

Built-in

`assert (v [, message])`

`getmetatable (object) -> table`

`ipairs (t) -> func, t, 0`

`load (chunk [, chunkname [, mode [, env]]])
-> func | nil, err_msg`

`loadfile ([filename [, mode [, env]]]) -> func |
nil, err_msg`

`next (table [, index]) -> value, next_i`

`pairs (t) -> v1, v2, v3 | next, t, nil`

`rawget (table, index) -> val`

`rawset (table, index, value)`

`select (index, ...) -> [i...]`

`setmetatable (table, metatable) -> table`

`type (v) -> t_str`

`pcall (f [, arg1, ...]) -> true, val | false`

`xpcall (f, msgf [, arg1, ...]) -> true, val | false`

table

`table.concat (list [, sep [, i [, j]]]) -> str`

`table.insert (list, [pos,] value)`

`table.move (a1, f, e, t [,a2])`

`table.remove (list [, pos])`

`table.unpack (list [, i [, j]]) -> ...`

`table.pack (...) -> table`

`table.sort (list [, comp])`

coroutine

`coroutine.create (closure_gen_func) ->
thread`

`coroutine.wrap (func) -> thread`

`coroutine.resume (co [, val1, ...]) -> true,
val1, ... | false, err`

`coroutine.yield (...)`

`coroutine.isyieldable () -> bool`

`coroutine.running () -> bool`

`coroutine.status (co) -> "running" | "suspended"
| "normal" | "dead"`

os

`os.clock () -> timeval`

`os.time ([table]) -> timeval`

`os.date ([format [, time]]) -> time_str`

`os.setlocale (locale [, category]) -> name |
nil`

`os.difftime (t2, t1) -> t2-t1`

`os.execute ([command]) -> true, "exit" | nil,
"signal", signo`

`os.exit ([code [, close]])`

`os.getenv (varname) -> env_arg | nil`

`os.remove (filename) -> true | nil, err,
err_code`

`os.rename (oldname, newname) -> true |
nil, err, err_code`

`os.tmpname () -> temp_name`

Command Line

`lua -e <statements>`

`arg` is a table contain command line
arguments

string

`string.match (s, pattern [, init]) -> cap1,
cap2, ... | nil`

`string.gmatch (s, pattern) -> match_iter_func
| nil`

`string.sub (s, i [, j]) -> sub_str`

`string.gsub (s, pattern, repl [, n]) -> sub_str`

`string.find (s, pattern [, init [, plain]]) -> start,
end | nil`

`string.byte (s [, i [, j]]) -> byte1, ...`

`string.char (...) -> str`

`string.dump (function [, strip]) -> func_str`

`string.pack (fmt, v1, v2, ...) -> bin_str`

`string.packsize (fmt) -> str_len`

`string.rep (s, n [, sep]) -> n_s`

`string.reverse (s) -> r_s`

`string.lower (s) -> l_s`

`string.upper (s) -> u_s`

string (cont)

`string.format (formatstring, ...) -> f_str`

`gmatch: match_iter_func() -> cap1, cap2, ...`

`gsub: repl`

table: use capture as key, retrieve

function: use captures as parameters

str_pattern: use %n as captures, %0 is
whole

io

`io.open (filename [, mode]) -> File`

`io.close ([file])`

`io.flush ()`

`io.input ([file]) -> in_File`

`io.output ([file]) -> out_File`

`io.lines ([filename ...]) -> iter_line_func`

`io.popen (prog [, mode]) -> File`

`io.read (...)`

`io.write (...)`

`io.tmpfile () -> tmp_File`

`io.type (File) -> "file" | "close_file" | nil`

`file:close ()`

`file:flush ()`

`file:lines (...) -> iter_line_func`

`file:read (...)`

`file:seek ([whence [, offset]])`

`file:setvbuf (mode [, size])`

`file:write (...) -> file | nil, err`

math

`math.abs (x)`

`math.acos (x)`

`math.asin (x)`

`math.atan (y [, x])`

`math.ceil (x)`

`math.cos (x)`

`math.deg (x)`

`math.exp (x)`

`math.floor (x)`

`math.fmod (x, y) -> (integer/float)`

math (cont)

`math.log (x [, base])`

`math.max (x, ...)`

`math.min (x, ...)`

`math.modf (x) -> int_part, frac_part`

`math.rad (x)`

`math.random ([m [, n]])`

`math.randomseed (x)`

`math.sin (x)`

`math.sqrt (x)`

`math.tan (x)`

`math.tointeger (x)`

`math.type (x) -> "integer" | "float" | nil`

`math.ult (m, n) -- unsigned less`

`math.huge`

`math.maxinteger`

`math.mininteger`

`math.pi`

字符类

`x`: (这里 `x` 不能是 魔法字符 `^$()%.[]*+-?` 中的一员) 表示字符 `x` 自身。

`.`: (一个点) 可表示任何字符。

`%a`: 表示任何字母。

`%c`: 表示任何控制字符。

`%d`: 表示任何数字。

`%g`: 表示任何除空白符外的可打印字符。

`%l`: 表示所有小写字母。

`%p`: 表示所有标点符号。

`%s`: 表示所有空白字符。

`%u`: 表示所有大写字母。

`%w`: 表示所有字母及数字。

`%x`: 表示所有 16 进制数字符号。

`%x`: (这里的 `x` 是任意非字母或数字的字符) 表示字符 `x`。

字符类 (cont)

`[set]`: 表示 `set` 中所有字符的联合。 可以用 `'` 连接, 升序书写范围两端的字符来表示一个范围的字符集。

`[^set]`: 表示 `set` 的补集。

模式条目

单个字符类匹配该类别中任意单个字符;

单个字符类跟一个 `*`, 将匹配零或多个该类的字符。 这个条目总是匹配尽可能长的串;

单个字符类跟一个 `+`, 将匹配一或多个该类的字符。 这个条目总是匹配尽可能长的串;

单个字符类跟一个 `.`, 将匹配零或多个该类的字符。 和 `*` 不同, 这个条目总是匹配尽可能短的串;

单个字符类跟一个 `?`, 将匹配零或一个该类的字符。 只要有可能, 它会匹配一个;
 `%n`, 这里的 `n` 可以从 1 到 9; 这个条目匹配一个等于 `n` 号捕获物 (后面有描述) 的子串。

`%bxy`, 这里的 `x` 和 `y` 是两个明确的字符; 这个条目匹配以 `x` 开始 `y` 结束

`%[set]`, 指 边境模式; 这个条目会匹配到一个位于 `set` 内某个字符之前的一个空串, 且这个位置的前一个字符不属于 `set`。

模式与捕获

模式: 指一个模式条目的序列。如果 `^` 和 `$` 出现在其它位置, 它们均没有特殊含义, 只表示自身。

捕获: 模式可以在内部用小括号括起一个子模式; 这些子模式被称为 捕获物。

debug

`debug.debug ()`

`debug.gethook ([thread]) -> hk_func, hk_mask, hk_cnt`

`debug.getinfo ([thread,] f [, what])`

`debug.getlocal ([thread,] f, local) -> name, val`

`debug.getmetatable (value) -> mt`

`debug.getregistry () -> registry_table`

`debug.getupvalue (f, up) -> up_name, up_val`

`debug.getuservalue (u) -> usr_val | nil`

`debug.sethook ([thread,] hook, mask [, count])`

`debug.setlocal ([thread,] level, local, value) -> local_name | nil`

`debug.setmetatable (value, table)`

`debug.setupvalue (f, up, value) -> up_name | nil`

`debug.setuservalue (udata, value) -> udata`

`debug.traceback ([thread,] [message [, level]])`

`debug.upvalueid (f, n)`

`debug.upvaluejoin (f1, n1, f2, n2)`