Funções Globais

Esta seção descreve as funções globais disponíveis no ambiente Lua do Monsta. Estas são funções que não pertencem a módulos específicos, mas estão disponíveis diretamente no escopo global do Lua. Existem funcionalidades para controle de fluxo, manipulação de tempo entre outras.

Funções Disponíveis

`sleep(seconds)`

Descrição: Pausa a execução do script por um número especificado de segundos.

Parâmetros:

- `seconds` (número): Número de segundos para pausar a execução

Retorno: `nil` (não retorna valor)

Exemplo:

```
-- Pausar por 5 segundos
sleep(5)
print("Execução retomada após 5 segundos")
```

`signal(signal_name)`

Descrição: Sinaliza o término da execução do script com um nome de sinal específico.

Parâmetros:

- `signal_name` (string): Nome do sinal a ser emitido

Retorno: Aborta a execução do script com nome do sinal

Características:

- Chama a função `_execution_done` se disponível no ambiente global
- Sempre lança um erro, interrompendo a execução normal

- Útil para controle de fluxo e sinalização de estados

Nota: Atualmente, esta função tem um único caso de uso específico: sinalizar com o nome `"RepeatPrevValue"`. Quando um script emite este sinal, o sistema interpreta que a coleta atual deve repetir o último valor válido da métrica, em vez de gerar um novo ponto de dados. Isso é útil em situações onde a fonte de dados está temporariamente indisponível ou a coleta falhou, mas não se deseja interromper a série temporal.

Exemplo de uso específico:

```
-- Tentar coletar um valor
local value, err = collect_metric()
if err then
    -- Em caso de falha, repetir o valor anterior
    signal("RepeatPrevValue")
end
```

`with_timeout(timeout_ms, func, ...)`

Descrição: Executa uma função com um limite de tempo (timeout).

Parâmetros:

- `timeout_ms` (número): Tempo limite em milissegundos
- `func` (função): Função Lua a ser executada
- `...` (opcional): Argumentos para passar para a função

Retorno: Retorna o resultado da função executada

Exemplo:

```
-- Executar uma função com timeout de 2 segundos
local result = with_timeout(2000, function()
    -- Operação que pode demorar
    return some_long_running_operation()
end)

-- Executar com argumentos
local data = with_timeout(1000, http.get, "https://api.example.com/data")

-- Tratamento de timeout
local success, result = pcall(function()
```

```
    return with_timeout(500, function()
        -- Operação que deve completar rapidamente
        return critical_operation()
    end)
end)

if not success then
    print("Operação excedeu o timeout de 500ms")
end
```

Características:

- Lança um erro se o timeout for excedido
- Preserva os argumentos passados para a função
- Útil para operações de rede ou I/O que podem travar

now()

Descrição: Retorna o número de segundos não bissextos desde 1 de janeiro de 1970 00:00:00 UTC (também conhecido como "timestamp UNIX").

Parâmetros: Nenhum

Retorno: Número representando segundos desde a época Unix

Exemplo:

```
-- Obter timestamp atual
local current_time = now()
print("Timestamp atual:", current_time)

-- Calcular duração de operação
local start_time = now()
-- Executar alguma operação
local end_time = now()
local duration = end_time - start_time
print("Operação levou", duration, "segundos")
```

Características:

- Mesma função disponível em `time.now()`
- Útil para medição de performance

`print(...)`

Descrição: Função de impressão que formata múltiplos argumentos.

Parâmetros:

- `...` (múltiplos valores): Valores a serem impressos

Retorno: `nil` (não retorna valor)

Exemplo:

```
print("Valor:", 42, "Status:", true, "Lista:", {1, 2, 3})
```

Características:

- Separa múltiplos argumentos com tabulação
- Útil para debugging

`diff(lhs, rhs)`

Descrição: Calcula a diferença entre dois valores numéricos, com tratamento especial para contadores que podem sofrer rollover.

Parâmetros:

- `lhs` (número): Valor atual (left-hand side)
- `rhs` (número): Valor anterior (right-hand side)

Retorno: Número representando a diferença entre os valores

Comportamento:

- Calcula `lhs - rhs`
- Se o resultado for negativo, emite o sinal `"RepeatPrevValue"`
- Usado internamente por `snmp.diff` e `wmi.diff` para cálculo de diferenças em contadores
- Útil para métricas de contador que podem sofrer rollover (como contadores de 32 ou 64 bits)

Exemplo:

```
-- Calcular diferença entre leituras de contador
local current_value = 4294967290  -- Valor atual (próximo do rollover de 32 bits)
local previous_value = 4294967280  -- Valor anterior
```

```
local difference = diff(current_value, previous_value)
-- difference = 10 (4294967290 - 4294967280)

-- Caso com rollover (valor diminuiu)
local current_with_rollover = 10  -- Após rollover
local previous_before_rollover = 4294967295  -- Antes do rollover

local rollover_diff = diff(current_with_rollover, previous_before_rollover)
-- Emite sinal "RepeatPrevValue" pois 10 - 4294967295 é negativo
```

Variáveis Globais

EXEC_IDENT

Descrição: Identificador único da execução atual do script.

Tipo: String

Exemplo:

```
-- Usar o identificador em logs
print("Execução ID:", EXEC_IDENT)

-- Incluir em dados de monitoramento
local metrics = {
    ident = EXEC_IDENT,
    timestamp = now(),
    value = collected_data
}

-- Usar como chave para armazenamento
store.put("results_" .. EXEC_IDENT, processing_result)
```

Características:

- Definida automaticamente pelo ambiente
- Única para cada execução de script
- Útil para rastreamento e correlação de logs

Limitações

1. `sleep` **Não é Preciso**: Devido à natureza assíncrona do sistema, `sleep` pode ter variações pequenas.
2. `signal` **Interrompe Execução**: Uma vez chamada, a execução normal é interrompida.

Exemplo

```
-- Monitorar serviço com backoff em caso de falha
local function monitor_with_backoff(service_url, max_attempts)
    local attempt = 1
    local backoff = 1  -- segundos

    while attempt <= max_attempts do
        log.info("Tentativa", attempt, "de", max_attempts)

        local success, status = pcall(function()
            return with_timeout(5000, function()
                return check_service(service_url)
            end)
        end)

        if success and status == "healthy" then
            log.info("Serviço saudável")
            return true
        end

        -- Incrementar backoff exponencial
        sleep(backoff)
        backoff = math.min(backoff * 2, 60)  -- Máximo 60 segundos
        attempt = attempt + 1
    end

    -- Todas as tentativas falharam
    return false
end

-- Executar monitoramento
```

```
monitor_with_backoff("https://api.example.com", 5)
```

Revision #1
Created 3 February 2026 16:42:08 by Marc
Updated 3 February 2026 16:42:27 by Marc