Módulo SNMP

O módulo **snmp** fornece uma interface completa para monitoramento e gerenciamento de dispositivos de rede através do protocolo SNMP (Simple Network Management Protocol). Este módulo é útil para coleta de métricas de roteadores, switches, servidores, impressoras e qualquer dispositivo que suporte SNMP.

- Suporte a SNMP v1, v2c e v3
- Operações GET, GET BULK e WALK
- Autenticação e criptografia SNMP v3
- Configuração flexível de timeout e retentativas
- Retorno seguro com tratamento de erros

Protocolos suportados:

- **SNMP v1**: Protocolo básico com comunidade pública/privada
- **SNMP v2c**: Melhorias de performance com operações BULK
- **SNMP v3**: Segurança avançada com autenticação e criptografia

Configuração SNMP

Todas as funções SNMP requerem um objeto de configuração que define os parâmetros de conexão. A configuração é uma tabela Lua com os seguintes campos:

Campos de Configuração Básicos:

Campo	Tipo	Padrão	Descrição
`address`	string	**obrigatório**	Endereço IP ou hostname do dispositivo
`snmpVersion`	número	1	Versão SNMP (1=v1, 2=v2c, 3=v3)
`snmpPort`	número	161	Porta SNMP
`snmpCommunity`	string	nil	Comunidade SNMP (v1/v2c)
`snmpTimeout`	número	5	Timeout em segundos
`snmpRetryCount`	número	3	Número de retentativas

Campo	Tipo	Padrão	Descrição
`snmpMaxBulkItems`	número	nil	Máximo de itens por operação BULK
`snmpExponentialBackoff`	booleano	false	Habilitar backoff exponencial

Campos para SNMP v3:

Campo	Tipo	Descrição
`snmpSecurityLevel`	string	Nível de segurança: "NoAuthNoPriv", "AuthNoPriv", "AuthPriv"
`snmpAuthProtocol`	string	Protocolo de autenticação: "MD5", "SHA1"
`snmpAuthUser`	string	Usuário de autenticação
`snmpAuthPassword`	string	Senha de autenticação
`snmpPrivProtocol`	string	Protocolo de criptografia: "DES", "AES"
`snmpPrivPassword`	string	Senha de criptografia

Configuração via Tabela `params`:

O ambiente Lua inclui uma tabela predefinida chamada `params` que contém os detalhes do dispositivo atual. Esta tabela pode ser usada diretamente como configuração SNMP, pois já possui os campos necessários no formato esperado.

Exemplo de uso direto:

```
-- Usar a tabela device diretamente como configuração
local sys_descr = snmp.getex(device, "1.3.6.1.2.1.1.1.0")
print("Descrição do dispositivo:", sys_descr)

-- Versão que não lança erro
local valor, erro = snmp.get_safe(device, "1.3.6.1.2.1.1.3.0")
if not erro then
    print("Uptime do dispositivo:", valor)
end
```

Combinando com configurações adicionais:

```
-- Criar configuração baseada em device com ajustes
local config = {
    address = device.address,
    snmpVersion = device.snmpVersion,
    snmpCommunity = device.snmpCommunity,
    snmpTimeout = device.snmpTimeout or 5,  -- Usar padrão se não definido
    snmpRetryCount = 2,  -- Sobrescrever valor padrão
    snmpMaxBulkItems = 50  -- Adicionar configuração extra
}

-- Usar para operação BULK
local oids = {"1.3.6.1.2.1.1.1.0", "1.3.6.1.2.1.1.3.0"}
local resultados = snmp.get_bulk(config, oids)
```

Exemplos de Configuração:

```
-- Configuração básica SNMP v2c
local config_v2c = {
    address = "192.168.1.1",
    snmpVersion = 2,
    snmpCommunity = "public",
    snmpTimeout = 3,
    snmpRetryCount = 2
}

-- Configuração SNMP v3 com autenticação e criptografia
local config_v3 = {
    address = "10.0.0.254",
    snmpVersion = 3,
    snmpSecurityLevel = "AuthPriv",
    snmpAuthProtocol = "SHA1",
    snmpAuthUser = "monitor",
    snmpAuthPassword = "senha123",
    snmpPrivProtocol = "AES",
    snmpPrivPassword = "chave456",
    snmpTimeout = 5
}
```

```
-- Configuração para dispositivo com porta não padrão
local config_custom_port = {
    address = "switch.piso1.local",
    snmpVersion = 2,
    snmpCommunity = "internal",
    snmpPort = 8161,  -- Porta customizada
    snmpTimeout = 10  -- Timeout maior para rede lenta
}
```

Funções Disponíveis

1. `snmp.getex(config, oid)`

Realiza uma consulta SNMP GET para um OID específico.

Parâmetros:

- **config** (tabela): Configuração SNMP (ver seção acima)
- **oid** (string): OID a ser consultado (formato numérico ou nomeado)

Retorno:

- **valor**: Valor retornado pelo dispositivo SNMP (número, string, etc.)

Erros:

- Lança erro se o OID não existir ou houver falha na comunicação

Exemplo de Uso:

```
-- Consultar sysDescr (descrição do sistema)
local config = {
    address = "192.168.1.1",
    snmpVersion = 2,
    snmpCommunity = "public"
}

local sys_descr = snmp.getex(config, "1.3.6.1.2.1.1.1.0")
-- ou usando OID nomeado
```

```
local sys_descr = snmp.getex(config, ".1.3.6.1.2.1.1.1.0")

print("Descrição do sistema:", sys_descr)
-- Exemplo de saída: "Cisco IOS Software, C3750 Software (C3750-IPSERVICESK9-M), Version
12.2(55)SE10, RELEASE SOFTWARE (fc2)"

-- Consultar uptime do sistema
local sys_uptime = snmp.getex(config, "1.3.6.1.2.1.1.3.0")
print("Uptime:", sys_uptime, "centésimos de segundo")

-- Consultar nome do host
local sys_name = snmp.getex(config, "1.3.6.1.2.1.1.5.0")
print("Nome do host:", sys_name)

-- Consultar localização
local sys_location = snmp.getex(config, "1.3.6.1.2.1.1.6.0")
print("Localização:", sys_location)
```

2. `snmp.get_safe(config, oid)`

Versão segura de `getex` que não lança exceções, retornando erro como segundo valor.

Parâmetros:

- **config** (tabela): Configuração SNMP
- **oid** (string): OID a ser consultado

Retorno:

- **tuple**: `(valor, erro)` onde:
  - **valor** (qualquer tipo ou nil): Valor retornado se bem-sucedido
  - **erro** (string ou nil): Mensagem de erro se falhar, nil se bem-sucedido

Exemplo de Uso:

```
local config = {
    address = "192.168.1.1",
    snmpVersion = 2,
    snmpCommunity = "public"
}
```

```
-- Consulta segura que não quebra o script em caso de erro
local valor, erro = snmp.get_safe(config, "1.3.6.1.2.1.1.1.0")

if erro then
    log.error("Falha na consulta SNMP:", erro)
    -- Tomar ação alternativa
else
    print("Valor obtido:", valor)
end

-- Consultar múltiplos OIDs com tratamento de erro individual
local oids = {
    "1.3.6.1.2.1.1.1.0",  -- sysDescr
    "1.3.6.1.2.1.1.3.0",  -- sysUpTime
    "1.3.6.1.2.1.1.5.0",  -- sysName
    "1.3.6.1.2.1.1.6.0"   -- sysLocation
}

local resultados = {}
for _, oid in ipairs(oids) do
    local valor, erro = snmp.get_safe(config, oid)
    if erro then
        log.warn("Falha no OID", oid, ":", erro)
        resultados[oid] = {erro = erro}
    else
        resultados[oid] = {valor = valor}
    end
end
```

3. `snmp.get_bulk(config, oids)`

Realiza operação SNMP GET BULK para múltiplos OIDs de uma vez (SNMP v2c/v3).

Parâmetros:

- **config** (tabela): Configuração SNMP (deve ser v2 ou v3)
- **oids** (array de strings): Lista de OIDs para consulta

Retorno:

- **tabela**: Mapa OID → valor para todos os OIDs consultados

Comportamento:

- Mais eficiente que múltiplas chamadas `getex` para muitos OIDs
- Suportado apenas em SNMP v2c e v3
- Usa `snmpMaxBulkItems` da configuração para limitar tamanho

Exemplo de Uso:

```
local config = {
    address = "192.168.1.1",
    snmpVersion = 2,  -- Deve ser v2 ou v3 para GET BULK
    snmpCommunity = "public",
    snmpMaxBulkItems = 50  -- Limitar a 50 OIDs por operação
}

-- Consultar múltiplas informações do sistema de uma vez
local oids = {
    "1.3.6.1.2.1.1.1.0",  -- sysDescr
    "1.3.6.1.2.1.1.3.0",  -- sysUpTime
    "1.3.6.1.2.1.1.5.0",  -- sysName
    "1.3.6.1.2.1.1.6.0",  -- sysLocation
    "1.3.6.1.2.1.1.7.0"   -- sysServices
}

local resultados = snmp.get_bulk(config, oids)

for oid, valor in pairs(resultados) do
    print("OID:", oid, "=", valor)
end

-- Consultar informações de múltiplas interfaces
local function obter_info_interfaces(config, indices)
    local oids = {}
    for _, idx in ipairs(indices) do
        table.insert(oids, "1.3.6.1.2.1.2.2.1.2." .. idx)   -- ifDescr
        table.insert(oids, "1.3.6.1.2.1.2.2.1.3." .. idx)   -- ifType
        table.insert(oids, "1.3.6.1.2.1.2.2.1.5." .. idx)
    end
    return snmp.get_bulk(config, oids)
```

```
end
```

4. `snmp.get(oid)`

Versão simplificada de `snmp.getex` que usa automaticamente a configuração do dispositivo atual (`params`).

Parâmetros:

- **oid** (string): OID a ser consultado

Retorno:

- **valor**: Valor retornado pelo dispositivo SNMP

Comportamento:

- Usa `params` como configuração
- Lança erro se o OID não existir ou houver falha na comunicação

Exemplo de Uso:

```
-- Consulta simplificada usando a configuração do dispositivo atual
local sys_descr = snmp.get("1.3.6.1.2.1.1.1.0")
print("Descrição do sistema:", sys_descr)

-- Consultar múltiplos OIDs
local uptime = snmp.get("1.3.6.1.2.1.1.3.0")
local hostname = snmp.get("1.3.6.1.2.1.1.5.0")

print("Uptime:", uptime, "Hostname:", hostname)
```

5. `snmp.walk(oid, cache_ttl, enforce_ordering)`

Realiza uma operação SNMP WALK com suporte a cache e ordenação.

Parâmetros:

- **oid** (string): OID base para o walk

- **cache_ttl** (número, opcional): Tempo de vida do cache em segundos
- **enforce_ordering** (booleano, opcional): Forçar ordenação dos resultados (ordem natural)

Comportamento do cache:

- Usa cache quando `cache_ttl` é especificado

Exemplo de Uso:

```
-- Walk com cache de 30 segundos
local interfaces = snmp.walk("1.3.6.1.2.1.2.2.1.2", 30)
for oid, ifname in pairs(interfaces) do
    print("Interface", oid, ":", ifname)
end

-- Walk com ordenação forçada (sem cache)
local ordered_interfaces = snmp.walk("1.3.6.1.2.1.2.2.1.2", nil, true)
print("Total de interfaces:", #ordered_interfaces)
```

Comportamento de Ordenação:

O parâmetro `enforce_ordering` controla como os resultados são estruturados:

- **Quando `false` (padrão)**: Os resultados são retornados como uma tabela Lua onde cada OID é uma chave que mapeia para seu valor. Esta estrutura é eficiente para acesso aleatório, mas **perde a ordenação natural** dos OIDs, pois as tabelas Lua não preservam a ordem de inserção das chaves.
- **Quando `true`**: Os resultados são retornados como uma **lista de pares** (tabela de tabelas), onde cada elemento é uma tabela contendo 2 elementos. Esta estrutura preserva a ordem natural dos OIDs conforme retornados pelo dispositivo SNMP.

Exemplo de diferença:

```
-- Com enforce_ordering = false (padrão)
local resultado_tabela = snmp.walk("1.3.6.1.2.1.2.2.1.2", nil, false)
-- Estrutura: { ["1.3.6.1.2.1.2.2.1.2.1"] = "eth0", ["1.3.6.1.2.1.2.2.1.2.2"] = "eth1" }
-- A ordem das chaves não é garantida

-- Com enforce_ordering = true
local resultado_lista = snmp.walk("1.3.6.1.2.1.2.2.1.2", nil, true)
-- Estrutura: { {"1.3.6.1.2.1.2.2.1.2.1", "eth0"},
--              {"1.3.6.1.2.1.2.2.1.2.2", "eth1"} }
-- A ordem dos elementos é preservada
```

Quando usar cada modo:

- Use `enforce_ordering = false` quando você só precisa acessar valores por OID específico e a ordem não importa.
- Use `enforce_ordering = true` quando você precisa processar os resultados na mesma ordem em que foram retornados pelo dispositivo, como para:
  - Gerar relatórios ordenados
  - Processar sequências de índices consecutivos
  - Manter correspondência com outras listas ordenadas

Retorno:

- **tabela**: Mapa OID → valor para todos os OIDs encontrados

6. `snmp.walkex(device, oid, cache_ttl)`

Função estendida de walk com sistema de cache avançado e prevenção de execuções concorrentes.

Parâmetros:

- **device** (tabela): Configuração do dispositivo
- **oid** (string): OID base para o walk
- **cache_ttl** (número, opcional): Tempo de vida do cache em segundos

Retorno:

- **tabela**: Mapa OID → valor para todos os OIDs encontrados

Comportamento:

- Implementa cache global compartilhado entre execuções
- Previne execuções concorrentes do mesmo walk
- Usa `registry` para coordenar execuções simultâneas

Exemplo de Uso:

```
-- Walk estendido com cache de 60 segundos
local device_config = {
    address = "192.168.1.1",
    snmpVersion = 2,
    snmpCommunity = "public"
}
```

```
local sys_oids = snmp.walkex(device_config, "1.3.6.1.2.1.1", 60)
for oid, value in pairs(sys_oids) do
    print("OID:", oid, "Valor:", value)
end
```

7. `snmp.count(oid)`

Conta o número de itens retornados por um walk.

Parâmetros:

- **oid** (string): OID base para contar

Retorno:

- **número**: Quantidade de itens encontrados

Exemplo de Uso:

```
-- Contar número de interfaces
local num_interfaces = snmp.count("1.3.6.1.2.1.2.2.1.2")
print("Número de interfaces:", num_interfaces)

-- Contar número de processos
local num_processes = snmp.count("1.3.6.1.2.1.25.4.2.1.2")
print("Número de processos:", num_processes)
```

8. `snmp.diff(typ, lhs, rhs)`

Calcula a diferença entre dois valores, tratando rollover de contadores.

Parâmetros:

- **typ** (número): Tipo do contador (32 ou 64 bits)
- **lhs** (número): Valor atual
- **rhs** (número): Valor anterior

Retorno:

- **número**: Diferença entre os valores

Comportamento:

- Trata rollover de contadores de 32 e 64 bits
- Sinaliza `RepeatPrevValue` se a diferença for negativa

Exemplo de Uso:

```
-- Calcular diferença para contador de 32 bits
local current_bytes = snmp.get("1.3.6.1.2.1.2.2.1.10.1")  -- ifInOctets.1
local prev_bytes = prev("1.3.6.1.2.1.2.2.1.10.1")
local bytes_diff = snmp.diff(32, current_bytes, prev_bytes)

print("Bytes recebidos desde última leitura:", bytes_diff)
```

9. `inst(oid)`

Resolve dinamicamente OIDs de instâncias baseado no nome da instância.

Parâmetros:

- **oid** (string): OID base (sem índice de instância)

Retorno:

- **string**: OID completo com índice de instância

Comportamento:

- Usa `params.InstanceName` e `params.snmpOIDDesc` para resolução
- Suporta cache de instâncias
- Lança erro se a instância não for encontrada

Exemplo de Uso:

```
-- Resolver OID para instância específica
-- params.InstanceName = "eth0"
-- params.snmpOIDDesc = "1.3.6.1.2.1.2.2.1.2"  -- ifDescr

local if_in_octets_oid = inst("1.3.6.1.2.1.2.2.1.10")  -- ifInOctets
print("OID resolvido:", if_in_octets_oid)
-- Saída: "1.3.6.1.2.1.2.2.1.10.1" (se eth0 for índice 1)
```

```
-- Consultar usando OID resolvido
local bytes_in = snmp.get(if_in_octets_oid)
print("Bytes recebidos na interface eth0:", bytes_in)
```

10. `prev(oid)`

Obtém o valor anterior de um OID armazenado.

Parâmetros:

- **oid** (string): OID para obter valor anterior

Retorno:

- **qualquer tipo**: Valor anterior armazenado, ou 0 se não existir

Comportamento:

- Busca valor em `store.get("snmp.value." .. oid)`
- Retorna 0 se não encontrar valor armazenado

Exemplo de Uso:

```
-- Obter valor anterior para cálculo de taxa
local current_value = snmp.get("1.3.6.1.2.1.2.2.1.16.1")  -- ifOutOctets.1
local previous_value = prev("1.3.6.1.2.1.2.2.1.16.1")

local bytes_out_diff = current_value - previous_value
print("Bytes enviados desde última leitura:", bytes_out_diff)
```

11. `lapsed(oid)`

Obtém o tempo decorrido desde a última leitura de um OID.

Parâmetros:

- **oid** (string): OID para verificar tempo decorrido

Retorno:

- **número**: Tempo em segundos desde última leitura, ou 1 se não houver registro

Exemplo de Uso:

```
-- Calcular taxa por segundo
local current_counter = snmp.get("1.3.6.1.2.1.2.2.1.10.1")  -- ifInOctets.1
local previous_counter = prev("1.3.6.1.2.1.2.2.1.10.1")
local time_elapsed = lapsed("1.3.6.1.2.1.2.2.1.10.1")

local bytes_per_second = (current_counter - previous_counter) / time_elapsed
print("Taxa de recebimento:", bytes_per_second, "bytes/segundo")
```

Exemplos Completos

Monitoramento de Interface de Rede:

```
-- Resolver OID da interface eth0
local if_index_oid = inst("1.3.6.1.2.1.2.2.1.1")  -- ifIndex
local if_descr_oid = inst("1.3.6.1.2.1.2.2.1.2")  -- ifDescr

-- Obter informações da interface
local interface_index = snmp.get(if_index_oid)
local interface_name = snmp.get(if_descr_oid)

print("Monitorando interface:", interface_name, "(índice", interface_index, ")")

-- Coletar estatísticas
local in_octets = snmp.get(inst("1.3.6.1.2.1.2.2.1.10"))  -- ifInOctets
local out_octets = snmp.get(inst("1.3.6.1.2.1.2.2.1.16")) -- ifOutOctets
local in_errors = snmp.get(inst("1.3.6.1.2.1.2.2.1.14"))  -- ifInErrors
local out_errors = snmp.get(inst("1.3.6.1.2.1.2.2.1.20")) -- ifOutErrors

-- Calcular diferenças desde última leitura
local time_elapsed = lapsed(inst("1.3.6.1.2.1.2.2.1.10"))
local prev_in = prev(inst("1.3.6.1.2.1.2.2.1.10"))
local prev_out = prev(inst("1.3.6.1.2.1.2.2.1.16"))
```

```
local in_rate = (in_octets - prev_in) / time_elapsed
local out_rate = (out_octets - prev_out) / time_elapsed

print("Taxa de entrada:", in_rate, "bytes/seg")
print("Taxa de saída:", out_rate, "bytes/seg")
print("Erros de entrada:", in_errors)
print("Erros de saída:", out_errors)
```

Inventário de Interfaces com Walk:

```
-- Listar todas as interfaces com walk
local interfaces = snmp.walk("1.3.6.1.2.1.2.2.1.2", 300)  -- ifDescr com cache de 5 minutos

print("=== Inventário de Interfaces ===")
for oid, ifname in pairs(interfaces) do
    -- Extrair índice da interface do OID
    local index = string.match(oid, "(%d+)$")

    -- Obter tipo e status da interface
    local iftype = snmp.get("1.3.6.1.2.1.2.2.1.3." .. index)   -- ifType
    local ifstatus = snmp.get("1.3.6.1.2.1.2.2.1.8." .. index)  -- ifOperStatus

    local status_text = "DOWN"
    if ifstatus == 1 then status_text = "UP" end

    print(string.format("Interface %s: %s (Tipo: %d, Status: %s)",
                        index, ifname, iftype, status_text))
end

print("Total de interfaces:", snmp.count("1.3.6.1.2.1.2.2.1.2"))
```

Monitoramento de Uso de CPU com Múltiplas Instâncias:

```
-- Usar walk para obter todas as CPUs
local cpu_oids = snmp.walk("1.3.6.1.2.1.25.3.3.1.2", 30)  -- hrProcessorLoad
```

```
local total_load = 0
local cpu_count = 0

for oid, load in pairs(cpu_oids) do
    cpu_count = cpu_count + 1
    total_load = total_load + load

    local cpu_index = string.match(oid, "(%d+)$")
    print(string.format("CPU %d: %d%%", cpu_index, load))
end

if cpu_count > 0 then
    local avg_load = total_load / cpu_count
    print(string.format("Média de uso de CPU: %.1f%%", avg_load))
end
```

Revision #1
Created 3 February 2026 16:45:02 by Marc
Updated 3 February 2026 16:45:14 by Marc