Módulo WMI

O módulo **wmi** fornece funções para interagir com o Windows Management Instrumentation (WMI), permitindo consultar informações do sistema, hardware, software e configurações em máquinas Windows remotas ou locais. Este módulo suporta tanto a execução cross-platform (usando o utilitário `wmic`) quanto a execução nativa no Windows (usando a API COM).

Funções Disponíveis

1. `wmi.buildwql(instance_id, table, fields)`

Constrói uma consulta WQL (WMI Query Language) a partir de parâmetros simplificados.

Parâmetros:

- `instance_id` (string opcional): Identificador de instância no formato "Tabela|Campo|Valor"
- `table` (string opcional): Nome da tabela WMI (usado quando não há instance_id)
- `fields` (tabela): Lista de campos a serem selecionados

Retorno:

- `string`: Consulta WQL formatada

Exceções:

- Lança erro se nenhum dos parâmetros `instance_id` ou `table` for fornecido
- Lança erro se `instance_id` estiver em formato inválido

Exemplos:

```
-- Consulta simples a uma tabela
local wql1 = wmi.buildwql(nil, "Win32_OperatingSystem", {"Caption", "Version", "BuildNumber"})
-- Resultado: "select Caption,Version,BuildNumber from Win32_OperatingSystem"

-- Consulta a uma instância específica
local wql2 = wmi.buildwql("Win32_Process|Name|explorer.exe", nil, {"ProcessId",
```

```
"WorkingSetSize"})
-- Resultado: "select ProcessId,WorkingSetSize from Win32_Process where Name =
\"explorer.exe\""

-- Consulta com múltiplos campos
local wql3 = wmi.buildwql(nil, "Win32_LogicalDisk", {"DeviceID", "Size", "FreeSpace",
"FileSystem"})
-- Resultado: "select DeviceID,Size,FreeSpace,FileSystem from Win32_LogicalDisk"
```

2. `wmi.exec(config, wql, namespace)`

Executa uma consulta WMI usando o utilitário `wmic`

Parâmetros:

- `config` (tabela): Configuração de conexão contendo:
  - `net.address` (string): Endereço IP ou hostname do alvo
  - `wmi.username` (string): Nome de usuário para autenticação
  - `wmi.password` (string): Senha para autenticação
  - `wmi.timeout` (número opcional, padrão: 10): Timeout em segundos
- `wql` (string): Consulta WQL a ser executada
- `namespace` (string opcional): Namespace WMI (padrão: "root\cimv2")

Retorno:

- `tabela`: Array de resultados, onde cada elemento é uma tabela com pares campo-valor

Nota: Em vez de criar manualmente uma tabela de configuração, você pode usar a tabela global `params` que já contém todos os campos necessários (`net.address`, `wmi.username`, `wmi.password`, etc.). Esta tabela é automaticamente disponibilizada pelo sistema quando o script é executado no contexto de um dispositivo gerenciado.

Exemplo usando `params`:

```
-- A tabela 'device' já contém as credenciais e endereço do dispositivo alvo
local success, results = pcall(function()
    return wmi.exec(params, wql, "root\\cimv2")
end)
```

Exemplo prático:

```
-- Consulta simplificada usando a tabela device
local wql = "select Caption,Version from Win32_OperatingSystem"
```

```
local results = wmi.exec(device, wql)

-- Para consultas locais no próprio dispositivo
if device["net.address"] == "127.0.0.1" or device["net.address"] == "localhost" then
    -- Pode-se usar exec_native para melhor performance
    if wmi.exec_native then
        results = wmi.exec_native(device, wql)
    end
end
```

Exceções:

- Lança erro se a conexão falhar
- Lança erro "Timeout" se a consulta exceder o tempo limite
- Lança erro se o utilitário `wmic` retornar código de erro

Exemplos:

```
-- Configuração básica
local config = {
    ["net.address"] = "192.168.1.100",
    ["wmi.username"] = "Administrator",
    ["wmi.password"] = "senha123",
    ["wmi.timeout"] = 15
}

-- Consulta informações do sistema operacional
local wql = "select Caption,Version,BuildNumber,OSArchitecture from Win32_OperatingSystem"
local success, results = pcall(function()
    return wmi.exec(config, wql, "root\\cimv2")
end)

if success then
    for _, row in ipairs(results) do
        print("Sistema: " .. row.Caption)
        print("Versão: " .. row.Version)
        print("Build: " .. row.BuildNumber)
        print("Arquitetura: " .. row.OSArchitecture)
    end
else
    print("Erro na consulta WMI: " .. results)
```

```
end

-- Consulta processos em execução
local process_wql = "select Name,ProcessId,WorkingSetSize,CommandLine from Win32_Process"
local process_results = wmi.exec(config, process_wql)

-- Consulta discos lógicos
local disk_wql = [[
select DeviceID,Size,FreeSpace,FileSystem
from Win32_LogicalDisk
where DriveType = 3
]]
local disk_results = wmi.exec(config, disk_wql, nil)  -- namespace padrão
```

3. `wmi.exec_native(config, wql, namespace)` (Apenas no agente Windows)

Executa uma consulta WMI usando a API nativa do Windows (COM). Esta função está disponível apenas em sistemas Windows e oferece melhor performance e integração.

Parâmetros:

- `config` (tabela): Configuração de conexão (ignorada na execução local)
- `wql` (string): Consulta WQL a ser executada
- `namespace` (string opcional): Namespace WMI

Retorno:

- `tabela`: Array de resultados, onde cada elemento é uma tabela com pares campo-valor

Exceções:

- Lança erro se a API COM falhar
- Lança erro se a consulta for inválida

Exemplos:

```
-- Apenas funciona em Windows
if wmi.exec_native then
    -- Consulta local (config é ignorado)
```

```
    local config = nil  -- Deixar vazio para consultas do agente
    local wql = "select Name,Manufacturer,Model from Win32_ComputerSystem"

    local success, results = pcall(function()
        return wmi.exec_native(config, wql)
    end)

    if success and #results > 0 then
        local computer = results[1]
        print("Computador: " .. computer.Name)
        print("Fabricante: " .. computer.Manufacturer)
        print("Modelo: " .. computer.Model)
    end
end
```

Informações Adicionais

1. **Suporte Cross-Platform**

- A função `exec` usa o utilitário `wmic` que funciona em sistemas Linux
- Permite consultar máquinas Windows remotamente
- Esta funcionalidade é legada, pois versões mais recentes do Windows não permitem conexões WMI remotas.

2. **Execução Nativa no Windows**

- A função `exec_native` oferece melhor performance em sistemas Windows
- Não requer autenticação para consultas locais
- Usa a API COM do Windows diretamente

3. **Timeout Configurável**

- Timeout padrão de 10 segundos
- Configurável via parâmetro `wmi.timeout` na configuração

Melhores Práticas

1. **Otimização de Consultas**

```
-- RUIM: Seleciona todas as colunas
local bad_wql = "select * from Win32_Process"

-- BOM: Seleciona apenas colunas necessárias
local good_wql = "select Name,ProcessId,WorkingSetSize from Win32_Process"

-- MELHOR: Adiciona filtros para reduzir resultados
local best_wql = [[
select Name,ProcessId,WorkingSetSize
from Win32_Process
where WorkingSetSize > 10485760  -- > 10MB
]]
```

4. `wmi.exec(config, wql, namespace, replace_backslash)`

Versão estendida da função `exec` com suporte a timeout e opção para substituir barras invertidas.

Parâmetros:

- **config** (tabela): Configuração de conexão
- **wql** (string): Consulta WQL a ser executada
- **namespace** (string, opcional): Namespace WMI (usa `wmi._namespace` ou `params.wmiNamespace` se não especificado)
- **replace_backslash** (booleano, opcional): Se `true`, substitui `\` por `\\` na WQL (padrão: `true`)

Retorno:

- **tabela**: Resultados da consulta WMI

Comportamento:

- Suporta execução via probe WMI quando `params["wmi.type"] == 0`

- Para localhost (`127.0.0.1`), usa execução nativa

Exemplo de Uso:

```
-- Executar com timeout configurado
-- params.wmiTimeout = 10 (10 segundos)

local config = {
    address = "192.168.1.100",
    username = "administrator",
    password = "senha123"
}

local wql = "select Name, ProcessId, WorkingSetSize from Win32_Process"
local results = wmi.exec(config, wql, "root\\cimv2")

for _, process in ipairs(results) do
    print(string.format("Processo: %s (PID: %d, Memória: %d bytes)",
                        process.Name, process.ProcessId, process.WorkingSetSize))
end
```

5. `wmi.query(wmiobj, ...)`

Consulta simplificada a uma tabela WMI sem instância específica.

Parâmetros:

- **wmiobj** (string): Nome da tabela WMI
- **...** (strings): Campos a serem selecionados (pode incluir `namespace=...`)

Retorno:

- **qualquer tipo**: Valor único se apenas um campo for selecionado, tabela se múltiplos campos

Comportamento:

- Constrói WQL automaticamente com `wmi.buildwql`
- Armazena resultado internamente para uso com `prev` e `lapsed`
- Suporta especificação de namespace via `namespace=` no início dos campos

Exemplo de Uso:

```
-- Consultar informações do sistema operacional
local os_name = wmi.query("Win32_OperatingSystem", "Caption")
print("Sistema Operacional:", os_name)

-- Consultar múltiplos campos
local disk_info = wmi.query("Win32_LogicalDisk", "DeviceID", "Size", "FreeSpace")
for _, disk in ipairs(disk_info) do
    local used_percent = 100 - (disk.FreeSpace / disk.Size * 100)
    print(string.format("Disco %s: %.1f%% usado", disk.DeviceID, used_percent))
end

-- Consultar com namespace específico
local cluster_info = wmi.query("MSCluster_Cluster", "namespace=root\\MSCluster", "Name",
"State")
```

6. `wmi.queryinst(...)`

Consulta WMI para instância específica definida em `params.InstanceId`.

Parâmetros:

- **...** (strings): Campos a serem selecionados (pode incluir `namespace=...`)

Retorno:

- **qualquer tipo**: Valor único se apenas um campo for selecionado, tabela se múltiplos campos

Comportamento:

- Usa `params.InstanceId` para construir consulta de instância
- Lança erro se instância não for encontrada
- Armazena resultado em `_mem_store` para uso com `prev` e `lapsed`

Exemplo de Uso:

```
-- params.InstanceId = "Win32_Process|Name|explorer.exe"

-- Consultar informações do processo explorer.exe
local pid = wmi.queryinst("ProcessId")
local memory = wmi.queryinst("WorkingSetSize")
```

```
print(string.format("Explorer.exe - PID: %d, Memória: %d bytes", pid, memory))

-- Consultar múltiplos campos
local process_info = wmi.queryinst("ProcessId", "WorkingSetSize", "ThreadCount", "Priority")
```

7. `wmi.prev(wmiobj, ...)`

Obtém o valor anterior de uma consulta WMI.

Parâmetros:

- **wmiobj** (string): Nome da tabela WMI
- **...** (strings): Campos da consulta original

Retorno:

- **qualquer tipo**: Valor anterior armazenado

Comportamento:

- Reconstrói a WQL original
- Busca valor em `store.get("wmi.value." .. wql)`
- Retorna valor único se consulta original retornou único valor

Exemplo de Uso:

```
-- Obter valor atual
local current_memory = wmi.query("Win32_OperatingSystem", "TotalVisibleMemorySize")

-- Obter valor anterior
local previous_memory = wmi.prev("Win32_OperatingSystem", "TotalVisibleMemorySize")

-- Calcular diferença
local memory_diff = current_memory - previous_memory
print("Variação de memória:", memory_diff, "bytes")
```

8. `wmi.previnst(...)`

Obtém o valor anterior de uma consulta WMI de instância.

Parâmetros:

- **...** (strings): Campos da consulta original

Retorno:

- **qualquer tipo**: Valor anterior armazenado

Exemplo de Uso:

```
-- params.InstanceId = "Win32_Process|Name|svchost.exe"

-- Obter uso atual de CPU
local current_cpu = wmi.queryinst("PercentProcessorTime")

-- Obter uso anterior
local previous_cpu = wmi.previnst("PercentProcessorTime")

-- Calcular variação
local cpu_change = current_cpu - previous_cpu
print("Variação no uso de CPU:", cpu_change, "%")
```

9. `wmi.lapsed(wmiobj, ...)`

Obtém o tempo decorrido desde a última consulta WMI.

Parâmetros:

- **wmiobj** (string): Nome da tabela WMI
- **...** (strings): Campos da consulta original

Retorno:

- **número**: Tempo em segundos desde última consulta

Exemplo de Uso:

```
-- Calcular taxa de transferência de disco
local current_reads = wmi.query("Win32_PerfRawData_PerfDisk_LogicalDisk",
"DiskReadBytesPerSec")
local previous_reads = wmi.prev("Win32_PerfRawData_PerfDisk_LogicalDisk",
```

```
"DiskReadBytesPerSec")
local time_elapsed = wmi.lapsed("Win32_PerfRawData_PerfDisk_LogicalDisk",
"DiskReadBytesPerSec")

local read_rate = (current_reads - previous_reads) / time_elapsed
print("Taxa de leitura de disco:", read_rate, "bytes/segundo")
```

10. `wmi.lapsedinst(...)`

Obtém o tempo decorrido desde a última consulta WMI de instância.

Parâmetros:

- **...** (strings): Campos da consulta original

Retorno:

- **número**: Tempo em segundos desde última consulta

Exemplo de Uso:

```
-- params.InstanceId = "Win32_PerfRawData_PerfDisk_LogicalDisk|Name|C:"

-- Calcular taxa de escrita para disco C:
local current_writes = wmi.queryinst("DiskWriteBytesPerSec")
local previous_writes = wmi.previnst("DiskWriteBytesPerSec")
local time_elapsed = wmi.lapsedinst("DiskWriteBytesPerSec")

local write_rate = (current_writes - previous_writes) / time_elapsed
print("Taxa de escrita no disco C:", write_rate, "bytes/segundo")
```

11. `wmi.diff(typ, lhs, rhs)`

Calcula a diferença entre dois valores WMI, 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:

- Usa a mesma implementação que `snmp.diff`
- Trata rollover de contadores de 32 e 64 bits
- Sinaliza `RepeatPrevValue` se a diferença for negativa

Exemplo de Uso:

```
-- Calcular diferença para contador de 64 bits
local current_bytes = wmi.queryinst("DiskReadBytesPerSec")
local prev_bytes = wmi.previnst("DiskReadBytesPerSec")
local bytes_diff = wmi.diff(64, current_bytes, prev_bytes)

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

Exemplos Completos

Monitoramento de Processo Específico:

```
-- Configurar instância para monitorar processo específico
-- params.InstanceId = "Win32_PerfRawData_PerfProc_Process|Name|chrome.exe"

-- Obter métricas atuais
local cpu_usage = wmi.queryinst("PercentProcessorTime")
local memory_usage = wmi.queryinst("WorkingSetPrivate")
local thread_count = wmi.queryinst("ThreadCount")

-- Calcular variações
local prev_cpu = wmi.previnst("PercentProcessorTime")
local prev_memory = wmi.previnst("WorkingSetPrivate")
local time_elapsed = wmi.lapsedinst("PercentProcessorTime")

local cpu_delta = wmi.diff(32, cpu_usage, prev_cpu)
local memory_delta = memory_usage - prev_memory
```

```
print(string.format("Chrome.exe - CPU: %d%%, Memória: %d bytes, Threads:
%d",
                    cpu_delta / time_elapsed, memory_delta, thread_count))
```

Inventário de Hardware com Cache:

```
-- Função para obter informações de hardware com cache
local function get_hardware_info()
    local cache_key = "hardware_info_" .. params.device.address
    local cached = cache.get(cache_key)

    if cached then
        return cached
    end

    -- Coletar informações diversas
    local hardware_info = {
        os = wmi.query("Win32_OperatingSystem", "Caption", "Version", "BuildNumber"),
        cpu = wmi.query("Win32_Processor", "Name", "NumberOfCores", "MaxClockSpeed"),
        memory = wmi.query("Win32_ComputerSystem", "TotalPhysicalMemory"),
        disks = wmi.query("Win32_LogicalDisk", "DeviceID", "Size", "FreeSpace",
"FileSystem")
    }

    -- Armazenar em cache por 1 hora
    cache.put(cache_key, hardware_info, 3600)

    return hardware_info
end

-- Usar informações em cache
local info = get_hardware_info()
print("Sistema:", info.os.Caption, info.os.Version)
print("Processador:", info.cpu.Name, "(" .. info.cpu.NumberOfCores .. " núcleos)")
print("Memória total:", info.memory / (1024*1024*1024), "GB")
```

Revision #1
Created 3 February 2026 16:47:57 by Marc
Updated 3 February 2026 16:48:09 by Marc