Módulo Registry

O módulo **registry** fornece um sistema de armazenamento de dados persistente e compartilhado entre execuções de scripts Lua. Este módulo é útil para manter estado entre diferentes execuções, compartilhar dados entre scripts e implementar mecanismos de cache e configuração persistente.

- Armazenamento chave-valor persistente
- Timestamp automático para cada entrada
- Thread-safe (seguro para uso concorrente)
- Dados compartilhados entre todos os scripts Lua
- Persistência entre reinícios do agente

Funções Disponíveis

1. `registry.put(chave, valor)`

Armazena um valor no registry com a chave especificada.

Parâmetros:

- **chave** (string): Identificador único para o valor
- **valor** (qualquer tipo Lua): Valor a ser armazenado (string, número, booleano, tabela, nil)

Retorno:

- **nil**: A função não retorna valor

Comportamento:

1. Armazena o valor associado à chave
2. Registra automaticamente o timestamp atual (UTC)
3. Sobrescreve qualquer valor existente com a mesma chave
4. Aceita qualquer tipo de dado Lua suportado pelo sistema de valores do Monagent

Exemplo de Uso:

```
-- Armazenar valores básicos
```

```
registry.put("ultima_execucao", os.time())
registry.put("hostname", "servidor-producao")
registry.put("ativo", true)
registry.put("versao", 2.5)

-- Armazenar tabelas complexas
local config = {
    timeout = 30,
    retries = 3,
    servidores = {"srv1", "srv2", "srv3"},
    limites = {cpu = 80, memoria = 90, disco = 95}
}
registry.put("configuracao_monitoramento", config)

-- Armazenar resultados de processamento
local metricas = {
    cpu_usage = 45.6,
    memory_usage = 78.3,
    disk_usage = 62.1,
    timestamp = os.time()
}
registry.put("metricas_recentes", metricas)

-- Sobrescrever valor existente
registry.put("contador", 1)
-- ... mais tarde ...
local valor, timestamp = registry.get("contador")
registry.put("contador", (valor or 0) + 1)
```

2. `registry.get(chave)`

Recupera um valor do registry pela chave especificada.

Parâmetros:

- **chave** (string): Chave do valor a ser recuperado

Retorno:

- **tuple**: `(valor, timestamp)` onde:

- **valor** (qualquer tipo ou nil): Valor armazenado, ou `nil` se a chave não existir
- **timestamp** (número ou nil): Timestamp Unix (segundos) da última atualização, ou `nil` se a chave não existir

Comportamento:

1. Retorna o valor e timestamp se a chave existir
2. Retorna `(nil, nil)` se a chave não existir
3. Preserva o tipo original do valor armazenado
4. O timestamp é em segundos desde a epoch Unix (UTC)

Exemplo de Uso:

```
-- Recuperar valor simples
local valor, timestamp = registry.get("config_timeout")
if valor then
    print("Timeout configurado:", valor, "desde", os.date("%H:%M:%S", timestamp))
else
    print("Configuração não encontrada")
end

-- Recuperar tabela
local config, ts = registry.get("configuracao_sistema")
if config then
    print("Configuração carregada:")
    for chave, valor in pairs(config) do
        print("  ", chave, "=", valor)
    end
    print("Última atualização:", os.date("%Y-%m-%d %H:%M:%S", ts))
end

-- Verificar existência antes de usar
local dados, timestamp = registry.get("dados_processamento")
if dados then
    -- Processar dados existentes
    processar_dados(dados)
else
    -- Inicializar novos dados
    dados = inicializar_dados()
    registry.put("dados_processamento", dados)
end
```

```
-- Usar valor padrão se não existir
local limite, _ = registry.get("limite_cpu")
limite = limite or 80  -- Valor padrão 80% se não configurado
```

3. `registry.delete(chave)`

Remove uma entrada do registry pela chave especificada.

Parâmetros:

- **chave** (string): Chave da entrada a ser removida

Retorno:

- **nil**: A função não retorna valor

Comportamento:

1. Remove completamente a entrada do registry
2. Não faz nada se a chave não existir
3. Libera a memória associada à chave
4. Operação atômica e thread-safe

Exemplo de Uso:

```
-- Remover entrada específica
registry.delete("cache_temporario")
print("Cache temporário removido")

-- Limpar todas as entradas de um prefixo
local function limpar_prefixo(prefixo)
    -- Nota: O registry não suporta listagem direta
    -- Em um caso real, você precisaria manter uma lista de chaves
    log.info("Limpando entradas com prefixo:", prefixo)

    -- Exemplo com chaves conhecidas
    local chaves_conhecidas = {
        prefixo .. "_cache_dns",
        prefixo .. "_cache_http",
        prefixo .. "_metricas",
        prefixo .. "_config"
```

```
    }

    for _, chave in ipairs(chaves_conhecidas) do
        registry.delete(chave)
        log.debug("Removido:", chave)
    end
end

-- Limpar entradas expiradas
local function limpar_expirados(ttl_segundos)
    local agora = os.time()
    -- Nota: Novamente, sem listagem direta, precisamos de estratégia
    log.info("Limpeza de expirados não suportada diretamente")
    log.info("Use timestamps nas chaves ou mantenha lista de chaves")
end

-- Remover após uso
local function processar_e_limpar(chave)
    local dados, timestamp = registry.get(chave)

    if dados then
        -- Processar dados
        local resultado = processar(dados)

        -- Remover após processamento
        registry.delete(chave)
        log.info("Dados processados e removidos:", chave)

        return resultado
    else
        log.warn("Chave não encontrada:", chave)
        return nil
    end
end
```

Revision #1
Created 3 February 2026 16:44:42 by Marc
Updated 3 February 2026 16:44:55 by Marc