**Openresty 与 Tengine**

Openresty和Tengine基于 Nginx 的两个衍生版本，某种意义上他们都和淘宝有关系，前者是前淘宝工程师agentzh主导开发的，后者是淘宝的一个开源项目；

Openresty的最大特点是引入了ngx\_lua模块，支持使用lua开发插件；

Tengine的特点是融入了因淘宝自身的一些业务带来的新功能；

**Tengine 简介**

tengine官方网站：<http://tengine.taobao.org/index_cn.html>

在 Nginx官方版本的基础上增加的一些定制模块如下：

1、支持动态加载模块：通过加载so文件实现，不用再重新编译整个项目了，配置如下：

dso {

load ngx\_http\_lua\_module.so;

load ngx\_http\_memcached\_module.so;

}

2、ngx\_proc\_daytime\_module模块，这个模块允许开一个独立的服务进程，该模块本身并未实现具体的业务逻辑，而是构建了一个TCP Server框架，等待开发者来实现自己的业务；

3、ngx\_http\_concat\_module模块，用于合并多个文件的响应；

4、ngx\_http\_upstream\_session\_sticky\_module模块，该模块是一个负载均衡模块，通过cookie实现客户端与后端服务器的会话保持, 在一定条件下可以保证同一个客户端访问的都是同一个后端服务器。

5、ngx\_http\_upstream\_check\_module模块，用于检查upstream上游服务器的健康状况，如下是一个配置的例子：

[复制代码](javascript:void(0);)

upstream cluster1 {

# simple round-robin

server 192.168.0.1:80;

server 192.168.0.2:80;

check interval=3000 rise=2 fall=5 timeout=1000 type=http;

check\_http\_send "HEAD / HTTP/1.0\r\n\r\n";

check\_http\_expect\_alive http\_2xx http\_3xx;

}

[复制代码](javascript:void(0);)

6、trim filter模块，用于删除 html，内嵌 javascript 和 css 中的注释以及重复的空白符，例如：

location / {

trim on;

trim\_js on;

trim\_css on;

}

7、ngx\_http\_headers\_module模块，支持根据Content-Type来设置过期时间，例如：

expires\_by\_types 24h text/html;

expires\_by\_types modified +24h text/xml;

expires\_by\_types @15h30m text/xml;

expires\_by\_types 0 text/xml;

expires\_by\_types -1 text/xml;

expires\_by\_types epoch text/xml;

8、ngx\_http\_limit\_req\_module模块，限制访问

9、扩展了ngx\_http\_log\_module模块，支持syslog和pipe；

**Openresty 简介**

openresty官方网站：<http://openresty.org/cn/index.html>

agentzh自己对openresty的介绍：<http://blog.zoomquiet.org/pyblosxom/oss/openresty-intro-2012-03-06-01-13.html>

agentzh（章亦春）的openresty开源项目（基于nginx），通过各种插件、模块，极大的扩展了nginx能干的事情，而lua扩展更是可以用来定制非常复杂的业务逻辑。作者给nginx赋予的这些新的特性，使openresty在业务开发上变得更加简单，对程序员更加友好，开发者可以在不需要对nginx源码熟悉的情况下就直接使用一些高级特性，比如并发subrequest、dns异步解析、对第三方数据库（如mysql、redis、memcached）等的访问。

先简单看一下openresty扩展的模块：

* ArrayVarNginxModule　　            支持数组变量
* AuthRequestNginxModule            鉴权模块
* CoolkitNginxModule                     一些插件
* NginxDevelKit                            封装了一些面对开发者的C接口
* **EchoNginxModule**                     一堆类似于shell命令的指令，支持（串行/并行）子请求等，这个模块很酷，也很实用
* EncryptedSessionNginxModule      会话加密
* FormInputNginxModule                解析POST请求中的参数
* HeadersMoreNginxModule            http头处理
* IconvNginxModule                       设置响应编码集
* MemcNginxModule                      支持对Memcached的访问
* DrizzleNginxModule　　                支持对mysql的访问
* PostgresNginxModule                   支持对Postgres数据库的访问
* RdsCsvNginxModule                     filter模块，将Drizzle和Postgres两个模块的响应格式化为csv输出
* RdsJsonNginxModule                    filter模块，将Drizzle和Postgres两个模块的响应格式化为json输出
* RedisNginxModule                        支持对redis的访问（只支持select和get命令）
* Redis2NginxModule                      支持对redis的访问（支持所有redis命令）
* SetMiscNginxModule                     一堆set命令，用于设置变量
* SrcacheNginxModule                    通过子请求进行cache
* XssNginxModule                          防跨站攻击

* LuaJIT                                       luaJIT解释器
* StandardLuaInterpreter               标准lua解释器
* LuaCjsonLibrary                          json库
* **LuaNginxModule                       lua接口库 （该模块本身用C实现）**
* LuaRdsParserLibrary                   rds格式解析
* LuaRedisParserLibrary                 redis响应解析库
* LuaRestyCoreLibrary                   LuaNginxModule模块的lua实现
* LuaRestyDNSLibrary                   dns解析库
* LuaRestyLockLibrary
* LuaRestyLrucacheLibrary             LRU cache库
* LuaRestyMemcachedLibrary 　　  memcached访问接口
* LuaRestyMySQLLibrary                mysql访问接口
* LuaRestyRedisLibrary                  redis访问接口
* LuaRestyStringLibrary                 一些hash函数的接口
* LuaRestyUploadLibrary
* LuaRestyUpstreamHealthcheckLibrary
* LuaRestyWebSocketLibrary           ws协议解析库
* LuaUpstreamNginxModule             扩展了对upstream的支持

浏览一下上述模块，有几个特点：

1、通过upstream机制已经可以支持对mysql、redis、postgreSQL、memcached 等数据库的访问（全都是异步无阻塞的）；

2、跟lua扩展有关的模块，提供给lua脚本调用的库，api非常丰富，涉及各种的操作；

**ngx\_lua 模块**

 ngx\_lua模块的原理：

1、每个worker（工作进程）创建一个Lua VM，worker内所有协程共享VM；  
2、将Nginx I/O原语封装后注入 Lua VM，允许Lua代码直接访问；  
3、每个外部请求都由一个Lua协程处理，协程之间数据隔离；  
4、Lua代码调用I/O操作等异步接口时，会挂起当前协程（并保护上下文数据），而不阻塞worker；  
5、I/O等异步操作完成时还原相关协程上下文数据，并继续运行；

**ngx\_lua 模块提供的指令和API等：**

|  |  |
| --- | --- |
| 指令名称 | 说明 |
| lua\_use\_default\_type | 是否使用default\_type指令定义的Content-Type默认值 |
| lua\_code\_cache | \*\_by\_lua\_file文件是否cache |
| lua\_regex\_cache\_max\_entries |  |
| lua\_regex\_match\_limit |  |
| lua\_package\_path | 用Lua写的lua外部库路径（.lua文件） |
| lua\_package\_cpath | 用C写的lua外部库路径（.so文件） |
| init\_by\_lua | master进程启动时挂载的lua代码 |
| init\_by\_lua\_file |  |
| init\_worker\_by\_lua | worker进程启动时挂载的lua代码，常用来执行一些定时器任务 |
| init\_worker\_by\_lua\_file |  |
| set\_by\_lua | 设置变量 |
| set\_by\_lua\_file |  |
| content\_by\_lua | handler模块 |
| content\_by\_lua\_file |  |
| rewrite\_by\_lua |  |
| rewrite\_by\_lua\_file |  |
| access\_by\_lua |  |
| access\_by\_lua\_file |  |
| header\_filter\_by\_lua | header filter模块 |
| header\_filter\_by\_lua\_file |  |
| body\_filter\_by\_lua | body filter模块，ngx.arg[1]代表输入的chunk，ngx.arg[2]代表当前chunk是否为last |
| body\_filter\_by\_lua\_file |  |
| log\_by\_lua |  |
| log\_by\_lua\_file |  |
| lua\_need\_request\_body | 是否读请求体，跟ngx.req.read\_body()函数作用类似 |
| lua\_shared\_dict | 创建全局共享的table（多个worker进程共享） |
| lua\_socket\_connect\_timeout | TCP/unix 域socket对象connect方法的超时时间 |
| lua\_socket\_send\_timeout | TCP/unix 域socket对象send方法的超时时间 |
| lua\_socket\_send\_lowat | 设置cosocket send buffer的low water值 |
| lua\_socket\_read\_timeout | TCP/unix 域socket对象receive方法的超时时间 |
| lua\_socket\_buffer\_size | cosocket读buffer大小 |
| lua\_socket\_pool\_size | cosocket连接池大小 |
| lua\_socket\_keepalive\_timeout | cosocket长连接超时时间 |
| lua\_socket\_log\_errors | 是否打开cosocket错误日志 |
| lua\_ssl\_ciphers |  |
| lua\_ssl\_crl |  |
| lua\_ssl\_protocols |  |
| lua\_ssl\_trusted\_certificate |  |
| lua\_ssl\_verify\_depth |  |
| lua\_http10\_buffering |  |
| rewrite\_by\_lua\_no\_postpone |  |
| lua\_transform\_underscores\_in\_response\_headers |  |
| lua\_check\_client\_abort | 是否监视client提前关闭请求的事件，如果打开监视，会调用ngx.on\_abort()注册的回调 |
| lua\_max\_pending\_timers |  |
| lua\_max\_running\_timers |  |

|  |  |
| --- | --- |
| table | 说明 |
| ngx.arg | 指令参数，如跟在content\_by\_lua\_file后面的参数 |
| ngx.var | 变量，ngx.var.VARIABLE引用某个变量 |
| ngx.ctx | 请求的lua上下文 |
| ngx.header | 响应头，ngx.header.HEADER引用某个头 |
| ngx.status | 响应码 |
|  |  |
| API | 说明 |
| ngx.log | 输出到error.log |
| print | 等价于 ngx.log(ngx.NOTICE, ...) |
| ngx.send\_headers | 发送响应头 |
| ngx.headers\_sent | 响应头是否已发送 |
| ngx.resp.get\_headers | 获取响应头 |
| ngx.timer.at | 注册定时器事件 |
| ngx.is\_subrequest | 当前请求是否是子请求 |
| ngx.location.capture | 发布一个子请求 |
| ngx.location.capture\_multi | 发布多个子请求 |
| ngx.exec |  |
| ngx.redirect |  |
| ngx.print | 输出响应 |
| ngx.say | 输出响应，自动添加'\n' |
| ngx.flush | 刷新响应 |
| ngx.exit | 结束请求 |
| ngx.eof |  |
| ngx.sleep | 无阻塞的休眠（使用定时器实现） |
| ngx.get\_phase |  |
| ngx.on\_abort | 注册client断开请求时的回调函数 |
| ndk.set\_var.DIRECTIVE |  |
| ngx.req.start\_time | 请求的开始时间 |
| ngx.req.http\_version | 请求的HTTP版本号 |
| ngx.req.raw\_header | 请求头（包括请求行） |
| ngx.req.get\_method | 请求方法 |
| ngx.req.set\_method | 请求方法重载 |
| ngx.req.set\_uri | 请求URL重写 |
| ngx.req.set\_uri\_args |  |
| ngx.req.get\_uri\_args | 获取请求参数 |
| ngx.req.get\_post\_args | 获取请求表单 |
| ngx.req.get\_headers | 获取请求头 |
| ngx.req.set\_header |  |
| ngx.req.clear\_header |  |
| ngx.req.read\_body | 读取请求体 |
| ngx.req.discard\_body | 扔掉请求体 |
| ngx.req.get\_body\_data |  |
| ngx.req.get\_body\_file |  |
| ngx.req.set\_body\_data |  |
| ngx.req.set\_body\_file |  |
| ngx.req.init\_body |  |
| ngx.req.append\_body |  |
| ngx.req.finish\_body |  |
| ngx.req.socket |  |
| ngx.escape\_uri | 字符串的url编码 |
| ngx.unescape\_uri | 字符串url解码 |
| ngx.encode\_args | 将table编码为一个参数字符串 |
| ngx.decode\_args | 将参数字符串编码为一个table |
| ngx.encode\_base64 | 字符串的base64编码 |
| ngx.decode\_base64 | 字符串的base64解码 |
| ngx.crc32\_short | 字符串的crs32\_short哈希 |
| ngx.crc32\_long | 字符串的crs32\_long哈希 |
| ngx.hmac\_sha1 | 字符串的hmac\_sha1哈希 |
| ngx.md5 | 返回16进制MD5 |
| ngx.md5\_bin | 返回2进制MD5 |
| ngx.sha1\_bin | 返回2进制sha1哈希值 |
| ngx.quote\_sql\_str | SQL语句转义 |
| ngx.today | 返回当前日期 |
| ngx.time | 返回UNIX时间戳 |
| ngx.now | 返回当前时间 |
| ngx.update\_time | 刷新时间后再返回 |
| ngx.localtime |  |
| ngx.utctime |  |
| ngx.cookie\_time | 返回的时间可用于cookie值 |
| ngx.http\_time | 返回的时间可用于HTTP头 |
| ngx.parse\_http\_time | 解析HTTP头的时间 |
| ngx.re.match |  |
| ngx.re.find |  |
| ngx.re.gmatch |  |
| ngx.re.sub |  |
| ngx.re.gsub |  |
| ngx.shared.DICT |  |
| ngx.shared.DICT.get |  |
| ngx.shared.DICT.get\_stale |  |
| ngx.shared.DICT.set |  |
| ngx.shared.DICT.safe\_set |  |
| ngx.shared.DICT.add |  |
| ngx.shared.DICT.safe\_add |  |
| ngx.shared.DICT.replace |  |
| ngx.shared.DICT.delete |  |
| ngx.shared.DICT.incr |  |
| ngx.shared.DICT.flush\_all |  |
| ngx.shared.DICT.flush\_expired |  |
| ngx.shared.DICT.get\_keys |  |
| ngx.socket.udp |  |
| udpsock:setpeername |  |
| udpsock:send |  |
| udpsock:receive |  |
| udpsock:close |  |
| udpsock:settimeout |  |
| ngx.socket.tcp |  |
| tcpsock:connect |  |
| tcpsock:sslhandshake |  |
| tcpsock:send |  |
| tcpsock:receive |  |
| tcpsock:receiveuntil |  |
| tcpsock:close |  |
| tcpsock:settimeout |  |
| tcpsock:setoption |  |
| tcpsock:setkeepalive |  |
| tcpsock:getreusedtimes |  |
| ngx.socket.connect |  |
| ngx.thread.spawn |  |
| ngx.thread.wait |  |
| ngx.thread.kill |  |
| coroutine.create |  |
| coroutine.resume |  |
| coroutine.yield |  |
| coroutine.wrap |  |
| coroutine.running |  |
| coroutine.status |  |
| ngx.config.debug | 编译时是否有 --with-debug选项 |
| ngx.config.prefix | 编译时的 --prefix选项 |
| ngx.config.nginx\_version | 返回nginx版本号 |
| ngx.config.nginx\_configure | 返回编译时 ./configure的命令行选项 |
| ngx.config.ngx\_lua\_version | 返回ngx\_lua模块版本号 |
| ngx.worker.exiting | 当前worker进程是否正在关闭（如reload、shutdown期间） |
| ngx.worker.pid | 返回当前worker进程的pid |
|  |  |
|  |  |
|  |  |
| 常量 | 说明 |
| Core constants | ngx.OK (0) ngx.ERROR (-1) ngx.AGAIN (-2) ngx.DONE (-4) ngx.DECLINED (-5) ngx.nil |
| HTTP method constants | ngx.HTTP\_GET ngx.HTTP\_HEAD ngx.HTTP\_PUT ngx.HTTP\_POST ngx.HTTP\_DELETE ngx.HTTP\_OPTIONS   ngx.HTTP\_MKCOL     ngx.HTTP\_COPY       ngx.HTTP\_MOVE      ngx.HTTP\_PROPFIND  ngx.HTTP\_PROPPATCH  ngx.HTTP\_LOCK  ngx.HTTP\_UNLOCK     ngx.HTTP\_PATCH    ngx.HTTP\_TRACE |
| HTTP status constants | ngx.HTTP\_OK (200) ngx.HTTP\_CREATED (201) ngx.HTTP\_SPECIAL\_RESPONSE (300) ngx.HTTP\_MOVED\_PERMANENTLY (301) ngx.HTTP\_MOVED\_TEMPORARILY (302) ngx.HTTP\_SEE\_OTHER (303) ngx.HTTP\_NOT\_MODIFIED (304) ngx.HTTP\_BAD\_REQUEST (400) ngx.HTTP\_UNAUTHORIZED (401) ngx.HTTP\_FORBIDDEN (403) ngx.HTTP\_NOT\_FOUND (404) ngx.HTTP\_NOT\_ALLOWED (405) ngx.HTTP\_GONE (410) ngx.HTTP\_INTERNAL\_SERVER\_ERROR (500) ngx.HTTP\_METHOD\_NOT\_IMPLEMENTED (501) ngx.HTTP\_SERVICE\_UNAVAILABLE (503) ngx.HTTP\_GATEWAY\_TIMEOUT (504) |
| Nginx log level constants | ngx.STDERR ngx.EMERG ngx.ALERT ngx.CRIT ngx.ERR ngx.WARN ngx.NOTICE ngx.INFO ngx.DEBUG |

======专注高性能web服务器架构和开发=====