**Biblioteca XML-RPC do Python (Built-in Library)**

## **Introdução**

XML-RPC (Remote Procedure Call usando XML) é um protocolo leve que permite a execução remota de métodos através de chamadas HTTP. O Python fornece suporte nativo para XML-RPC por meio do módulo xmlrpc, que inclui xmlrpc.server para servidores e xmlrpc.client para clientes.

O XML-RPC funciona convertendo chamadas de função em XML e transmitindo-as via HTTP. O servidor recebe a requisição, processa-a e responde com um XML contendo o resultado.

## **1. Principais Classes e Métodos**

A biblioteca xmlrpc do Python possui duas classes principais:

### **1.1 Servidor (xmlrpc.server)**

* SimpleXMLRPCServer: Cria um servidor XML-RPC.
  + register\_function(func, name=None): Registra uma função para chamadas remotas.
    - func: A função Python a ser registrada.
    - name (opcional): Nome pelo qual a função será acessada remotamente. Se não for fornecido, usa o nome original da função.
  + register\_instance(obj): Registra todas as funções de um objeto (classe).
    - obj: Instância de uma classe cujos métodos serão expostos remotamente.
  + serve\_forever(): Mantém o servidor rodando indefinidamente, processando requisições.
* SimpleXMLRPCRequestHandler: Lida com as requisições recebidas.

### **1.2 Cliente (xmlrpc.client)**

* ServerProxy: Conecta-se a um servidor XML-RPC remoto.
  + url: URL do servidor XML-RPC remoto.
  + allow\_none (opcional, padrão False): Se True, permite valores None em chamadas.
  + use\_builtin\_types (opcional, padrão False): Se True, converte nativamente tipos como bytes.
  + Métodos remotos podem ser chamados diretamente pelo objeto ServerProxy.

## **2. Criando um Servidor XML-RPC**

O servidor XML-RPC escuta chamadas remotas e executa métodos definidos.

### **Exemplo de servidor:**

from xmlrpc.server import SimpleXMLRPCServer

def soma(a, b):

return a + b

def multiplica(a, b):

return a \* b

server = SimpleXMLRPCServer(("localhost", 8000))

print("Servidor XML-RPC rodando em localhost:8000")

server.register\_function(soma, "soma")

server.register\_function(multiplica, "multiplica")

server.serve\_forever()

Neste exemplo, criamos um servidor XML-RPC que expõe duas funções, soma e multiplica, para chamadas remotas.

## **3. Criando um Cliente XML-RPC**

O cliente pode se conectar ao servidor XML-RPC e chamar métodos remotamente.

### **Exemplo de cliente:**

from xmlrpc.client import ServerProxy

proxy = ServerProxy("http://localhost:8000/")

print("Soma:", proxy.soma(5, 3))

print("Multiplicação:", proxy.multiplica(4, 2))

O ServerProxy permite que o cliente chame funções disponíveis no servidor remoto.

## **4. Exemplo de Servidor com Classe**

Podemos organizar os métodos dentro de uma classe para maior modularidade.

from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCRequestHandler

class Operacoes:

def soma(self, a, b):

return a + b

def multiplica(self, a, b):

return a \* b

server = SimpleXMLRPCServer(("localhost", 8000), requestHandler=SimpleXMLRPCRequestHandler)

server.register\_instance(Operacoes())

print("Servidor XML-RPC rodando em localhost:8000")

server.serve\_forever()

Aqui, register\_instance(Operacoes()) registra todas as funções da classe Operacoes.

## **5. Métodos Úteis no XML-RPC**

### **No Servidor**

* register\_function(func, name=None): Registra uma única função.
  + func: A função a ser disponibilizada remotamente.
  + name (opcional): Nome pelo qual a função será acessada no cliente.
* register\_instance(obj): Registra todas as funções de um objeto.
  + obj: Instância contendo métodos que podem ser chamados remotamente.
* serve\_forever(): Mantém o servidor rodando.
* handle\_request(): Processa uma única requisição e depois continua disponível.
* shutdown(): Para o servidor.

### **No Cliente**

* ServerProxy(url, allow\_none=False, use\_builtin\_types=False): Cria um proxy para o servidor XML-RPC.
  + url: Endereço do servidor remoto.
  + allow\_none: Se True, permite valores None em chamadas XML-RPC.
  + use\_builtin\_types: Se True, converte bytes e outros tipos automaticamente.
* Chamadas remotas: Os métodos do servidor podem ser chamados diretamente via proxy.metodo(), por exemplo, proxy.soma(2, 3).

## **6. Segurança e Considerações**

* Sem autenticação: XML-RPC padrão não implementa autenticação. Se necessário, use HTTPS e autenticação personalizada.
* Sem criptografia: Todas as chamadas são enviadas em texto puro. Use SSL/TLS se estiver transmitindo dados sensíveis.
* Não recomendado para alta performance: XML-RPC é mais lento que alternativas modernas como REST e gRPC.
* Portas expostas: Evite expor um servidor XML-RPC publicamente sem proteção adequada.

## **7. Conclusão**

A biblioteca XML-RPC do Python é uma solução simples para chamadas remotas, sendo útil para comunicação entre sistemas de forma leve e estruturada. Entretanto, para aplicações modernas, alternativas como REST e gRPC são mais flexíveis e seguras.