认 PCRE JIT 是否已经启用，需要在 Nginx 或 OpenResty 的 `./configure` 配置脚本中，添加 `--with-debug` 选项激活 Nginx 的调试日志。然后，在 `error_log` 指令中启用 `error` 错误日志级别。当 PCRE JIT 启用时，将出现下述信息：

```
pcre JIT compiling result: 1
```

自 **0.9.4** 版本开始, 此函数接受第五个参数, **res_table**, 让调用者可以自己指定存储所有匹配结果的 Lua 表。自 **0.9.6** 版本开始, 调用者需要自己确保这个表是空的。这个功能对表预分配、重用以及节省 Lua 回收机制 (GC) 非常有用。

这个功能最早出现在 **v0.2.1rc11** 版本中。

[返回目录 \(page 77\)](#)

ngx.re.find

语法: *from, to, err = ngx.re.find(subject, regex, options?, ctx?, nth?)*

环境: *init_worker_by_lua*, set_by_lua*, rewrite_by_lua*, access_by_lua*, content_by_lua*, header_filter_by_lua*, body_filter_by_lua*, log_by_lua*, ngx.timer.*, balancer_by_lua*, ssl_certificate_by_lua*, ssl_session_fetch_by_lua*, ssl_session_store_by_lua**

与 [ngx.re.match \(page 145\)](#) 类似但只返回匹配结果子字符串的开始索引 (**from**) 和结束索引 (**to**)。返回的索引值是基于 1 的, 可以直接被用于 Lua 的 [string.sub](#) API 函数来获取匹配结果子串。

当出现任何错误时 (例如错误的正则表达式或任何 PCRE 运行时错误), 这个 API 函数返回两个 **nil** 以及一个描述错误的字符串。

如果匹配不成功, 此函数返回一个 **nil** 值。

下面是一个示例:

```
local s = "hello, 1234"
local from, to, err = ngx.re.find(s, "([0-9]+)", "jo")
if from then
    ngx.say("from: ", from)
    ngx.say("to: ", to)
    ngx.say("matched: ", string.sub(s, from, to))
else
    if err then
        ngx.say("error: ", err)
        return
    end
    ngx.say("not matched!")
end
```

此示例将输出

```

from: 8
to: 11
matched: 1234

```

因此此 API 函数并不创建任何新 Lua 字符串或 Lua 表，运行速度大大快于 [ngx.re.match \(page 145\)](#)。所以如果可能请尽量使用本函数。

自 **0.9.3** 版本开始，添加第 5 个可选参数 **nth**，用来指定第几个子匹配结果索引被返回。当 **nth** 为 0 (默认值) 时，返回完整匹配子串索引；当 **nth** 为 1 时，第一个(括号内的)子匹配结果索引被返回；当 **nth** 为 2 时，第二个子匹配结果索引被返回，以此类推。当被指定的子匹配没有结果时，返回 **nil**。下面是一个例子：

```

local str = "hello, 1234"
local from, to = ngx.re.find(str, "([0-9])([0-9]+)", "jo", nil, 2)
if from then
    ngx.say("matched 2nd submatch: ", string.sub(str, from, to)) -- yields "234"
end

```

此 API 函数自 **v0.9.2** 版开始提供。

[返回目录 \(page 77\)](#)

ngx.re.gmatch

语法: *iterator, err = ngx.re.gmatch(subject, regex, options?)*

环境: *init_worker_by_lua**, *set_by_lua**, *rewrite_by_lua**, *access_by_lua**, *content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**, *ngx.timer.**, *balancer_by_lua**, *ssl_certificate_by_lua**, *ssl_session_fetch_by_lua**, *ssl_session_store_by_lua**

与 [ngx.re.match \(page 145\)](#) 行为类似，不同的是本函数返回一个 Lua 迭代器，使用户程序可以自行迭代 PCRE 正则表达式 **regex** 匹配字符串参数 **<subject>** 产生的所有结果。

当出现错误时，例如发现错误的正则表达式时，返回 **nil** 和一个描述错误的字符串。

下面用一个小例子演示基本用法：

```
local iterator, err = ngx.re.gmatch("hello, world!", "([a-z]+)", "i")
if not iterator then
    ngx.log(ngx.ERR, "error: ", err)
    return
end

local m
m, err = iterator()    -- m[0] == m[1] == "hello"
if err then
    ngx.log(ngx.ERR, "error: ", err)
    return
end

m, err = iterator()    -- m[0] == m[1] == "world"
if err then
    ngx.log(ngx.ERR, "error: ", err)
    return
end

m, err = iterator()    -- m == nil
if err then
    ngx.log(ngx.ERR, "error: ", err)
    return
end
```

更常见的是使用 Lua 循环：

```

    local it, err = ngx.re.gmatch("hello, world!", "([a-z]+)",
    "i")
    if not it then
        ngx.log(ngx.ERR, "error: ", err)
        return
    end

    while true do
        local m, err = it()
        if err then
            ngx.log(ngx.ERR, "error: ", err)
            return
        end

        if not m then
            -- no match found (any more)
            break
        end

        -- found a match
        ngx.say(m[0])
        ngx.say(m[1])
    end
end

```

可选参数 `options` 含义与使用方法与 [ngx.re.match \(page 145\)](#) 相同。

在当前实现中，本函数返回的迭代器仅可被用于单一请求。也就是说，此迭代器 不能被赋值给属于持久命名空间的变量，例如 Lua 包(模块)。

这个方法需要在 Nginx 中启用 PCRE 库。([Known Issue With Special Escaping Sequences \(page 23\)](#))。

这个功能最早出现在 `v0.2.1rc12` 版本中。

[返回目录 \(page 77\)](#)

ngx.re.sub

语法: `newstr, n, err = ngx.re.sub(subject, regex, replace, options?)`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

使用 Perl 兼容正则表达式 `regex` 匹配字符串 `subject`，将第一个结果替换为字符串或函数类型参数 `replace`。可选参数 `options` 含义与 [ngx.re.match \(page 145\)](#) 相同。

这个方法返回结果字符串以及成功替换的数量。当发生失败时，例如正则表达式或 `<replace>` 字符串参数语法错，将返回 `nil` 以及一个描述错误的字符串。

当 `replace` 是一个字符串时，它们被视为一个特殊字符串替换模板。例如：

```
local newstr, n, err = ngx.re.sub("hello, 1234",
"([0-9])[0-9]", "$0[$1]")
if newstr then
    -- newstr == "hello, [12][1]34"
    -- n == 1
else
    ngx.log(ngx.ERR, "error: ", err)
    return
end
```

其中 `$0` 指模板匹配的完整子字符串，`$1` 指第一个括号内匹配的子串。

花括号可以被用来从背景字符串中消除变量名歧义。

```
local newstr, n, err = ngx.re.sub("hello, 1234", "[0-9]",
"${0}00")
-- newstr == "hello, 100234"
-- n == 1
```

要在 `replace` 中使用美元字符(`$`)，可以使用另外一个该符号作为转义，例如，

```
local newstr, n, err = ngx.re.sub("hello, 1234", "[0-9]",
"$")
-- newstr == "hello, $234"
-- n == 1
```

不要使用反斜线转义美元字符；它不会象你想象的那样工作。

当 `replace` 参数是一个“函数”时，它将被通过参数“匹配表”调用，用来生成替换字符串。被送入 `replace` 函数的“匹配表”与 [ngx.re.match \(page 145\)](#) 的返回值相同。例如：

```

local func = function (m)
    return "[" .. m[0] .. "]"[" .. m[1] .. "]"
end
local newstr, n, err = ngx.re.sub("hello, 1234", "( [0-9] )
[0-9]", func, "x")
    -- newstr == "hello, [12][1]34"
    -- n == 1

```

在 `replace` 函数返回值中的美元字符没有任何特殊含义。

这个方法需要在 Nginx 中启用 PCRE 库。([Known Issue With Special Escaping Sequences \(page 23\)](#)).

这个功能最早出现在 `v0.2.1rc13` 版本中。

[返回目录 \(page 77\)](#)

ngx.re.gsub

语法: `newstr, n, err = ngx.re.gsub(subject, regex, replace, options?)`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`,
`content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`,
`ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

就象 [ngx.re.sub \(page 152\)](#), 但执行全局替换。

下面是例子:

```

local newstr, n, err = ngx.re.gsub("hello, world", "([a-z])[a-
z]+", "$0,$1", "i")
if newstr then
    -- newstr == "[hello,h], [world,w]"
    -- n == 2
else
    ngx.log(ngx.ERR, "error: ", err)
return
end

```



```
local func = function (m)
    return "[" .. m[0] .. "," .. m[1] .. "]"
end
local newstr, n, err = ngx.re.gsub("hello, world", "([a-z])[a-z]+", func, "i")
-- newstr == "[hello,h], [world,w]"
-- n == 2
```

这个方法需要在 Nginx 中启用 PCRE 库。([Known Issue With Special Escaping Sequences \(page 23\)](#)).

这个功能最早出现在 `v0.2.1rc15` 版本中。

[返回目录 \(page 77\)](#)

ngx.shared.DICT

语法: `dict = ngx.shared.DICT`

语法: `dict = ngx.shared[name_var]`

环境: `init_by_lua*`, `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

获取基于共享内存名为 `DICT` 的 Lua 字典对象，它是一个共享内存区块，通过 [lua_shared_dict \(page 69\)](#) 指令定义。

所有共享内存区块均被当前 nginx 服务器实例的所有 nginx worker 进程共享。

获取的 `dict` 对象有如下方法：

- [get \(page 157\)](#)
- [get_stale \(page 158\)](#)
- [set \(page 158\)](#)
- [safe_set \(page 160\)](#)
- [add \(page 160\)](#)
- [safe_add \(page 160\)](#)
- [replace \(page 161\)](#)
- [delete \(page 161\)](#)
- [incr \(page 162\)](#)

- [lpush \(page 163\)](#)
- [rpush \(page 163\)](#)
- [lpop \(page 163\)](#)
- [rpop \(page 164\)](#)
- [llen \(page 164\)](#)
- [flush_all \(page 165\)](#)
- [flush_expired \(page 165\)](#)
- [get_keys \(page 166\)](#)

例如：

```
http {
    lua_shared_dict dogs 10m;
    server {
        location /set {
            content_by_lua_block {
                local dogs = ngx.shared.dogs
                dogs:set("Jim", 8)
                ngx.say("STORED")
            }
        }
        location /get {
            content_by_lua_block {
                local dogs = ngx.shared.dogs
                ngx.say(dogs:get("Jim"))
            }
        }
    }
}
```

测试结果如下：

```
$ curl localhost/set
STORED

$ curl localhost/get
8

$ curl localhost/get
8
```

当访问 `/get` 时，无论有多少 Nginx worker 进程，将一直输出数字 `8`，因为 `dogs` 字典存储在共享内存中，对 所有 worker 进程可见。

当服务器配置重新加载时共享内存字典内容不会丢失 (不管是通过送 `HUP` 信号给 Nginx 进程或使用命令行 `-s reload` 属性)。

但是，当 Nginx 服务器退出时，字典存储内容将丢失。

这个功能最早出现在 `v0.3.1rc22` 版本中。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.get

语法: `value, flags = ngx.shared.DICT:get(key)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

从 `ngx.shared.DICT` (page 155) 字典中获取名为 `key` 的键 (key) 值。如果此 key 不存在或已过期，返回 `nil`。

当发生错误时，将返回 `nil` 和一个描述错误的字符串。

返回值将保持写入字典时的原始数据类型，例如，Lua 布尔型、数字型、或字符串型。

此方法的第一个参数必须是该字典对象本身，例如，

```
local cats = ngx.shared.cats
local value, flags = cats.get(cats, "Marry")
```

或使用 Lua 的“语法糖”形式来调用方法：

```
local cats = ngx.shared.cats
local value, flags = cats:get("Marry")
```

这两种形式完全等价。

如果用户标志 (`flags`) 是 `0` (默认值), 将不会返回 `flags` 值。

这个功能最早出现在 `v0.3.1rc22` 版本中。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.get_stale

语法: `value, flags, stale = ngx.shared.DICT:get_stale(key)`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

与 [get \(page 157\)](#) 方法类似, 但即使 `key` 已经过期依然返回值。

返回第三个值, `stale`, 来标识该 `key` 是否已经过期。

需要注意的是, 已过期的 `key` 无法保证存在, 所以永远不应该依赖已过期项的可用性。

此方法最早出现在 `0.8.6` 版本中。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.set

语法: `success, err, forcible = ngx.shared.DICT:set(key, value, exptime?, flags?)`

环境: `init_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`,
`balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`,
`ssl_session_store_by_lua*`

无条件在基于共享内存的字典 [ngx.shared.DICT \(page 155\)](#) 中设置一个 key-value (键-值)对。返回三个值:

- `success`: 布尔值, 用以标识 key-value 对是否存储成功;

- `err`: 文字描述的错误信息, 例如 `"no memory"` (内存不足);
- `forcible`: 布尔值, 用以标识是否有其他可用项被强制删除, 因为共享内存区块中的存储空间不足。

`value` 参数可以是 Lua 布尔值、数字、字符串, 或 `nil`。其类型也将被存储在字典中, 之后通过 [get \(page 157\)](#) 方法可以原样取出。

通过可选的 `exptime` 参数给写入的 key-value 对设定过期时间 (单位秒)。时间的精度是 `0.001` 秒。如果 `exptime` 送入 `0` (默认值), 此项将永不过期。

通过可选的 `flags` 参数给写入项设定一个用户标志, 之后可以与值一起被取出。用户标志在内部被存储为一个 32 位无符号整数。默认值是 `0`。此参数最早出现在 `v0.5.0rc2` 版本中。

当无法给当前的 key-value 项分配内存时, `set` 方法将根据最近最少使用 (Least-Recently Used, LRU) 算法, 删除存储中已有的项。需要注意的是, LRU 算法在这里优先考虑过期时间。如果在删除了最多数十项以后剩余存储空间依旧不足 (由于 [lua_shared_dict \(page 69\)](#) 定义的总存储空间限制或内存碎片化), `success` 将返回 `false`, 同时 `err` 将返回 `no memory`。

如果此方法执行成功的同时, 根据 LRU 算法强制移除了字典中其他尚未过期的项, `forcible` 返回值将是 `true`。如果存储时没有移除其他有效项, `forcible` 返回值将是 `false`。

此方法第一个参数必须是该字典对象本身, 例如,

```
local cats = ngx.shared.cats
local succ, err, forcible = cats.set(cats, "Marry", "it is a nice cat!")
```

或使用 Lua 的“语法糖”形式来调用方法:

```
local cats = ngx.shared.cats
local succ, err, forcible = cats:set("Marry", "it is a nice cat!")
```

这两种形式完全等价。

这个功能最早出现在 `v0.3.1rc22` 版本中。

这里需要注意的是, 虽然在内部 key-value 对设置是原子性的, 但这种原子性无法跨过方法调用边界。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.safe_set

语法: *ok, err = ngx.shared.DICT:safe_set(key, value, exptime?, flags?)*

环境: *init_by_lua*, set_by_lua*, rewrite_by_lua*, access_by_lua*, content_by_lua*, header_filter_by_lua*, body_filter_by_lua*, log_by_lua*, ngx.timer.*, balancer_by_lua*, ssl_certificate_by_lua*, ssl_session_fetch_by_lua*, ssl_session_store_by_lua**

类似 [set \(page 158\)](#) 方法，但当共享内存区块存储空间不足时，不覆盖 (最近最少使用的) 有效的项 (非过期项)。此时，它将返回 `nil` 和字符串 “no memory” (内存不足)。

此方法最早出现在 `0.7.18` 版本中。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.add

语法: *success, err, forcible = ngx.shared.DICT:add(key, value, exptime?, flags?)*

环境: *init_by_lua*, set_by_lua*, rewrite_by_lua*, access_by_lua*, content_by_lua*, header_filter_by_lua*, body_filter_by_lua*, log_by_lua*, ngx.timer.*, balancer_by_lua*, ssl_certificate_by_lua*, ssl_session_fetch_by_lua*, ssl_session_store_by_lua**

类似 [set \(page 158\)](#) 方法，但仅当存储字典 [ngx.shared.DICT \(page 155\)](#) 中 不存在该 key 时执行存储 key-value 对。

如果参数 `key` 在字典中已经存在 (且没有过期)，`success` 返回值为 `false`，同时 `err` 返回 `"exist"` (已存在)。

这个功能最早出现在 `v0.3.1rc22` 版本中。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.safe_add

语法: *ok, err = ngx.shared.DICT:safe_add(key, value, exptime?, flags?)*

环境: `init_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

类似 [add \(page 160\)](#) 方法，但当共享内存块存储空间不足时，不覆盖 (最近最少使用的) 有效的项 (非过期项)。此时，它将返回 `nil` 和字符串 “no memory” (内存不足)。

这个功能最早出现在 `v0.7.18` 版本中。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.replace

语法: `success, err, forcible = ngx.shared.DICT:replace(key, value, exptime?, flags?)`

环境: `init_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

与 [set \(page 158\)](#) 方法类似，但仅当存储字典 [ngx.shared.DICT \(page 155\)](#) 中 存在该 key 时执行存储 key-value 对。

如果参数 `key` 在字典中 不存在 (或已经过期)，`success` 返回值为 `false`，同时 `err` 返回 “not found” (没找到)。

这个功能最早出现在 `v0.3.1rc22` 版本中。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.delete

语法: `ngx.shared.DICT:delete(key)`

环境: `init_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

从基于同享内存的字典 [ngx.shared.DICT \(page 155\)](#) 中无条件移除 key-value 对。

同 `ngx.shared.DICT:set(key, nil)` 等价。

这个功能最早出现在 `v0.3.1rc22` 版本中。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.incr

语法: `newval, err, forcible? = ngx.shared.DICT:incr(key, value, init?)`

环境: `init_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

在基于共享内存的字典 [ngx.shared.DICT \(page 155\)](#) 中递增 `key` 的 (数字) 值, 步长为 `value`。当操作成功时返回结果数字, 否则返回 `nil` 和错误信息字符串。

当 `key` 在共享内存字典中不存在或已经过期: When the key does not exist or has already expired in the shared dictionary,

1. 如果 `init` 参数没有指定或使用 `nil`, 该方法将返回 `nil` 并返回错误信息 `"not found"`。
2. 如果 `init` 参数被指定是一个 `number` 类型, 该方法将创建一个以 `init + value` 为值的新 `key`。

如同 [add \(page 160\)](#) 方法, 当共享内存区出现空间不足时, 他也会覆盖存储中未过期的数据项 (最近最少使用规则)。

当 `init` 参数没有指定时, `forcible` 参数将永远返回 `nil`。

如果该方法调用成功, 但它是通过数据字典的 LRU 方式强制删除其他未完结过期的数据项, `forcible` 的返回值将是 `true`。如果本次数据项存储没有强制删除任何其他有效数据, `forcible` 的返回值将是 `false`。

如果 `key` 的原始值不是一个有效的 Lua 数字, 返回 `nil` 和 `"not a number"` (不是数字)。

`value` 和 `init` 参数可以是任意有效的 Lua 数字, 包括负数和浮点数。

这个功能最早出现在 `v0.3.1rc22` 版本中。

在 `v0.10.6` 版本首次加入可选 `init` 参数。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.lpush

syntax: *length, err = ngx.shared.DICT:lpush(key, value)*

context: *init_by_lua*, set_by_lua*, rewrite_by_lua*, access_by_lua*, content_by_lua*, header_filter_by_lua*, body_filter_by_lua*, log_by_lua*, ngx.timer.*, balancer_by_lua*, ssl_certificate_by_lua*, ssl_session_fetch_by_lua*, ssl_session_store_by_lua**

在基于共享字典 [ngx.shared.DICT \(page 155\)](#) 命名是 **key** 的链表头部插入指定的（数字或字符串）**value**，返回值是插入后链表包含的对象数量。

如果 **key** 不存在，会在执行插入操作之前创建一个空的链表。当 **key** 已经有值但不是链表，会返回 **nil** 和 **"value not a list"**。

当共享内存区间中的存储空间不足时，它永远不会覆盖这里未过期数据（最近最少使用）。这种情况，它将直接返回 **nil** 和字符串 **"no memory"**。

该特性在 **v0.10.6** 版本首次引入。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.rpush

syntax: *length, err = ngx.shared.DICT:rpush(key, value)*

context: *init_by_lua*, set_by_lua*, rewrite_by_lua*, access_by_lua*, content_by_lua*, header_filter_by_lua*, body_filter_by_lua*, log_by_lua*, ngx.timer.*, balancer_by_lua*, ssl_certificate_by_lua*, ssl_session_fetch_by_lua*, ssl_session_store_by_lua**

与 [lpush \(page 163\)](#) 方法相似，但该方法将指定 **value**（数字或字符串）插入到命名为 **key** 的链表末尾。

该特性在 **v0.10.6** 版本首次引入。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.lpop

syntax: *val, err = ngx.shared.DICT:lpop(key)*

context: *init_by_lua**, *set_by_lua**, *rewrite_by_lua**, *access_by_lua**,
*content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**,
*ngx.timer.**, *balancer_by_lua**, *ssl_certificate_by_lua**, *ssl_session_fetch_by_lua**,
*ssl_session_store_by_lua**

删除并返回基于共享字典 [ngx.shared.DICT \(page 155\)](#) 命名为 **key** 链表的第一个对象。

如果 **key** 不存在，它将返回 **nil**。当 **key** 已经存在却不是链表时，将返回 **nil** 和 **"value not a list"**。

该特性在 **v0.10.6** 版本首次引入。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.rpop

syntax: *val, err = ngx.shared.DICT:rpop(key)*

context: *init_by_lua**, *set_by_lua**, *rewrite_by_lua**, *access_by_lua**,
*content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**,
*ngx.timer.**, *balancer_by_lua**, *ssl_certificate_by_lua**, *ssl_session_fetch_by_lua**,
*ssl_session_store_by_lua**

删除并返回基于共享字典 [ngx.shared.DICT \(page 155\)](#) 命名为 **key** 链表的最后一个对象。

如果 **key** 不存在，它将返回 **nil**。当 **key** 已经存在却不是链表时，将返回 **nil** 和 **"value not a list"**。

该特性在 **v0.10.6** 版本首次引入。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.llen

syntax: *len, err = ngx.shared.DICT:llen(key)*

context: *init_by_lua**, *set_by_lua**, *rewrite_by_lua**, *access_by_lua**,
*content_by_lua**, *header_filter_by_lua**, *body_filter_by_lua**, *log_by_lua**,
*ngx.timer.**, *balancer_by_lua**, *ssl_certificate_by_lua**, *ssl_session_fetch_by_lua**,
*ssl_session_store_by_lua**

返回基于共享字典 [ngx.shared.DICT \(page 155\)](#) 命名为 **key** 的链表长度。

如果 `key` 不存在, 将被解释为一个空链表, 所以返回 0。当 `key` 已经存在却不是链表时, 将返回 `nil` 和 `"value not a list"`。

该特性在 `v0.10.6` 版本首次引入。

更多功能请参考 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.flush_all

语法: `ngx.shared.DICT:flush_all()`

环境: `init_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

清空字典中的所有内容。这个方法并不实际释放字典占用的内存块, 而是标记所有存在的内容为已过期。

这个功能最早出现在 `v0.5.0rc17` 版本中。

更多功能请参考 [ngx.shared.DICT.flush_expired \(page 165\)](#) 和 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.flush_expired

语法: `flushed = ngx.shared.DICT:flush_expired(max_count?)`

环境: `init_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

清除字典中已过期的内容, 最多清除可选参数 `max_count` (最大数量) 个。当参数 `max_count` 值为 `0` 或者未指定时, 意为无数量限制。返回值为实际清除的数量。

与 [flush_all \(page 165\)](#) 方法不同, 此方法释放删除掉的已过期内容占用的内存。

这个功能最早出现在 `v0.6.3` 版本中。

更多功能请参考 [ngx.shared.DICT.flush_all \(page 165\)](#) 和 [ngx.shared.DICT \(page 155\)](#)。

[返回目录 \(page 77\)](#)

ngx.shared.DICT.get_keys

语法: `keys = ngx.shared.DICT:get_keys(max_count?)`

环境: `init_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

获取字典中存储的 key 列表，最多 `<max_count>` 个。

默认时，前 1024 个 key (如果有) 被返回。当参数 `<max_count>` 值为 `0` 时，字典中所有的 key 被返回，即使超过 1024 个。

警告 在包含非常多 key 的字典中调用此方法要非常小心。此方法会锁定字典一段时间，会阻塞所有访问字典的 nginx worker 进程。

这个功能最早出现在 `v0.7.3` 版本中。

[返回目录 \(page 77\)](#)

ngx.socket.udp

语法: `udpsock = ngx.socket.udp()`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

创建并得到一个 UDP 或 unix 域数据报 socket 对象（也被称为“cosocket”对象的一种类型）。该对象支持下面这些方法：

- [setpeername \(page 167\)](#)
- [send \(page 168\)](#)
- [receive \(page 168\)](#)
- [close \(page 169\)](#)
- [settimeout \(page 170\)](#)

不仅完整兼容 `LuaSocket` 库的 UDP API，而且还是 100% 非阻塞的。

该特性是在 `v0.5.7` 版本首次引入的。

也可以看看 [ngx.socket.tcp \(page 170\)](#)。

[返回目录 \(page 77\)](#)

udpsock:setpeername

语法: `ok, err = udpsock:setpeername(host, port)`

语法: `ok, err = udpsock:setpeername("unix:/path/to/unix-domain.socket")`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`,
`ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

尝试对远端服务或 unix 域数据报 socket 文件建立 UDP socket 对象。因为 UDP 实际上是 无连接 的，该方法并没有真正建立一条连接，但为了后续读/写操作只是设置了远程端点的名称。

对 `host` 参数 IP 地址和域名在这里都是可以使用的。当使用域名时，该方法将使用 Nginx 内部的动态域名解析器（非阻塞并且需要在 `nginx.conf` 文件中配置 `resolver` 指令），例如：

```
resolver 8.8.8.8; # use Google's public DNS nameserver
```

如果域名服务器对这个主机名返回多个 IP 地址，该方法将从中随机挑选一个。

失败情况下，该方法返回 `nil` 和 错误字符描述信息。成功情况下，该方法返回 `1`。

这里是个连接到 UDP（memcached）服务的示例：

```
location /test {
    resolver 8.8.8.8;

    content_by_lua_block {
        local sock = ngx.socket.udp()
        local ok, err = sock:setpeername("my.memcached.server.domain", 11211)
        if not ok then
            ngx.say("failed to connect to memcached: ", err)
            return
        end
        ngx.say("successfully connected to memcached!")
        sock:close()
    }
}
```

自 **v0.7.18** 版本以来，在 Linux 平台连接到 unix 域数据报 socket 文件也是可能的：

```
local sock = ngx.socket.udp()
local ok, err = sock:setpeername("unix:/tmp/some-datagram-service.sock")
if not ok then
    ngx.say("failed to connect to the datagram unix domain socket: ", err)
    return
end
```

假设数据报服务在 unix domain socket 文件 **/tmp/some-datagram-service.sock** 上监听，并且客户端 socket 将在 Linux 上使用“autobind”。

对已经连接的 socket 对象调用该方法将导致原始连接将先被关闭。

该特性是在 **v0.5.7** 版本首次引入的。

[返回目录 \(page 77\)](#)

udpsock:send

语法: *ok, err = udpsock:send(data)*

环境: *rewrite_by_lua*, access_by_lua*, content_by_lua*, ngx.timer.*, ssl_certificate_by_lua*, ssl_session_fetch_by_lua**

对当前 UDP 或 unix 域数据报 socket 对象发送数据。

成功情况下，返回 **1**。其他情况，返回 **nil** 和错误描述信息。

输入参数 **data** 可以是 Lua 字符串，也可以是包含字符串的（嵌套）Lua 表。对于输入参数是表的情况，该方法将逐一拷贝所有的字符串对象到底层的 Nginx socket 发送缓冲区，这是比 Lua 层面完成字符串拼接更好的优化方案。

该特性是在 **v0.5.7** 版本首次引入的。

[返回目录 \(page 77\)](#)

udpsock:receive

语法: *data, err = udpsock:receive(size?)*

环境: *rewrite_by_lua*, access_by_lua*, content_by_lua*, ngx.timer.*, ssl_certificate_by_lua*, ssl_session_fetch_by_lua**

使用一个可选的接收缓冲区大小参数 `size`，从 UDP 或 unix 域数据报 socket 对象中读取数据。

该方法是同步操作并且 100% 非阻塞。

成功情况，返回已经接收的数据；错误情况，返回 `nil` 和错误描述信息。

如果指定了 `size` 参数，该方法将使用它作为缓冲区大小。但是当该值比 `8192` 大时，将继续使用 `8192` 这个大小。

如果没有参数指定，那么最大的缓冲区大小，将假定 `8192`。

读操作超时控制，是由 `lua_socket_read_timeout` (page 70) 配置指令和 `settimeout` (page 170) 方法设置的。而后者有更高的优先级，例如：

```
sock:settimeout(1000)  -- one second timeout
local data, err = sock:receive()
if not data then
    ngx.say("failed to read a packet: ", err)
    return
end
ngx.say("successfully read a packet: ", data)
```

调用这个方法 之前 调用 `settimeout` (page 170) 方法设置超时时间，是非常重要的。

该特性是在 `v0.5.7` 版本首次引入的。

[返回目录 \(page 77\)](#)

udpsock:close

语法: `ok, err = udpsock:close()`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

关闭当前 UDP 或 unix 域数据报 socket。成功情况返回 `1`，反之错误情况返回 `nil` 和 错误描述信息。

当 socket 对象被 Lua GC（垃圾回收）释放或当前客户端请求处理完毕，还没有调用该方法的 socket 对象（关联连接）都将被关闭。

该特性是在 `v0.5.7` 版本首次引入的。

[返回目录 \(page 77\)](#)

udpsock:settimeout

语法: `udpsock:settimeout(time)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

设置随后的 socket 操作（例如 [receive \(page 168\)](#)）的超时时间（毫秒为单位）。

通过该方法设置内容相比这些配置指令有更高的优先级，例如：
[lua_socket_read_timeout \(page 70\)](#)。

该 API 在 `v0.5.7` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.socket.stream

只是 [ngx.socket.tcp \(page 170\)](#) 的一个别名。如果是 stream 类型的 cosocket 并有可能连接到 unix domain socket，这时候该 API 名称更贴切。

该 API 在 `v0.10.1` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.socket.tcp

语法: `tcpsock = ngx.socket.tcp()`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

创建并得到一个 TCP 或 unix 域流式 socket 对象（也被称为“cosocket”对象的一种类型）。该对象支持下面这些方法：

- [connect \(page 171\)](#)
- [sslhandshake \(page 173\)](#)
- [send \(page 174\)](#)
- [receive \(page 175\)](#)
- [close \(page 178\)](#)
- [settimeout \(page 179\)](#)
- [settimeouts \(page 179\)](#)

- [setoption \(page 180\)](#)
- [receiveuntil \(page 176\)](#)
- [setkeepalive \(page 180\)](#)
- [getreusedtimes \(page 181\)](#)

不仅完整兼容 [LuaSocket](#) 库的 TCP API，而且还是 100% 非阻塞的。此外，我们引入了一些新的API，提供更多功能。

通过该 API 函数创建的 `cosocket` 对象，与创造它的 Lua 环境拥有相同的生命周期。所以永远不要把 `cosocket` 对象传递给其他 Lua 环境（包括 `ngx.timer` 回调函数），并且永远不要在两个不同的 Nginx 请求之间共享 `cosocket` 对象。

对于任何一个 `cosocket` 对象的底层连接，如果你没有显式关闭（通过 [close \(page 178\)](#)）或把它放到连接池中（通过 [setkeepalive \(page 180\)](#)），一旦下面的两个事件中任何一个发生，它将被自动关闭：

- 当前请求处理执行完毕
- Lua `cosocket` 对象被 Lua GC（垃圾回收机制）回收

进行 `cosocket` 操作时发生致命错误总是会自动关闭当前连接（注意，读超时是这里唯一的非致命错误），并且如果你对一个已经关闭的连接调用 [close \(page 178\)](#)，你将得到“closed”的错误信息。

从 [0.9.9](#) 版本开始，`cosocket` 对象是全双工的，也就是说，一个专门读取的“light thread”，一个专门写入的“light thread”，它们可以同时为同一个 `cosocket` 对象进行操作（两个“light threads”必须运行在同一个 Lua 环境中，原因见上）。但是你不能让两个“light threads”对同一个 `cosocket` 对象都进行读（或者写入、或者连接）操作，否则当调用 `cosocket` 对象时，你将得到一个类似“socket busy reading”的错误。

该特性在 [v0.5.0rc1](#) 版本首次引入。

也可以看看 [ngx.socket.udp \(page 166\)](#)。

[返回目录 \(page 77\)](#)

tcpsock:connect

语法: `ok, err = tcpsock:connect(host, port, options_table?)`

语法: `ok, err = tcpsock:connect("unix:/path/to/unix-domain.socket", options_table?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

尝试以非阻塞的方式，对远端服务或 unix domain socket 文件建立 TCP socket 对象。

在真正解析主机名并连接到后端服务之前，该方法将永远优先在连接池内（都是调用该方法或 `ngx.socket.connect` (page 181) 函数的连接）查找符合条件的空闲连接。

对 `host` 参数 IP 地址和域名在这里都是可以使用的。当使用域名时，该方法将使用 Nginx 内部的动态域名解析器（非阻塞并且需要在 `nginx.conf` 文件中配置 `resolver` 指令），例如：

```
resolver 8.8.8.8; # 使用 Google 的公用域名解析服务器
```

如果域名服务器对这个主机名返回多个 IP 地址，该方法将从中随机挑选一个。

错误情况下，该方法返回 `nil` 及错误描述信息。成功情况下，该方法返回 `1`。

这里是个连接到 TCP 服务的示例：

```
location /test {
    resolver 8.8.8.8;

    content_by_lua_block {
        local sock = ngx.socket.tcp()
        local ok, err = sock:connect("www.google.com", 80)
        if not ok then
            ngx.say("failed to connect to google: ", err)
            return
        end
        ngx.say("successfully connected to google!")
        sock:close()
    }
}
```

连接到 Unix Domain Socket 文件也是可能的：

```
local sock = ngx.socket.tcp()
local ok, err = sock:connect("unix:/tmp/memcached.sock")
if not ok then
    ngx.say("failed to connect to the memcached unix domain socket: ", err)
    return
end
```

假设 memcached（或其他服务）正在 Unix Domain Socket 文件 `/tmp/memcached.sock` 监听。

连接操作超时控制，是由 `lua_socket_connect_timeout` (page 69) 配置指令和 `settimeout` (page 179) 方法设置的。而后者有更高的优先级，例如：

```
local sock = ngx.socket.tcp()
sock:settimeout(1000) -- one second timeout
local ok, err = sock:connect(host, port)
```

调用这个方法 之前 调用 `settimeout` (page 179) 方法设置超时时间，是非常重要的。

对已经连接状态的 socket 对象再次调用该方法，将导致原本的连接首先被关闭。

对于该方法的最后一个参数是个可选的 Lua 表，用来指定各种连接选项：

- `pool` 对即将被使用的连接池指定一个名字。如果没有指定该参数，连接池的名字将自动生成，使用 `"<host>:<port>"` 或 `"<unix-socket-path>"` 的命名方式。

支持可选的表参数，是在 `v0.5.7` 版本首次引入。

该特性在 `v0.5.0rc1` 版本首次引入。

[返回目录 \(page 77\)](#)

tcpsock:sslhandshake

语法: `session, err = tcpsock:sslhandshake(reused_session?, server_name?, ssl_verify?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

对当前建立的连接上完成 SSL/TLS 握手。

可选参数 `reused_session` 可以是一个前任 SSL session 用户数据，是由访问同一个目的地的前一个 `sslhandshake` 调用返回的。对于短链接，重复使用 SSL session 通常可以加速握手速度，但对于开启连接池情况情况下不是很有用。该参数默认使用 `nil`。如果该参数使用布尔值 `false`，将不返回 SSL 会话用户数据，只返回一个 Lua 布尔值；其他情况下，成功时第一个参数将返回当前的 SSL session。

可选参数 `server_name` 被用来指名新的 TLS 扩展 Server Name Indication (SNI) 的服务名。使用 SNI 可以使得在服务端不同的服务可以共享同一个 IP 地址。同样，当 SSL 验证启用，参数 `server_name` 也被用来验证从远端服务端发送出来的证书中的服务名。

可选参数 `ssl_verify`，通过一个 Lua 布尔值来控制是否启用 SSL 验证。当设置为 `true` 时，服务证书将根据 `lua_ssl_trusted_certificate` (page 73) 指令指定的 CA 证书进行验证。你可能需要调整 `lua_ssl_verify_depth` (page 74) 指令来控制我们对证书链的验证深度。同样，当 `ssl_verify` 参数为 `true` 并且也指名了 `server_name`，在后面服务端证书中将被用来验证服务名。

可选参数 `send_status_req`，可以通过一个布尔值控制是否在 SSL 握手请求（在请求 OCSP stapling 一方）中发送 OCSP 状态。

对已经完成 SSL/TLS 握手的连接，该方法立即返回。

该特性在 `v0.9.11` 版本首次引入。

[返回目录 \(page 77\)](#)

tcpsock:send

语法: `bytes, err = tcpsock:send(data)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

在当前 TCP 或 Unix Domain Socket 连接上非阻塞的发送数据。

该方法是个同步操作，直到 所有的数据全部被刷写到系统 socket 发送缓冲区或有错误发生，否则不会返回。

成功情况下，返回已经发送数据字节数的总数。其他情况，返回 `nil` 和错误描述信息。

输入参数 `data` 可以是 Lua 字符串，也可以是包含字符串的（嵌套）Lua 表。对于输入参数是表的情况，该方法将逐一拷贝所有的字符串对象到底层的 Nginx socket 发送缓冲区，这是比 Lua 层面完成字符串拼接更好的优化方案。

发送超时控制，是由 `lua_socket_send_timeout` (page 70) 配置指令和 `settimeout` (page 179) 方法设置的。而后者有更高的优先级，例如：

```
sock:settimeout(1000) -- one second timeout
local bytes, err = sock:send(request)
```

调用这个方法 之前 调用 `settimeout` (page 179) 方法设置超时时间，是非常重要的。

一旦有任何错误发生，该方法将自动关闭当前连接。

该特性在 `v0.5.0rc1` 版本首次引入。

[返回目录 \(page 77\)](#)

tcpsock:receive

语法: `data, err, partial = tcpsock:receive(size)`

语法: `data, err, partial = tcpsock:receive(pattern?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`,
`ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

根据读取规则或大小，从当前连接 `socket` 中接收数据。

与 [send \(page 174\)](#) 方法一样是同步操作，并且 100% 非阻塞。

成功情况下，返回接收到的数据；失败情况，返回 `nil`、一个错误描述信息以及目前接收到的部分数据。

如果指定一个像数字的参数（里面是数字的字符串），它将解析为大小。该方法将不返回直到读取明确大小的数据或有错误发生。

如果一个不像数字的字符串参数被指定，它将解析为“规则”。支持下面的规则：

- `'*a'`：从 `socket` 中读取内容直到连接被关闭。这里是不执行 `end-of-line` 翻译。
- `'*l'`：从 `socket` 中读取一行文本信息。行结束符是 `Line Feed`（LF）字符（ASCII 10），前面是一个可选 `Carriage Return`（CR）字符（ASCII 13）。返回行信息中不包含 CR 和 LF 字符。实际上，所有的 CR 字符在此规则中都是被忽略的。

如果没有参数指定，它将被假定为规则 `'*l'`，也就是说，使用按行读取的规则。

读操作超时控制，是由 [lua_socket_read_timeout \(page 70\)](#) 配置指令和 [settimeout \(page 179\)](#) 方法设置的。而后者有更高的优先级，例如：

```
sock:settimeout(1000) -- one second timeout
local line, err, partial = sock:receive()
if not line then
    ngx.say("failed to read a line: ", err)
    return
end
ngx.say("successfully read a line: ", line)
```

调用这个方法 之前 调用 [settimeout \(page 179\)](#) 方法设置超时时间，是非常重要的。

自从 **v0.8.8** 版本，当出现读取超时错误时，该方法不再自动关闭当前连接。对于其他连接错误，该方法总是会关闭连接。

该特性是在 **v0.5.0rc1** 版本首次引入的。

[返回目录 \(page 77\)](#)

tcpsock:receiveuntil

语法: `iterator = tcpsock:receiveuntil(pattern, options?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

该方法返回一个迭代的 Lua 函数，该函数可以被调用读取数据流直到指定的规则或有错误发生。

这里有个例子，使用这个方法读取边界序列为 **--abcedhb** 数据流：

```
local reader = sock:receiveuntil("\r\n--abcedhb")
local data, err, partial = reader()
if not data then
    ngx.say("failed to read the data stream: ", err)
end
ngx.say("read the data stream: ", data)
```

当不使用任何参数调用时，迭代函数返回的接收数据是指定规则 *之前* 的输入数据流。所以对于上面的例子，如果输入数据流是 **'hello, world! -agentzh\r\n--abcedhb blah blah'**，然后将返回字符串 **'hello, world! -agentzh'**。

错误的情况下，迭代函数将返回 **nil**、错误描述信息以及已经读取到的部分数据内容。

迭代函数可以被多次调用，并且可以安全的与其他 `cosocket` 方法或其他迭代函数混合调用。

当这个迭代函数使用 **size** 参数时的行为有点不同（比如，真正的迭代）。就是说，每次调用它将读取 **size** 大小的数据，最后一次调用（无论找到边界规则或遇到错误）将返回 **nil**。该迭代函数的最后一次成功调用，**err** 的返回值也将是 **nil**。在最后一次成功调用（返回数据 **nil**，错误信息 **nil**）之后，该迭代函数将会被重置。细看下面的例子：

```
local reader = sock:receiveuntil("\r\n--abcedhb")

while true do
    local data, err, partial = reader(4)
    if not data then
        if err then
            ngx.say("failed to read the data stream: ", err)
            break
        end
        ngx.say("read done")
        break
    end
    ngx.say("read chunk: [", data, "]")
end
```

对于输入数据是 `'hello, world! -agentzh\r\n--abcedhb blah blah'` , 使用上面的示例代码我们将得到下面输出 :

```
read chunk: [hell]
read chunk: [o, w]
read chunk: [orld]
read chunk: [! -a]
read chunk: [gent]
read chunk: [zh]
read done
```

注意, 当边界规则对数据流解析有歧义时, 实际返回数据长度 可能会略大于 `size` 参数指定的大小限制。在数据流的边界, 返回的字符数据长度同样也可能小于 `size` 参数限制。

迭代函数的读操作超时控制, 是由 [lua_socket_read_timeout \(page 70\)](#) 指令配置和 [settimeout \(page 179\)](#) 方法设置的。而后者有更高的优先级, 例如 :

```

local readline = sock:receiveuntil("\r\n")

sock:settimeout(1000) -- one second timeout
line, err, partial = readline()
if not line then
    ngx.say("failed to read a line: ", err)
    return
end
ngx.say("successfully read a line: ", line)

```

在调用迭代函数（注意 `receiveuntil` 调用在这里是不相干的）之前调用 `settimeout` (page 179) 方法是非常重要的。

从 `v0.5.1` 版本开始，该方法接收一个可选的 `options` 表参数来控制一些行为。支持下面这些选项：

- `inclusive`

`inclusive` 用一个布尔值来控制返回数据串是否包含规则字符串，默认是 `false`。例如：

```

local reader = tcpsock:receiveuntil("_END_", { inclusive = true })
local data = reader()
ngx.say(data)

```

然后对于数据流 `"hello world _END_ blah blah blah"`，根据上面的示例代码将得到 `hello world _END_` 的输出，包含规则字符串 `_END_` 自身。

自从 `v0.8.8` 版本，当出现读取超时错误时，该方法不再自动关闭当前连接。对于其他连接错误，该方法总是会关闭连接。

该特性是在 `v0.5.0rc1` 版本首次引入的。

[返回目录 \(page 77\)](#)

tcpsock:close

语法: `ok, err = tcpsock:close()`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

关闭当前 TCP 或 unix domain socket。成功情况下返回 `1`，否则将返回 `nil` 和错误描述信息。

注意，调用了 [setkeepalive \(page 180\)](#) 方法的 socket 对象，不再需要调用这个方法，因为这个 socket 对象已经关闭（当前连接已经保存到内建的连接池内）。

当 socket 对象已经被 Lua GC（垃圾回收）或当前客户 HTTP 请求完成处理时，没有调用这个方法的 socket 对象（和其他关联连接）将会被关闭。

该特性是在 `v0.5.0rc1` 版本首次引入的。

[返回目录 \(page 77\)](#)

tcpsock:settimeout

语法: `tcpsock:settimeout(time)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

对 [connect \(page 171\)](#)、[receive \(page 175\)](#)、基于 [receiveuntil \(page 176\)](#) 迭代返回，设置随后的 socket 操作的超时时间（毫秒为单位）。

通过该方法设置内容相比这些配置指令有更高的优先级，例如：

[lua_socket_connect_timeout \(page 69\)](#)、[lua_socket_send_timeout \(page 70\)](#)、[lua_socket_read_timeout \(page 70\)](#)。

注意，该方法 不会对 [lua_socket_keepalive_timeout \(page 71\)](#) 有任何影响，这个目的应换用 [setkeepalive \(page 180\)](#) 的 `timeout` 参数。

该特性是在 `v0.5.0rc1` 版本首次引入的。

[返回目录 \(page 77\)](#)

tcpsock:settimeouts

syntax: `tcpsock:settimeouts(connect_timeout, send_timeout, read_timeout)`

context: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

设置连接超时阈值、发送超时阈值和读取超时阈值，以毫秒为单位控制随后的 socket 操作（[connect \(page 171\)](#)，[send \(page 174\)](#)，[receive \(page 175\)](#) 和 [receiveuntil \(page 176\)](#) 方法返回的迭代操作）。

通过该方法设置定的值，相比这些配置指令有更高的优先级，比如：

[lua_socket_connect_timeout \(page 69\)](#)、[lua_socket_send_timeout \(page 70\)](#) 和 [lua_socket_read_timeout \(page 70\)](#)。

推荐使用 [settimeouts \(page 179\)](#) 方法替代 [settimeout \(page 179\)](#)。

注意：该方法不影响 `lua_socket_keepalive_timeout` (page 71) 设定，这种情况应调用 `setkeepalive` (page 180) 方法完成目的。

该特性在 `v0.10.7` 版本首次引入。

[返回目录 \(page 77\)](#)

tcpsock:setoption

语法: `tcpsock:setoption(option, value?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`,
`ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

该函数是为兼容 `LuaSocket` API，目前没做任何事情。它的功能将在将来实现。

该特性是在 `v0.5.0rc1` 版本首次引入的。

[返回目录 \(page 77\)](#)

tcpsock:setkeepalive

语法: `ok, err = tcpsock:setkeepalive(timeout?, size?)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`,
`ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

把当前 socket 连接立即放到内建的 cosocket 连接池中，维持活动状态直到被其他 `connect` (page 171) 方法调用请求，或者达到自身绑定的最大空闲时间后连接过期。

第一个可选参数参数 `timeout`，可以用来指定当前连接的最大空闲时间（单位毫秒）。如果没指定该参数，将使用配置在 `lua_socket_keepalive_timeout` (page 71) 指令的值作为默认值使用。如果给定的值是 `0`，那么超时时间是没有限制的。

第二个参数 `size`，可以用来指定当前服务（例如，当前主机+端口配对或 unix socket 文件路径作为标识）在连接池中允许存放的最大连接数。注意，连接池大小一旦创建后，是不能被修改的。如果没指定该参数，将使用配置在 `lua_socket_pool_size` (page 71) 指令的值作为默认值使用。

当连接池中连接数超过限制大小，在连接池中最近最少使用的（空闲）连接将被关闭，给当前连接腾挪空间。

注意，cosocket 连接池是每个 Nginx 工作进程使用的，而不是每个 Nginx 服务实例，所以这里指定的限制也只能在每个独立的 nginx 工作进程上生效。

连接池中空闲连接的任何异常事件都将会被监控，例如连接终止、在线收到非预期数据，在这种情况下有问题的连接，将被关闭并从池子中移除。

成功情况，该方法返回 `1`；否则，将返回 `nil` 和错误描述字符信息。

对于当前连接，当系统接收缓冲区有未读取完的数据，这时该方法将返回“connection in dubious state”的错误信息（作为第二个返回值），因为前一个请求留下了一些未读取数据给下一个请求，这种连接被重用是不安全的。

该方法也可以让当前 `cosocket` 对象进入 `closed` 状态，所以这里不再需要事后手工调用 `close` (page 178) 方法。

该特性是在 `v0.5.0rc1` 版本首次引入的。

[返回目录 \(page 77\)](#)

tcpsock:getreusedtimes

语法: `count, err = tcpsock:getreusedtimes()`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

该方法返回当前连接的使用次数（调用成功）。失败时，返回 `nil` 和错误描述字符信息。

如果当前连接不是从内建连接池中获取的，该方法总是返回 `0`，也就是说，该连接还没有被使用过。如果连接来自连接池，那么返回值永远都是非零。所以这个方法可以用来确认当前连接是否来自池子。

该特性是在 `v0.5.0rc1` 版本首次引入的。

[返回目录 \(page 77\)](#)

ngx.socket.connect

语法: `tcpsock, err = ngx.socket.connect(host, port)`

语法: `tcpsock, err = ngx.socket.connect("unix:/path/to/unix-domain.socket")`

环境: `rewrite_by_lua`, `access_by_lua`, `content_by_lua`, `ngx.timer.**`

该函数是融合 `ngx.socket.tcp()` (page 170) 和 `connect()` (page 171) 方法到一个单独操作的快捷方式。它实际上可以这样实现：

```
local sock = ngx.socket.tcp()
local ok, err = sock:connect(...)
if not ok then
    return nil, err
end
return sock
```

这里没办法使用 [settimeout \(page 179\)](#) 方法来指定连接时间，只能通过指令 [lua_socket_connect_timeout \(page 69\)](#) 预先配置作为替代方案。

该特性是在 **v0.5.0rc1** 版本首次引入的。

[返回目录 \(page 77\)](#)

ngx.get_phase

语法: `str = ngx.get_phase()`

环境: `init_by_lua*`, `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`,
`access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`,
`log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`,
`ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

检索当前正在执行的阶段名称。返回值可能有：

- **init** `init_by_lua*` ([page 37](#)) 运行环境。
- **init_worker** `init_worker_by_lua*` ([page 40](#)) 运行环境。
- **ssl_cert** `ssl_certificate_by_lua_block*` ([page 64](#)) 运行环境。
- **ssl_session_fetch** `ssl_session_fetch_by_lua*` ([page 65](#)) 运行环境。
- **ssl_session_store** `ssl_session_store_by_lua*` ([page 67](#)) 运行环境。
- **set** `set_by_lua*` ([page 42](#)) 运行环境。
- **rewrite** `rewrite_by_lua*` ([page 46](#)) 运行环境。
- **balancer** `balancer_by_lua_*` ([page 61](#)) 运行环境。
- **access** `access_by_lua*` ([page 51](#)) 运行环境。
- **content** `content_by_lua*` ([page 45](#)) 运行环境。
- **header_filter** `header_filter_by_lua*` ([page 53](#)) 运行环境。
- **body_filter** `body_filter_by_lua*` ([page 55](#)) 运行环境。
- **log** `log_by_lua*` ([page 58](#)) 运行环境。
- **timer** `ngx.timer.*` ([page 190](#)) 类的用户回调函数运行环境。

该 API 是从 **v0.5.10** 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.thread.spawn

语法: `co = ngx.thread.spawn(func, arg1, arg2, ...)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`,
`ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

使用 Lua 函数 `func` 以及其他可选参数 `arg1`、`arg2` 等，产生一个新的用户“轻线程”。返回一个 Lua 线程（或者说是 Lua 协程）对象，这里称之为“轻线程”。

“轻线程”是特殊类型的 Lua 协程，它是由 `ngx_lua` 模块调度的。

在 `ngx.thread.spawn` 返回之前，`func` 将被其他可选参数调用直到它返回，例如错误的终止或调用 [Nginx API for Lua \(page 77\)](#) (比如 [tcpsock.receive \(page 175\)](#)) 引起了 I/O 操作被挂起。

`ngx.thread.spawn` 返回后，新创建的“轻线程”将开始异步方式在各个 I/O 事件上执行。

在 [rewrite_by_lua \(page 46\)](#)、[access_by_lua \(page 51\)](#) 中的 Lua 代码块是在 `ngx_lua` 自动创建的“轻线程”样板执行的。这类样板的“轻线程”也被称为“入口线程”。

默认的，相应的 Nginx 处理部分（例如 [rewrite_by_lua \(page 46\)](#) 部分）将不会终止，直到：

1. “入口线程”和用户所有的“轻线程”都终止了
2. 一个“轻线程”（无论“入口线程”或用户“轻线程”），由于调用 [ngx.exit \(page 132\)](#)、[ngx.exec \(page 126\)](#)、[ngx.redirect \(page 128\)](#)、[ngx.req.set_uri\(uri, true\) \(page 111\)](#) 中止执行
3. “入口线程”因为 Lua 错误的终止

当用户“轻线程”因为 Lua 错误而中止，无论如何，它将不会中止其他运行的“轻线程”，就像“入口线程”一样。

归咎于 Nginx 子请求模型的限制，一般来说终止一个正在运行的 Nginx 子请求是不被允许的。挂起在一个或多个 Nginx 子请求结果的“轻线程”，也是禁止终止的。你必须调用 [ngx.thread.wait \(page 186\)](#) 等待这些“轻线程”退出这个“世界”。这里有个非正式异常办法，你可以调用 [ngx.exit \(page 132\)](#) 完成挂起子请求的终止，可以使用的状态码有：`ngx.ERROR` (-1)，`408`，`444`，或 `499`。

这些“轻线程”不会以抢占方式调度。换句话说，不会有自动执行的时间分片。“轻线程”将继续保持运行状态，只有当 CPU 出现下面情况：

1. 一个 I/O 操作（异步）不能在独立操作执行里完成
2. 调用 [coroutine.yield \(page 200\)](#) 声明放弃执行

3. 因为 Lua 错误或 [ngx.exit \(page 132\)](#)、[ngx.exec \(page 126\)](#)、[ngx.redirect \(page 128\)](#)、或 [ngx.req.set_uri\(uri, true\) \(page 111\)](#) 导致的终止

对于前两个示例，“轻线程”将通常稍候被 ngx_lua 调度唤醒，除非发生“全局停止”事件。

用户“轻线程”可以创建它们自己的“轻线程”。并且由 [coroutine.create \(page 199\)](#) 创建的普通用户协程 也能“轻线程”。直接产生“轻线程”的协程（可以是普通 Lua 协程或“轻线程”）被称为“父协程”，新产生的是“轻线程”。

“父协程”可以调用 [ngx.thread.wait \(page 186\)](#) 等待子“轻线程”的终止。

你可以在“轻线程”协程中调用 `coroutine.status()` 和 `coroutine.yield()`。

下面情况，“轻线程”协程可能是“僵尸”状态：

1. 当前“轻线程”已经终止（无论成功还是错误地）
2. 它的父协程还存活，但是他的父协程已经不再使用 [ngx.thread.wait \(page 186\)](#) 进行等待

下面的例子说明在“轻线程”协程中通过 `coroutine.yield()` 手工完成时间-切片：

```
local yield = coroutine.yield

function f()
    local self = coroutine.running()
    ngx.say("f 1")
    yield(self)
    ngx.say("f 2")
    yield(self)
    ngx.say("f 3")
end

local self = coroutine.running()
ngx.say("0")
yield(self)

ngx.say("1")
ngx.thread.spawn(f)

ngx.say("2")
yield(self)

ngx.say("3")
yield(self)

ngx.say("4")
```

然后它将生成下面的输出：

```
0
1
f 1
2
f 2
3
f 3
4
```

在一个独立的 Nginx 请求中完成上游请求的并发执行，“轻线程”是非常有用，有点像 [ngx.location.capture_multi \(page 103\)](#) 的一个普通版本，而且它能使用所有 [Nginx Lua 的 API \(page 77\)](#)。下面的例子说明在一个独立的 Nginx 请求中并行请求到 MySQL、Memcached 和上游 HTTP 服务，并以他们实际返回结果的顺序进行输出（与 Facebook BigPipe 模型非常相像）：

```

-- 同时查询 mysql、 memcached 和一个远程 HTTP 服务
-- 以它们实际返回结果的顺序进行输出

local mysql = require "resty.mysql"
local memcached = require "resty.memcached"

local function query_mysql()
    local db = mysql:new()
    db:connect{
        host = "127.0.0.1",
        port = 3306,
        database = "test",
        user = "monty",
        password = "mypass"
    }
    local res, err, errno, sqlstate =
        db:query("select * from cats order by id asc")
    db:set_keepalive(0, 100)
    ngx.say("mysql done: ", cJSON.encode(res))
end

local function query_memcached()
    local memc = memcached:new()
    memc:connect("127.0.0.1", 11211)
    local res, err = memc:get("some_key")
    ngx.say("memcached done: ", res)
end

local function query_http()
    local res = ngx.location.capture("/my-http-proxy")
    ngx.say("http done: ", res.body)
end

ngx.thread.spawn(query_mysql)      -- 创建线程 1
ngx.thread.spawn(query_memcached) -- 创建线程 2
ngx.thread.spawn(query_http)      -- 创建线程 3

```

该 API 是从 **v0.7.0** 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.thread.wait

语法: `ok, res1, res2, ... = ngx.thread.wait(thread1, thread2, ...)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `ngx.timer.*`,
`ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`

等待一个或多个子“轻线程”，并等待第一个终止（无论成功或有错误）“轻线程”的返回结果。

参数 `thread1`、`thread2` 等都是之前调用 `ngx.thread.spawn` (page 183) 返回的 Lua 线程对象。

返回值与 `coroutine.resume` (page 200) 是完全一样的，也就是说，第一个返回值是一个布尔值，说明“轻线程”的终止是成功还是异常，随后的返回值是 Lua 函数的返回结果，该 Lua 函数是被用来产生“轻线程”（成功情况下）或错误对象（失败情况下）。

只有直属“父协程”才能等待它的子“轻线程”，否则将会有 Lua 异常抛出。

下面的示范例子，是使用 `ngx.thread.wait` 和 `ngx.location.capture` (page 96) 共同来模拟 `ngx.location.capture_multi` (page 103)：

```
local capture = ngx.location.capture
local spawn = ngx.thread.spawn
local wait = ngx.thread.wait
local say = ngx.say

local function fetch(uri)
    return capture(uri)
end

local threads = {
    spawn(fetch, "/foo"),
    spawn(fetch, "/bar"),
    spawn(fetch, "/baz")
}

for i = 1, #threads do
    local ok, res = wait(threads[i])
    if not ok then
        say(i, ": failed to run: ", res)
    else
        say(i, ": status: ", res.status)
        say(i, ": body: ", res.body)
    end
end
```

这里它实质实现是“等待所有”模型。

下面的例子是示范“等待任何”模型：

```
function f()
    ngx.sleep(0.2)
    ngx.say("f: hello")
    return "f done"
end

function g()
    ngx.sleep(0.1)
    ngx.say("g: hello")
    return "g done"
end

local tf, err = ngx.thread.spawn(f)
if not tf then
    ngx.say("failed to spawn thread f: ", err)
    return
end

ngx.say("f thread created: ", coroutine.status(tf))

local tg, err = ngx.thread.spawn(g)
if not tg then
    ngx.say("failed to spawn thread g: ", err)
    return
end

ngx.say("g thread created: ", coroutine.status(tg))

ok, res = ngx.thread.wait(tf, tg)
if not ok then
    ngx.say("failed to wait: ", res)
    return
end

ngx.say("res: ", res)

-- stop the "world", aborting other running threads
-- 停止这个“世界”，终止其他正在运行的线程
ngx.exit(ngx.OK)
```

它将生成下面的输出：

```
f thread created: running
g thread created: running
g: hello
res: g done
```

该 API 是从 **v0.7.0** 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.thread.kill

语法: `ok, err = ngx.thread.kill(thread)`

环境: `rewrite_by_lua`, `access_by_lua`, `content_by_lua`, `ngx.timer.**`

杀死一个正在运行的轻线程（通过 [ngx.thread.spawn \(page 183\)](#) 创建）。成功时返回一个 `true`，其他情况则返回一个错误字符描述信息。

根据目前的实现，只有父协程（或“轻线程”）可以终止一个“线程”。同样，正在挂起运行 Nginx 子请求（例如调用 [ngx.location.capture \(page 96\)](#)）的“轻线程”，是不能被杀死的，这要归咎于 Nginx 内核限制。

该 API 是从 **v0.9.9** 版本开始首次引入。

[返回目录 \(page 77\)](#)

ngx.on_abort

语法: `ok, err = ngx.on_abort(callback)`

环境: `rewrite_by_lua`, `access_by_lua*`, `content_by_lua**`

注册一个用户回调函数，当客户端过早关闭当前连接（下游）时自动调用。

如果回调函数注册成功返回 **1**，或反之将返回 **nil** 和一个错误描述字符信息。

所有的 [Nginx Lua 的 API \(page 77\)](#) 都可以在这个回调中使用，因为这个函数是以一个特定的“轻线程”方式运行，就像其他使用 [ngx.thread.spawn \(page 183\)](#) 创建的“轻线程”。

该回调函数能自己决定对客户端终止事件做什么处理。例如，它可以简单的不做任何事情从而忽略这个事件，使得当前 Lua 请求处理可以没有任何打扰的继续执行。当然该回调函数也可以调用 [ngx.exit \(page 132\)](#) 从而终止所有处理，例如：

```

local function my_cleanup()
    -- custom cleanup work goes here, like cancelling a pending DB transaction

    -- now abort all the "light threads" running in the current request handler
    ngx.exit(499)
end

local ok, err = ngx.on_abort(my_cleanup)
if not ok then
    ngx.log(ngx.ERR, "failed to register the on_abort callback: ", err)
    ngx.exit(500)
end

```

当 [lua_check_client_abort \(page 76\)](#) 被设置为 `off`（这是默认值），这时这个函数调用将永远返回错误信息“lua_check_client_abort is off”。

根据当前实现，这个函数在单个请求中能且只能调用一次；随后的调用将收到错误消息“duplicate call”。

该 API 是从 `v0.7.4` 版本首次引入。

也可以看看 [lua_check_client_abort \(page 76\)](#)。

[返回目录 \(page 77\)](#)

ngx.timer.at

语法: `ok, err = ngx.timer.at(delay, callback, user_arg1, user_arg2, ...)`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

使用一个可选参数的用户回调函数，创建一个 Nginx 定时器。

第一个参数 `delay`，指定这个定时器的延迟时间，秒为单位。我们可以指定带有小数部分的时间像 `0.001`，这里代表 1 毫秒。当需要立即执行当前挂起的处理，指定延迟为 `0` 即可。

第二个参数 `callback`，可以是任何 Lua 函数，在指定的延迟时间之后，将会在一个后台的“轻线程”中被调用。这个调用是 Nginx 内核使用参数 `premature`，`user_arg1`，`user_arg2` 等参数自动完成的，`premature` 参数代表这个定时器是否过期的布尔值，`user_arg1`，`user_arg2` 等其他参数是调用 `ngx.timer.at` 的其余参数。

当 Nginx 工作进程正在尝试关闭时，定时器会过早失效，例如 `HUP` 信号触发的 Nginx 配置文件重载或 Nginx 服务关闭。当 Nginx 工作进程正在关闭，是不能调用 `ngx.timer.at` 来创建新的非零时间延迟定时器，这种情况下 `ngx.timer.at` 将返回 `nil` 和一个描述这个错误的字符串信息：“process exiting”。

从 `v0.9.3` 版本开始，当 Nginx 工作进程开始关闭时，是允许创建零延迟定时器的。

当定时器到期，定时器中的 Lua 代码是在一个“请线程”中运行的，它与创造它的原始请求是完全分离的。因此，和创造它的请求有相同生命周期的对象，比如 `cosockets` (page 170)，是不能在原始请求和定时器中的回调函数共享使用的。

这是个简单的例子：

```
location / {
    ...
    log_by_lua '
        local function push_data(premature, uri, args, status)
            -- push the data uri, args, and status to the remote
            -- via ngx.socket.tcp or ngx.socket.udp
            -- (one may want to buffer the data in Lua a bit to
            -- save I/O operations)
        end
        local ok, err = ngx.timer.at(0, push_data,
                                    ngx.var.uri, ngx.var.args,
                                    ngx.header.status)
        if not ok then
            ngx.log(ngx.ERR, "failed to create timer: ", err)
            return
        end
    ';
}
```

我们可以创建反复循环使用的定时器，例如，得到一个每 5 秒触发一次的定时器，可以在定时器中递归调用 `ngx.timer.at`。这里有个这样的例子：

```

local delay = 5
local handler
handler = function (premature)
    -- do some routine job in Lua just like a cron job
    if premature then
        return
    end
    local ok, err = ngx.timer.at(delay, handler)
    if not ok then
        ngx.log(ngx.ERR, "failed to create the timer: ", err)
        return
    end
end

local ok, err = ngx.timer.at(delay, handler)
if not ok then
    ngx.log(ngx.ERR, "failed to create the timer: ", err)
    return
end

```

因为定时调用是在后台运行，并且他们的执行不会增加任何客户端的响应时长，所以它们很容易让服务端累积错误并耗尽系统资源，有可能是 Lua 编程错误也可能仅仅是太多客户端请求。为了防止这种极端的恶果（例如：Nginx 服务崩溃），在 Nginx 工作进程里内建了“pending timers”和“running timers”两个数量限制。这里的“pending timers”代表还没有过期的定时器，“running timers”代表用户回调函数当前正在运行的定时器。

在 Nginx 进程内“pending timers”的最大数控制是 [lua_max_pending_timers \(page 77\)](#) 指令完成的。“running timers”的最大数控制是 [lua_max_running_timers \(page 77\)](#) 指令完成的。

根据当前实现，每一个“running timer”，都将从全局连接列表中占用一个（假）连接，全局列表通过 [nginx.conf](#) 的标准指令 [worker_connections](#) 配置。所以要确保 [worker_connections](#) 指令设置了足够大的值，用来存放真正的连接和定时器需要假连接（受限于 [lua_max_running_timers \(page 77\)](#) 指令）。

在定时器的回调函数中，很多 Nginx 的 Lua API 是可用的，像数据流/数据报 [cosocket](#) 的 ([ngx.socket.tcp \(page 170\)](#) 和 [ngx.socket.udp \(page 166\)](#))，共享内存字典 ([ngx.shared.DICT \(page 155\)](#))，用户协同程序 ([coroutine.\]\(#coroutinecreate\)](#))，用户“轻线程” ([ngx.thread. \(page 183\)](#))，[ngx.exit \(page 132\)](#)，[ngx.now \(page 142\)](#)/[ngx.time \(page 141\)](#)，[ngx.md5 \(page 139\)](#)/[ngx.sha1_bin \(page 140\)](#)，都是允许的。但是子请求 API（如 [ngx.location.capture \(page 96\)](#)），[ngx.req.* \(page 108\)](#)，下游输出 API（如 [ngx.say \(page 131\)](#)，[ngx.print \(page 130\)](#) 和 [ngx.flush \(page 132\)](#)），在这个环境是明确被禁用的。

定时器的回调函数可传入大多数标准 Lua 值（空、布尔值、数字、字符串、表、闭包、文件句柄等），无论是明确的用户参数或回调闭包的隐式值。这里有几个例外，通过 [coroutine.create \(page 199\)](#) 和 [ngx.thread.spawn \(page 183\)](#) 的线程对象，或通过 [ngx.socket.tcp \(page 170\)](#)、[ngx.socket.udp \(page 166\)](#) 和 [ngx.req.socket \(page 125\)](#) 得到的 cosocket 对象，他们都是不能作为传入对象的，因为这些对象的生命周期都是绑定到创建定时器的请求环境的，但定时器回调与创建环境是完全隔离的（设计上），并且是在自己的（假）请求环境中运行。如果你尝试在创建请求边界共享线程或 cosocket 对象，你将得到错误信息 “no co ctx found”（对于线程），”bad request”（对于 cosockets）。它是好的，所以，在你的定时器回调中创建所有这些对象。

这个 API 是从 **v0.8.0** 首次引入的。

[返回目录 \(page 77\)](#)

ngx.timer.running_count

语法: `count = ngx.timer.running_count()`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

返回当前 **running timers** 总数。

该指令从 **v0.9.20** 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.timer.pending_count

语法: `count = ngx.timer.pending_count()`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

返回待定 **pending timers** 数量。

该指令从 **v0.9.20** 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.config.subsystem

语法: `subsystem = ngx.config.subsystem`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`,
`init_worker_by_lua*`

该字段的值用来表明当前 Nginx 子系统的基础运行环境。例如该模块环境下，该字段的返回值永远为 `"http"` 字符串。对于 `ngx_stream_lua_module`，无论如何，该字段返回值为 `"stream"`。

该字段在 `0.10.1` 版本中首次引入。

[返回目录 \(page 77\)](#)

ngx.config.debug

语法: `debug = ngx.config.debug`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`,
`init_worker_by_lua*`

这个布尔值代表当前 Nginx 是否为调试版本，既，在编译时使用 `./configure` 的可选项 `--with-debug`。

该 API 在 `0.8.7` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.config.prefix

语法: `prefix = ngx.config.prefix()`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`,
`init_worker_by_lua*`

返回 Nginx 服务的“prefix”路径，它可能是由 Nginx 启动时通过可选 `-p` 命令行确定的，也可能是由编译 Nginx 的 `./configure` 脚本中可选的 `--prefix` 命令行参数确定的。

该 API 在 `0.9.2` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.config.nginx_version

语法: `ver = ngx.config.nginx_version`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`,
`init_worker_by_lua*`

这个字段是当前正在使用的 Nginx 内核版本数字标识。例如，版本 `1.4.3` 用 Lua 数字表示就是 1004003。

该 API 在 `0.9.3` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.config.nginx_configure

语法: `str = ngx.config.nginx_configure()`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`

该 API 返回编译 Nginx 时的 `./configure` 命令参数字符串。

该 API 在 `0.9.5` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.config.ngx_lua_version

语法: `ver = ngx.config.ngx_lua_version`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`

这个字段是当前正在使用的 `ngx_lua` 模块版本数字标识。例如，版本 `0.9.3` 用 Lua 数字表示就是 9003。

该 API 在 `0.9.3` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.worker.exiting

语法: `exiting = ngx.worker.exiting()`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`,
`init_worker_by_lua*`

该函数返回一个布尔值，表示目前 Nginx 的工作进程是否已经开始退出。Nginx 的工作进程退出，发生在 Nginx 服务退出或配置重载（又名HUP重载）。

该 API 在 `v0.6.0` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.worker.pid

语法: `pid = ngx.worker.pid()`

语法: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`,
`init_worker_by_lua*`

这个函数返回一个Lua数字，它是当前 Nginx 工作进程的进程 ID（PID）。这个 API 比 `ngx.var.pid` 更有效，`ngx.var.VARIABLE` (page 88) API 不能使用的地方（例如 `init_worker_by_lua` (page 40)），该 API 是可以的。

该 API 在 `v0.6.0` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.worker.count

语法: `count = ngx.worker.count()`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`,
`init_worker_by_lua*`

返回当前 Nginx 工作进程数的数量（既：在 `nginx.conf` 配置中，使用 `worker_processes` 指令配置的值）。

该 API 从 `v0.9.20` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.worker.id

语法: `count = ngx.worker.id()`

环境: `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`,
`header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `init_by_lua*`

返回当前 Nginx 工作进程的一个顺序数字（从 0 开始）。

所以，如果工作进程总数是 `N`，那么该方法将返回 0 和 `N - 1`（包含）的一个数字。

该方法只对 Nginx 1.9.1+ 版本返回有意义的值。更早版本的 nginx，将总是返回 `nil`。

同样可以看看 [ngx.worker.count \(page 196\)](#)。

该 API 从 `v0.9.20` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.semaphore

语法: `local semaphore = require "ngx.semaphore"`

该 Lua 模块，实现了一个经典的 semaphore API，可以高效的完成不同“轻线程”之间的同步。在不同的请求环境中创建的“轻线程”，可以共享同一个 semaphore 对象是支持的，只要这些“轻线程”是在同一个 nginx 工作进程中即可，此外要求 [lua_code_cache \(page 34\)](#) 指令是开启的（默认是开启的）。

该模块没与 ngx_lua 模块一起发行，而是与 [lua-resty-core](#) 库一起发行。

获取更多信息，请参考 Lua 模块 [lua-resty-core](#) 的 `ngx.semaphore` 信息，[文档](#)。

该特性在 `v0.10.0` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.balancer

语法: `local balancer = require "ngx.balancer"`

这是一个允许使用纯 Lua 语言，完整定义一个动态负载均衡 Lua API 模块。

该模块没与 ngx_lua 模块一起发行，而是与 [lua-resty-core](#) 库一起发行。

为获取更多信息，请参考 [lua-resty-core](#) Lua 模块的 `ngx.balancer` 章节 [文档](#)。

该特性在 `v0.10.0` 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.ssl

语法: `local ssl = require "ngx.ssl"`

该 Lua 模块提供 API 函数，可以在诸如 [ssl_certificate_by_lua*](#) (page 64) 环境中完成 SSL 握手过程的控制。

该模块没与 ngx_lua 模块一起发行，而是与 [lua-resty-core](#) 库一起发行。

获取更多 [ngx.ssl](#) Lua 模块信息，请参考 [文档](#)。

该特性在 [v0.10.0](#) 版本首次引入。

[返回目录 \(page 77\)](#)

ngx.ocsp

语法: `local ocsp = require "ngx.ocsp"`

该模块提供 API 完成 OCSP 查询、OCSP 响应验证和 OCSP stapling planting。

通常，该模块与 [ngx.ssl](#) 模块一起配合在 [ssl_certificate_by_lua*](#) (page 64) 的环境中使用。

该模块没与 ngx_lua 模块一起发行，而是与 [lua-resty-core](#) 库一起发行。

获取更多 [ngx.ocsp](#) Lua 模块信息，请参考 [文档](#)。

该特性在 [v0.10.0](#) 版本首次引入。

[返回目录 \(page 77\)](#)

ndk.set_var.DIRECTIVE

语法: `res = ndk.set_var.DIRECTIVE_NAME`

环境: `init_worker_by_lua*`, `set_by_lua*`, `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `log_by_lua*`, `ngx.timer.*`, `balancer_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

该机制允许调用这类 nginx C 模块指令：使用 [Nginx Devel Kit](#) (NDK) 的 `set_var` 的子模块的 [ndk_set_var_value](#) 实现。

例如，下列[set-misc-nginx-module](#) 指令是可以通过这个方式调用的：

- [set_quote_sql_str](#)
- [set_quote_pgsql_str](#)
- [set_quote_json_str](#)
- [set_unescape_uri](#)
- [set_escape_uri](#)

- [set_encode_base32](#)
- [set_decode_base32](#)
- [set_encode_base64](#)
- [set_decode_base64](#)
- [set_encode_hex](#)
- [set_decode_hex](#)
- [set_sha1](#)
- [set_md5](#)

举例：

```
local res = ndk.set_var.set_escape_uri('a/b');  
-- now res == 'a%2fb'
```

相似的，下列指令是由 [encrypted-session-nginx-module](#) 提供，他们在 Lua 中也可以被调用：

- [set_encrypt_session](#)
- [set_decrypt_session](#)

这个特性需要 [ngx_devel_kit](#) 模块。

[返回目录 \(page 77\)](#)

coroutine.create

语法: `co = coroutine.create(f)`

环境: `rewrite_by_lua*`, `access_by_lua*`, `content_by_lua*`, `init_by_lua*`, `ngx.timer.*`, `header_filter_by_lua*`, `body_filter_by_lua*`, `ssl_certificate_by_lua*`, `ssl_session_fetch_by_lua*`, `ssl_session_store_by_lua*`

通过一个 Lua 函数创建一个用户的 Lua 协程，并返回一个协程对象。

类似标准的 Lua [coroutine.create](#) API，但它是在 `ngx_lua` 创建的 Lua 协程环境中运行。

该 API 在 `init_by_lua*` ([page 37](#)) 的环境中可用，是从 [0.9.2](#) 开始的。

该 API 在 [v0.6.0](#) 版本首次引入。

[返回目录 \(page 77\)](#)

coroutine.resume

语法: `ok, ... = coroutine.resume(co, ...)`

环境: `rewrite_by_lua`, `access_by_lua`, `content_by_lua`, `init_by_lua`, `ngx.timer.`, `header_filter_by_lua*`, `body_filter_by_lua**`

恢复以前挂起或刚创建的用户 Lua 协程对象的执行。Resume the execution of a user Lua coroutine object previously yielded or just created.

类似标准的 Lua [coroutine.resume](#) API，但它是在 `ngx_lua` 创建的 Lua 协程环境中运行。

该 API 在 [init_by_lua*](#) (page 37) 的环境中可用，是从 **0.9.2** 开始的。

该 API 在 **v0.6.0** 版本首次引入。

[返回目录 \(page 77\)](#)

coroutine.yield

语法: `... = coroutine.yield(...)`

环境: `rewrite_by_lua`, `access_by_lua`, `content_by_lua`, `init_by_lua`, `ngx.timer.`, `header_filter_by_lua*`, `body_filter_by_lua**`

挂起当前用户 Lua 协程的执行。

类似标准的 Lua [coroutine.yield](#) API，但它是在 `ngx_lua` 创建的 Lua 协程环境中运行。

该 API 在 [init_by_lua*](#) (page 37) 的环境中可用，是从 **0.9.2** 开始的。

该 API 在 **v0.6.0** 版本首次引入。

[返回目录 \(page 77\)](#)

coroutine.wrap

语法: `co = coroutine.wrap(f)`

环境: `rewrite_by_lua`, `access_by_lua`, `content_by_lua`, `init_by_lua`, `ngx.timer.`, `header_filter_by_lua*`, `body_filter_by_lua**`

类似标准的 Lua [coroutine.wrap](#) API，但它是在 `ngx_lua` 创建的 Lua 协程环境中运行。

该 API 在 [init_by_lua*](#) (page 37) 的环境中可用，是从 **0.9.2** 开始的。

该 API 在 **v0.6.0** 版本首次引入。

[返回目录](#) (page 77)

coroutine.running

语法: `co = coroutine.running()`

环境: `rewrite_by_lua`, `access_by_lua`, `content_by_lua`, `init_by_lua`, `ngx.timer.`, `header_filter_by_lua*`, `body_filter_by_lua**`

与标准的 Lua `coroutine.running` API 相同。

该 API 在 [init_by_lua*](#) (page 37) 的环境中可用，是从 **0.9.2** 开始的。

该 API 在 **v0.6.0** 版本首次引入。

[返回目录](#) (page 77)

coroutine.status

语法: `status = coroutine.status(co)`

环境: `rewrite_by_lua`, `access_by_lua`, `content_by_lua`, `init_by_lua`, `ngx.timer.`, `header_filter_by_lua*`, `body_filter_by_lua**`

与标准的 Lua `coroutine.status` API 相同。

该 API 在 [init_by_lua*](#) (page 37) 的环境中可用，是从 **0.9.2** 开始的。

该 API 在 **v0.6.0** 版本首次引入。

[返回目录](#) (page 77)

Obsolete Sections

这里保留一些过期文档小节，虽然他们可能被重命名、删除，但他们的链接将一直有效。

[返回目录](#) (page 3)

Special PCRE Sequences

本章节已经被重命名为 [特殊的转义序列](#) (page 23)。

[返回目录](#) (page 3)

Introduction

Overview

This site provides documentation, training, and other notes for the Jekyll Documentation theme. There's a lot of information about how to do a variety of things here, and it's not all unique to this theme. But by and large, understanding how to do things in Jekyll depends on how your theme is coded. As a result, these additional details are provided.

The instructions here are geared towards technical writers working on documentation. You may have a team of one or more technical writers working on documentation for multiple projects. You can use this same theme to author all of your documentation for each of your products. The theme is built to accommodate documentation for multiple products on the same site.

Survey of features

Some of the more prominent features of this theme include the following:

- Bootstrap framework
- [Navgoco multi-level sidebar](#) for table of contents
- Ability to specify different sidebars for different products
- Top navigation bar with drop-down menus
- Notes, tips, and warning information notes
- Tags for alternative navigation
- Advanced landing page layouts from the [Modern Business theme](#) .

Getting started

To get started, see [Getting Started \(page 3\)](#).

Supported features

Summary: If you're not sure whether Jekyll and this theme will support your requirements, this list provides a semi-comprehensive overview of available features.

Before you get into exploring Jekyll as a potential platform for help content, you may be wondering if it supports some basic features needed to fulfill your tech doc requirements. The following table shows what is supported in Jekyll and this theme.

Supported features

Features	Supported	Notes
Content re-use	Yes	Supports re-use through Liquid. You can re-use variables, snippets of code, entire pages, and more. In DITA speak, this includes conref and keyref. See Content reuse (page 248) for more details.
Markdown	Yes	You can author content using Markdown syntax, specifically kramdown . This is a wiki-like syntax for HTML that you can probably pick up in 10 minutes. Where Markdown falls short, you can use HTML. Where HTML falls short, you use Liquid, which is a scripting that allows you to incorporate more advanced logic.
Responsive design	Yes	Uses Bootstrap framework for responsive design.
Translation	Yes	To translate content, send the generated HTML to your translation group. You can translate the Markdown source if your translator accepts the format, but usually Markdown is problematic. Note that this theme isn't structured well to accommodate translation projects.

Features	Supported	Notes
Collaboration	Yes	You collaborate with Jekyll projects the same way that developers collaborate with software projects. (You don't need a CMS.) Because you're working with text file formats, you can use any version control software (Git, Mercurial, Perforce, Bitbucket, etc.) as a CMS for your files.
Scalability	Yes	Your site can scale to any size. It's up to you to determine how you will design the information architecture for your pages. You can choose what you display at first, second, third, fourth, and more levels, etc. Note that when your project has thousands of pages, the build time will be longer (maybe 1 minute per thousand pages?). It really depends on how many for loops you have iterating through the pages. I recommend that you use smaller repos in your content architecture.
Lightweight architecture	Yes	You don't need a LAMP stack (Linux, Apache, MySQL, PHP) architecture to get your site running. All of the building is done on your own machine, and you then push the static HTML files onto a server.
Skinnability	Yes	You can skin your Jekyll site to look identical to pretty much any other site online. If you have a UX team, they can really skin and design the site using all the tools familiar to the modern designer – JavaScript, HTML5, CSS, jQuery, and more. Jekyll is built on the modern web development stack rather than the XML stack (XSLT, XPath, XQuery). See this tutorial for details on how to create your own Jekyll theme.
Support	Yes	The community for your Jekyll site isn't so much other tech writers (as is the case with DITA) but rather the wider web development community. Jekyll Talk is a great resource. So is Stack Overflow . See the Getting Help section of Jekyll.

Features	Supported	Notes
Blogging features	Yes	There is a simple blogging feature. This appears as news (page 0) and is intended to promote news that applies across products.
Versioning	Yes	Jekyll doesn't version your files. You upload your files to a version control system such as Github. Your files are versioned there.
PC platform	Yes	Jekyll runs on Windows. Although the experience working on the command line is better on a Mac, Windows also works, especially now that Jekyll 3.0 dropped dependencies on Python, which wasn't available by default on Windows.
jQuery plug-ins	Yes	You can use any jQuery plugins you and other JavaScript, CMS, or templating tools. However, note that if you use Ruby plugins, you can't directly host the source files on Github Pages because Github Pages doesn't allow Ruby plugins. Instead, you can just push your output to any web server. If you're not planning to use Github Pages, there are no restrictions on any plugins of any sort. Jekyll makes it super easy to integrate every kind of plugin imaginable. This theme doesn't actually use any plugins, so you can publish on Github if you want.
Bootstrap integration	Yes	This theme is built on Bootstrap . If you don't know what Bootstrap is, basically this means there are hundreds of pre-built components, styles, and other elements that you can simply drop into your site. For example, the responsive quality of the site comes about from the Bootstrap code base.

Features	Supported	Notes
Fast-loading pages	Yes	This is one of the Jekyll's strengths. Because the files are static, they loading extremely fast, approximately 0.5 seconds per page. You can't beat this for performance. (A typically database-driven site like WordPress averages about 2.5 + seconds loading time per page.) Because the pages are all static, it means they are also extremely secure. You won't get hacked like you might with a WordPress site.
Themes	Yes	You can have different themes for different outputs. If you know CSS, theming both the web and print outputs is pretty easy.
Open source	Yes	This theme is entirely open source. Every piece of code is open, viewable, and editable. Note that this openness comes at a price — it's easy to make changes that break the theme or otherwise cause errors.
Offline viewing	Yes	This theme uses relative linking throughout, so you can view the content offline and on any web-server without configuring urls and baseurls in your configuration file.

Features not available

The following features are not available.

Features	Supported	Notes
CMS interface	No	Unlike with WordPress, you don't log into an interface and navigate to your files. You work with text files and preview the site dynamically in your browser. Don't worry – this is part of the simplicity that makes Jekyll awesome. I recommend using WebStorm as your text editor.

Features	Supported	Notes
WYSIWYG interface	No	I use WebStorm to author content, because I like working in text file formats. But you can use any Markdown editor you want (e.g., Lightpaper for Mac, Marked) to author your content.
Different outputs	No	This theme provides a single website output that contains documentation for multiple products. Unlike previous iterations of the theme, it's not intended to support different outputs from the same content. However, you can easily set things up to do this by simply creating multiple configuration files and running different builds for each configuration file.
Robust search	No	The search feature is a simplistic JSON search. For more robust search, you should integrate Swiftype or Algolia. However, those services aren't currently integrated into the theme.
Standardized templates	No	You can create pages with any structure you want. The theme does not enforce topic types such as a task or concept as the DITA specification does.
Integration with Swagger	No	You can link to a SwaggerUI output, but there is no built-in integration of SwaggerUI into this documentation theme.
Templates for endpoints	No	Although static site generators work well with API documentation, there aren't any built-in templates specific to endpoints in this theme. You could construct your own, though.
eBook output	No	There isn't an eBook output for the content.

About the theme's author

Summary: I have used this theme for projects that I've worked on as a professional technical writer.

My name is Tom Johnson, and I'm a technical writer, blogger, and podcaster based in San Jose, California. For more details, see my [technical writing blog](#) and my [course on API documentation](#). See [my blog's about page](#) for more details about me.

I have used this theme and variations of it for various documentation projects. This theme has undergone several major iterations, and now it's fairly stable and full of all the features that I need. You are welcome to use it for your documentation projects for free.

I think this theme does pretty much everything that you can do with something like OxygenXML, but without the constraints of structured authoring. Everything is completely open and changeable, so if you start tinkering around with the theme's files, you can break things. But it's completely empowering as well!

With a completely open architecture and code base, you can modify the code to make it do exactly what you want, without having to jump through all kinds of confusing or proprietary code.

If there's a feature you need but it isn't available here, let me know and I might add it. Alternatively, if you fork the theme, I would love to see your modifications and enhancements. Thanks for using Jekyll.

Support

Summary: Contact me for any support issues.

Release notes 6.0

Summary: Version 6.0 of the Documentation theme for Jekyll, released July 4, 2016, implements relative links so you can view the files offline or on any server without configuring urls and baseurls. Additionally, you can store pages in subdirectories. Templates for alerts and images are available.

Relative links

You can now view the site offline rather than solely through the Jekyll preview server or deployed on a web server. The linking approach in both the sidebar and with inline links uses relative linking throughout.

Subfolders for pages

You can create folders and subfolders for your pages, similar to how you can store posts in folders and subfolders. When Jekyll builds the site, all pages get pushed into the root directory as single html files (rather than being pushed inside folders, or remaining in subfolders). See [Pages \(page 230\)](#) for more details.

Alerts templates

You can use include templates for notes, tips, and warnings. These include templates make it easier to insert notes. If you make an error, you're immediately made aware since the site won't build. See [Alerts \(page 281\)](#) for more details.

Image templates

Similar to alerts, images also have include templates. You can insert both regular images and inline images, such as images that are a button or icon. See [Images \(page 296\)](#) for more details.

Automated links using Markdown formatting

Instead of using YAML references to handle links, I've switched to a Markdown reference style approach. A links.html file iterates through the sidebar files and formats the content in the Markdown reference. You then just use Markdown syntax for the links. See [Links \(page 303\)](#) for more details.

Workflow maps

If you want to display a workflow map for a process, you can do so by adding some properties in your frontmatter. The workflow map helps guide users through a process. Both simple and complex workflow maps are available. For more details, see [Workflow maps \(page 315\)](#).

Upgrading

If you want to upgrade from an earlier version of the theme, I recommend that you download the new theme and copy of your Markdown files into the new theme. You'll then need to make adjustments to your page frontmatter, to the sidebar table of contents, links, image references, and alert references. In short, there's no easy upgrade path. But all of this won't take too long if you don't have mountains of content.

Release notes 5.0

Summary: Version 5.0 of the Documentation theme for Jekyll changes some fundamental ways the theme works to provide product-specific sidebars, intended to accommodate a site where multiple products are grouped together on the same site rather than generated out as separate outputs.

Unique sidebars for each product

In previous versions of the theme, I built the theme to generate different outputs for different scenarios based on various filtering attributes that might include product, version, platform, and audience variants.

However, this model results in siloed outputs and lots of separate file directories to manage. Instead of having 30 separate sites for your content (or however many variants you might have been producing), in this version of the theme I've moved towards a strategy of having one site with multiple products.

For each product, you can associate a unique sidebar with each of the product's pages. This allows you to have all your documentation on one site, but with separate navigation that is tailored to a view of that product.

You can still output to both web and PDF. And if you really need multiple site outputs, you can still do so by using multiple configuration files that trigger different builds. But my conclusion after using the multiple site output model for some years is that it's a bad practice for tech comm.

Permalinks

With this theme, since you'll be publishing to one site, I've implement permalinks instead of relative links. Using permalinks means the way you store pages is much more flexible. You can store topics in folders and subfolders, etc., to any degree. But note that with permalinks you can't view the content offline (outside of Jekyll's preview server) nor on a separate site other than the one specified in the configuration file. Permalinks are how Jekyll was designed to work, and the sites just work better that way.

Kramdown and Rouge

I also switched from redcarpet and Pygments to Kramdown and Rouge to align with the current direction of Jekyll 3.0. Kramdown is a Markdown filter (it's slightly different from Github-flavored Markdown). Rouge is a syntax highlighter. Pygments had some dependencies on Python, which made it more cumbersome for Windows users.

Blog feature

I included a blog feature with this version of the theme. You can write posts and view them through the News link. There's also an archive for blog posts that sorts posts by year.

Additionally, the tagging system works across both the blog and pages, so your tags allow users to move laterally across the site based on topics they're interested in. When you view a tag archive, the sidebar shows a list of tags.

Updated documentation

I updated the documentation for the theme. The switch from the multi-site outputs to the single-site with multiple sidebars required updating a lot of different parts of the documentation and code.

Fixed errors

Previously I had some errors with the HTML that showed up in w3c HTML validator analyses. This caused some small problems in certain browsers or systems less tolerant of the errors. I fixed all of the errors.

Accessing the old theme

If you want to access the old theme, you can still find it [here](#).

About Ruby, Gems, Bundler, and other prerequisites

Summary: Ruby is a programming language you must have on your computer in order to build Jekyll locally. Ruby has various gems (or plugins) that provide various functionality. Each Jekyll project usually requires certain gems.

About Ruby

Jekyll runs on Ruby, a programming language. You have to have Ruby on your computer in order to run Ruby-based programs like Jekyll. Ruby is installed on the Mac by default, but you must add it to Windows.

About Ruby Gems

Ruby has a number of plugins referred to as “gems.” Just because you have Ruby doesn’t mean you have all the necessary Ruby gems that your program needs to run. Gems provide additional functionality for Ruby programs. There are thousands of [Rubygems](#) available for you to use.

Some gems depend on other gems for functionality. For example, the Jekyll gem might depend on 20 other gems that must also be installed.

Each gem has a version associated with it, and not all gem versions are compatible with each other.

Rubygem package managers

[Bundler](#) is a gem package manager for Ruby, which means it goes out and gets all the gems you need for your Ruby programs. If you tell Bundler you need the [jekyll gem](#), it will retrieve all the dependencies on the jekyll gem as well – automatically.

Not only does Bundler retrieve the right gem dependencies, but it’s smart enough to retrieve the right versions of each gem. For example, if you get the [github-pages](#) gem, it will retrieve all of these other gems:

```
github-pages-health-check = 1.1.0
jekyll = 3.0.3
jekyll-coffeescript = 1.0.1
jekyll-feed = 0.4.0
jekyll-gist = 1.4.0
jekyll-github-metadata = 1.9.0
jekyll-mentions = 1.1.2
jekyll-paginate = 1.1.0
jekyll-redirect-from = 0.10.0
jekyll-sass-converter = 1.3.0
jekyll-seo-tag = 1.3.2
jekyll-sitemap = 0.10.0
jekyll-textile-converter = 0.1.0
jemoji = 0.6.2
kramdown = 1.10.0
liquid = 3.0.6
mercenary ~> 0.3
rdiscout = 2.1.8
redcarpet = 3.3.3
RedCloth = 4.2.9
rouge = 1.10.1
terminal-table ~> 1.
```

See how Bundler retrieved version 3.0.3 of the jekyll gem, even though (as of this writing) the latest version of the jekyll gem is 3.1.2? That's because github-pages is only compatible up to jekyll 3.0.3. Bundler handles all of this dependency and version compatibility for you.

Trying to keep track of which gems and versions are appropriate for your project can be a nightmare. This is the problem Bundler solves. As explained on [Bundler.io](https://bundler.io):

Bundler provides a consistent environment for Ruby projects by tracking and installing the exact gems and versions that are needed.

Bundler is an exit from dependency hell, and ensures that the gems you need are present in development, staging, and production. Starting work on a project is as simple as `bundle install`.

Gemfiles

Bundler looks in a project's "Gemfile" (no file extension) to see which gems are required by the project. The Gemfile lists the source and then any gems, like this:

```
source "https://rubygems.org"

gem 'github-pages'
gem 'jekyll'
```

The source indicates the site where Bundler will retrieve the gems:

<https://rubygems.org>.

The gems it retrieves are listed separately on each line.

Here no versions are specified. Sometimes gemfiles will specify the versions like this:

```
gem 'kramdown', '1.0'
```

This means Bundler should get version 1.0 of the kramdown gem.

To specify a subset of versions, the Gemfile looks like this:

```
gem 'jekyll', '~> 2.3'
```

The  sign means greater than or equal to the *last digit before the last period in the number*.

Here it will get any gem equal to 2.3 but less than 3.0.

If it adds another digit, the scope is affected:

```
gem 'jekyll', '~>2.3.1'
```

This means to get any gem equal to 2.3.1 but less than 2.4.

If it looks like this:

```
gem 'jekyll', '~> 3.0', '>= 3.0.3'
```

This will get any Jekyll gem between versions 3.0 and up to 3.0.3.

See this [Stack Overflow post](#) for more details.

Gemfile.lock

After Bundler retrieves and installs the gems, it makes a detailed list of all the gems and versions it has installed for your project. The snapshot of all gems + versions installed is stored in your Gemfile.lock file, which might look like this:

GEM

```
remote: https://rubygems.org/
specs:
  RedCloth (4.2.9)
  activesupport (4.2.5.1)
    i18n (~> 0.7)
    json (~> 1.7, >= 1.7.7)
    minitest (~> 5.1)
    thread_safe (~> 0.3, >= 0.3.4)
    tzinfo (~> 1.1)
  addressable (2.3.8)
  coffee-script (2.4.1)
    coffee-script-source
    execjs
  coffee-script-source (1.10.0)
  colorator (0.1)
  ethon (0.8.1)
    ffi (>= 1.3.0)
  execjs (2.6.0)
  faraday (0.9.2)
    multipart-post (>= 1.2, < 3)
  ffi (1.9.10)
  gemoji (2.1.0)
  github-pages (52)
    RedCloth (= 4.2.9)
    github-pages-health-check (= 1.0.1)
    jekyll (= 3.0.3)
    jekyll-coffeescript (= 1.0.1)
    jekyll-feed (= 0.4.0)
    jekyll-gist (= 1.4.0)
    jekyll-mentions (= 1.0.1)
    jekyll-paginate (= 1.1.0)
    jekyll-redirect-from (= 0.9.1)
    jekyll-sass-converter (= 1.3.0)
    jekyll-seo-tag (= 1.3.1)
    jekyll-sitemap (= 0.10.0)
    jekyll-textile-converter (= 0.1.0)
    jemoji (= 0.5.1)
    kramdown (= 1.9.0)
    liquid (= 3.0.6)
    mercenary (~> 0.3)
    rdiscount (= 2.1.8)
    redcarpet (= 3.3.3)
    rouge (= 1.10.1)
    terminal-table (~> 1.4)
```



```
github-pages-health-check (1.0.1)
  addressable (~> 2.3)
  net-dns (~> 0.8)
  octokit (~> 4.0)
  public_suffix (~> 1.4)
  typhoeus (~> 0.7)
html-pipeline (2.3.0)
  activesupport (>= 2, < 5)
  nokogiri (>= 1.4)
i18n (0.7.0)
jekyll (3.0.3)
  colorator (~> 0.1)
  jekyll-sass-converter (~> 1.0)
  jekyll-watch (~> 1.1)
  kramdown (~> 1.3)
  liquid (~> 3.0)
  mercenary (~> 0.3.3)
  rouge (~> 1.7)
  safe_yaml (~> 1.0)
jekyll-coffeescript (1.0.1)
  coffee-script (~> 2.2)
jekyll-feed (0.4.0)
jekyll-gist (1.4.0)
  octokit (~> 4.2)
jekyll-mentions (1.0.1)
  html-pipeline (~> 2.3)
  jekyll (~> 3.0)
jekyll-paginate (1.1.0)
jekyll-redirect-from (0.9.1)
  jekyll (>= 2.0)
jekyll-sass-converter (1.3.0)
  sass (~> 3.2)
jekyll-seo-tag (1.3.1)
  jekyll (~> 3.0)
jekyll-sitemap (0.10.0)
jekyll-textile-converter (0.1.0)
  RedCloth (~> 4.0)
jekyll-watch (1.3.1)
  listen (~> 3.0)
jemoji (0.5.1)
  gemoji (~> 2.0)
  html-pipeline (~> 2.2)
  jekyll (>= 2.0)
json (1.8.3)
kramdown (1.9.0)
```

```
liquid (3.0.6)
listen (3.0.6)
  rb-fsevent (>= 0.9.3)
  rb-inotify (>= 0.9.7)
mercenary (0.3.5)
mini_portile2 (2.0.0)
minitest (5.8.4)
multipart-post (2.0.0)
net-dns (0.8.0)
nokogiri (1.6.7.2)
  mini_portile2 (~> 2.0.0.rc2)
octokit (4.2.0)
  sawyer (~> 0.6.0, >= 0.5.3)
public_suffix (1.5.3)
rb-fsevent (0.9.7)
rb-inotify (0.9.7)
  ffi (>= 0.5.0)
rdiscount (2.1.8)
redcarpet (3.3.3)
rouge (1.10.1)
safe_yaml (1.0.4)
sass (3.4.21)
sawyer (0.6.0)
  addressable (~> 2.3.5)
  faraday (~> 0.8, < 0.10)
terminal-table (1.5.2)
thread_safe (0.3.5)
typhoeus (0.8.0)
  ethon (>= 0.8.0)
tzinfo (1.2.2)
  thread_safe (~> 0.1)
```

PLATFORMS

ruby

DEPENDENCIES

github-pages
jekyll

BUNDLED WITH

1.11.2

You can always delete the Gemlock file and run Bundle install again to get the latest versions. You can also run `bundle update`, which will ignore the Gemlock file to get the latest versions of each gem.

To learn more about Bundler, see [Bundler's Purpose and Rationale](#) .

Install Jekyll on Mac

Summary: Installation of Jekyll on Mac is usually less problematic than on Windows. However, you may run into permissions issues with Ruby that you must overcome. You should also use Bundler to be sure that you have all the required gems and other utilities on your computer to make the project run.

Ruby and RubyGems

Ruby and [RubyGems](#) are usually installed by default on Macs. Open your Terminal and type `which ruby` and `which gem` to confirm that you have Ruby and Rubygems. You should get a response indicating the location of Ruby and Rubygems.

If you get responses that look like this:

```
/usr/local/bin/ruby
```

and

```
/usr/local/bin/gem
```

Great! Skip down to the [Bundler \(page 224\)](#) section.

However, if your location is something like `/Users/MacBookPro/.rvm/rubies/ruby-2.2.1/bin/gem`, which points to your system location of Rubygems, you will likely run into permissions errors when trying to get a gem. A sample permissions error (triggered when you try to install the jekyll gem such as `gem install jekyll`) might look like this for Rubygems:

```
>ERROR: While executing gem ... (Gem::FilePermissionError)
  You don't have write permissions for the /Library/Ruby/Gems/
  2.0.0 directory.
```

Instead of changing the write permissions on your operating system's version of Ruby and Rubygems (which could pose security issues), you can install another instance of Ruby (one that is writable) to get around this.

Install Homebrew

Homebrew is a package manager for the Mac, and you can use it to install an alternative instance of Ruby code. To install Homebrew, run this command:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

If you already had Homebrew installed on your computer, be sure to update it:

```
brew update
```

Install Ruby through Homebrew

Now use Homebrew to install Ruby:

```
brew install ruby
```

Log out of terminal, and then then log back in.

When you type `which ruby` and `which gem`, you should get responses like this:

```
/usr/local/bin/ruby
```

And this:

```
/usr/local/bin/gem
```

Now Ruby and Rubygems are installed under your username, so these directories are writeable.

Note that if you don't see these paths, try restarting your computer or try installing rbenv, which is a Ruby version management tool. If you still have issues getting a writeable version of Ruby, you need to resolve them before installing Bundler.

Install the Jekyll gem

At this point you should have a writeable version of Ruby and Rubygem on your machine.

Now use `gem` to install Jekyll:

```
gem install jekyll
```

You can now use Jekyll to create new Jekyll sites following the quick-start instructions on Jekyllrb.com.

Installing dependencies through Bundler

Some Jekyll themes will require certain Ruby gem dependencies. These dependencies are stored in something called a Gemfile, which is packaged with the Jekyll theme. You can install these dependencies through Bundler. (Although you don't need to install Bundler for this Documentation theme, it's a good idea to do so.)

[Bundler](#) is a package manager for RubyGems. You can use it to get all the gems (or Ruby plugins) that you need for your Jekyll project.

You install Bundler by using the gem command with RubyGems:

```
gem install bundler
```

If you're prompted to switch to superuser mode (`sudo`) to get the correct permissions to install Bundler in that directory, avoid doing this. All other applications that need to use Bundler will likely not have the needed permissions to run.

Bundler goes out and retrieves all the gems that are specified in a Jekyll project's Gemfile. If you have a gem that depends on other gems to work, Bundler will go out and retrieve all of the dependencies as well. (To learn more about Bundler, see [About Ruby Gems \(page 214\)](#)).

The vanilla Jekyll site you create through `jekyll new my-awesome-site` doesn't have a Gemfile, but many other themes (including the Documentation theme for Jekyll) do have a Gemfile.

Serve the Jekyll Documentation theme

1. Browse to the directory where you downloaded the Documentation theme for Jekyll.
2. Type `jekyll serve`
3. Go to the preview address in the browser. (Make sure you include the `/` at the end.)

Resolve “No Github API authentication” errors

After making an edit, Jekyll auto-rebuilds the site. If you have the Gemfile in the theme with the github-pages gem, you may see the following error:

```
GitHub Metadata: No GitHub API authentication could be found. Some fields may be missing or have incorrect data.
```

If you see this error, you will need to take some additional steps to resolve it. (Note that this error only appears if you have the github-pages gem in your gemfile.) The resolution involves adding a Github token and a cert file.

Note: These instructions apply to Mac OS X, but they're highly similar to Windows. These instructions are adapted from a post on [Knight Codes](#). If you're on Windows, see the Knight Codes post for details instead of following along below.

To resolve the “No Github API authentication” error:

1. Follow Github's instructions to [create a personal access token](#).
2. Open the `.bash_profile` file in your user directory:

```
open ~/.bash_profile
```

The file will open in your default terminal editor. If you don't have a `.bash_profile` file, you can just create a file with this name. Note that files that begin with `.` are hidden, so if you're looking in your user directory for the file, use `ls -a` to see hidden files.

3. In your `.bash_profile` file, reference your token as a system variable like this:

```
export JEKYLL_GITHUB_TOKEN=abc123abc123abc123abc123abc123abc123abc123abc123
```

Replace `abc123...` with your own token that you generated in step 1.

4. Go to [\[https://curl.haxx.se/ca/cacert.pem\]](https://curl.haxx.se/ca/cacert.pem)[\[https://curl.haxx.se/ca/cacert.pem\]](https://curl.haxx.se/ca/cacert.pem). **Right-click the page, select `**Save as`**, and save the file on your computer (save it somewhere safe, where you won't delete it). Name the file `cacert`.
5. Open your `.bash_profile` file again and add this line, replacing `Users/johndoe/projects/` with the path to your `cacert.pem` file:

```
export SSL_CERT_FILE=/Users/johndoe/projects/cacert.pem
```

6. Close and restart your terminal.

Browse to your jekyll project and run `bundle exec jekyll serve`. Make an edit to a file and observe that no Github API errors appear when Jekyll rebuilds the project.

Install Jekyll on Windows

✓ **Tip:** For a better terminal emulator on Windows, use [Git Bash](#) . Git Bash gives you Linux-like control on Windows.

Install Ruby and Ruby Development Kit

First you must install Ruby because Jekyll is a Ruby-based program and needs Ruby to run.

1. Go to [RubyInstaller for Windows](#) .
2. Under **RubyInstallers**, download and install one of the Ruby installers under the **WITH DEVKIT** list (usually the recommended/highlighted option).
3. Double-click the downloaded file and proceed through the wizard to install it. Run the `ridk install` step on the last stage of the installation wizard.
4. Open a new command prompt window or Git Bash session.

Install the Jekyll gem

At this point you should have Ruby and Rubygem on your machine.

Now use `gem` to install Jekyll:

```
gem install jekyll
```

You can now use Jekyll to create new Jekyll sites following the quick-start instructions on [Jekyllrb.com](#) .

Installing dependencies through Bundler

Some Jekyll themes will require certain Ruby gem dependencies. These dependencies are stored in something called a Gemfile, which is packaged with the Jekyll theme. You can install these dependencies through Bundler. (Although you don't need to install Bundler for this Documentation theme, it's a good idea to do so.)

[Bundler](#) is a package manager for RubyGems. You can use it to get all the gems (or Ruby plugins) that you need for your Jekyll project.

You install Bundler by using the gem command with RubyGems:

Install Bundler

1. Browse to the directory where you downloaded the Documentation theme for Jekyll.
2. Delete or rename the existing `Gemfile` and `Gemfile.lock` files.
3. Install Bundler: `gem install bundler`
4. Initialize Bundler: `bundle init`

This will create a new Gemfile.

5. Open the Gemfile in a text editor.

Typically you can open files from the Command Prompt by just typing the filename, but because Gemfile doesn't have a file extension, no program will automatically open it. You may need to use your File Explorer and browse to the directory, and then open the Gemfile in a text editor such as Notepad.

6. Remove the existing contents. Then paste in the following:

```
source "https://rubygems.org"

gem 'wdm'
gem 'jekyll'
```

The [wdm gem](#) allows for the polling of the directory and rebuilding of the Jekyll site when you make changes. This gem is needed for Windows users, not Mac users.

7. Save and close the file.
8. Type `bundle install`.

Bundle retrieves all the needed gems and gem dependencies and downloads them to your computer. At this time, Bundle also takes a snapshot of all the gems used in your project and creates a Gemfile.lock file to store this information.

Git Clients for Windows

Although you can use the default command prompt with Windows, it's recommended that you use [Git Bash](#) instead. The Git Bash client will allow you to run shell scripts and execute other Unix commands.

Serve the Jekyll Documentation theme

1. Browse to the directory where you downloaded the Documentation theme for Jekyll.
2. Type `jekyll serve`
3. Go to the preview address in the browser. (Make sure you include the `/` at the end.)

Unfortunately, the Command Prompt doesn't allow you to easily copy and paste the URL, so you'll have to type it manually.

Resolving Github Metadata errors

After making an edit, Jekyll auto-rebuilds the site. If you have the Gemfile in the theme with the github-pages gem, you may see the following error:

```
GitHub Metadata: No GitHub API authentication could be found. Some fields may be missing or have incorrect data.
```

If so, you will need to take some additional steps to resolve it. (Note that this error only appears if you have the github-pages gem in your gemfile.) The resolution involves adding a Github token and a cert file.

See this post on [Knight Codes](#) for instructions on how to fix the error. You basically generate a personal token on Github and set it as a system variable. You also download a certification file and set it as a system variable. This resolves the issue.

Pages

Summary: This theme primarily uses pages. You need to make sure your pages have the appropriate frontmatter. One frontmatter tag your users might find helpful is the summary tag. This functions similar in purpose to the shortdesc element in DITA.

Where to author content

Use a text editor such as Sublime Text, WebStorm, IntelliJ, or Atom to create pages. Atom is recommended because it's created by Github, which is driving some of the Jekyll development through Github Pages.

Where to save pages

You can store your pages in any folder structures you want, with any level of folder nesting. The site output will pull all of those pages out of their folders and put them into the root directory. Check out the `_site` folder, which is where Jekyll is generated, to see the difference between your project's structure and the resulting site output.

The listing of all pages in the root directory (which happens when you add a permalink property to the pages) is what allows the relative linking and offline viewing of the site to work.

Frontmatter

Make sure each page has frontmatter at the top like this:

```
---
title: Alerts
tags: [formatting]
keywords: notes, tips, cautions, warnings, admonitions
last_updated: July 3, 2016
summary: "You can insert notes, tips, warnings, and important alerts in your content."
sidebar: mydoc_sidebar
permalink: mydoc_alerts.html
---
```

Frontmatter is always formatted with three hyphens at the top and bottom. Your frontmatter must have a `title` and `permalink` value. All the other values are optional.

Note that you cannot use variables in frontmatter.

The following table describes each of the frontmatter that you can use with this theme:

Frontmatter	Required?	Description
title	Required	The title for the page
tags	Optional	Tags for the page. Make all tags single words, with underscores if needed (rather than spaces). Separate them with commas. Enclose the whole list within brackets. Also, note that tags must be added to <code>_data/tags_doc.yml</code> to be allowed entrance into the page. This prevents tags from becoming somewhat random and unstructured. You must create a tag page for each one of your tags following the pattern shown in the tags folder. (Tag pages aren't automatically created.)
keywords	Optional	Synonyms and other keywords for the page. This information gets stuffed into the page's metadata to increase SEO. The user won't see the keywords, but if you search for one of the keywords, it will be picked up by the search engine.
last_updated	Optional	The date the page was last updated. This information could helpful for readers trying to evaluate how current and authoritative information is. If included, the <code>last_updated</code> date appears in the footer of the page in small font.
sidebar	Required	Refers to the sidebar data file for this page. Don't include the <code>".yml"</code> file extension for the sidebar — just provide the file name. If no sidebar is specified, this value will inherit the <code>default</code> property set in your <code>_config.yml</code> file for the page's frontmatter.

Frontmatter	Required?	Description
summary	Optional	A 1-2 word sentence summarizing the content on the page. This gets formatted into the summary section in the page layout. Adding summaries is a key way to make your content more scannable by users (check out Jakob Nielsen's site for a great example of page summaries.) The only drawback with summaries is that you can't use variables in them.
permalink	Required	The permalink <i>must</i> match the filename in order for automated links to work. Additionally, you must include the ".html" in the filename. Do not put forward slashes around the permalink (this makes Jekyll put the file inside a folder in the output). When Jekyll builds the site, it will put the page into the root directory rather than leaving it in a subdirectory or putting it inside a folder and naming the file index.html. Having all files flattened in the root directory is essential for relative linking to work and for all paths to JS and CSS files to be valid.
datatable	Optional	'true'. If you add <code>datatable: true</code> in the frontmatter, scripts for the jQuery Datatables plugin get included on the page. You can see the scripts that conditionally appear by looking in the <code>_layouts/default.html</code> page.
toc	Optional	If you specify <code>toc: false</code> in the frontmatter, the page won't have the table of contents that appears below the title. The toc refers to the list of jump links below the page title, not the sidebar navigation. You probably want to hide the TOC on the homepage and product landing pages.

Colons in page titles

If you want to use a colon in your page title, you must enclose the title's value in quotation marks.

Page names and excluding files from outputs

By default, everything in your project is included in the output. You can exclude all files that don't belong to that project by specifying the file name, the folder name, or by using wildcards in your configuration file:

```
exclude:  
  
- filename.md  
- subfolder_name/  
- mydoc_*  
- gitignore
```

Wildcards will exclude every match after the `*`.

Saving pages as drafts

If you add `published: false` in the frontmatter, your page won't be published. You can also move draft pages into the `_drafts` folder to exclude them from the build. With posts, you can also keep them as drafts by omitting the date in the title.

Markdown or HTML format

Pages can be either Markdown or HTML format (specified through either an `.md` or `.html` file extension).

If you use Markdown, you can also include HTML formatting where needed. But if your format is HTML, you must add a `markdown="1"` attribute to the element in order to use Markdown inside that HTML element:

```
<div markdown="1">This is a [link](http://exmaple.com).</div>
```

For your Markdown files, note that a space or two indent will set text off as code or blocks, so avoid spacing indents unless intentional.

If you have a lot of HTML, as long as the top and bottom tags of the HTML are flush left in a Markdown file, all the tags inside those bookend HTML tags will render as HTML, regardless of their indentation. (This can be especially useful for tables.)

Page names

I recommend prefixing your page names with the product, such as “mydoc_pages” instead of just “pages.” This way if you have other products that also have topics with generic names such as “pages,” there won’t be naming conflicts.

Additionally, consider adding the product name in parentheses after the title, such as “Pages (Mydoc)” so that users can clearly navigate different topics for each product.

Kramdown Markdown

Kramdown is the Markdown flavor used in the theme. This mostly aligns with Github-flavored Markdown, but with some differences in the indentation allowed within lists. Basically, Kramdown requires you to line up the indent between list items with the first starting character after the space in your list item numbering. See this [blog post on Kramdown and Rouge](#) for more details.

You can use standard Multimarkdown syntax for tables. You can also use fenced code blocks with lexers specifying the type of code. The configuration file shows the Markdown processor and extension:

```
highlighter: rouge
markdown: kramdown
kramdown:
  input: GFM
  auto_ids: true
  hard_wrap: false
  syntax_highlighter: rouge
```

Automatic mini-TOCs

By default, a TOC appears at the top of your pages and posts. If you don’t want the TOC to appear for a specific page, such as for a landing page or other homepage, add `toc: false` in the frontmatter of the page.

The mini-TOC requires you to use the `##` Markdown syntax for headings. If you use `<h2>` elements, you must add an ID attribute for the heading element in order for it to appear in the mini-TOC (for example, `<h2 id="mysampleid">Heading</h2>`).

Headings

Use pound signs before the heading title to designate the level. Note that kramdown requires headings to have one space before and after the heading. Without this space above and below, the heading won't render into HTML.

```
## Second-level heading
```

Result:

Second-level heading

```
### Third-level heading
```

Result:

Third-level heading

```
#### Fourth-level heading
```

Result:

Fourth-level heading

Headings with ID Tags

If you want to use a specific ID tag with your heading, add it like this:

```
## Headings with ID Tags {#someIdTag}
```

Then you can reference it with a link like this on the same page:

```
[Some link](#someIdTag)
```

Result:

[Some link \(page 235\)](#)

For details about linking to headings on different pages, see [Automated links to headings on pages \(page 0\)](#).

Specify a particular page layout

The configuration file sets the default layout for pages as the “page” layout.

You can create other layouts inside the layouts folder. If you create a new layout, you can specify that your page use your new layout by adding `layout: mylayout.html` in the page’s frontmatter. Whatever layout you specify in the frontmatter of a page will override the layout default set in the configuration file.

Comments

Disqus, a commenting system, is integrated into the theme. In the configuration file, specify the Disqus code for the universal code, and Disqus will appear. If you don’t add a Disqus value, the Disqus form isn’t included.

Posts

Summary: You can use posts when you want to create blogs or news type of content.

About posts

Posts are typically used for blogs or other news information because they contain a date and are sorted in reverse chronological order.

You create a post by adding a file in the `_posts` folder that is named `yyyy-mm-dddd-permalink.md`, which might be `2016-02-25-my-latest-updates.md`. You can use any number of subfolders here that you want.

Posts use the `post.html` layout in the `_layouts` folder when you are viewing the post.

The `news.html` file in the root directory shows a reverse chronological listing of the 10 latest posts

Allowed frontmatter

The frontmatter you can use with posts is as follows:

```
---
title: My sample post
tags: content_types
keywords: pages, authoring, exclusion, frontmatter
sidebar: mydoc_sidebar
permalink: mydoc_pages.html
summary: "This is some summary frontmatter for my sample post."
---
```

Frontmatter	Required?	Description
title	Required	The title for the page

Frontmatter	Required?	Description
tags	Optional	Tags for the page. Make all tags single words, with underscores if needed. Separate them with commas. Enclose the whole list within brackets. Also, note that tags must be added to <code>_data/tags_doc.yml</code> to be allowed entrance into the page. This prevents tags from becoming somewhat random and unstructured. You must create a tag page for each one of your tags following the sample pattern in the tabs folder. (Tag pages aren't automatically created.)
keywords	Optional	Synonyms and other keywords for the page. This information gets stuffed into the page's metadata to increase SEO. The user won't see the keywords, but if you search for one of the keywords, it will be picked up by the search engine.
sidebar	Required	Refers to the sidebar data file for this page. Don't include the ".yml" file extension for the sidebar — just provide the file name. If no sidebar is specified, this value will inherit the <code>default</code> property set in your <code>_config.yml</code> file for the page's frontmatter.
permalink	Required	This theme uses permalinks to facilitate the linking. You specify the permalink want for the page, and the <code>_site</code> output will put the page into the root directory when you publish. Follow the same convention here as you do with page permalinks — list the file name followed by the <code>.html</code> extension.
summary	Optional	A 1-2 word sentence summarizing the content on the page. This gets formatted into the summary section in the page layout. Adding summaries is a key way to make your content more scannable by users (check out Jakob Nielsen's site for a great example of page summaries.) The only drawback with summaries is that you can't use variables in them.

Lists

Summary: This page shows how to create both bulleted and numbered lists

Bulleted Lists

This is a bulleted list:

```
* first item  
* second item  
* third item
```

Result:

- first item
- second item
- third item

Numbered list

This is a simple numbered list:

```
1. First item.  
1. Second item.  
1. Third item.
```

Result:

1. First item.
2. Second item.
3. Third item.

You can use whatever numbers you want — when the Markdown filter processes the content, it will assign the correct numbers to the list items.

Complex Lists

Here's a more complex list:

1. Sample first item.
 - * sub-bullet one
 - * sub-bullet two
2. Continuing the list
 1. sub-list numbered one
 2. sub-list numbered two
3. Another list item.

Result:

1. Sample first item.
 - sub-bullet one
 - sub-bullet two
2. Continuing the list
 - a. sub-list numbered one
 - b. sub-list numbered two
3. Another list item.

Another Complex List

Here's a list with some intercepting text:

1. Sample first item.

This is a result statement that talks about something....

2. Continuing the list

```
<div markdown="span" class="alert alert-info" role="aler
t"><i class="fa fa-info-circle"></i> <b>Note:</b> Remember to d
o this. If you have "quotes", you must escape them.</div>
```

Here's a list in here:

- * first item
- * second item

3. Another list item.

```
```js
function alert("hello");
```
```

4. Another item.

Result:

1. Sample first item.

This is a result statement that talks about something....

2. Continuing the list

Note: Remember to do this. If you have “quotes”, you must escape them.

Here's a list in here:

- first item
- second item

3. Another list item.

```
function alert("hello");
```

4. Another item.

Key Principle to Remember with Lists

The key principle is to line up the first character after the dot following the number:

```
61
62 Here's a list with some intercepting text:
63
64 1. Sample first item.
65
66   This is a result statement that talks about something...
67
68 2. Continuing the list
69
70   {% include note.html content="Remember to do this. If you have \"quotes\", you must escape
71
72   Here's a list in here:
73
74   * first item
75   * second item
76
77 3. Another list item.
78
79   ```js
80   function alert("hello");
81   ```
82
83 4. Another item.
84
85 The key principle is to line up |
86
87 ## Links
88
```

Lining up the left edge ensures the list stays in tact.

For the sake of simplicity, use two spaces after the dot for numbers 1 through 9. Use one space for numbers 10 and up. If any part of your list doesn't align symmetrically on this left edge, the list will not render correctly. Also note that this is characteristic of kramdown-flavored Markdown and may not yield the same results in other Markdown flavors.

Conditional logic

Summary: You can implement advanced conditional logic that includes if statements, or statements, unless, and more. This conditional logic facilitates single sourcing scenarios in which you're outputting the same content for different audiences.

About Liquid and conditional statements

If you want to create different outputs for different audiences, you can do all of this using a combination of Jekyll's Liquid markup and values in your configuration file. This is how I previously configured the theme. I had different configuration files for each output. Each configuration file specified different values for product, audience, version, and so on. Then I had different build processes that would leverage the different configuration files. It seemed like a perfect implementation of DITA-like techniques with Jekyll.

But I soon found that having lots of separate outputs for a project was undesirable. If you have 10 different outputs that have different nuances for different audiences, it's hard to manage and maintain. In this latest version of the theme, I consolidated all information into the same output to explicitly do away with the multi-output approach.

As such, the conditional logic won't have as much play as it previously did. Instead of conditions, you'll probably want to incorporate [navtabs \(page 0\)](#) to split up the information.

However, you can still of course use conditional logic as needed.

✓ **Tip:** Definitely check out [Liquid's documentation](#) for more details about how to use operators and other liquid markup. The notes here are a small, somewhat superficial sample from the site.

Where to store filtering values

You can filter content based on values that you have set either in your page's frontmatter, a config file, or in a file in your `_data` folder. If you set the attribute in your config file, you need to restart the Jekyll server to see the changes. If you set the value in a file in your `_data` folder or page frontmatter, you don't need to restart the server when you make changes.

Conditional logic based on config file value

Here's an example of conditional logic based on a value in the page's frontmatter. Suppose you have the following in your frontmatter:

```
platform: mac
```

On a page in my site (it can be HTML or markdown), you can conditionalize content using the following:

```
{% if page.platform == "mac" %}  
Here's some info about the Mac.  
{% elsif page.platform == "windows" %}  
Here's some info about Windows ...  
{% endif %}
```

This uses simple `if-elsif` logic to determine what is shown (note the spelling of `elsif`). The `else` statement handles all other conditions not handled by the `if` statements.

Here's an example of `if-else` logic inside a list:

```
To bake a casserole:  
  
1. Gather the ingredients.  
{% if page.audience == "writer" %}  
2. Add in a pound of meat.  
{% elsif page.audience == "designer" %}  
3. Add in an extra can of beans.  
{% endif %}  
3. Bake in oven for 45 min.
```

You don't need the `elsif` or `else`. You could just use an `if` (but be sure to close it with `endif`).

Or operator

You can use more advanced Liquid markup for conditional logic, such as an `or` command. See [Shopify's Liquid documentation](#) for more details.

For example, here's an example using `or`:

```
{% if page.audience contains "vegan" or page.audience == "vegetarian" %}  
  Then run this...  
{% endif %}
```

Note that you have to specify the full condition each time. You can't shorten the above logic to the following:

```
{% if page.audience contains "vegan" or "vegetarian" %}  
  // run this.  
{% endif %}
```

This won't work.

Unless operator

You can also use `unless` in your logic, like this:

```
{% unless site.output == "pdf" %}  
...do this  
{% endunless %}
```

When figuring out this logic, read it like this: “Run the code here *unless* this condition is satisfied.”.

Don't read it the other way around or you'll get confused. (It's *not* executing the code only if the condition is satisfied.)

Storing conditions in the `_data` folder

Here's an example of using conditional logic based on a value in a data file:

```
{% if site.data.options.output == "alpha" %}  
show this content...  
{% elsif site.data.options.output == "beta" %}  
show this content...  
{% else %}  
this shows if neither of the above two if conditions are met.  
{% endif %}
```

To use this, I would need to have a `_data` folder called `options` where the `output` property is stored.

Specifying the location for `_data`

You can also specify a `data_source` for your data location in your configuration file. Then you aren't limited to simply using `_data` to store your data files.

For example, suppose you have 2 projects: `alpha` and `beta`. You might store all the data files for `alpha` inside `data_alpha`, and all the data files for `beta` inside `data_beta`.

In your `alpha` configuration file, specify the data source like this:

```
data_source: data_alpha
```

Then create a folder called `_data_alpha`.

For your `beta` configuration file, specify the data source like this:

```
data_source: data_beta
```

Then create a folder called `_data_beta`.

Conditions versus includes

If you have a lot of conditions in your text, it can get confusing. As a best practice, whenever you insert an `if` condition, add the `endif` at the same time. This will reduce the chances of forgetting to close the if statement. Jekyll won't build if there are problems with the liquid logic.

If your text is getting busy with a lot of conditional statements, consider putting a lot of content into includes so that you can more easily see where the conditions begin and end.

Content reuse

Summary: You can reuse chunks of content by storing these files in the includes folder. You then choose to include the file where you need it. This works similar to conref in DITA, except that you can include the file in any content type.

About content reuse

You can embed content from one file inside another using includes. Put the file containing content you want to reuse (e.g., mypage.html) inside the `_includes/` custom folder and then use a tag like this:

```
{% include custom/mypage.html %}
```

With content in your `_includes` folder, you don't add any frontmatter to these pages because they will be included on other pages already containing frontmatter.

Also, when you include a file, all of the file's contents get included. You can't specify that you only want a specific part of the file included. However, you can use parameters with includes.

Page-level variables

You can also create custom variables in your frontmatter like this:

```
---  
title: Page-level variables  
permalink: page_level_variables/  
thing1: Joe  
thing2: Dave  
---
```

You can then access the values in those custom variables using the `page` namespace, like this:

```
thing1: {{page.thing1}}  
thing2: {{page.thing2}}
```

I use includes all the time. Most of the includes in the `_includes` directory are pulled into the theme layouts. For those includes that change, I put them inside `custom` and then inside a specific project folder.

Collections

Summary: Collections are useful if you want to loop through a special folder of pages that you make available in a content API. You could also use collections if you have a set of articles that you want to treat differently from the other content, with a different layout or format.

What are collections

Collections are custom content types different from pages and posts. You might create a collection if you want to treat a specific set of articles in a unique way, such as with a custom layout or listing. For more detail on collections, see [Ben Balter's explanation of collections here](#).

Create a collection

To create a collection, add the following in your configuration file:

```
collections:
  tooltips:
    output: true
```

In this example, “tooltips” is the name of the collection.

Interacting with collections

You can interact with collections by using the `site.collectionname` namespace, where `collectionname` is what you've configured. In this case, if I wanted to loop through all tooltips, I would use `site.tooltips` instead of `site.pages` or `site.posts`.

See [Collections in the Jekyll documentation](#) for more information.

How to use collections

I haven't found a huge use for collections in normal documentation. However, I did find a use for collections in generating a tooltip file that would be used for delivering tooltips to a user interface from text files in the documentation. See [Help APIs and UI tooltips \(page 339\)](#) for details.

Video tutorial on collections

See this [video tutorial on Jekyll.tips](#) for more details on collections.

WebStorm Text Editor

Summary: You can use a variety of text editors when working with a Jekyll project. WebStorm from IntelliJ offers a lot of project-specific features, such as find and replace, that make it ideal for working with tech comm projects.

About text editors and WebStorm

There are a variety of text editors available, but I like WebStorm the best because it groups files into projects, which makes it easy to find all instances of a text string, to do find and replace operations across the project, and more.

If you decide to use WebStorm, here are a few tips on configuring the editor.

Remove unnecessary plugins

By default, WebStorm comes packaged with a lot more functionality than you probably need. You can lighten the editor by removing some of the plugins. Go to **WebStorm > Preferences > Plugins** and clear the check boxes of plugins you don't need.

Set default tab indent to 3 spaces instead of 4

You can set the way the tab works, and whether it uses spaces or a tab character. For details, see [Code Style. JavaScript](#) in WebStorm's help.

On a Mac, go to **WebStorm > Preferences > Editor > Code Style > Other File Types**. Don't select the "Use tab character" check box. Set **4** for the **Tab size** and **Indent** check boxes.

On Windows, go to **File > Settings > Editor > Code Style > Other File Types** to access the same menu.

Add the Markdown Support plugin

Since you'll be writing in Markdown, having color coding and other support for Markdown is important. Install the Markdown Support plugin by going to **WebStorm > Preferences > Plugins** and clicking **Install JetBrains Plugin**. Search for **Markdown Support**. You can also implement the Markdown Navigator plugin.

Enable Soft Wraps (word wrapping)

Most likely you'll want to enable soft wraps, which wraps lines rather than extending them out forever and requiring you to scroll horizontally to see the text. To enable softwrapping, go to **WebStorm > Preferences > Editor > General** and see the Soft Wraps section. Select the **Use soft wraps in editor** check box.

Exclude a directory

When you're searching for content, you don't want to edit any file that appears in the `_site` directory. You can exclude a directory from Webstorm by right-clicking the directory and choosing **Mark Directory As** and then selecting **Excluded**.

Set tabs to 4 spaces

You can set the default number of spaces a tab sets, including whether Webstorm uses a tab character or spaces. You want spaces, and you want to set this to default number of spaces to `4`. Note that this is due to the way Kramdown handles the continuation of lists.

To set the indentation, see the "Tabs and Indents" topic in this [Code Style Javascript](#) topic in Webstorm's help.

Shortcuts

It can help to learn a few key shortcuts:

| Command | Shortcuts |
|---------------------|---|
| Shift + Shift | Allows you to find a file by searching for its name. |
| Shift + Command + F | Find in whole project. (WebStorm uses the term "Find in path".) |
| Shift + Command + R | Replace in whole project. (Again, WebStorm calls it "Replace in path".) |
| Command + F | Find on page |
| Shift + R | Replace on page |

| Command | Shortcuts |
|-----------------------------------|--|
| Right-click >
Add to Favorites | Allows you to add files to a Favorites section, which expands below the list of files in the project pane. |
| Shift + tab | Applies outdenting (opposite of tabbing) |
| Shift + Function
+ F6 | Rename a file |
| Command +
Delete | Delete a file |
| Command + 2 | Show Favorites pane |
| Shift + Option +
F | Add to Favorites |

✔ **Tip:** If these shortcut keys aren't working for you, make sure you have the "Max OS X 10.5+" keymap selected. Go to **WebStorm > Preferences > Keymap** and select it there.

Finding files

When I want to find a file, I browse to the file in the preview site and copy the page name in the URL. Then in Webstorm I press **Shift** twice and paste in the file name. The search feature automatically highlights the file I want, and I press **Enter**.

Identifying changed files

When you have the Git and Github integration, changed files appear in blue. This lets you know what needs to be committed to your repository.

Creating file templates

Rather than insert the frontmatter by hand each time, it's much faster to simply create a Jekyll template. To create a Jekyll template in WebStorm:

1. Right-click a file in the list of project files, and select **New > Edit File Templates**.

If you don't see the Edit File Templates option, you may need to create a file template first. Go to **File > Default Settings > Editor > File and Code Templates**. Create a new file template with an md extension, and then close and restart WebStorm. Then repeat this step and you will see the File Templates option appear in the right context menu.

2. In the upper-left corner of the dialog box that appears, click the + button to create a new template.
3. Name it something like Jekyll page. Insert the frontmatter you want, and save it.

To use the Jekyll template, when you create a new file in your WebStorm project, you can select your Jekyll file template.

Disable pair quotes

By default, each time you type , WebStorm will pair the quote (creating two quotes). You can disable this by going to **WebStorm > Preferences > Editor > Smartkeys**. Clear the **Insert pair quotes** check box.

Atom Text Editor

Summary: Atom is a free text editor that is a favorite tool of many writers because it is free. This page provides some tips for using Atom.

If you haven't downloaded [Atom](#), download and install it. Use this as your editor when working with Jekyll. The syntax highlighting is probably the best among the available editors, as it was designed with Jekyll-authoring in mind. However, if you prefer Sublime Text, WebStorm, or some other editor, you can also use that.

Customize the invisibles and tab spacing in Atom:

1. Go to **Atom > Preferences**.
2. On the **Settings** tab, keep the default options but also select the following:
 - **Show Invisibles**
 - **Soft Wrap**
 - For the **Tab Length**, type **4**.
 - For the **Tab Type**, select **soft**.

Turn off auto-complete:

1. Go to **Atom > Preferences**.
2. Click the **Packages** tab.
3. Search for **autocomplete-plus**.
4. Disable the autocomplete package.

Atom Shortcuts

- **Cmd + T**: Find file
- **Cmd + Shift + F**: Find across project
- **Cmd + Alt + S**: Save all

(For Windows, replace “Cmd” with “Ctrl”.)

Sidebar Navigation

Summary: The sidebar navigation uses a jQuery component called Navgoco. The sidebar is a somewhat complex part of the theme that remembers your current page, highlights the active item, stays in a fixed position on the page, and more. This page explains a bit about how the sidebar was put together.

Navgoco foundation

The sidebar uses the [Navgoco jQuery plugin](#) as its basis. Why not use Bootstrap? Navgoco provides a few features that I couldn't find in Bootstrap:

- Navgoco sets a cookie to remember the user's position in the sidebar. If you refresh the page, the cookie allows the plugin to remember the state.
- Navgoco inserts an `active` class based on the navigation option that's open. This is essential for keeping the accordion open.
- Navgoco includes the expand and collapse features of a sidebar.

In short, the sidebar has some complex logic here. I've integrated Navgoco's features with the sidebar.html and sidebar data files to build the sidebar. It's probably the most impressive part of this theme. (Other themes usually aren't focused on creating hierarchies of pages, but this kind of hierarchy is important in a documentation site.)

Accordion sidebar feature

The sidebar.html file (inside the `_includes` folder) contains the `.navgoco` method called on the `#mysidebar` element.

There are some options to set within the `.navgoco` method. The only noteworthy option is `accordion`. This option makes it so when you expand a section, the other sections collapse. It's a way of keeping your navigation controls condensed.

The value for `accordion` is a Boolean (`true` or `false`). By default, the `accordion` option is set as `true`. If you don't want the accordion, set it to `false`. Note that there's also a block of code near the bottom of sidebar.html that is commented out. Uncomment out that section to have the Collapse all and Expand All buttons appear.

There's a danger with setting the accordion to `false`. If you click Expand All and the sidebar expands beyond the dimensions of the browser, users will be stuck. When that happens, it's hard to collapse it. As a best practice, leave the sidebar's accordion option set to `true`.

Fixed position sidebar

The sidebar has one other feature — this one from Bootstrap. If the user's viewport is tall enough, the sidebar remains fixed on the page. This allows the user to scroll down the page and still keep the sidebar in view.

In the `customscripts.js` file in the `js` folder, there's a function that adds an `affix` class if the height of the browser window is greater than 800 pixels. If the browser's height is less than 800 pixels, the `nav affix` class does not get inserted. As a result, the sidebar can slide up and down as the user scrolls up and down the page.

Depending on your content, you may need to adjust `800` pixel number. If your sidebar is so long that having it in a fixed position makes it so the bottom of the sidebar gets cut off, increase the `800` pixel number here to a higher number.

Opening sidebar links into external pages

In the attributes for each sidebar item, if you use `external_url` instead of `url`, the theme will insert the link into an `a href` element that opens in a blank target.

For example, the `sidebar.html` file contains the following code:

```
{% if folderitem.external_url %}
  <li><a href="{{folderitem.external_url}}" target="_blank" r
el="noopener">{{folderitem.title}}</a></li>
```

You can see that the `external_url` is a condition that applies a different formatting. Although this feature is available, I recommend putting any external navigation links in the top navigation bar instead of the side navigation bar.

Sidebar item highlighting

The `sidebar.html` file inserts an `active` class into the sidebar element when the `url` attribute in the sidebar data file matches the page URL.

For example, the `sidebar.html` file contains the following code:


```
{% elsif page.url == folderitem.url %}  
  <li class="active"><a href="{{folderitem.url | remove:  
  "/" }}">{{folderitem.title}}</a></li>
```

If the `page.url` matches the `subfolderitem.url`, then an `active` class gets applied. If not, the `active` class does not get applied.

The `page.url` in Jekyll is a site-wide variable. If you insert `{{page.url}}` on a page, it will render as follows: `/mydoc_sidebar_navigation.html`. The `url` attribute in the sidebar item must match the page URL in order to get the `active` class applied.

This is why the `url` value in the sidebar data file looks something like this:

```
- title: Understanding how the sidebar works  
  permalink: mydoc_understand_sidebar.html  
  output: web, pdf
```

Note that the url does not include the project folder where the file is stored. This is because the site uses permalinks, which pulls the topics out of subfolders and places them into the root directory when the site builds.

Now the `page.url` and the `item.url` can match and the `active` class can get applied. With the `active` class applied, the sidebar section remains open.

YAML tutorial in the context of Jekyll

Summary: YAML is a format that relies on white spacing to separate out the various elements of content. Jekyll lets you use Liquid with YAML as a way to parse through the data. Storing items for your table of contents is one of the most common uses of YAML with Jekyll.

Overview

One of the most interesting features of Jekyll is the ability to separate out data elements from formatting elements using a combination of YAML and Liquid. This setup is most common when you're trying to create a table of contents.

Not many Jekyll themes actually have a robust table of contents, which is critical when you are creating any kind of documentation or reference material that has a lot of pages.

Here's the basic approach in creating a table of contents. You store your data items in a YAML file using YAML syntax. (I'll go over more about YAML syntax in a later section.) You then create your HTML structure in another file, such as sidebar.html. You might leverage one of the many different table of content frameworks (such as [Navgoco](#)) that have been created for this HTML structure.

Then, using Liquid syntax for loops and conditions, you access all of those values from the data file and splice them into HTML formatting. This will become more clear as we go through some examples.

YAML overview

Rather than just jump into YAML at the most advanced level, I'm going to start from ground zero with an introduction to YAML and how you access basic values in your data files using Jekyll.

Note that you don't actually have to use Jekyll when using YAML. YAML is used in a lot of other systems and is a format completely independent of Jekyll. However, because Jekyll uses Liquid, it gives you a lot of power to parse through your YAML data and make use of it.

YAML itself doesn't do anything on its own — it's just a way of storing your data in a specific structure that other utilities can parse.

YAML basics

You can read about YAML from a lot of different sources. Here are some basic characteristics of YAML:

- YAML (“YAML Ain’t Markup Language”) doesn’t use markup tags. This means you won’t see any kind of angle brackets. It uses white space as a way to form the structure. This makes YAML much more human readable.
- Because YAML does use white space for the structure, YAML is extremely picky about the exactness of spaces. If you have just one extra space somewhere, it can cause the whole file to be invalid.
- For each new level in YAML, you indent two spaces. Each level provides a different access point for the content. You use dot notation to access each new level.
- Because tabs are not universally implemented the same way in editors, a tab might not equate to two spaces. In general, it’s best to manually type two spaces to create a new level in YAML.
- YAML has several types of elements. The most common are mappings and lists. A mapping is simply a key-value pair. A list is a sequence of items. List start with hyphens.
- Items at each level can have various properties. You can create conditions based on the properties.
- You can use “for” loops to iterate through a list.

I realize a lot of this vague and general; however, it will become a lot more clear as we go through some concrete examples.

In the `_data` folder, there’s a file called `samplelist.yml`. All of these examples come from that file.

Example 1: Simple mapping

YAML:

```
name:  
  husband: Tom  
  wife: Shannon
```

Markdown + Liquid:

```
<p>Husband's name: {{site.data.samplelist.name.husband}}</p>
<p>Wife's name: {{site.data.samplelist.name.wife}}</p>
```

Notice that in order to access the data file, you use `site.data.samplelist` where as `samplelist` is the name of the YAML file.

Result:

```
Husband's name: Tom
Wife's name: Shannon
```

Example 2: Line breaks**YAML:**

```
feedback: >
  This is my feedback to you.
  Even if I include linebreaks here,
  all of the linebreaks will be removed when the value is inser
ted.

block: |
  This pipe does something a little different.
  It preserves the breaks.
  This is really helpful for code samples,
  since you can format the code samples with
  the appropriate
```

Markdown:

```
<p><b>Feedback</b></p>
<p>{{site.data.samplelist.feedback}}</p>

<p><b>Block</b></p>
<p>{{site.data.samplelist.block}}</p>
```

Result:

Feedback

This is my feedback to you. Even if I include linebreaks here, all of the linebreaks will be removed when the value is inserted.

Block

This pipe does something a little different. It preserves the breaks. This is really helpful for code samples, since you can format the code samples with the appropriate white spacing.

The right angle bracket `>` allows you to put the value on the next lines (which must be indented). Even if you create a line break, the output will remove all of those line breaks, creating one paragraph.

The pipe `|` functions like the angle bracket in that it allows you to put the values for the mapping on the next lines (which again must be indented). However, the pipe does preserve all of the line breaks that you use. This makes the pipe method ideal for storing code samples.

Example 3: Simple list

YAML:

```
bikes:
- title: mountain bikes
- title: road bikes
- title: hybrid bikes
```

Markdown + Liquid:

```
<ul>
{% for item in site.data.samplelist.bikes %}
<li>{{item.title}}</li>
{% endfor %}
</ul>
```

Result:

- mountain bikes

- road bikes
- hybrid bikes

Here we use a “for” loop to get each item in the bikes list. By using `.title` we only get the `title` property from each list item.

Example 4: List items

YAML:

```
salesteams:
- title: Regions
  subfolderitems:
    - location: US
    - location: Spain
    - location: France
```

Markdown + Liquid:

```
{% for item in site.data.samplelist.salesteams %}
<h3>{{item.title}}</h3>
<ul>
  {% for entry in item.subfolderitems %}
  <li>{{entry.location}}</li>
  {% endfor %}
</ul>
{% endfor %}
```

Result:

Regions

- US
- Spain
- France

Hopefully you can start to see how to wrap more complex formatting around the YAML content. When you use a “for” loop, you choose the variable of what to call the list items. The variable you choose to use becomes how you access the properties of each list item. In this case, I decided to use the variable `item`. In order to get each property of the list item, I used `item.subfolderitems`.

Each list item starts with the hyphen `-`. You cannot directly access the list item by referring to a mapping. You only loop through the list items. If you wanted to access the list item, you would have to use something like `[1]`, which is how you access the position in an array. You cannot access a list item like you can access a mapping key.

Example 5: Table of contents

YAML:

```
toc:
  - title: Group 1
    subfolderitems:
      - page: Thing 1
      - page: Thing 2
      - page: Thing 3
  - title: Group 2
    subfolderitems:
      - page: Piece 1
      - page: Piece 2
      - page: Piece 3
  - title: Group 3
    subfolderitems:
      - page: Widget 1
      - page: Widget 2 it's
      - page: Widget 3
```

Markdown + Liquid:

```
{% for item in site.data.samplelist.toc %}  
<h3>{{item.title}}</h3>  
<ul>  
  {% for entry in item.subfolderitems %}  
    <li>{{entry.page}}</li>  
  {% endfor %}  
</ul>  
{% endfor %}
```

Result:**Group 1**

- Thing 1
- Thing 2
- Thing 3

Group 2

- Piece 1
- Piece 2
- Piece 3

Group 3

- Widget 1
- Widget 2
- Widget 3

This example is similar to the previous one, but it's more developed as a real table of contents.

Example 6: Variables

YAML:

```
something: &hello Greetings earthling!  
myref: *hello
```


Markdown:

```
{{ site.data.samplelist.myref }}
```

Result:

```
Greetings earthling!
```

This example is notably different. Here I'm showing how to reuse content in YAML file. If you have the same value that you want to repeat in other mappings, you can create a variable using the `&` symbol. Then when you want to refer to that variable's value, you use an asterisk `*` followed by the name of the variable.

In this case the variable is `&hello` and its value is `Greetings earthling!` In order to reuse that same value, you just type `*hello`.

I don't use variables much, but that's not to say they couldn't be highly useful. For example, let's say you put name of the product in parentheses after each title (because you have various products that you're providing documentation for in the same site). You could create a variable for that product name so that if you change how you're referring to it, you wouldn't have to change all instances of it in your YAML file.

Example 7: Positions in lists

YAML:

```
about:
- zero
- one
- two
- three
```

Markdown:

```
{{ site.data.samplelist.about[0] }}
```

Result:

```
zero
```

You can see that I'm accessing one of the items in the list using `[0]`. This refers to the position in the array where a list item is. Like most programming languages, you start counting at zero, not one.

I wanted to include this example because it points to the challenge in getting a value from a specific list item. You can't just call out a specific item in a list like you can with a mapping. This is why you usually iterate through the list items using a "for" loop.

Example 8: Properties from list items at specific positions

YAML:

```
numbercolors:
- zero:
  properties: red
- one:
  properties: yellow
- two:
  properties: green
- three:
  properties: blue
```

Markdown + Liquid:

```
{{ site.data.samplelist.numbercolors[0].properties }}
```

Result:

```
red
```

This example is similar as before; however, in this case we were getting a specific property from the list item in the zero position.

Example 9: Conditions

YAML:

```
mypages:
- section1: Section 1
  audience: developers
  product: acme
  url: facebook.com
- section2: Section 2
  audience: writers
  product: acme
  url: google.com
- section3: Section 3
  audience: developers
  product: acme
  url: amazon.com
- section4: Section 4
  audience: writers
  product: gizmo
  url: apple.com
- section5: Section 5
  audience: writers
  product: acme
  url: microsoft.com
```

Markdown + Liquid:

```
<ul>
{% for sec in site.data.samplelist.mypages %}
{% if sec.audience == "writers" %}
<li>{{sec.url}}</li>
{% endif %}
{% endfor %}
</ul>
```

Result:

- google.com
- apple.com
- microsoft.com

This example shows how you can use conditions in order to selectively get the YAML content. In your table of contents, you might have a lot of different pages. However, you might only want to get the pages for a particular audience. Conditions lets you get only the items that meet those audience attributes.

Now let's adjust the condition just a little. Let's add a second condition so that the `audience` property has to be `writers` and the `product` property has to be `gizmo`. This is how you would write it:

```
<ul>
{% for sec in site.data.samplelist.mypages %}
{% if sec.audience == "writers" and sec.product == "gizmo" %}
<li>{{sec.url}}</li>
{% endif %}
{% endfor %}
</ul>
```

And here is the result:

- [apple.com](#)

More resources

For more examples and explanations, see this helpful post on [tournemille.com](#): [How to create data-driven navigation in Jekyll](#) .

Tags

Summary: Tags provide another means of navigation for your content. Unlike the table of contents, tags can show the content in a variety of arrangements and groupings. Implementing tags in this Jekyll theme is somewhat of a manual process.

Add a tag to a page

You can add tags to pages by adding `tags` in the frontmatter with values inside brackets, like this:

```
---
title: 5.0 Release Notes
permalink: release_notes_5_0.html
tags: [formatting, single_sourcing]
---
```

Tags overview

Note: With posts, tags have a namespace that you can access with `posts.tags.tagname`, where `tagname` is the name of the tag. You can then list all posts in that tag namespace. But pages don't off this same tag namespace, so you could actually use another key instead of `tags`. Nevertheless, I'm using the same tags approach for posts as with pages.

To prevent tags from getting out of control and inconsistent, first make sure the tag appears in the `_data/tags.yml` file. If it's not there, the tag you add to a page won't be read. I added this check just to make sure I'm using the same tags consistently and not adding new tags that don't have tag archive pages.

Note: In contrast to WordPress, with Jekyll to get tags on pages you have to build out the functionality for tags so that clicking a tag name shows you all pages with that tag. Tags in Jekyll are much more manual.

Additionally, you must create a tag archive page similar to the other pages named `tag_{tagname}.html` folder. This theme doesn't auto-create tag archive pages.

For simplicity, make all your tags single words (connect them with hyphens if necessary).

Setting up tags

Tags have a few components.

1. In the `_data/tags.yml` file, add the tag names you want to allow. For example:

```
allowed-tags:
  - getting_started
  - overview
  - formatting
  - publishing
  - single_sourcing
  - special_layouts
  - content_types
```

2. Create a tag archive file for each tag in your `tags_doc.yml` list. Name the file following the same pattern in the `tags` folder, like this:
`tag_collaboration.html`.

Each tag archive file needs only this:

```
---
title: "Collaboration pages"
tagName: collaboration
search: exclude
permalink: tag_collaboration.html
sidebar: mydoc_sidebar
---
{% include taglogic.html %}
```

❗ Note: In the `_includes/mydoc` folder, there's a `taglogic.html` file. This file (included in each tag archive file) has common logic for getting the tags and listing out the pages containing the tag in a table with summaries or truncated excerpts. You don't have to do anything with the file — just leave it there because the tag archive pages reference it.

3. Change the title, tagName, and permalink values to be specific to the tag name you just created.

By default, the `_layouts/page.html` file will look for any tags on a page and insert them at the bottom of the page using this code:

```
<div class="tags">
{% if page.tags != null %}
<b>Tags: </b>
{% assign projectTags = site.data.tags.allowed-tags %}
{% for tag in page.tags %}
{% if projectTags contains tag %}
<a href="{{ "tag_" | append: tag | append: ".html" }}" class="btn btn-default navbar-btn cursorNorm" role="button">{{page.tagName}}
{{tag}}</a>
{% endif %}
{% endfor %}
{% endif %}
</div>
```

Because this code appears on the `_layouts/page.html` file by default, you don't need to do anything in your page to get the tags to appear. However, if you want to alter the placement or change the button color, you can do so within the `_includes/taglogic.html` file.

You can change the button color by changing the class on the button from `btn-info` to one of the other button classes bootstrap provides. See [Labels \(page 302\)](#) for more options on button class names.

Retrieving pages for a specific tag

If you want to retrieve pages outside of a particular `tag_archive` page, you could use this code:

```
Getting started pages:
<ul>
{% for page in site.pages %}
{% for tag in page.tags %}
{% if tag == "getting_started" %}
<li><a href="{{page.url | remove: "/" }}">{{page.title}}</a></li>
{% endif %}
{% endfor %}
{% endfor %}
</ul>
```

Here's how that code renders:

Getting started pages:

- [Getting started with the Documentation Theme for Jekyll \(page 0\)](#)
- [About the theme's author \(page 208\)](#)
- [About Ruby, Gems, Bundler, and other prerequisites \(page 214\)](#)
- [Install Jekyll on Mac \(page 222\)](#)
- [Pages \(page 230\)](#)
- [Posts \(page 237\)](#)
- [Release notes 5.0 \(page 212\)](#)
- [Release notes 6.0 \(page 210\)](#)
- [Sidebar Navigation \(page 257\)](#)
- [Support \(page 209\)](#)
- [Supported features \(page 203\)](#)

If you want to sort the pages alphabetically, you have to apply a `sort` filter:


```
Getting started pages:
<ul>
{% assign sorted_pages = site.pages | sort: 'title' %}
{% for page in sorted_pages %}
{% for tag in page.tags %}
{% if tag == "getting_started" %}
<li><a href="{{page.url | remove: '/' }}">{{page.title}}</a></li>
{% endif %}
{% endfor %}
{% endfor %}
</ul>
```

Here's how that code renders:

Getting started pages:

- [About Ruby, Gems, Bundler, and other prerequisites \(page 214\)](#)
- [About the theme's author \(page 208\)](#)
- [Getting started with the Documentation Theme for Jekyll \(page 0\)](#)
- [Install Jekyll on Mac \(page 222\)](#)
- [Pages \(page 230\)](#)
- [Posts \(page 237\)](#)
- [Release notes 5.0 \(page 212\)](#)
- [Release notes 6.0 \(page 210\)](#)
- [Sidebar Navigation \(page 257\)](#)
- [Support \(page 209\)](#)
- [Supported features \(page 203\)](#)

Efficiency

Although the tag approach here uses `for` loops, these are somewhat inefficient on a large site. Most of my tech doc projects don't have hundreds of pages (like my blog does). If your project does have hundreds of pages, this `for` loop approach with tags is going to slow down your build times.

Without the ability to access pages inside a universal namespace with the page type, there aren't many workarounds here for faster looping.

With posts (instead of pages), since you can access just the posts inside `posts.tag.tagname`, you can be a lot more efficient with the looping.

Still, if the build times are getting long (e.g., 1 or 2 minutes per build), look into reducing the number of `for` loops on your site.

Empty tags?

If your page shows “tags:” at the bottom without any value, it could mean a couple of things:

- You’re using a tag that isn’t specified in your allowed tags list in your tags.yml file.
- You have an empty `tags: []` property in your frontmatter.

If you don’t want tags to appear at all on your page, remove the tags property from your frontmatter.

Remembering the right tags

Since you may have many tags and find it difficult to remember what tags are allowed, I recommend creating a template that prepopulates all your frontmatter with all possible tags. Then just remove the tags that don’t apply.

See [WebStorm Text Editor \(page 252\)](#) for tips on creating file templates in WebStorm.

Series

Summary: You can automatically link together topics belonging to the same series. This helps users know the context within a particular process.

Using series for pages

You create a series by looking for all pages within a tag namespace that contain certain frontmatter. Here's a [demo \(page 0\)](#).

1. Create the series button

First create an include that contains your series button:

```
<div class="seriesContext">
  <div class="btn-group">
    <button type="button" data-toggle="dropdown" class="btn btn-primary dropdown-toggle">Series Demo <span class="caret"></span></button>
    <ol class="dropdown-menu">
      {% assign pages = site.pages | sort:"weight" %}
      {% for p in pages %}
      {% if p.series == "ACME series" %}
      {% if p.url == page.url %}
      <li class="active"> → {{p.weight}}. {{p.title}}</li>
      {% else %}
      <li>
        <a href="{{p.url | remove: '/' }}">{{p.weight}}. {{p.title}}</a>
      </li>
      {% endif %}
      {% endif %}
      {% endfor %}
    </ol>
  </div>
</div>
```

Change “ACME series” to the name of your series.

Save this in your `_includes/custom` folder as something like `series_acme.html`.

⚠ Warning: With pages, there isn't a universal namespace created from tags or categories like there is with Jekyll posts. As a result, you have to loop through all pages. If you have a lot of pages in your site (e.g., 1,000+), then this looping will create a slow build time. If this is the case, you will need to rethink the approach to looping here.

2. Create the “next” include

Now create another include for the Next button at the bottom of the page. Copy the following code, changing the series name to your series' name:

```
<p>{% assign series_pages = site.tags.series_acme %}
  {% for p in pages %}
    {% if p.series == "ACME series" %}
      {% assign nextTopic = page.weight | plus: "1" %}
      {% if p.weight == nextTopic %}
        <a href="{{p.url}}"><button type="button" class="btn btn-primary">Next: {{p.weight}} {{p.title}}</button></a>
      {% endif %}
    {% endif %}
  {% endfor %}
</p>
```

Change “acme” to the name of your series.

Save this in your `_includes/custom/mydoc` folder as `series_acme_next.html`.

3. Add the correct frontmatter to each of your series pages

Now add the following frontmatter to each page in the series:

```
series: "ACME series"
weight: 1.0
```

With weights, Jekyll will treat 10 as coming after 1. If you have more than 10 items, consider changing `plus: "1.0"` to `plus: "0.1"`.

Additionally, if your page names are prefaced with numbers, such as “1. Download the code,” then the `{{p.weight}}` will create a duplicate number. In that case, just remove the `{{p.weight}}` from both code samples here.

4. Add links to the series button and next button on each page.

On each series page, add a link to the series button at the top and a link to the next button at the bottom.

```
<!-- your frontmatter goes here -->

{% include custom/series_acme.html %}

<!-- your page content goes here ... -->

{% include custom/series_acme_next.html %}
```

Changing the series drop-down color

The Bootstrap menu uses the `primary` class for styling. If you change this class in your theme, the Bootstrap menu should automatically change color as well. You can also just use another Bootstrap class in your button code. Instead of `btn-primary`, use `btn-info` or `btn-warning`. See [Labels \(page 302\)](#) for more Bootstrap button classes.

Using a collection with your series

Instead of copying and pasting the button includes on each of your series, you could also create a collection and define a layout for the collection that has the include code. For more information on creating collections, see [Collections \(page 250\)](#) for more details.

Tooltips

Summary: You can add tooltips to any word, such as an acronym or specialized term. Tooltips work well for glossary definitions, because you don't have to keep repeating the definition, nor do you assume the reader already knows the word's meaning.

Creating tooltips

Because this theme is built on Bootstrap, you can simply use a specific attribute on an element to insert a tooltip.

Suppose you have a glossary.yml file inside your `_data` folder. You could pull in that glossary definition like this:

```
<a href="#" data-toggle="tooltip" data-original-title="{{site.data.glossary.jekyll_platform}}">Jekyll</a> is my favorite tool for building websites.
```

This renders to the following:

Jekyll is my favorite tool for building websites.

Alerts

Summary: You can insert notes, tips, warnings, and important alerts in your content. These notes make use of Bootstrap styling and are available through data references such as `site.data.alerts.note`.

About alerts

Alerts are little warnings, info, or other messages that you have called out in special formatting. In order to use these alerts or callouts, reference the appropriate value stored in the `alerts.yml` file as described in the following sections.

Alerts

Similar to [inserting images \(page 296\)](#), you insert alerts through various includes that have been developed. These includes provide templates through which you pass parameters to easily populate the right HTML code.

```
{% include note.html content="This is my note. All the content  
I type here is treated as a single paragraph." %}
```

Here's the result:

❗ Note: This is my note. All the content I type here is treated as a single paragraph.

With alerts, there's just one include property:

Property	description
content	The content for the alert.

Using block level tags inside the alerts

If you need multiple paragraphs, enter `<br/><br/>` tags. This is because block level tags aren't allowed here, as Kramdown is processing the content as Markdown despite the fact that the content is surrounded by HTML tags. Here's an example with a break:

```
{% include note.html content="This is my note. All the content
I type here is treated as a single paragraph. <br/><br/> Now
I'm typing on a new line." %}
```

Here's the result:

📌 Note: This is my note. All the content I type here is treated as a single paragraph.

Now I'm typing on a new line.

The include uses `markdown="span"` as an attribute, which means kramdown will process the entire `content` as a span. You can't use block elements such as `p` or `div` or `pre`. If you need these elements, you can either manually surround the content with the HTML from the include, or you can use these tags:

[illegible]

Result:

Note:
This is my note.


```
def foo(x):  
    return x+1
```

The same Bootstrap code from the alert is stored in yaml files inside the `_data` folder. (This was how I previously implemented this code, but since this method was prone to error and didn't trigger any build warnings or failures when incorrectly coded, I changed the approach to use includes instead.)

Types of alerts available

There are four types of alerts you can leverage:

- `note.html`
- `tip.html`
- `warning.html`
- `important.html`

They function the same except they have a different color, icon, and alert word. You include the different types by selecting the include template you want. Here are samples of each alert:

Note: This is my note.

Tip: This is my tip.

Warning: This is my warning.

Important: This is my important info.

These alerts leverage includes stored in the `_include` folder. The `content` option is a parameter that you pass to the include. In the include, the parameter is passed like this:

```
<div markdown="span" class="alert alert-info" role="alert"><i class="fa fa-info-circle"></i> <b>Note:</b> {{include.content}}</div>
```

The content in `content="This is my note."` gets inserted into the `{{include.content}}` part of the template. You can follow this same pattern to build additional includes. See this [Jekyll screencast on includes](#) or [this screencast](#) for more information.

Callouts

There's another type of callout available called callouts. This format is typically used for longer callout that spans more than one or two paragraphs, but really it's just a stylistic preference whether to use an alert or callout.

Here's the syntax for a callout:

```
{% include callout.html content="This is my callout. It has a border on the left whose color you define by passing a type parameter. I typically use this style of callout when I have more information that I want to share, often spanning multiple paragraphs. " type="primary" %}
```

Here's the result:

This is my callout. It has a border on the left whose color you define by passing a type parameter. I typically use this style of callout when I have more information that I want to share, often spanning multiple paragraphs.

The available properties for callouts are as follows:

Property	description
content	The content for the callout.
type	The style for the callout. Options are <code>danger</code> , <code>default</code> , <code>primary</code> , <code>success</code> , <code>info</code> , and <code>warning</code> .

The types just define the color of the left border. Each of these callout types get inserted as a class name in the callout template. These class names correspond with styles in Bootstrap. These classes are common Bootstrap class names whose style attributes differ depending on your Bootstrap theme and style definitions.

Here's an example of each different type of callout:

This is my **danger** type callout. It has a border on the left whose color you define by passing a type parameter.

This is my **default** type callout. It has a border on the left whose color you define by passing a type parameter.

This is my **primary** type callout. It has a border on the left whose color you define by passing a type parameter.

This is my **success** type callout. It has a border on the left whose color you define by passing a type parameter.

This is my **info** type callout. It has a border on the left whose color you define by passing a type parameter.

This is my **warning** type callout. It has a border on the left whose color you define by passing a type parameter.

Now that in contrast to alerts, callouts don't include the alert word (note, tip, warning, or important). You have to manually include it inside **content** if you want it.

To include paragraph breaks, use `<br/><br/>` inside the callout:

```
{% include callout.html content="**Important information**: This is my callout. It has a border on the left whose color you define by passing a type parameter. I typically use this style of callout when I have more information that I want to share, often spanning multiple paragraphs. <br/><br/>Here I am starting a new paragraph, because I have lots of information to share. You may wonder why I'm using line breaks instead of paragraph tags. This is because Kramdown processes the Markdown here as a span rather than a div (for whatever reason). Be grateful that you can be using Markdown at all inside of HTML. That's usually not allowed in Markdown syntax, but it's allowed here." type="primary" %}
```

Here's the result:

Important information: This is my callout. It has a border on the left whose color you define by passing a type parameter. I typically use this style of callout when I have more information that I want to share, often spanning multiple paragraphs.

Here I am starting a new paragraph, because I have lots of information to share. You may wonder why I'm using line breaks instead of paragraph tags. This is because Kramdown processes the Markdown here as a span rather than a div (for whatever reason). Be grateful that you can be using Markdown at all inside of HTML. That's usually not allowed in Markdown syntax, but it's allowed here.

Use Liquid variables inside parameters with includes

Suppose you have a product name or some other property that you're storing as a variable in your configuration file (`_config.yml`), and you want to use this variable in the `content` parameter for your alert or callout. You will get an error if you use Liquid syntax inside a include parameter. For example, this syntax will produce an error:

```
{% include note.html content="The {{site.company}} is pleased to announce an upcoming release." %}
```

The error will say something like this:

```
Liquid Exception: Invalid syntax for include tag. File contains invalid characters or sequences: ... Valid syntax: {% include file.ext param='value' param2='value' %}
```

To use variables in your include parameters, you must use the “variable parameter” approach. First you use a `capture` tag to capture some content. Then you reference this captured tag in your include. Here’s an example.

In my site configuration file (`_config.yml`), I have a property called `company_name`.

```
company_name: Your company
```

I want to use this variable in my note include.

First, before the note I capture the content for my note’s include like this:

```
{% capture company_note %}The {{site.company_name}} company is pleased to announce an upcoming release.{% endcapture %}
```

Now reference the `company_note` in your `include` parameter like this:

```
{% include note.html content=company_note %}
```

Here’s the result:

Note: The OpenResty 中文 is pleased to announce an upcoming release.

Note the omission of quotation marks with variable parameters.

Also note that instead of storing the variable in your site’s configuration file, you could also put the variable in your page’s frontmatter. Then instead of using `{{site.company_name}}` you would use `{{page.company_name}}`.

Markdown inside of callouts and alerts

You can use Markdown inside of callouts and alerts, even though this content actually gets inserted inside of HTML in the include. This is one of the advantages of kramdown Markdown. The include template has an attribute of `markdown="span"` that allows for the processor to parse Markdown inside of HTML.

Validity checking

If you have some of the syntax wrong with an alert or callout, you'll see an error when Jekyll tries to build your site. The error may look like this:

```
Liquid Exception: Invalid syntax for include tag: content="This is my **info** type callout. It has a border on the left whose color you define by passing a type parameter. type="info" Valid syntax: {% include file.ext param='value' param2='value' %} in mydoc/mydoc_alerts.md
```

These errors are a good thing, because it lets you know there's an error in your syntax. Without the errors, you may not realize that you coded something incorrectly until you see the lack of alert or callout styling in your output.

In this case, the quotation marks aren't set correctly. I forgot the closing quotation mark for the content parameter include.

Blast a warning to users on every page

If you want to blast a warning to users on every page, add the alert or callout to the `_layouts/page.html` page right below the frontmatter. Every page using the page layout (all, by default) will show this message.

Icons

Summary: You can integrate font icons through the Font Awesome and Glyphical Halflings libraries. These libraries allow you to embed icons through their libraries delivered as a link reference. You don't need any image libraries downloaded in your project.

Font icon options

The theme has two font icon sets integrated: Font Awesome and Glyphicons Halflings. The latter is part of Bootstrap, while the former is independent. Font icons allow you to insert icons drawn as vectors from a CDN (so you don't have any local images on your own site).

External icons

When you link to an external site, like [Jekyll](#), an icon appears after the link. If you want to remove this icon, comment out this style in `css/customstyles.css`.

```
/* this part adds an icon after external links, using FontAwesome */  
a[href^="http://"]:after, a[href^="https://"]:after {  
  content: "\f08e";  
  font-family: FontAwesome;  
  font-weight: normal;  
  font-style: normal;  
  display: inline-block;  
  text-decoration: none;  
  padding-left: 3px;  
}
```

See Font Awesome icons available

Go to the [Font Awesome library](#) to see the available icons.

The Font Awesome icons allow you to adjust their size by simply adding `fa-2x`, `fa-3x` and so forth as a class to the icon to adjust their size to two times or three times the original size. As vector icons, they scale crisply at any size.

Here's an example of how to scale up a camera icon:

```

<i class="fa fa-camera-retro"></i> normal size (1x)
<i class="fa fa-camera-retro fa-lg"></i> fa-lg
<i class="fa fa-camera-retro fa-2x"></i> fa-2x
<i class="fa fa-camera-retro fa-3x"></i> fa-3x
<i class="fa fa-camera-retro fa-4x"></i> fa-4x
<i class="fa fa-camera-retro fa-5x"></i> fa-5x

```

Here's what they render to:



With Font Awesome, you always use the `i` tag with the appropriate class. You also implement `fa` as a base class first. You can use font awesome icons inside other elements. Here I'm using a Font Awesome class inside a Bootstrap alert:

```

<div class="alert alert-danger" role="alert"><i class="fa fa-exclamation-circle"></i> <b>Warning: </b>This is a special warning message.

```

Here's the result:

! This is a special warning message.

The notes, tips, warnings, etc., are pre-coded with Font Awesome and stored in the alerts.yml file. That file includes the following:


```
tip: '<div class="alert alert-success" role="alert"><i class="fa fa-check-square-o"></i> <b>Tip: </b>'
note: '<div class="alert alert-info" role="alert"><i class="fa fa-info-circle"></i> <b>Note: </b>'
important: '<div class="alert alert-warning" role="alert"><i class="fa fa-warning"></i> <b>Important: </b>'
warning: '<div class="alert alert-danger" role="alert"><i class="fa fa-exclamation-circle"></i> <b>Warning: </b>'
end: '</div>'

callout_danger: '<div class="bs-callout bs-callout-danger">'
callout_default: '<div class="bs-callout bs-callout-default">'
callout_primary: '<div class="bs-callout bs-callout-primary">'
callout_success: '<div class="bs-callout bs-callout-success">'
callout_info: '<div class="bs-callout bs-callout-info">'
callout_warning: '<div class="bs-callout bs-callout-warning">'

hr_faded: '<hr class="faded"/>'
hr_shaded: '<hr class="shaded"/>'
```

This means you can insert a tip, note, warning, or important alert simply by using these tags.

```
{% include note.html content="Add your note here." %}
```

```
{% include tip.html content="Add your tip here." %}
```

```
{% include important.html content="Add your important info here." %}
```

```
{% include warning.html content="Add your warning here." %}
```

Here's the result:

Note: Add your note here.

Tip: Here's my tip.

⚠ Important: This information is very important.

⚠ Warning: If you overlook this, you may die.

The color scheme is the default colors from Bootstrap. You can modify the icons or colors as needed.

Creating your own combinations

You can innovate with your own combinations. Here's a similar approach with a file download icon:

```
<div class="alert alert-success" role="alert"><i class="fa fa-download fa-lg"></i> This is a special tip about some file to download....</div>
```

And the result:

 This is a special tip about some file to download....

Grab the right class name from the [Font Awesome library](#) and then implement it by following the pattern shown previously.

If you want to make your fonts even larger than the 5x style, add a custom style to your stylesheet like this:

```
.fa-10x{font-size:1700%;}
```

Then any element with the attribute `fa-10x` will be enlarged 1700%.

Glyphicon icons available

Glyphicons work similarly to Font Awesome. Go to the [Glyphicons library](#) to see the icons available.

Although the Glyphicon Halflings library doesn't provide the scalable classes like Font Awesome, there's a [StackOverflow trick](#) to make the icons behave in a similar way. This theme's stylesheet (customstyles.css) includes the following to the stylesheet:

```
.gi-2x{font-size: 2em;}  
.gi-3x{font-size: 3em;}  
.gi-4x{font-size: 4em;}  
.gi-5x{font-size: 5em;}
```

Now you just add `gi-5x` or whatever to change the size of the font icon:

```
<span class="glyphicon glyphicon-globe gi-5x"></span>
```

And here's the result:



Glyphicons use the `span` element instead of `i` to attach their classes.

Here's another example:

```
<span class="glyphicon glyphicon-download"></span>
```



And magnified:

```
<span class="glyphicon glyphicon-download gi-3x"></span>
```



You can also put glyphicons inside other elements:

```
<div class="alert alert-danger" role="alert">
  <span class="glyphicon glyphicon-exclamation-sign" aria-hidden="true"></span>
  <b>Error:</b> Enter a valid email address
</div>
```

❗ Error: Enter a valid email address

Callouts

The previously shown alerts might be fine for short messages, but with longer notes, the solid color takes up a bit of space. In this theme, you also have the option of using callouts, which are pretty common in Bootstrap's documentation but surprisingly not offered as an explicit element. Their styles have been copied into this theme, in a way similar to the alerts:

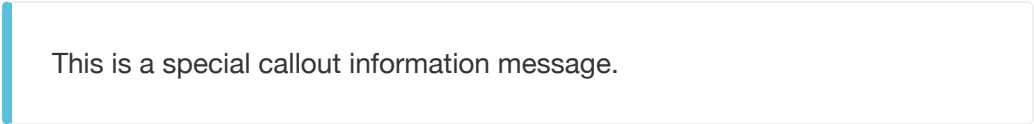
```
<div class="bs-callout bs-callout-info">
  This is a special info message. This is a special info message. This is a special info message. This is a special info message. This is a special info message. This is a special info message. This is a special info message. This is a special info message. </div>
```

❓ This is a special info message. This is a special info message. This is a special info message. This is a special info message. This is a special info message. This is a special info message. This is a special info message. This is a special info message.

And here's the shortcode:

```
{{site.data.alerts.callout_info}}This is a special callout information message.{{site.data.alerts.end}}
```

Here's the result:



This is a special callout information message.

You can use any of the following:

```
{{site.data.alerts.callout_default}}  
{{site.data.alerts.callout_primary}}  
{{site.data.alerts.callout_success}}  
{{site.data.alerts.callout_info}}  
{{site.data.alerts.callout_warning}}
```

The only difference is the color of the left bar.

Callouts are explained in a bit more detail in [Alerts \(page 281\)](#).

Images

Summary: Store images in the images folder and use the image.html include to insert images. This include has several options, including figcaptions, that extract the content from the formatting.

Image Include Template

Instead of using Markdown or HTML syntax directly in your page for images, the syntax for images has been extracted out into an image include that allows you to pass the parameters you need. Include the image.html like this:

```
{% include image.html file="jekyll.png" url="http://jekyllrb.com" alt="Jekyll" caption="This is a sample caption" %}
```

The available include properties are as follows:

Property	description
file	The name of the file. Store it in the /images folder. If you want to organize your images in subfolders, reference the subfolder path here, like this: <code>mysubfolder/jekyllrb.png</code>
url	Whether to link the image to a URL
alt	Alternative image text for accessibility and SEO
caption	A caption for the image
max-width	a maximum width for the image (in pixels). Just specify the number, not px.

The properties of the include get populated into the image.html template.

Here's the result:



This is a sample caption

Inline image includes

For inline images, such as with a button that you want to appear inline with text, use the `inline_image.html` include, like this:

```
Click the Android SDK Manager button {% include inline_image.html
file="androidsdkmanagericon.png" alt="SDK button" %}
```

Click the **Android SDK Manager** button 

The `inline_image.html` include properties are as follows:

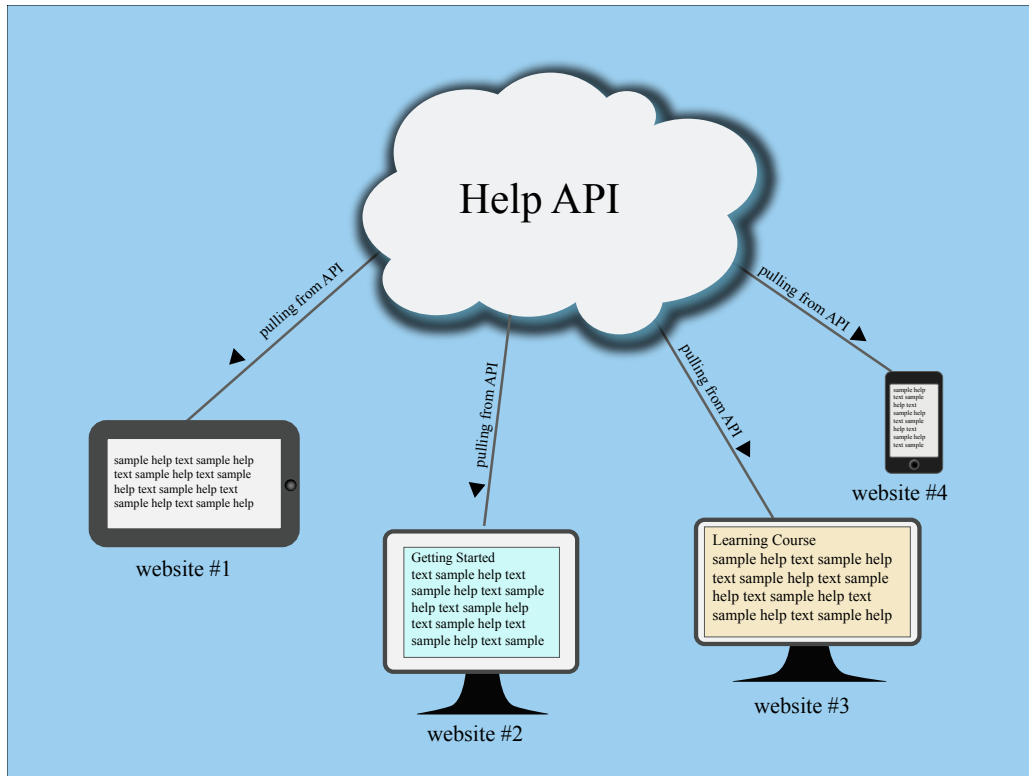
Property	description
file	The name of the file
type	The type of file (png, svg, and so on)
alt	Alternative image text for accessibility and SEO

SVG Images

You can also embed SVG graphics. If you use SVG, you need to use the HTML syntax so that you can define a width/container for the graphic. Here's a sample embed:

```
{% include image.html file="helpapi.svg" url="http://idratherbe
writing.com/documentation-theme-jekyll/mydoc_help_api/" alt="Bu
ilding a Help API" caption="A help API provides a JSON file at
a web URL with content that can be pulled into different target
s" max-width="600" %}
```

Here's the result:



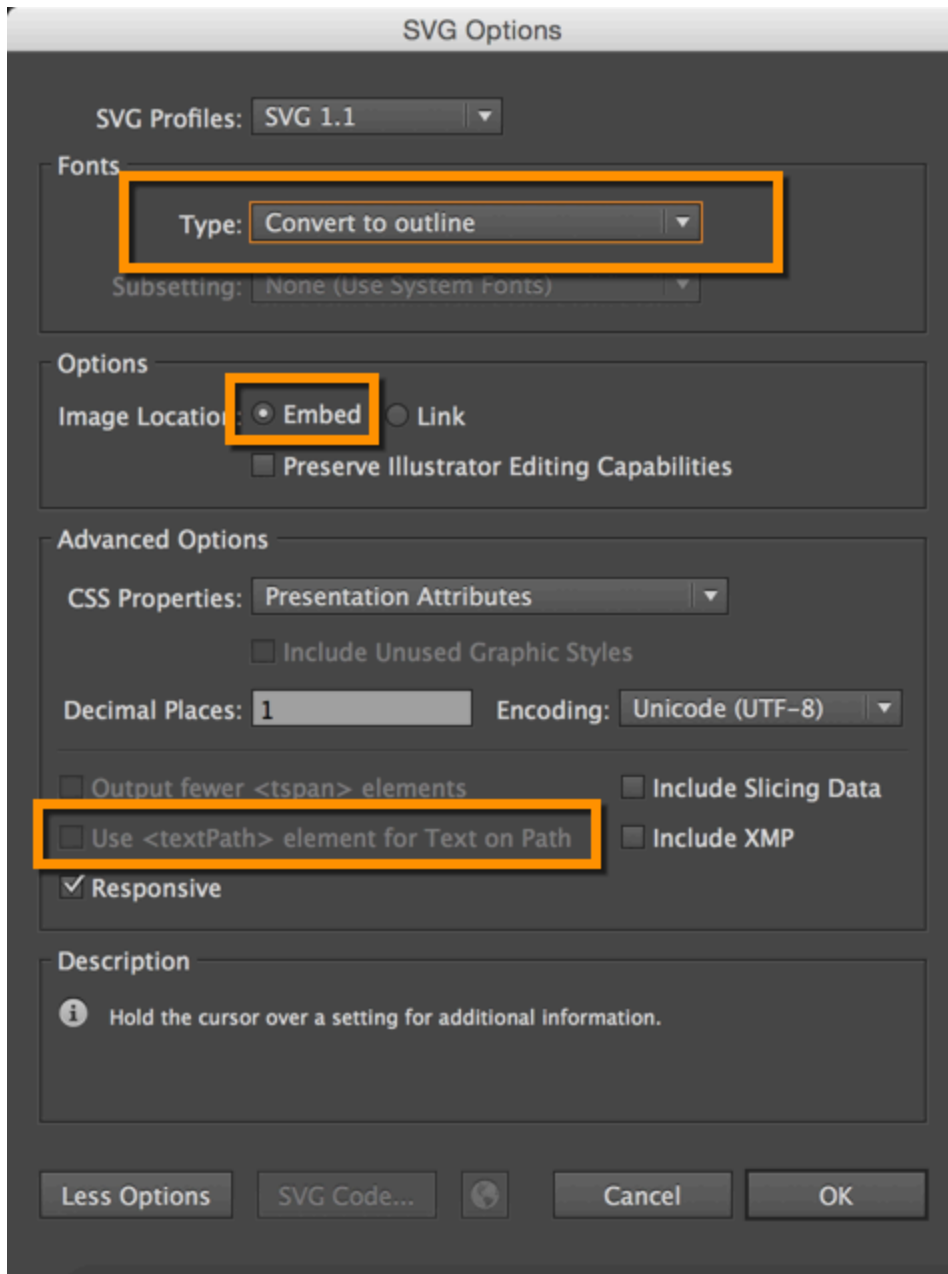
A help API provides a JSON file at a web URL with content that can be pulled into different targets

The stylesheet even handles SVG display in IE 9 and earlier through the following style (based on this [gist](#)):


```
/*
 * Let's target IE to respect aspect ratios and sizes for img tags containing SVG files
 *
 * [1] IE9
 * [2] IE10+
 */
/* 1 */
.ie9 img[src$=".svg"] {
    width: 100%;
}
/* 2 */
@media screen and (-ms-high-contrast: active), (-ms-high-contrast: none) {
    img[src$=".svg"] {
        width: 100%;
    }
}
```

Also, if you're working with SVG graphics, note that Firefox does not support SVG fonts. In Illustrator, when you do a Save As with your AI file and choose SVG, to preserve your fonts, in the Font section, select "Convert to outline" as the Type (don't choose SVG in the Font section).

Also, remove the check box for "Use textpath element for text on a path". And select "Embed" rather than "Link." The following screenshot shows the settings I use. Your graphics will look great in Firefox.



Essential options for SVG with Illustrator

Code samples

Summary: You can use fenced code blocks with the language specified after the first set of backtick fences.

Code Samples

Use fenced code blocks with the language specified, like this:

```
```js
console.log('hello');
```
```

Result:

```
console.log('hello');
```

For the list of supported languages you can use (similar to `js` for JavaScript), see [Supported languages](#).

Labels

Summary: Labels are just a simple Bootstrap component that you can include in your pages as needed. They represent one of many Bootstrap options you can include in your theme.

About labels

Labels might come in handy for adding button-like tags next to elements, such as POST, DELETE, UPDATE methods for endpoints. You can use any classes from Bootstrap in your content.

```
<span class="label label-default">Default</span>
<span class="label label-primary">Primary</span>
<span class="label label-success">Success</span>
<span class="label label-info">Info</span>
<span class="label label-warning">Warning</span>
<span class="label label-danger">Danger</span>
```

Default Primary Success Info Warning Danger

You can have a label appear within a heading simply by including the span tag in the heading. However, you can't mix Markdown syntax with HTML, so you'd have to hard-code the heading ID for the auto-TOC to work.

Links

Summary: When creating links, you can use standard HTML or Markdown formatting. However, you can also implement an automated approach to linking that makes linking much less error-prone (meaning less chances of broken links in your output) and requiring less effort.

Create an external link

When linking to an external site, use Markdown formatting because it's simplest:

```
[Google](http://google.com)
```

Linking to internal pages

When linking to internal pages, you can manually link to the pages like this:

```
[Icons](mydoc_icons.html)
```

However, if you change the file name, you'll have to update all of your links.

Navtabs

Summary: Navtabs provide a tab-based navigation directly in your content, allowing users to click from tab to tab to see different panels of content. Navtabs are especially helpful for showing code samples for different programming languages. The only downside to using navtabs is that you must use HTML instead of Markdown.

Common uses

Navtabs are particularly useful for scenarios where you want to show a variety of options, such as code samples for Java, .NET, or PHP, on the same page.

While you could resort to single-source publishing to provide different outputs for each unique programming language or role, you could also use navtabs to allow users to select the content you want.

Navtabs are better for SEO since you avoid duplicate content and drive users to the same page.

Navtabs demo

The following is a demo of a navtab. Refresh your page to see the tab you selected remain active.



Profile

Praesent sit amet fermentum leo. Aliquam feugiat,

1. nibh in u ltrices mattis
2. felis ipsum venenatis metus, vel vehicula libero mauris a enim. Sed placerat est ac lectus vestibulum tempor.
 - Quisque ut condimentum massa.
 - ut condimentum massa.

Proin venenatis leo id urna cursus blandit. Vivamus sit amet hendrerit metus.

Code

Here's the code for the above (with the filler text abbreviated):

```
<ul id="profileTabs" class="nav nav-tabs">
  <li class="active"><a href="#profile" data-toggle="tab">Pro
file</a></li>
  <li><a href="#about" data-toggle="tab">About</a></li>
  <li><a href="#match" data-toggle="tab">Match</a></li>
</ul>
<div class="tab-content">
<div role="tabpanel" class="tab-pane active" id="profile">
  <h2>Profile</h2>
<p>Praesent sit amet fermentum leo....</p>
</div>

<div role="tabpanel" class="tab-pane" id="about">
  <h2>About</h2>
  <p>Lorem ipsum ...</p></div>

<div role="tabpanel" class="tab-pane" id="match">
  <h2>Match</h2>
  <p>Vel vehicula ....</p>
</div>
</div>
```

Design constraints

Bootstrap automatically clears any floats after the navtab. Make sure you aren't trying to float any element to the right of your navtabs, or there will be some awkward space in your layout.

Appearance in the mini-TOC

If you put a heading in the navtab content, that heading will appear in the mini-TOC as long as the heading tag has an ID. If you don't want the headings for each navtab section to appear in the mini-TOC, omit the ID attribute from the heading tag. Without this ID attribute in the heading, the mini-TOC won't insert the heading title into the mini-TOC.

Must use HTML

You must use HTML within the navtab content because each navtab section is surrounded with HTML, and you can't use Markdown inside of HTML.

Match up ID tags

Each tab's `href` attribute must match the `id` attribute of the tab content's `div` section. So if your tab has `href="#acme"`, then you add `acme` as the ID attribute in `<div role="tabpanel" class="tab-pane" id="acme">`.

Set an active tab

One of the tabs needs to be set as active, depending on what tab you want to be open by default (usually the first one).

```
<div role="tabpanel" class="tab-pane active" id="acme">
```

Sets a cookie

The navtabs are part of Bootstrap, but this theme sets a cookie to remember the last tab's state. The `js/customscripts.js` file has a long chunk of JavaScript that sets the cookie. The JavaScript comes from [this StackOverflow thread](#).

By setting a cookie, if the user refreshes the page, the active tab is the tab the user last selected (rather than defaulting to the default active tab).

Functionality to implement

One piece of functionality I'd like to implement is the ability to set site-wide nav tab options. For example, if the user always chooses PHP instead of Java in the code samples, it would be great to set this option site-wide by default. However, this functionality isn't yet coded.

Tables

Summary: You can format tables using either multimarkdown syntax or HTML. You can also use jQuery datatables (a plugin) if you need more robust tables.

Multimarkdown Tables

You can use Multimarkdown syntax for tables. The following shows a sample:

```
| Priority apples | Second priority | Third priority |  
|-----|-----|-----|  
| ambrosia | gala | red delicious |  
| pink lady | jazz | macintosh |  
| honeycrisp | granny smith | fuji |
```

Result:

Priority apples	Second priority	Third priority
ambrosia	gala	red delicious
pink lady	jazz	macintosh
honeycrisp	granny smith	fuji

Note: You can't use block level tags (paragraphs or lists) inside Markdown tables, so if you need separate paragraphs inside a cell, use `<br/><br/>`.

HTML Tables

If you need a more sophisticated table syntax, use HTML syntax for the table. Although you're using HTML, you can use Markdown inside the table cells by adding `markdown="span"` as an attribute for the `td` tag, as shown in the following table. You can also control the column widths.

```

<table>
<colgroup>
<col width="30%" />
<col width="70%" />
</colgroup>
<thead>
<tr class="header">
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td markdown="span">First column fields</td>
<td markdown="span">Some descriptive text. This is a markdown link to [Google](http://google.com). Or see [some link][mydoc_tags].</td>
</tr>
<tr>
<td markdown="span">Second column fields</td>
<td markdown="span">Some more descriptive text.
</td>
</tr>
</tbody>
</table>

```

Result:

Field	Description
First column fields	Some descriptive text. This is a markdown link to Google . Or see some link (page 271) .
Second column fields	Some more descriptive text.

jQuery DataTables

You also have the option of using a [jQuery DataTable](#) , which gives you some additional capabilities. To use a jQuery DataTable in a page, include `datatable: true` in a page's frontmatter. This tells the default layout to load the necessary CSS and javascript bits and to include a `$(document).ready()` function that initializes the DataTables library.

You can change the options used to initialize the DataTables library by editing the call to `$('#table.display').DataTable()` in the default layout. The available options for Datatables are described in the [DataTable documentation](#), which is excellent.

You also must add a class of `display` to your tables. You can change the class, but then you'll need to change the trigger defined in the `$(document).ready()` function in the default layout from `table.display` to the class you prefer.

You can also add page-specific triggers (by copying the `<script></script>` block from the default layout into the page) and classes, which lets you use different options on different tables.

If you use an HTML table, adding `class="display"` to the `<table>` tag is sufficient.

Markdown, however, doesn't allow you to add classes to tables, so you'll need to use a trick: add `<div class="datatable-begin"></div>` before the table and `<div class="datatable-end"></div>` after the table. The default layout includes a jQuery snippet that automagically adds the `display` class to any table it finds between those two markers. So you can start with this (we've trimmed the descriptions for display):

```
<div class="datatable-begin"></div>
```

Food sample type	Description	Category	Sample
-----	-----	-----	-----
Apples	A small, somewhat round ...	Fruit	Fuji
Bananas	A long and curved, often-yellow ...	Fruit	Snow
Kiwis	A small, hairy-skinned sweet ...	Fruit	Golden
Oranges	A spherical, orange-colored sweet ...	Fruit	Navel

```
<div class="datatable-end"></div>
```

and get this:

Food	Description	Category	Sample type
Apples	A small, somewhat round and often red-colored, crispy fruit grown on trees.	Fruit	Fuji
Bananas	A long and curved, often-yellow, sweet and soft fruit that grows in bunches in tropical climates.	Fruit	Snow
Kiwis	A small, hairy-skinned sweet fruit with green-colored insides and seeds.	Fruit	Golden
Oranges	A spherical, orange-colored sweet fruit commonly grown in Florida and California.	Fruit	Navel

Notice a few features:

- You can keyword search the table. When you type a word, the table filters to match your word.
- You can sort the column order.
- You can page the results so that you show only a certain number of values on the first page and then require users to click next to see more entries.

Read more of the [DataTable documentation](#) to get a sense of the options you can configure. You should probably only use DataTables when you have long, massive tables full of information.

❗ Note: Try to keep the columns to 3 or 4 columns only. If you add 5+ columns, your table may create horizontal scrolling with the theme. Additionally, keep the column heading titles short.

Syntax highlighting

Summary: You can apply syntax highlighting to your code. This theme uses pygments and applies color coding based on the lexer you specify.

About syntax highlighting

For syntax highlighting, use fenced code blocks optionally followed by the language syntax you want:

```
```java
import java.util.Scanner;

public class ScannerAndKeyboard
{
 public static void main(String[] args)
 {
 Scanner s = new Scanner(System.in);
 System.out.print("Enter your name: ");
 String name = s.nextLine();
 System.out.println("Hello " + name + "!");
 }
}
```
```

This looks as follows:

```
import java.util.Scanner;

public class ScannerAndKeyboard
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.print( "Enter your name: " );
        String name = s.nextLine();
        System.out.println( "Hello " + name + "!" );
    }
}
```

Fenced code blocks require a blank line before and after.

If you're using an HTML file, you can also use the `highlight` command with Liquid markup.

```
{% highlight java %}
import java.util.Scanner;

public class ScannerAndKeyboard
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.print( "Enter your name: " );
        String name = s.nextLine();
        System.out.println( "Hello " + name + "!" );
    }
}
{% endhighlight %}
```

Result:

```
import java.util.Scanner;

public class ScannerAndKeyboard
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.print( "Enter your name: " );
        String name = s.nextLine();
        System.out.println( "Hello " + name + "!" );
    }
}
```

The theme has syntax highlighting specified in the configuration file as follows:

```
highlighter: rouge
```

The syntax highlighting is done via the `css/syntax.css` file.

Available lexers

The keywords you must add to specify the highlighting (in the previous example, `ruby`) are called “lexers.” You can search for “lexers.” Here are some common ones I use:

- js
- html
- yaml
- css
- json
- php
- java
- cpp
- dotnet
- xml
- http

Workflow maps

Summary: Version 6.0 of the Documentation theme for Jekyll reverts back to relative links so you can view the files offline. Additionally, you can store pages in subdirectories. Templates for alerts and images are available.

Workflow maps overview

You can implement workflow maps at the top of your pages. This is helpful if you're describing a process that involves multiple topics. See the following demos:

- [Simple workflow maps \(page 0\)](#)
- [Complex workflow maps \(page 0\)](#)

Simple workflow maps

1. Create an include at `_includes/custom/usermap.html`, where `usermap.html` contains the workflow and links you want. See the `usermap.html` as an example. It should look something like this:

```
<div id="userMap">
<div class="content"><a href="p2_sample1.html"><div clas
s="box box1">Connect to ADB</div></a></div>
<div class="arrow">→</div>
<div class="content"><a href="p2_sample2.html"><div clas
s="box box2">Download and Build the Starter Kit</di
v></a></div>
<div class="arrow">→</div>
<div class="content"><a href="p2_sample3.html"><div clas
s="box box3">Take a Tour</div></a></div>
<div class="arrow">→</div>
<div class="content"><a href="p2_sample4.html"><div clas
s="box box4">Load Your Widgets</div></a></div>
<div class="arrow">→</div>
<div class="content"><a href="p2_sample5.html"><div clas
s="box box5">Query for Something</div></a></div>
<div class="clearfix"></div>
</div>
```

You can have only 5 possible workflow squares across. Also, the links must be manually coded HTML like those shown, not automated Markdown links. (This is because the boxes are linked.)

2. Where you want the user maps to appear, add the sidebar properties shown in red below:

```
---
title: Sample 1 Topic
keywords: sample
summary: "This is just a sample topic..."
sidebar: product2_sidebar
permalink: p2_sample1
folder: product2
simple_map: true
map_name: usermap
box_number: 1
---
```

In the page.html layout, the following code gets activated when `simple_map` equals `true`:

```
{% if page.simple_map == true %}

<script>
  $(document).ready ( function(){
    $('.box{{page.box_number}}').addClass('active');
  });
</script>

{% include custom/{{page.map_name}}.html %}

{% endif %}
```

The script adds an `active` class to the box number, which automatically makes the active workflow box become highlighted based on the page you're viewing.

The `map_name` gets used as the name of the included file.

Complex workflow maps

The simpler user workflow allows for 5 workflow steps. If you have a more complex workflow, with multiple possible steps, branching, and more, consider using a complex workflow map. This map uses modals to show a list of instructions and links for each step.

1. Create an include at `_includes/custom/usermapcomplex.html`, where `usermapcomplex.html` contains the workflow and links you want. See the `usermapcomplex.html` as an example. The code in that file simply implements Bootstrap modals to create the pop-up boxes. Add your custom content inside the modal body:

```
<div class="modal-body">
<p>This is just dummy text ... Your first steps should be
to get started. You will need to do the following:</p>

  <ul>
    <li><a href="p2_sample6.html">Sample 6</a></li>
    <li><a href="p2_sample7.html">Sample 7</a></li>
    <li><a href="p2_sample8.html">Sample 8</a></li>
  </ul>
  <p>If you run into any of these setup issues, you must
  solve them before you can continue on.</p>

</div>
```

The existing `usermapcomplex.html` file just has 3 workflow square modals. If you need more, duplicate the modal code. In the duplicated code, make sure you make the following values in red unique (but the same within the same modal):

```
<button type="button" class="btn btn-default btn-lg modalButton3" data-toggle="modal" data-target="#myModal3">Publish your app</button>

  <div class="modal fade" id="myModal3" tabindex="-1" role="dialog" aria-labelledby="myModalLabel">
```

2. For each topic where you want the modal to appear, insert the following properties in your frontmatter:

```
---
title: Sample 6 Topic
keywords: sample
summary: "This is just a sample topic..."
sidebar: product2_sidebar
permalink: p2_sample6
complex_map: true
map_name: usermapcomplex
box_number: 1
toc: false
folder: product2
---
```

When your frontmatter contains `complex_map` equal to `true`, the following code gets activated in the `page.layout.html` file:

In the `page.html` layout, the following code gets activated when ``map`` equals ``true``:

```
{% if page.complex_map == true %}

<script>
  $(document).ready ( function(){
    $('.modalButton{{page.box_number}}').addClass('active');
  });
</script>

{% include custom/{{page.map_name}}.html %}

{% endif %}
```
```

# Commenting on files

**Summary:** You can add a button to your pages that allows people to add comments.

## About the review process

If you're using the doc as code approach, you might also consider using the same techniques for reviewing the doc as people use in reviewing code. This approach will involve using Github to edit the files.

There's an Edit me button on each page on this theme. This button allows collaborators to edit the content on Github.

Here's the code for that button on the page.html layout for GitHub:

```
{% if site.github_editme_path %}

<i class="fa fa-github fa-lg"></i> Edit me

{% endif %}
```

and here for GitLab:

```
{% if site.gitlab_editme_path %}

<i class="fa fa-gitlab fa-lg"></i> Edit me

{% endif %}
```

In your configuration file, edit the value for `github_editme_path` (or for Gitlab: `gitlab_editme_path`). For example, you might create a branch called “reviews” on your Github repo. Then you would add something like this in your configuration

file for the 'github\_editme\_path': tomjoht/documentation-theme-jekyll/edit/reviews. Here "tomjoht" is my github account name. The repo name is "documentation-theme-jekyll". The "reviews" name is the branch.

To suppress this button, comment out the `github_editme_path` in the `_config.yml` file.

## Add reviewers as collaborators

If you want people to collaborate on your project so that their edits get committed to a branch on your project, you need to add them as collaborators. For your Github repo, click **Settings** and add the collaborators on the Collaborators tab using their Github usernames.

If you don't want to allow anyone to commit to your Github branch, don't add the reviewers as collaborators. When someone makes an edit, Github will fork the theme. The person's edit then will appear as a pull request to your repo. You can then choose to merge the change indicated in the pull or not.

**Note:** When you process pull requests, you have to accept everything or nothing. You can't pick and choose which changes you'll merge. Therefore you'll probably want to edit the branch you're planning to merge or ask the contributor to make some changes to the fork before processing the pull request.

## Workflow

Users will make edits in your "reviews" branch (or whatever you want to call it). You can then commit those edits as you make updates.

When you're finished making all updates in the branch, you can merge the branch into the master.

Note that if you're making updates online, those updates will be out of sync with any local edits.

**Warning:** Don't make edits both online using Github's browser-based interface AND offline on your local machine using your local tools. When you try to push from your local, you'll likely get a merge conflict error. Instead, make sure you do a pull and update on your local after making any edits online.

## Prose.io

Prose.io is an overlay on Github that would allow people to make comments in an easier interface. If you simply go to [prose.io](https://prose.io), it asks to authorize your Github account, and so it will read files directly from Github but in the Prose.io interface.

## Build arguments

**Summary:** You use various build arguments with your Jekyll project. You can also create shell scripts to act as shortcuts for long build commands. You can store the commands in iTerm as profiles as well.

### How to build Jekyll sites

The normal way to build the Jekyll site is through the build command:

```
jeekyll build
```

To build the site and view it in a live server so that Jekyll rebuilds that site each time you make a change, use the `serve` command:

```
jeekyll serve
```

By default, the `_config.yml` in the root directory will be used, Jekyll will scan the current directory for files, and the folder `_site` will be used as the output. You can customize these build commands like this:

```
jeekyll serve --config configs/myspecialconfig.yml --destination ../doc_outputs
```

Here the `configs/myspecialconfig.yml` file is used instead of `_config.yml`. The destination directory is `../doc_outputs`, which would be one level up from your current directory.

### Shortcuts for the build arguments

If you have a long build argument and don't want to enter it every time in Jekyll, noting all your configuration details, you can create a shell script and then just run the script. Simply put the build argument into a text file and save it with the `.sh` extension (for Mac) or `.bat` extension (for Windows). Then run it like this:



```
. myscript.sh
```

My preference is to add the scripts to profiles in iTerm. See [iTerm Profiles \(page 355\)](#) for more details.

## Stop a server

When you're done with the preview server, press **Ctrl+C** to exit out of it. If you exit iTerm or Terminal without shutting down the server, the next time you build your site, or if you build multiple sites with the same port, you may get a server-already-in-use message.

You can kill the server process using these commands:

```
ps aux | grep jekyll
```

Find the PID (for example, it looks like “22298”).

Then type `kill -9 22298` where “22298” is the PID.

To kill all Jekyll instances, use this:

```
kill -9 $(ps aux | grep '[j]ekyll' | awk '{print $2}')
```

I recommend creating a profile in iTerm that stores this command. Here's what the iTerm settings look like:

The screenshot shows the 'General' tab of the iTerm2 settings window. The profile name is 'Kill all Jekyll'. The shortcut key is set to ^⌘. The tags field contains 'Example: linux, dark bg, tall window'. Under the 'Command' section, 'Login shell' is selected. The 'Send text at start' field contains the command 'kill -9 \$(ps aux | grep '[j]ekyll' | awk '{print \$2}')'. Under the 'Working Directory' section, 'Directory:' is selected with the path '/Users/tjohnson/projects/docs'. The 'URL Schemes' section shows 'Schemes handled:' set to 'Select URL Schemes...'.

**General** Colors Text Window Terminal Session Keys Advanced

**Basics**

Name: Kill all Jekyll

Shortcut key: ^⌘

Tags: Example: linux, dark bg, tall window

**Command**

☒ Login shell

☐ Command:

Send text at start: kill -9 \$(ps aux | grep '[j]ekyll' | awk '{print \$2}')

**Working Directory**

☐ Home directory

☐ Reuse previous session's directory

☒ Directory: /Users/tjohnson/projects/docs

☐ Advanced Configuration [Edit...](#)

**URL Schemes**

Schemes handled: Select URL Schemes...

*iTerm profile settings to kill all Jekyll*

# Themes

**Summary:** You can choose between two different themes (one green, the other blue) for your projects. The theme CSS is stored in the CSS folder and configured in the configuration file for each project.

**Note:** The [gem-based theme](#) approach is not yet integrated into this theme.

## Theme options

You can choose a green or blue theme, or you can create your own. In the css folder, there are two theme files: theme-blue.css and theme-green.css. These files have the most common CSS elements extracted in their own CSS file. Just change the hex colors to the ones you want.

In the \_includes/head.html file, specify the theme file you want the output to use — for example, `theme_file: theme-green.css`. See this line:

```
<link rel="stylesheet" href="css/theme-green.css" />
```

## Theme differences

The differences between the themes is fairly minimal. The main navigation bar, sidebar, buttons, and heading colors change color. That's about it.

In a more sophisticated theming approach, you could use Sass files to generate rules based on options set in a data file, but I kept things simple here.

# Generating PDFs

**Summary:** You can generate a PDF from your Jekyll project. You do this by creating a web version of your project that is printer friendly. You then use utility called Prince to iterate through the pages and create a PDF from them. It works quite well and gives you complete control to customize the PDF output through CSS, including page directives and dynamic tags from Prince.

## PDF overview

This process for creating a PDF relies on Prince XML to transform the HTML content into PDF. Prince costs about \$500 per license. That might seem like a lot, but if you're creating a PDF, you're probably working for a company that sells a product, so you likely have access to some resources. There's also a free license that prints a small "P" watermark on your title page, so if you're fine with that, great.

The basic approach is to generate a list of all web pages that need to be added to the PDF, and then add leverage Prince to package them up into a PDF. Once you set it up, building a pdf is just a matter of running a couple of commands. Also, creating a PDF this way gives you a lot more control and customization capabilities than with other methods for creating PDFs. If you know CSS, you can entirely customize the output.

## Demo

You can see an example of the finished product here:

 PDF Download

To generate the PDF, browse to the theme's directory in your terminal and run this script:

```
• pdf-mydoc.sh
```

This builds a PDF for the documentation in the theme. Look in the **pdf** folder for the output, and see the "last generated date" to confirm that you generated the PDF.

To build a PDF for the other sample projects, run these commands:

```
• pdf-product1.sh
```

or

```
• pdf-product2.sh
```

You can see the details of the script in these files in the theme's root directory. For example, open pdf-mydoc.sh. It contains the following:

```
Note that .sh scripts work only on Mac. If you're on Window
s, install Git Bash and use that as your client.

echo 'Kill all Jekyll instances'
kill -9 $(ps aux | grep '[j]ekyll' | awk '{print $2}')
clear

echo "Building PDF-friendly HTML site for Mydoc ...";
bundle exec jekyll serve --detach --config _config.yml,pdfconfi
gs/config_mydoc_pdf.yml;
echo "done";

echo "Building the PDF ...";
prince --javascript --input-list=_site/pdfconfigs/prince-list.t
xt -o pdf/mydoc.pdf;

echo "Done. Look in the pdf directory to see if it printed succ
essfully."
```

After stopping all Jekyll instances, we build Jekyll using a special configuration file that specifies a unique stylesheet. The build contains a file (prince-list.txt) that contains a list of all pages to be included in the PDF. We feed this list into a Prince command to build the PDF.

The following sections explain more about the setup.

## 1. Set up Prince

Download and install [Prince](#).

You can install a fully functional trial version. The only difference is that the title page will have a small Prince PDF watermark.

## 2. Create a new configuration file for each of your PDF targets

The PDF configuration file will build on the settings in the regular configuration file but will have some additional fields. Here's the configuration file for the mydoc product within this theme. This configuration file is located in the pdfconfigs folder.

```
destination: _site/
url: "http://127.0.0.1:4010"
baseurl: "/mydoc-pdf"
port: 4010
output: pdf
product: mydoc
print_title: Jekyll theme for documentation – mydoc product
print_subtitle: version 5.0
output: pdf
defaults:
 -
 scope:
 path: ""
 type: "pages"
 values:
 layout: "page_print"
 comments: true
 search: true

pdf_sidebar: mydoc_sidebar
```

**Note:** Although you're creating a PDF, you must still build an HTML web target before running Prince. Prince will pull from the HTML files and from the file-list for the TOC.

Note that the default page layout specified by this configuration file is `page_print`. This layout strips out all the sections that shouldn't appear in the print PDF, such as the sidebar and top navigation bar.

Also note that there's a `output: pdf` property in case you want to make some of your content unique to PDF output. For example, you could add conditional logic that checks whether `site.output` is `pdf` or `web`. If it's `pdf`, then include information only for the PDF, and so on. If you're using nav tabs, you'll definitely want to create an alternative experience in the PDF.

In the configuration file, customize the values for the `print_title` and `print_subtitle` that you want. These will appear on the title page of the PDF.

We will access this configure file in the PDF generation script.

### 3. Make sure your sidebar data file has titlepage.html and tocpage.html entries

There are two template pages in the root directory that are critical to the PDF:

- titlepage.html
- tocpage.html

These pages should appear in your sidebar YML file (in this product, `mydoc_sidebar.yml`):

```
- title:
 output: pdf
 type: frontmatter
 folderitems:
 - title:
 url: /titlepage.html
 output: pdf
 type: frontmatter
 - title:
 url: /tocpage.html
 output: pdf
 type: frontmatter
```

Leave these pages here in your sidebar. (The `output: pdf` property means they won't appear in your online TOC because the conditional logic of the sidebar.html checks whether `web` is equal to `pdf` or not before including the item in the web version of the content.)

The code in the tocpage.html is mostly identical to that of the sidebar.html page. This is essential for Prince to create the page numbers correctly with cross references.

There's another file (in the root directory of the theme) that is critical to the PDF generation process: `prince-list.txt`. This file simply iterates through the items in your sidebar and creates a list of links. Prince will consume the list of links from `prince-list.txt` and create a running PDF that contains all of the pages listed, with appropriate cross references and styling for them all.

**Note:** If you have any files that you do not want to appear in the PDF, add `output: web` (rather than `output: pdf`) in the list of attributes in your sidebar. The `prince-list.txt` file that loops through the `mydoc_sidebar.yml` file to grab the URLs of each page that should appear in the PDF will skip over any items that do not list `output: pdf` in the item attributes. For example, you might not want your tag archives to appear in the PDF, but you probably will want to list them in the online help navigation.

## 4. Customize your headers and footers

Open up the `css/printstyles.css` file and customize what you want for the headers and footers. At the very least, customize the email address ( `youremail@domain.com` ) that appears in the bottom left.

Exactly how the print styling works here is pretty nifty. You don't need to understand the rest of the content in this section unless you want to customize your PDFs to look different from what I've configured. But I'm adding this information here in case you want to understand how to customize the look and feel of the PDF output.

This style creates a page reference for a link:

```
a[href]::after {
 content: " (page " target-counter(attr(href), page) ")"
}
```

You don't want cross references for any link that doesn't reference another page, so this style specifies that the content after should be blank:

```
a[href*="mailto"]::after, a[data-toggle="tooltip"]::after, a[href].noCrossRef::after {
 content: "";
}
```



✓ **Tip:** If you have a link to a file download, or some other link that shouldn't have a cross reference (such as link used in JavaScript for navtabs or collapsible sections, for example, add `noCrossRef` as a class to the link to avoid having it say "page 0" in the cross reference.

This style specifies that after links to web resources, the URL should be inserted instead of the page number:

```
a[href^="http:"]::after, a[href^="https:"]::after {
 content: " (" attr(href) ")";
}
```

This style sets the page margins:

```
@page {
 margin: 60pt 90pt 60pt 90pt;
 font-family: sans-serif;
 font-style:none;
 color: gray;
}
```

To set a specific style property for a particular page, you have to name the page. This allows Prince to identify the page.

First you add frontmatter to the page that specifies the type. For the `titlepage.html`, here's the frontmatter:

```

type: title

```

For the `tocpage`, here's the frontmatter:

```

type: frontmatter

```

For the index.html page, we have this type tag (among others):

```

type: first_page

```

The default\_print.html layout will change the class of the `body` element based on the type value in the page's frontmatter:

```
<body class="{% if page.type == "title"%}title{% elsif page.type == "frontmatter" %}frontmatter{% elsif page.type == "first_page" %}first_page{% endif %} print">
```

Now in the `css/printstyles.css` file, you can assign a page name based on a specific class:

```
body.title { page: title }
```

This means that for content inside of `body class="title"`, we can style this page in our stylesheet using `@page title`.

Here's how that title page is styled:

```
@page title {
 @top-left {
 content: " ";
 }
 @top-right {
 content: " "
 }
 @bottom-right {
 content: " ";
 }
 @bottom-left {
 content: " ";
 }
}
```

As you can see, we don't have any header or footer content, because it's the title page.

For the `tocpage.html`, which has the `type: frontmatter`, this is specified in the stylesheet:

```
body.frontmatter { page: frontmatter }
body.frontmatter {counter-reset: page 1}

@page frontmatter {
 @top-left {
 content: prince-script(guideName);
 }
 @top-right {
 content: prince-script(datestamp);
 }
 @bottom-right {
 content: counter(page, lower-roman);
 }
 @bottom-left {
 content: "youremail@domain.com"; }
}
```

With `counter(page, lower-roman)`, we reset the page count to 1 so that the title page doesn't start the count. Then we also add some header and footer info. The page numbers start counting in lower-roman numerals.

Finally, for the first page (which doesn't have a specific name), we restart the counting to 1 again and this time use regular numbers.

```
body.first_page {counter-reset: page 1}

h1 { string-set: doctitle content() }

@page {
 @top-left {
 content: string(doctitle);
 font-size: 11px;
 font-style: italic;
 }
 @top-right {
 content: prince-script(datestamp);
 font-size: 11px;
 }

 @bottom-right {
 content: "Page " counter(page);
 font-size: 11px;
 }
 @bottom-left {
 content: prince-script(guideName);
 font-size: 11px;
 }
}
```

You'll see some other items in there such as `prince-script`. This means we're using JavaScript to run some functions to dynamically generate that content. These JavaScript functions are located in the `_includes/head_print.html`:

```
<script>
 Prince.addScriptFunc("datestamp", function() {
 return "PDF last generated: April 13, 2021";
 });
</script>

<script>
 Prince.addScriptFunc("guideName", function() {
 return "Jekyll theme for documentation – mydoc product
User Guide";
 });
</script>
```

There are a couple of Prince functions that are default functions from Prince. This gets the heading title of the page:

```
content: string(doctype);
```

This gets the current page:

```
content: "Page " counter(page);
```

Because the theme uses JavaScript in the CSS, you have to add the `--javascript` tag in the Prince command (detailed later on this page).

## 5. Customize and run the PDF script

Duplicate the `pdf-mydoc.sh` file in the root directory and customize it for your specific configuration files.

```
echo 'Killing all Jekyll instances'
kill -9 $(ps aux | grep '[j]ekyll' | awk '{print $2}')
clear

echo "Building PDF-friendly HTML site for Mydoc ...";
jekyll serve --detach --config _config.yml,pdfconfigs/config_mydoc_pdf.yml;
echo "done";

echo "Building the PDF ...";
prince --javascript --input-list=_site/pdfconfigs/prince-list.txt -o pdf/mydoc.pdf;
echo "done";
```

Note that the first part kills all Jekyll instances. This way you won't try to serve Jekyll at a port that is already occupied.

The `jekyll serve` command serves up the HTML-friendly PDF configurations for our two projects. This web version is where Prince will go to get its content.

The prince script issues a command to the Prince utility. JavaScript is enabled (`--javascript`), and we tell it exactly where to find the list of files (`--input-list`) — just point to the `prince-list.txt` file. Then we tell it where and what to output (`-o`).

Make sure that the path to the `prince-list.txt` is correct. For the output directory, I like to output the PDF file into my project's source (into the `files` folder). Then when I build the web output, the PDF is included and something I can refer to.

**Note:** You might not want to include the PDF in your project files, since you're likely committing the PDF to Github and as a result saving the changes from one PDF version to another with each save.

## 6. Add conditions for your new builds in the PDF config file

In the PDF configuration file, there's a section that looks like this:

```
{% if site.product == "mydoc" %}
pdf_sidebar: product2_sidebar
{% endif %}
```

Point to the sidebar you want here.

What this does is allow the prince-list.txt and toc.html files to iterate through the right sidebar. Otherwise, you would need to create a unique prince-list.txt and toc.html file for each separate PDF output you have.

## 7. Add a download button for the PDF

You can add a download button for your PDF using some Bootstrap button code:

```
<button type="button" class="btn btn-default" aria-label="Left Align"> PDF Download</button>
```

Here's what that looks like:

```

```

 PDF Download </a>

## JavaScript conflicts

If you have JavaScript on any of your pages, Prince will note errors in Terminal like this:

```
error: TypeError: value is not an object
```

However, the PDF will still build.

You need to conditionalize out any JavaScript from your PDF web output before building your PDFs. Make sure that the PDF configuration files have the `output: pdf` property.

Then surround the JavaScript with conditional tags like this:

```
{% raw %}{% unless site.output == "pdf" %}
javascript content here ...
{% endunless %}
```

For more detail about using `unless` in conditional logic, see [Conditional logic \(page 243\)](#). What this code means is “run this code unless this value is the case.”

## Overriding Bootstrap Print Styles

The theme relies on Bootstrap’s CSS for styling. However, for print media, Bootstrap applies the following style:

```
@media print{*,:after,:before{color:#000!important;text-shado
w:none!important;background:0 0!important;-webkit-box-shadow:no
ne!important;box-shadow:none!important}
```

This is minified, but basically the `*` (asterisk) means select all, and applied the color `#000` (black). As a result, the Bootstrap style strips out all color from the PDF (for Bootstrap elements).

This is problematic for code snippets that have syntax highlighting. I decided to remove this de-coloring from the print output. I commented out the Bootstrap style:

```
@media print{*,:after,:before{/*color:#000!important;*/text-sha
dow:none!important;*/background:0 0!important;*/-webkit-box-sha
dow:none!important;box-shadow:none!important}
```

If you update Bootstrap, make sure you make this edit. (Sorry, admittedly I couldn’t figure out how to simply overwrite the `*` selector with a later style.)

I did, however, remove the color from the alerts and lighten the background shading for `pre` elements. The `printstyles.css` has this setting.



# Help APIs and UI tooltips

**Summary:** You can loop through files and generate a JSON file that developers can consume like a help API. Developers can pull in values from the JSON into interface elements, styling them as popovers for user interface text, for example. The beauty of this method is that the UI text remains in the help system (or at least in a single JSON file delivered to the dev team) and isn't hard-coded into the UI.

## Full code demo of content API

You can create a help API that developers can use to pull in content.

For the full code demo, see the notes in the [Tooltips file \(page 0\)](#).

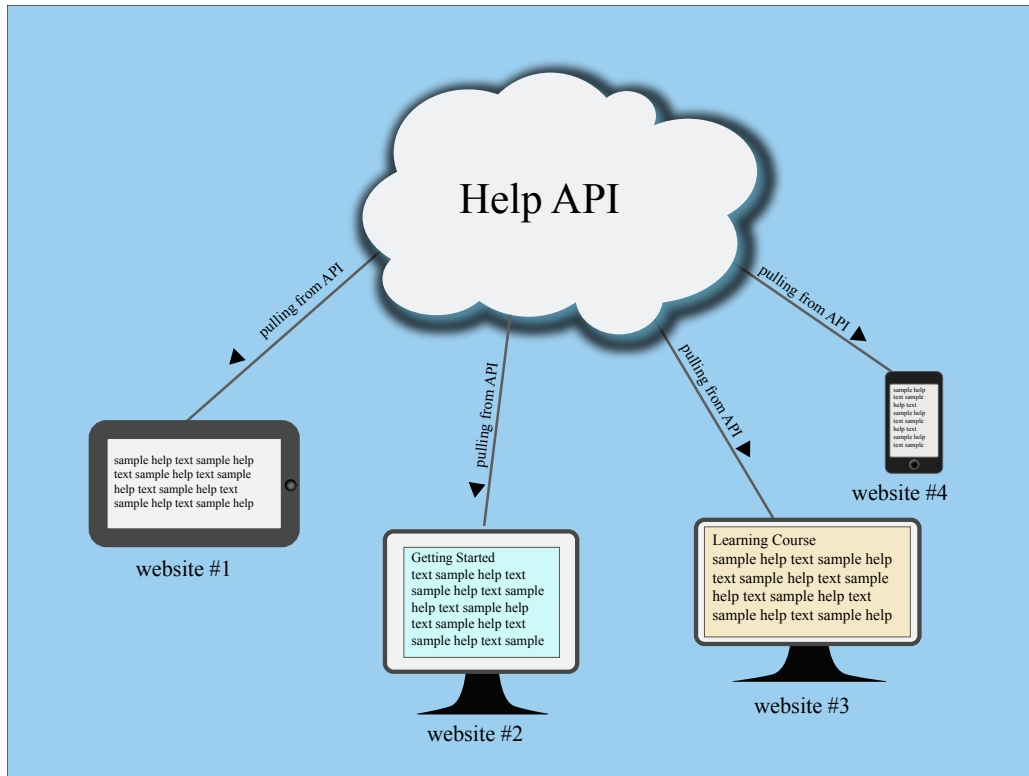
In this demo, the popovers pull in and display content from the information in a [tooltips.json \(page 0\)](#) file located in the same directory.

Instead of placing the JSON source in the same directory, you could also host the JSON file on another site.

Additionally, instead of tooltip popovers, you could also print content directly to the page. Basically, whatever you can stuff into a JSON file, developers can integrate it onto a page.

## Diagram overview

Here's a diagram showing the basic idea of the help API:



Is this really an API? Well, sort of. The help content is pushed out into a JSON file that other websites and applications can easily consume. The endpoints don't deliver different data based on parameters added to a URL. But the overall concept is similar to an API: you have a client requesting resources from a server.

Note that in this scenario, the help is openly accessible on the web. If you have a private system, it's more complicated.

To deliver help this way using Jekyll, follow the steps in each of the sections below.

## 1. Create a “collection” for the help content

A collection is another content type that extends Jekyll beyond the use of pages and posts. Call the collection “tooltips.”

Add the following information to your configuration file to declare your collection:

```
collections:
 tooltips:
 output: false
```

In your Jekyll project's root directory, create a new folder called “\_tooltips” and put every page that you want to be part of that tooltips collection inside that folder.

In Jekyll, folders that begin with an underscore (“\_”) aren't included in the output. However, in the collection information that you add to your configuration file, if you change `output` to `true`, the tooltips folder will appear in the output, and each page inside tooltips will be generated. You most likely don't want this for tooltips (you just want the JSON file), so make the `output` setting `false`.

## 2. Create tooltip definitions in a YAML file

Inside the `_data` folder, create a YAML file called something like `definitions.yml`. Add the definitions for each of your tooltips here like this:

```
basketball: "Basketball is a sport involving two teams of five
players each competing to put a ball through a small circular r
im 10 feet above the ground. Basketball requires players to be
in top physical condition, since they spend most of the game ru
nning back and forth along a 94-foot-long floor."
```

The definition of basketball is stored this data file so that you can re-use it in other parts of the help as well. You'll likely want the definition to appear not only in the tooltip in the UI, but also in the regular documentation as well.

## 3. Create pages in your collection

Create pages inside your new tooltips collection (that is, inside the `_tooltips` folder). Each page needs to have a unique `id` in the frontmatter as well as a `product`. Then reference the definition you created in the `definitions.yml` file.

Here's an example:

```

doc_id: basketball
product: mydoc

{{site.data.definitions.basketball}}
```

(Note: Avoid using `id`, as it seems to generate out as `/tooltips/basketball` instead of just ``basketball``.)

You need to create a separate file for each tooltip you want to deliver.

The product attribute is required in the frontmatter to distinguish the tooltips produced here from the tooltips for other products in the same `_tooltips` folder. When creating the JSON file, Jekyll will iterate through all the pages inside `_tooltips`, regardless of any subfolders included here.

## 4. Create a JSON file that loops through your collection pages

Now it's time to create a JSON file with Liquid code that iterates through our tooltip collection and grabs the information from each tooltip file.

Inside your project's pages directory (e.g., `mydoc`), add a file called "tooltips.json." (You can use whatever name you want.) Add the following to your JSON file:

```

layout: null
search: exclude

{
 "entries":
 [
 {% for page in site.tooltips %}
 {
 "doc_id": "{{ page.doc_id }}",
 "body": "{{ page.content | strip_newlines | replace: '\n',
 '\\\\n' | replace: '\"', '\\\"' }}"
 } {% unless forloop.last %},{% endunless %}
 {% endfor %}
]
}
```

This code will loop through all pages in the `tooltips` collection and insert the `id` and `body` into key-value pairs for the JSON code. Here's an example of what that looks like after it's processed by Jekyll in the site build:

```
{
 "entries": [
 {
 "doc_id": "baseball",
 "body": "Baseball is considered America's pasttime sport, though that may be more of a historical term than a current one. There's a lot more excitement about football than baseball. A baseball game is somewhat of a snooze to watch, for the most part."
 },
 {
 "doc_id": "basketball",
 "body": "Basketball is a sport involving two teams of five players each competing to put a ball through a small circular rim 10 feet above the ground. Basketball requires players to be in top physical condition, since they spend most of the game running back and forth along a 94-foot-long floor."
 },
 {
 "doc_id": "football",
 "body": "No doubt the most fun sport to watch, football also manages to accrue the most injuries with the players. From concussions to blown knees, football players have short sport lives."
 },
 {
 "doc_id": "soccer",
 "body": "If there's one sport that dominates the world landscape, it's soccer. However, US soccer fans are few and far between. Apart from the popularity of soccer during the World Cup, most people don't even know the name of the professional soccer organization in their area."
 }
]
}
```

You can also view the same JSON file here: [tooltips.json \(page 0\)](#).

You can add different fields depending on how you want the JSON to be structured. Here we just have two fields: `doc_id` and `body`. And the JSON is looking just in the `tooltips` collection that we created.

✓ **Tip:** Check out [Google's style guide](#) for JSON. These best practices can help you keep your JSON file valid.

Note that you can create different JSON files that specialize in different content. For example, suppose you have some getting started information. You could put that into a different JSON file. Using the same structure, you might add an `if` tag that checks whether the page has frontmatter that says `type: getting_started` or something. Or you could put the content into separate collection entirely (different from tooltips).

By chunking up your JSON files, you can provide a quicker lookup. (I'm not sure how big the JSON file can be before you experience any latency with the jQuery lookup.)

## 5. Build your site and look for the JSON file

When you build your site, Jekyll will iterate through every page in your `_tooltips` folder and put the page id and body into this format. In the output, look for the JSON file in the `tooltips.json` file. You'll see that Jekyll has populated it with content. This is because of the triple hyphen lines in the JSON file — this instructs Jekyll to process the file.

## 6. Allow CORS access to your help if stored on a remote server

You can simply deliver the JSON file to devs to add to the project. But if you have the option, it's best to keep the JSON file stored in your own help system. Assuming you have the ability to update your content on the fly, this will give you completely control over the tooltips without being tied to a specific release window.

When people make calls to your site *from other domains*, you must allow them access to get the content. To do this, you have to enable something called CORS (cross origin resource sharing) within the server where your help resides.

In other words, people are going to be executing calls to reach into your site and grab your content. Just like the door on your house, you have to unlock it so people can get in. Enabling CORS is unlocking it.

How you enable CORS depends on the type of server.

If your server setup allows `htaccess` files to override general server permissions, create an `.htaccess` file and add the following:

```
Header set Access-Control-Allow-Origin "*"
```

Store this in the same directory as your project. This is what I've done in a directory on my web host (bluehost.com). Inside `http://idratherassets.com/wp-content/apidemos/`, I uploaded a file called ".htaccess" with the preceding code.

After I uploaded it, I renamed it to .htaccess, right-clicked the file and set the permissions to 774.

To test whether your server permissions are set correctly, open a terminal and run the following curl command pointing to your tooltips.json file:

```
curl -I http://idratherassets.com/wp-content/apidemos/tooltips.json
```

The `-I` command tells cURL to return the request header only.

If the server permissions are set correctly, you should see the following line somewhere in the response:

```
Access-Control-Allow-Origin: *
```

If you don't see this response, CORS isn't allowed for the file.

If you have an AWS S3 bucket, you can supposedly add a CORS configuration to the bucket permissions. Log into AWS S3 and click your bucket. On the right, in the Permissions section, click **Add CORS Configuration**. In that space, add the following policy:

```
<CORSConfiguration>
 <CORSRule>
 <AllowedOrigin>*</AllowedOrigin>
 <AllowedMethod>GET</AllowedMethod>
 </CORSRule>
</CORSConfiguration>
```

(Although this should work, in my experiment it doesn't. And I'm not sure why...)

In other server setups, you may need to edit one of your Apache configuration files. See [Enable CORS](#) or search online for ways to allow CORS for your server.

If you don't have CORS enabled, users will see a CORS error/warning message in the console of the page making the request.

✓ **Tip:** If enabling CORS is problematic, you could always just send developers the tooltips.json file and ask them to place it on their own server.

## 7. Explain how developers can access the help

Developers can access the help using the `.get` method from jQuery, among other methods. Here's an example of how to get tooltips for basketball, baseball, football, and soccer:

```
var url = "tooltips.json";

$.get(url, function(data) {

 /* Bootstrap popover text is defined inside a data-content attribute inside an element. That's why I'm using attr here. If you just want to insert content on the page, use append and remove the data-content argument from the parentheses.*/

 $.each(data.entries, function(i, page) {
 if (page.doc_id == "basketball") {
 $("#basketball").attr("data-content", page.body);
 }

 if (page.doc_id == "baseball") {
 $("#baseball").attr("data-content", page.body);
 }

 if (page.doc_id == "football") {
 $("#football").attr("data-content", page.body);
 }

 if (page.doc_id == "soccer") {
 $("#soccer").attr("data-content", page.body);
 }

 });
});
```



View the [tooltip demo](#) for a demonstration. See the source code for full code details.

The `url` in the demo is relative, but you could equally point it to an absolute path on a remote host assuming CORS is enabled on the host.

The `each` method looks through all the JSON content to find the item whose `page.id` is equal to `basketball`. It then looks for an element on the page named `#basketball` and adds a `data-content` attribute to that element.

**⚠ Warning:** Make sure your JSON file is valid. Otherwise, this method won't work. I use the [JSON Formatter extension for Chrome](#). When I go to the `tooltips.json` page in my browser, the JSON content — if valid — is nicely formatted (and includes some color coding). If the file isn't valid, it's not formatted and there isn't any color. You can also check the JSON formatting using [JSON Formatter and Validator](#). If your JSON file isn't valid, identify the problem area using the validator and troubleshoot the file causing issues. It's usually due to some code that isn't escaping correctly.

Why `data-content`? Well, in this case, I'm using [Bootstrap popovers](#) to display the tooltip content. The `data-content` attribute is how Bootstrap injects popovers.

Here's the section on the page where the popover is inserted:

```
<p>Basketball </p>
```

Notice that I just have `id="basketball"` added to this popover element. Developers merely need to add a unique ID to each tooltip they want to pull in the help content. Either you tell developers the unique ID they should add, or ask them what IDs they added (or just tell them to use an ID that matches the field's name).

In order to use jQuery and Bootstrap, you'll need to add the appropriate references in the head tags of your page:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/js/bootstrap.min.js"></script>

<script type="text/javascript">
$(document).ready(function(){
 $('[data-toggle="popover"]').popover({
 placement : 'right',
 trigger: 'hover',
 html: true
 });
</script>
```

Again, see the [Tooltip Demo](#) for a demo of the full code.

Note that even though you reference a Bootstrap JS script, Bootstrap's popovers require you to initialize them using the above code as well — they aren't turned on by default.

View the source code of the [tooltip demo](#) for the full comments.

## 8. Create easy links to embed the help in your help site

You might also want to insert the same content into different parts of your help site. For example, if you have tooltips providing definitions for fields, you'll probably want to create a page in your help that lists those same definitions.

You could use the same method developers use to pull help content into their applications. But it will probably be easier to simply use Jekyll's tags for doing it.

Here's how you would reuse the content:

```
<h2>Reuse Demo</h2>

<table>
<thead>
<tr>
<th>Sport</th>
<th>Comments</th>
</tr>
</thead>
<tbody>

<tr>
<td>Basketball</td>
<td>{{site.data.definitions.basketball}}</td>
</tr>

<tr>
<td>Baseball</td>
<td>{{site.data.definitions.baseball}}</td>
</tr>

<tr>
<td>Football</td>
<td>{{site.data.definitions.football}}</td>
</tr>

<tr>
<td>Soccer</td>
<td>{{site.data.definitions.soccer}}</td>
</tr>
</tbody>
</table>
```

And here's the code:

## Reuse Demo

Sport	Comments
Basketball	Basketball is a sport involving two teams of five players each competing to put a ball through a small circular rim 10 feet above the ground. Basketball requires players to be in top physical condition, since they spend most of the game running back and forth along a 94-foot-long floor.
Baseball	Baseball is considered America's pasttime sport, though that may be more of a historical term than a current one. There's a lot more excitement about football than baseball. A baseball game is somewhat of a snooze to watch, for the most part.
Football	No doubt the most fun sport to watch, football also manages to accrue the most injuries with the players. From concussions to blown knees, football players have short sport lives.
Soccer	If there's one sport that dominates the world landscape, it's soccer. However, US soccer fans are few and far between. Apart from the popularity of soccer during the World Cup, most people don't even know the name of the professional soccer organization in their area.

Now you have both documentation and UI tooltips generated from the same definitions file.

# Search configuration

**Summary:** The search feature uses JavaScript to look for keyword matches in a JSON file. The results show instant matches, but it doesn't provide a search results page like Google. Also, sometimes invalid formatting can break the JSON file.

## About search

The search is configured through the search.json file in the root directory. The search is a simple search that looks at content in pages. It looks at titles, summaries, keywords, and tags.

However, the search doesn't work like google — you can't hit return and see a list of results on the search results page, with the keywords in bold. Instead, this search shows a list of page titles that contain keyword matches. It's fast, but simple.

## Excluding pages from search

By default, every page is included in the search. Depending on the type of content you're including, you may find that some pages will break the JSON formatting. If that happens, then the search will no longer work.

If you want to exclude a page from search add `search: exclude` in the page's frontmatter.

## Troubleshooting search

You should exclude any files from search that you don't want appearing in the search results. For example, if you have a tooltips.json file or prince-list.txt, don't include it, as the formatting will break the JSON format.

If any formatting in the search.json file is invalid (in the build), search won't work. You'll know that search isn't working if no results appear when you start typing in the search box.

If this happens, point your browser to your build's search.json file at `http://localhost:4000/search.json` (or your public github pages site), copy the entire search.json page as it's output on the browser, and then use a [JSON validator](#) to validate the output by pasting what you copied into the validator. Look for the line causing trouble. It will be highlighted. Edit the file that's causing

the problem to either exclude it from the search or fix the syntax so that it doesn't invalidate the JSON. (Note that tabs in the body will invalidate JSON, as will certain characters in the file's front matter. For example, the summary: cannot have " quotes in it.

The search.json file already tries to strip out content that would otherwise make the JSON invalid.

## Including the body field in search

I've found that including the `body` field in the search creates too many problems, and so I've removed `body` from the search. You can see the results of including the `body` by adding this along with the other fields in search.json:

```
"body": "{ { page.content | strip_html | strip_newlines |
replace: '\\', '\\\\' | replace: '\"', '\\\"' | replace: '
, ' ' } }",
```

Note that the last replace, `| replace: '^t', ' '`, looks for any tab character and replaces it with four spaces. (Tab characters invalidate JSON.) If you run into other problematic formatting, you can use regex expressions to find and replace the content. See [Regular Expressions](#) for details on finding and replacing code.

It's possible that the formatting may not account for all the scenarios that would invalidate the JSON. (Sometimes it's an extra comma after the last item that makes it invalid.)

Note that including the body in the search creates other problems as well. The search results show the most immediate matches in the JSON file. If several topics have matches for the keyword in the body, these matches might appear before other files that have matches in the title, summary, or keywords. This is because this simple search does not provide any weighting mechanisms for the content.

## Customizing search results

At some point, you may want to customize the search results more. Here's a little more detail that will be helpful. The search.json file retrieves various page values:

```

title: search
layout: none
search: exclude

[
{% for page in site.pages %}
{% unless page.search == "exclude" %}
{
"title": "{{ page.title | escape }}",
"tags": "{{ page.tags }}",
"keywords": "{{page.keywords}}",
"url": "{{ page.url | remove: "/" }}",
"summary": "{{page.summary | strip }}"
},
{% endunless %}
{% endfor %}

{% for post in site.posts %}

{
"title": "{{ post.title | escape }}",
"tags": "{{ post.tags }}",
"keywords": "{{post.keywords}}",
"url": "{{ post.url }}",
"summary": "{{post.summary | strip }}"
}
{% unless forloop.last %},{% endunless %}
{% endfor %}

]
```

The `_includes/topnav.html` file then makes use of these values:

```

 <!--start search-->
 <div id="search-demo-container">
 <input type="text" id="search-input" placeholder="查找文档...">
 <ul id="results-container">
 </div>
 <script src="js/jekyll-search.js" type="text/javascript"></script>
 <script type="text/javascript">
 SimpleJekyllSearch.init({
 searchInput: document.getElementById('search-input'),
 resultsContainer: document.getElementById('results-container'),
 dataSource: 'search.json',
 searchResultTemplate: '{title}',
 noResultsText: '找不到相关文档。',
 limit: 10,
 fuzzy: true,
 })
 </script>
 <!--end search-->

```

Where you see `{url}` and `{title}`, the search is retrieving the values for these as specified in the `search.json` file.

## More robust search

Overall, the built-in search only works for small documentation projects. If you have more robust search needs, consider integrating [Google Custom Search](#), [Algolia](#), or [Swifttype](#).



# iTerm profiles

**Summary:** You can set up profiles in iTerm to facilitate the build process with just a few clicks. This can make it a lot easier to quickly build multiple outputs.

## About iTerm profiles

When you're working with tech docs, a lot of times you have builds that push files onto different servers, or that build the content for different environments. It can be a hassle to type out these commands each time. Instead, it's easier to configure iTerm with profiles that initiate the scripts.

## Set up profiles

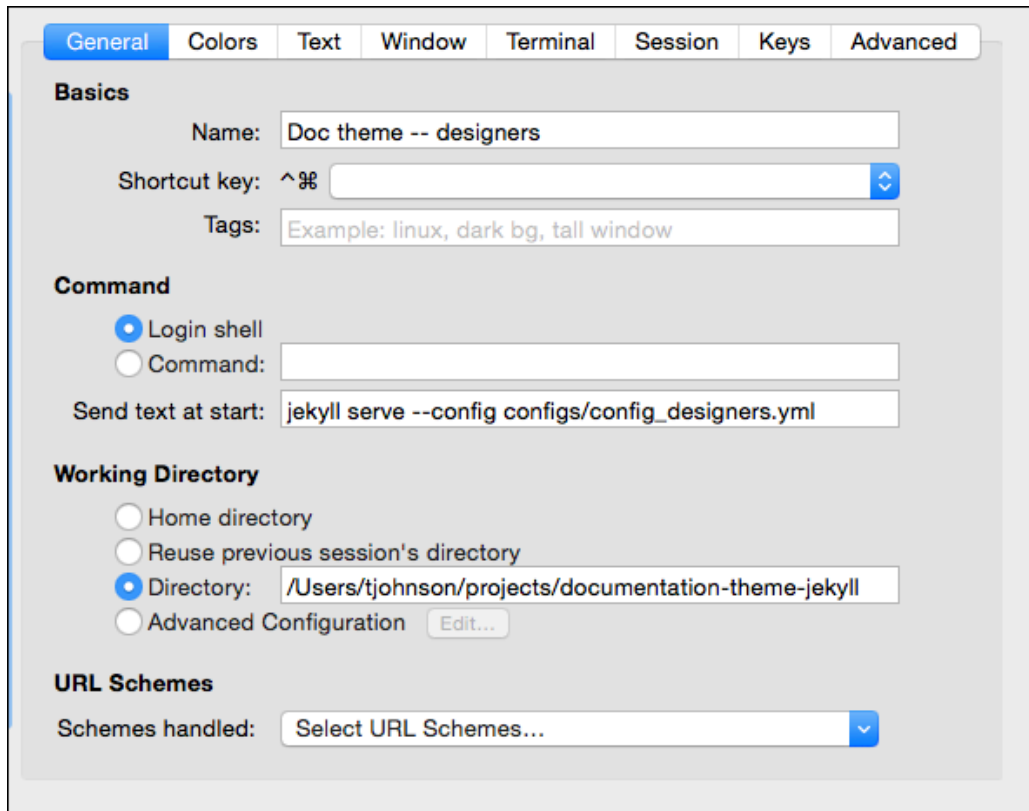
1. Open iTerm and go to **Profiles > Open Profiles**.
2. Click **Edit Profiles**.
3. Click the + button in the lower-left corner to create a new profile.
4. In the **Name** field, type a name describing the output, such as `Doc theme -- designers`.
5. In the **Send text at start** field, type the command for the build script, such as this:

```
JEKYLL_ENV=production jekyll serve
```

Leave the Login shell option selected.

6. In the Working Directory section, select **Directory** and enter the directory for your project, such as `/Users/tjohnson/projects/documentation-theme-jekyll`.
7. Close the profiles panel.

Here's an example:



The screenshot shows the 'General' tab of the iTerm profile configuration window. The tabs at the top are: General (selected), Colors, Text, Window, Terminal, Session, Keys, and Advanced. The 'Basics' section contains: Name: 'Doc theme -- designers', Shortcut key: '^⌘' (with a dropdown arrow), and Tags: 'Example: linux, dark bg, tall window'. The 'Command' section has radio buttons for 'Login shell' (selected) and 'Command:', with a text field below it containing 'jekyll serve --config configs/config\_designers.yml'. The 'Working Directory' section has radio buttons for 'Home directory', 'Reuse previous session's directory', 'Directory:' (selected), and 'Advanced Configuration' (with an 'Edit...' button). The 'Directory:' text field contains '/Users/tjohnson/projects/documentation-theme-jekyll'. The 'URL Schemes' section has a 'Schemes handled:' dropdown menu with 'Select URL Schemes...'.

*iTerm profile example*

## Launching a profile

1. In iTerm, make sure the Toolbar is shown. Go to **View > Toggle Toolbar**.
2. Click the **New** button and select your profile.

✓ **Tip:** When you're done with the session, make sure to click **Ctrl+C**.

# Pushing builds to server

**Summary:** You can push your build to AWS using commands from the command line. By including your copy commands in commands, you can package all of the build and deploy process into executable scripts.

## Pushing to AWS S3

If you have the AWS Command Line Interface installed and are pushing your builds to AWS, the following commands show how you can build and push to an AWS location from the command line:

```
aws s3 cp ~/users/tjohnson/projects/mydocproject/ s3://[aws path]docpath/mydocproject --recursive

aws s3 cp ~/users/tjohnson/projects/anotherdocproject2/ s3://[aws path]docpath/anotherdocproject --recursive
```

The first path in the argument is the local location; the second path is the destination.

## Pushing to a regular server

If you're pushing to a regular server that you can ssh into, you can use `scp` commands to push your build. Here's an example:

```
scp -r /users/tjohnson/projects/mydocproject/ name@domain:/var/www/html/mydocproject
```

Similar to the above, the first path is the local location; the second path is the destination.

# Publishing on Github Pages

**Summary:** You can publish your project on Github Pages, which is a free web hosting service provided by Github. All you need is to put your content into a Github repo branch called gh-pages and make this your default branch in your repo. With a Jekyll site, you just commit your entire project into the gh-pages branch and Github Pages will build the site for you.

## Set up your Github repo

1. Make sure you have Git installed. You can download and install [Git for Windows here](#) and [Git for Mac here](#). If you're on a Mac, chances are you might already have git installed. You can check by opening up a terminal and typing `which git`.
2. Go to [Github.com](#) and sign up for an account.
3. Click the + button in the upper-right corner and select **New repository**.
4. Name the repository something like **mydocthem**.
5. Type a description..
6. Select the **Initialize this repository with a README** check box.
7. Add a license if desired.
8. Leave the other options at the defaults and click **Create repository**.
9. Click the **Settings** button.
10. Go to your repository's home page, and click the branch drop-down menu.
11. Create a new branch called **gh-pages**.
12. Click **Settings** and change the default branch to **gh-pages**.
13. Go back to your repository's homepage. With the gh-pages branch selected, copy the **https clone url**:
14. Open a terminal, browse to a convenient location for your project, and type `git clone https://github.com/tomjoht/myreponame.git`, replacing the `https://github.com/tomjoht/myreponame.git` with your repository's https clone URL that you copied.

15. Move the jekyll theme files into this new folder that you just created in the previous step.
16. Open the `_config.yml` file and add the following:

```
url: tomjoht.github.io
baseurl: /myreponame
```

Change the url to your github account name, and the baseurl to your repo name.

## Install Bundler

Bundler is a package manager for Ruby that will install all dependencies you might need to build your site locally. I recommend installing Bundler through homebrew. (Sorry, these instructions apply to Mac only.)

1. Install [homebrew](#) :

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. Install Bundler:

```
gem install bundler
```

## Add the github pages gem

1. In terminal, browse to your Jekyll project directory.
2. Type `bundle init` . This creates a Gemfile and Gemfile.lock in your project.
3. Type `open gemfile` . This opens the gemfile in your default text editor.
4. Add the following in the gemfile (replacing the existing contents):

```
source 'https://rubygems.org'
gem 'github-pages'
```

5. Run `bundle install`.
6. Add the new jekyll files to git: `git add --all`.
7. Commit the files: `git commit -m "committing my jekyll theme"`.
8. Push the files up to your github repo: `git push`.

Github Pages will now automatically build your site. Wait a minute or two, and then visit `tomjoht.github.io/yourreponame`, replacing this path with your github account and branch.

## Customize your URL

You can also customize your Github URL. More instructions on this later....

## Knowledge-base layout

**Summary:** This shows a sample layout for a knowledge base. Each square could link to a tag archive page. In this example, font icons from Font Awesome are used for the graphics, and the layout is pulled from the Modern Business theme. .

Here's the sample knowledge-base style layout:

### Knowledge Base Categories



*Getting started*

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

[Learn More \(page 0\)](#)



### *Navigation*

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

[Learn More \(page 0\)](#)



### *Single sourcing*

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

[Learn More \(page 0\)](#)



### *Formatting*

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

[Learn More \(page 0\)](#)

## Generating a list of all pages with a certain tag

If you don't want to link to a tag archive index, but instead want to list all pages that have a certain tag, you could use this code:



Getting started pages:

```

{% assign sorted_pages = site.pages | sort: 'title' %}
{% for page in sorted_pages %}
{% for tag in page.tags %}
{% if tag == "getting_started" %}
{{page.titl
e}}
{% endif %}
{% endfor %}
{% endfor %}

```

Here's the result:

Getting started pages:

- [About Ruby, Gems, Bundler, and other prerequisites \(page 214\)](#)
- [About the theme's author \(page 208\)](#)
- [Getting started with the Documentation Theme for Jekyll \(page 0\)](#)
- [Install Jekyll on Mac \(page 222\)](#)
- [Pages \(page 230\)](#)
- [Posts \(page 237\)](#)
- [Release notes 5.0 \(page 212\)](#)
- [Release notes 6.0 \(page 210\)](#)
- [Sidebar Navigation \(page 257\)](#)
- [Support \(page 209\)](#)
- [Supported features \(page 203\)](#)

# Glossary layout

**Summary:** Your glossary page can take advantage of definitions stored in a data file. This gives you the ability to reuse the same definition in multiple places. Additionally, you can use Bootstrap classes to arrange your definition list horizontally.

You can create a glossary for your content. First create your glossary items in a data file such as `glossary.yml`.

Then create a page and use definition list formatting, like this:

**fractious**

Like a little mischevious child, full of annoying and constant trouble.

**gratuitous**

Something that is unwarranted and uncouth, like the social equivalent of a flagrant foul.

**haughty**

Proud and flaunting it. Holding your head high up like a snooty, too-good-for-everything rich person.

**gratuitous**

Something that is unwarranted and uncouth, like the social equivalent of a flagrant foul.

**impertinent**

Brave and courageous especially in a difficult, dangerous situation.

Here's the code:

```
fractious
: {{site.data.glossary.fractious}}

gratuitous
: {{site.data.glossary.gratuitous}}

haughty
: {{site.data.glossary.haughty}}

gratuitous
: {{site.data.glossary.gratuitous}}

impertinent
: {{site.data.glossary.intrepid}}
```

The glossary works well as a link in the top navigation bar.

## Horizontally styled definiton lists

You can also change the definition list ( `dl` ) class to `dl-horizontal` . This is a Bootstrap specific class. If you do, the styling looks like this:

### **fractious**

Like a little mischevious child, full of annoying and constant trouble.

### **gratuitous**

Something that is unwarranted and uncouth, like the social equivalent of a flagrant foul.

### **haughty**

Proud and flaunting it. Holding your head high up like a snooty, too-good-for-everything rich person.

### **gratuitous**

Something that is unwarranted and uncouth, like the social equivalent of a

flagrant foul.

### **impertinent**

Someone acting rude and insensitive to others.

### **intrepid**

Brave and courageous especially in a difficult, dangerous situation.

For this type of list, you must use HTML. The list would then look like this:

```
<dl class="dl-horizontal">

<dt id="fractious">fractious</dt>
<dd>{{site.data.glossary.fractious}}</dd>

<dt id="gratuitous">gratuitous</dt>
<dd>{{site.data.glossary.gratuitous}}</dd>

<dt id="haughty">haughty</dt>
<dd>{{site.data.glossary.haughty}}</dd>

<dt id="benchmark_id">gratuitous</dt>
<dd>{{site.data.glossary.gratuitous}}</dd>

<dt id="impertinent">impertinent</dt>
<dd>{{site.data.glossary.impertinent}}</dd>

<dt id="intrepid">intrepid</dt>
<dd>{{site.data.glossary.intrepid}}</dd>

</dl>
```

If you squish your screen small enough, at a certain breakpoint this style reverts to the regular `dl` class.

Although I like the side-by-side view for shorter definitions, I found it problematic with longer definitions.

## FAQ layout

**Summary:** You can use an accordion-layout that takes advantage of Bootstrap styling. This is useful for an FAQ page.

If you want to use an FAQ format, use the syntax shown on the `faq.html` page. Rather than including code samples here (which are bulky with a lot of nested `div` tags), just look at the source in the `mydoc_faq.html` theme file.

Lorem ipsum dolor sit amet, consectetur adipiscing elit?

Curabitur eget leo at velit imperdiet varius. In eu ipsum vitae velit congue iaculis vitae at risus?

Aenean consequat lorem ut felis ullamcorper?

Lorem ipsum dolor sit amet, consectetur adipiscing elit?

Curabitur eget leo at velit imperdiet varius. In eu ipsum vitae velit congue iaculis vitae at risus?

Aenean consequat lorem ut felis ullamcorper?

Lorem ipsum dolor sit amet, consectetur adipiscing elit?

Curabitur eget leo at velit imperdiet varius. In eu ipsum vitae velit congue iaculis vitae at risus?

Aenean consequat lorem ut felis ullamcorper?

# Shuffle layout

**Summary:** This layout shows an example of a knowledge-base style navigation system, where there is no hierarchy, just groups of pages that have certain tags.

**Note:** The content on this page doesn't display well on PDF, but I included it anyway so you could see the problems this layout poses if you're including it in PDF.

[All](#)[Getting Started](#)[Formatting](#)[Publishing](#)[Content types](#)[Single Sourcing](#)[Special Layouts](#)

## Getting started

If you're getting started with Jekyll, see the links in this section. It will take you from the beginning level to comfortable.

- [Getting started with the Documentation Theme for Jekyll \(page 0\)](#)
- [About the theme's author \(page 208\)](#)
- [About Ruby, Gems, Bundler, and other prerequisites \(page 214\)](#)
- [Install Jekyll on Mac \(page 222\)](#)
- [Pages \(page 230\)](#)
- [Posts \(page 237\)](#)
- [Release notes 5.0 \(page 212\)](#)

## Content types

This section lists different content types and how to work with them.

## Formatting

These topics get into formatting syntax, such as images and tables, that you'll use on each of your pages:

- [Tooltips \(page 280\)](#)
- [Alerts \(page 281\)](#)
- [Code samples \(page 301\)](#)
- [Glossary layout \(page 364\)](#)

- [Release notes 6.0 \(page 210\)](#)
- [Sidebar Navigation \(page 257\)](#)
- [Support \(page 209\)](#)
- [Supported features \(page 203\)](#)

- [Links \(page 303\)](#)
- [Icons \(page 289\)](#)
- [Images \(page 296\)](#)
- [Labels \(page 302\)](#)
- [Lists \(page 239\)](#)
- [Navtabs \(page 304\)](#)
- [Pages \(page 230\)](#)
- [Posts \(page 237\)](#)
- [Syntax highlighting \(page 312\)](#)
- [Tables \(page 308\)](#)
- [Workflow maps \(page 315\)](#)
- [YAML tutorial in the context of Jekyll \(page 260\)](#)

## Single Sourcing

These topics cover strategies for single\_sourcing. Single sourcing refers to strategies for re-using the same source in different outputs for different audiences or purposes.

- [Conditional logic \(page 243\)](#)
- [Content reuse \(page 248\)](#)
- [Excluding files \(page 0\)](#)
- [Generating PDFs \(page](#)

## Publishing

When you're building, publishing, and deploying your Jekyll site, you might find these topics helpful.

- [Build arguments \(page 322\)](#)
- [10. Configure the build scripts \(page 0\)](#)
- [Generating PDFs \(page 326\)](#)
- [Help APIs and UI tooltips \(page 339\)](#)

326)

- [Help APIs and UI tooltips \(page 339\)](#)

- [iTerm profiles \(page 355\)](#)
- [Pushing builds to server \(page 357\)](#)
- [Search configuration \(page 351\)](#)
- [Themes \(page 325\)](#)

### Special Layouts

These pages highlight special layouts outside of the conventional page and TOC hierarchy.

- [FAQ layout \(page 367\)](#)
- [Glossary layout \(page 364\)](#)
- [Knowledge-base layout \(page 361\)](#)
- [Shuffle layout \(page 368\)](#)
- [Special layouts overview \(page 0\)](#)

**❗ Note:** This was mostly an experiment to see if I could break away from the hierarchical TOC and provide a different way of arranging the content. However, this layout is somewhat problematic because it doesn't allow you to browse other navigation options on the side while viewing a topic.



# Troubleshooting

**Summary:** This page lists common errors and the steps needed to troubleshoot them.

## Issues building the site

### Address already in use

When you try to build the site, you get this error in iTerm:

```
jekyll 2.5.3 | Error: Address already in use - bind(2)
```

This happens if a server is already in use. To fix this, edit your config file and change the port to a unique number.

If the previous server wasn't shut down properly, you can kill the server process using these commands:

```
ps aux | grep jekyll
```

Find the PID (for example, it looks like "22298").

Then type `kill -9 22298` where "22298" is the PID.

Alternatively, type the following to stop all Jekyll servers:

```
kill -9 $(ps aux | grep '[j]ekyll' | awk '{print $2}')
```

### shell file not executable

If you run into permissions errors trying to run a shell script file (such as `mydoc_multibuild_web.sh`), you may need to change the file permissions to make the sh file executable. Browse to the directory containing the shell script and run the following:

```
chmod +x build_writer.sh
```

## shell file not runnable

If you're using a PC, rename your shell files with a .bat extension.

### "page 0" cross references in the PDF

If you see "page 0" cross-references in the PDF, the URL doesn't exist. Check to make sure you actually included this page in the build.

If it's not a page but rather a file, you need to add a `noCrossRef` class to the file so that your print stylesheet excludes the counter from it. Add

`class="noCrossRef"` as an attribute to the link. In the `css/printstyles.css` file, there is a style that should remove the counter from anchor elements with this class.

### The PDF is blank

Check the `prince-list.txt` file in the output to see if it contains links. If not, you have something wrong with the logic in the `prince-list.txt` file. Check the `conditions.html` file in your `_includes` to see if the audience specified in your configuration file aligns with the `buildAudience` in the `conditions.html` file

### Sidebar not appearing

If you build your site but the sidebar doesn't appear, check the following:

Look in your PDF config file and make sure you have a sidebar property, such as this:

```
pdf_sidebar: product2_sidebar
```

Make sure each TOC item has an output property that specifies web or pdf.

Understanding how the theme works can be helpful in troubleshooting. The `_includes/sidebar.html` file loops through the values in the `_data/sidebar.yml` file. There are `if` statements that check whether the conditions (as specified in the `conditions.html` file) are met. If the `sidebar.yml` item doesn't have the right output, then it won't get displayed in the sidebar. It would instead get skipped.

### Sidebar isn't collapsed

If the sidebar levels aren't collapsed, usually your JavaScript is broken somewhere. Open the JavaScript Console and look to see where the problem is. If one script breaks, then other scripts will break too, so troubleshooting it is a little tricky.

### Search isn't working

If the search isn't working, check the JSON validity in the search.json file in your output folder. Usually something is invalid. Identify the problematic line, fix the file, or put `search: exclude` in the frontmatter of the file to exclude it from search.