Módulo TCP

Este módulo fornece funcionalidades para conexões TCP e TCP+TLS a partir de scripts Lua. Oferece suporte para comunicação com sockets brutos, leitura/escrita de strings, JSON e pacotes JSON com tamanho prefixado.

O módulo **tcp** permite:

- Conexões TCP simples com timeout configurável
- Conexões TLS
- Leitura e escrita de strings simples
- Leitura e escrita de JSON
- Leitura e escrita de pacotes JSON com tamanho prefixado (para comunicação binária segura)
- Resolução DNS automática para conexões TLS

Funções Disponíveis

`tcp.connect(host, port, [timeout_secs])`

Estabelece uma conexão TCP simples com um servidor remoto.

Parâmetros:

- `host` (string): Endereço do host ou IP do servidor
- `port` (number): Porta TCP para conexão
- `timeout_secs` (number, opcional): Timeout em segundos (padrão: 5)

Valor de retorno:

- Objeto `Connection` que pode ser usado para leitura/escrita
- Lança erro em caso de falha ou timeout

Exemplo:

```
-- Conectar a um servidor na porta 8080 com timeout de 10 segundos
local conn = tcp.connect("servidor.exemplo.com", 8080, 10)
```

```
-- Conectar com timeout padrão (5 segundos)
local conn2 = tcp.connect("192.168.1.100", 3000)
```

`tcp.connect_tls(host, port, [timeout_secs])`

Estabelece uma conexão TLS (SSL) com um servidor remoto.

Parâmetros:

- `host` (string): Endereço do host do servidor
- `port` (number): Porta TLS para conexão
- `timeout_secs` (number, opcional): Timeout em segundos (padrão: 5)

Valor de retorno:

- Objeto `TlsConnection` que pode ser usado para leitura/escrita
- Lança erro em caso de falha ou timeout

Exemplo:

```
-- Conectar via TLS na porta 443
local conn = tcp.connect_tls("api.exemplo.com", 443)

-- Conectar com timeout personalizado
local conn2 = tcp.connect_tls("servidor-seguro.com", 8443, 15)
```

`tcp.send(data)`

Envia dados através da última conexão TCP aberta com a função `connect()`. Esta função é mantida para compatibilidade com versões anteriores. Para maior clareza e controle, prefira a forma `local conn = connect("host", porta)` e use os métodos do objeto de conexão retornado (ex: `conn:write_json_packet()`).

Parâmetros:

- `data` (string): Dados a serem enviados

Valor de retorno:

- `nil` em caso de sucesso
- Lança erro se não houver conexão ativa

Nota: Esta função usa a última conexão TCP aberta com a função `connect()`. É útil para scripts simples que mantêm uma única conexão.

Exemplo:

```
-- Primeiro estabelece uma conexão
local conn = tcp.connect("servidor.exemplo.com", 8080)


-- Envia dados
send("GET / HTTP/1.1\r\nHost: servidor.exemplo.com\r\n\r\n")
```

`tcp.recv()`

Recebe dados da última conexão TCP aberta com a função `connect()` e armazenada no registro Lua. Esta função é mantida para compatibilidade com versões anteriores. Para maior clareza e controle, prefira a forma `local conn = connect("host", porta)` e use os métodos do objeto de conexão retornado (ex: `conn:read_str()` ou `conn:read_json_packet()`).

Parâmetros:

- Nenhum

Valor de retorno:

- `string` com os dados recebidos
- Lança erro se não houver conexão ativa

Nota:

- Esta função usa a última conexão TCP aberta com a função `connect()`. É útil para scripts simples que mantêm uma única conexão.
- Lê até 8192 bytes por chamada. Para leitura completa, pode ser necessário chamar múltiplas vezes.

Exemplo:

```
-- Recebe resposta
local resposta = tcp.recv()
print("Resposta recebida:", resposta)
```

Métodos dos Objetos Connection

Os objetos retornados por `connect()` e `connect_tls()` possuem os seguintes métodos:

`conn:read_str()`

Lê uma string completa do socket até EOF.

Valor de retorno:

- `string` com os dados lidos
- Lança erro em caso de falha de leitura

Exemplo:

```
local conn = tcp.connect("servidor.exemplo.com", 8080)
local dados = conn:read_str()
print("Dados recebidos:", dados)
```

`conn:read_json()`

Lê uma string do socket e a interpreta como JSON.

Valor de retorno:

- Valor Lua decodificado do JSON
- Lança erro se os dados não forem JSON válido

Exemplo:

```
local conn = tcp.connect("api.exemplo.com", 3000)
local dados_json = conn:read_json()

-- Acessar dados decodificados
print("Status:", dados_json.status)
print("Mensagem:", dados_json.message)
```

`conn:read_json_packet()`

Lê um pacote JSON com tamanho prefixado (formato binário).

Formato do pacote:

1. 4 bytes (uint32): Tamanho dos dados JSON

2. N bytes: Dados JSON serializados

Limite de tamanho: 512KB (512.000 bytes)

Valor de retorno:

- Valor Lua decodificado do JSON
- Lança erro se o pacote for muito grande ou JSON inválido

Exemplo:

```
local conn = tcp.connect("servidor-binario.com", 9000)
local pacote = conn:read_json_packet()
print("Pacote recebido:", pacote)
```

`conn:write_json_packet(packet)`

Escreve um pacote JSON com tamanho prefixado (formato binário).

Parâmetros:

- `packet` (qualquer valor Lua): Dados a serem serializados como JSON

Valor de retorno:

- `nil` em caso de sucesso
- Lança erro em caso de falha de serialização ou escrita

Exemplo:

```
local conn = connect("servidor-binario.com", 9000)

-- Enviar um pacote JSON
local dados = {
    comando = "atualizar",
    id = 123,
    valores = {10, 20, 30}
}
conn:write_json_packet(dados)
```

Informações Adicionais

Timeout Configurável

Todas as funções de conexão aceitam timeout personalizado. Se não especificado, usa 5 segundos como padrão.

Resolução DNS Automática

A função `connect_tls()` resolve automaticamente o nome do host para um endereço IP antes de estabelecer a conexão.

Tamanho Máximo de Pacote

Pacotes JSON com tamanho prefixado têm um limite de 512KB para prevenir ataques de negação de serviço.

Exemplos de Uso

Comunicação HTTP Simples

```
function fazer_requisicao_http(host, porta, caminho)
    local conn = connect(host, porta, 10)

    -- Enviar requisição HTTP
    local requisicao = string.format(
        "GET %s HTTP/1.1\r\nHost: %s\r\nConnection: close\r\n\r\n",
        caminho, host
    )

    -- Usar o método send (conexão está no registro)
    send(requisicao)

    -- Ler resposta
    local resposta = recv()

    -- Extrair corpo da resposta (simplificado)
    local corpo = resposta:match("\r\n\r\n(.+)$")
```

```
    return corpo
end
```

Cliente de API JSON

```
function consultar_api_json(endpoint, dados)
    local conn = connect_tls("api.exemplo.com", 443)

    -- Enviar dados como pacote JSON
    conn:write_json_packet({
        endpoint = endpoint,
        dados = dados,
        timestamp = os.time()
    })

    -- Aguardar resposta
    local resposta = conn:read_json_packet()

    return resposta
end

-- Exemplo de uso
local resultado = consultar_api_json("/usuarios", {id = 123})
if resultado.success then
    print("Usuário:", resultado.usuario.nome)
end
```

Monitoramento de Serviço TCP

```
function verificar_servico_tcp(host, porta)
    local inicio = os.time()
    local sucesso, conn = pcall(connect, host, porta, 5)

    if sucesso then
        local tempo_resposta = os.time() - inicio
```

```
        -- Testar comunicação básica
        conn:write_json_packet({ping = true})
        local resposta = conn:read_json_packet()

        if resposta and resposta.pong then
            return {
                status = "online",
                tempo_resposta = tempo_resposta,
                versao = resposta.versao
            }
        end
    end

    return {
        status = "offline",
        erro = conn  -- conn contém a mensagem de erro quando pcall falha
    }
end
```

Comunicação Bidirecional

```
function chat_client(host, porta)
    local conn = connect(host, porta)

    -- Thread para receber mensagens
    local function receber_mensagens()
        while true do
            local mensagem = conn:read_json_packet()
            print("Recebido:", mensagem.texto)
        end
    end

    -- Thread para enviar mensagens (simplificado)
    local function enviar_mensagens()
        while true do
            io.write("Digite mensagem: ")
            local texto = io.read()
            if texto == "sair" then break end

            conn:write_json_packet({
```

```
                tipo = "mensagem",
                texto = texto,
                timestamp = os.time()
            })
        end
    end

    -- Em um ambiente real, isso seria feito com corrotinas
    -- Esta é uma simplificação para demonstração
end
```

Revision #1
Created 3 February 2026 16:46:59 by Marc
Updated 3 February 2026 16:47:10 by Marc