Módulo String

O módulo **string** fornece funções utilitárias para manipulação de strings em Lua, complementando as funções nativas da linguagem.

Importante: Este módulo estende a tabela `string` padrão do Lua, portanto as funções são acessadas através da tabela global `string` (ex: `string.split()`).

Funções Disponíveis

1. `string.split(s, sep)`

Divide uma string em substrings com base em um separador especificado.

Parâmetros:

- **s** (string): A string original que será dividida
- **sep** (string): O separador usado para dividir a string

Retorno:

- **table**: Um array (tabela indexada numericamente) contendo todas as substrings resultantes da divisão

Comportamento:

- Se o separador for uma string vazia (`""`), a função retorna um array com cada caractere individual
- Se o separador não for encontrado na string, retorna um array contendo apenas a string original
- A divisão é feita em todas as ocorrências do separador

Exemplo de Uso:

```
-- Dividir uma string por vírgula
local resultado = string.split("maçã,banana,laranja,uva", ",")
-- resultado = {"maçã", "banana", "laranja", "uva"}
```

```
-- Dividir por espaço
local palavras = string.split("Olá mundo do Lua", " ")
-- palavras = {"Olá", "mundo", "do", "Lua"}

-- Dividir por nova linha (processamento de logs)
local linhas = string.split("2024-01-15 ERROR: Falha na conexão\n2024-01-15 INFO:
Reconectando", "\n")
-- linhas = {"2024-01-15 ERROR: Falha na conexão", "2024-01-15 INFO: Reconectando"}

-- Separador vazio (dividir em caracteres)
local chars = string.split("teste", "")
-- chars = {"t", "e", "s", "t", "e"}

-- Separador não encontrado
local unico = string.split("texto_sem_separador", "|")
-- unico = {"texto_sem_separador"}
```

2. `string.trim(s)`

Remove espaços em branco (whitespace) do início e do final de uma string.

Parâmetros:

- **s** (string): A string que será limpa

Retorno:

- **string**: A string original sem espaços em branco no início e no final

Comportamento:

- Remove espaços (` `), tabs (`\t`), novas linhas (`\n`), retornos de carro (`\r`)
- Não remove espaços no meio da string
- Retorna string vazia se a entrada for apenas espaços em branco

Exemplo de Uso:

```
-- Remover espaços extras
local limpo = string.trim("  texto com espaços  ")
-- limpo = "texto com espaços"
```

```
-- Limpar entrada de usuário
local entrada = "\t\n valor digitado \r\n"
local processado = string.trim(entrada)
-- processado = "valor digitado"

-- Processar configurações
local config_line = "   timeout = 30   "
local chave_valor = string.trim(config_line)
-- chave_valor = "timeout = 30"

-- String apenas com espaços
local vazio = string.trim("   \t\n   ")
-- vazio = ""
```

3. `string.starts(String, Start)`

Verifica se uma string começa com um prefixo específico.

Parâmetros:

- **String** (string): A string a ser verificada
- **Start** (string): O prefixo a ser procurado no início da string

Retorno:

- **boolean**: `true` se a string começar com o prefixo especificado, `false` caso contrário

Comportamento:

- A comparação é case-sensitive (diferencia maiúsculas de minúsculas)
- Retorna `true` se o prefixo for uma string vazia (`""`)
- Funciona com strings multibyte (UTF-8)

Exemplo de Uso:

```
-- Verificar se uma string começa com "http"
local url = "https://example.com"
local is_http = string.starts(url, "http")
-- is_http = true
```

```
-- Verificar prefixo em caminhos de arquivo
local path = "/var/log/app.log"
local is_absolute = string.starts(path, "/")
-- is_absolute = true

-- Verificar em processamento de logs
local log_line = "ERROR: Database connection failed"
local is_error = string.starts(log_line, "ERROR:")
-- is_error = true

-- Prefixo vazio sempre retorna true
local always_true = string.starts("qualquer string", "")
-- always_true = true

-- Case-sensitive
local case_check = string.starts("Hello World", "hello")
-- case_check = false (diferencia maiúsculas/minúsculas)
```

4. `string.ends(string, end)`

Verifica se uma string termina com um sufixo específico.

Parâmetros:

- **string** (string): A string a ser verificada
- **end** (string): O sufixo a ser procurado no final da string

Retorno:

- **boolean**: `true` se a string terminar com o sufixo especificado, `false` caso contrário

Comportamento:

- A comparação é case-sensitive (diferencia maiúsculas de minúsculas)
- Retorna `true` se o sufixo for uma string vazia (`""`)
- Funciona com strings multibyte (UTF-8)

Exemplo de Uso:

```
-- Verificar extensão de arquivo
local filename = "document.pdf"
```

```lua
local is_pdf = string.ends(filename, ".pdf")
-- is_pdf = true

-- Verificar sufixo em URLs
local url = "https://api.example.com/v1/data.json"
local is_json = string.ends(url, ".json")
-- is_json = true

-- Verificar terminação em strings de log
local log_entry = "Process completed successfully."
local is_success = string.ends(log_entry, "successfully.")
-- is_success = true

-- Sufixo vazio sempre retorna true
local always_true = string.ends("qualquer string", "")
-- always_true = true

-- Case-sensitive
local case_check = string.ends("Hello World", "world")
-- case_check = false (diferencia maiúsculas/minúsculas)
```

Vantagens do Módulo String do Monsta:

1. `string.split()` - Não existe nativamente no Lua, precisa ser implementada manualmente
2. `string.trim()` - Mais performática que implementações em Lua puro
3. **Consistência** - Mesma interface para todas as funções

Funções Complementares:

Use em conjunto com funções nativas do Lua:

- `string.find()` - Para buscas complexas
- `string.gsub()` - Para substituições
- `string.match()` - Para extração com padrões
- `string.gmatch()` - Para iteração sobre padrões

Exemplos Completos

Exemplo 1: Processamento de Log de Sistema

```
-- Simular leitura de log do sistema
local log_data = [[
Jan 15 10:30:45 servidor kernel: [12345.67890] CPU temperature: 65.5C
Jan 15 10:31:15 servidor sshd[1234]: Accepted password for user from 192.168.1.100
Jan 15 10:32:00 servidor crond[5678]: (root) CMD (/usr/bin/backup.sh)
]]

-- Processar cada linha do log
local function analisar_logs(logs)
    local eventos = {}

    for linha in string.gmatch(logs, "[^\n]+") do
        local linha_limpa = string.trim(linha)
        if linha_limpa ~= "" then
            -- Dividir por espaços (formato syslog)
            local partes = string.split(linha_limpa, " ")

            if #partes >= 5 then
                local evento = {
                    data = partes[1] .. " " .. partes[2] .. " " ..
partes[3],
                    host = partes[4],
                    servico = partes[5],
                    mensagem = table.concat(partes, " ", 6)
                }
                table.insert(eventos, evento)
            end
        end
    end

    return eventos
end

local eventos = analisar_logs(log_data)
```

Exemplo 2: Parser de Configuração Simples

```
-- Parser para arquivos de configuração no formato chave=valor
local function parse_config(conteudo)
    local config = {}

    for linha in string.gmatch(conteudo, "[^\n]+") do
        local linha_limpa = string.trim(linha)

        -- Ignorar linhas vazias e comentários
        if linha_limpa ~= "" and not string.starts(linha_limpa, "#") then
            -- Dividir por "="
            local partes = string.split(linha_limpa, "=")

            if #partes == 2 then
                local chave = string.trim(partes[1])
                local valor = string.trim(partes[2])

                -- Remover aspas se existirem
                if string.starts(valor, "\"") and string.ends(valor, "\"")
then
                    valor = string.sub(valor, 2, -2)
                end

                config[chave] = valor
            end
        end
    end

    return config
end

-- Exemplo de uso
local config_text = [[
# Configurações do monitor
hostname = "servidor-prod"
port = 8080
timeout = 30
```

```
debug = false
]]

local config = parse_config(config_text)
-- config.hostname = "servidor-prod", config.port = "8080", etc.
```

Revision #1
Created 3 February 2026 16:46:03 by Marc
Updated 3 February 2026 16:46:15 by Marc