21.27. `xmlrpc.server` — Basic XML-RPC servers

Source code: Lib/xmlrpc/server.py

The `xmlrpc.server` module provides a basic server framework for XML-RPC servers written in Python. Servers can either be free standing, using `SimpleXMLRPCServer`, or embedded in a CGI environment, using `CGIXMLRPCRequestHandler`.

Warning: The `xmlrpc.server` module is not secure against maliciously constructed data. If you need to parse untrusted or unauthenticated data see XML vulnerabilities.

class `xmlrpc.server.`**`SimpleXMLRPCServer`**(*addr*, *requestHandler=SimpleXMLRPCRequestHandler*, *logRequests=True*, *allow_none=False*, *encoding=None*, *bind_and_activate=True*, *use_builtin_types=False*)

Create a new server instance. This class provides methods for registration of functions that can be called by the XML-RPC protocol. The *requestHandler* parameter should be a factory for request handler instances; it defaults to `SimpleXMLRPCRequestHandler`. The *addr* and *requestHandler* parameters are passed to the `socketserver.TCPServer` constructor. If *logRequests* is true (the default), requests will be logged; setting this parameter to false will turn off logging. The *allow_none* and *encoding* parameters are passed on to `xmlrpc.client` and control the XML-RPC responses that will be returned from the server. The *bind_and_activate* parameter controls whether `server_bind()` and `server_activate()` are called immediately by the constructor; it defaults to true. Setting it to false allows code to manipulate the *allow_reuse_address* class variable before the address is bound. The *use_builtin_types* parameter is passed to the `loads()` function and controls which types are processed when date/times values or binary data are received; it defaults to false.

Changed in version 3.3: The *use_builtin_types* flag was added.

class `xmlrpc.server.`**`CGIXMLRPCRequestHandler`**(*allow_none=False*, *encoding=None*, *use_builtin_types=False*)

Create a new instance to handle XML-RPC requests in a CGI environment. The *allow_none* and *encoding* parameters are passed on to `xmlrpc.client` and control the XML-RPC responses that will be returned from the server. The *use_builtin_types* parameter is passed to the `loads()` function and controls

which types are processed when date/times values or binary data are received; it defaults to false.

Changed in version 3.3: The *use_builtin_types* flag was added.

class `xmlrpc.server.`**`SimpleXMLRPCRequestHandler`**

Create a new request handler instance. This request handler supports `POST` requests and modifies logging so that the *logRequests* parameter to the `SimpleXMLRPCServer` constructor parameter is honored.

21.27.1. SimpleXMLRPCServer Objects

The `SimpleXMLRPCServer` class is based on `socketserver.TCPServer` and provides a means of creating simple, stand alone XML-RPC servers.

`SimpleXMLRPCServer.`**`register_function`**(*function*, *name=None*)

Register a function that can respond to XML-RPC requests. If *name* is given, it will be the method name associated with *function*, otherwise `function.__name__` will be used. *name* can be either a normal or Unicode string, and may contain characters not legal in Python identifiers, including the period character.

`SimpleXMLRPCServer.`**`register_instance`**(*instance*, *allow_dotted_names=False*)

Register an object which is used to expose method names which have not been registered using `register_function()`. If *instance* contains a `_dispatch()` method, it is called with the requested method name and the parameters from the request. Its API is `def _dispatch(self, method, params)` (note that *params* does not represent a variable argument list). If it calls an underlying function to perform its task, that function is called as `func(*params)`, expanding the parameter list. The return value from `_dispatch()` is returned to the client as the result. If *instance* does not have a `_dispatch()` method, it is searched for an attribute matching the name of the requested method.

If the optional *allow_dotted_names* argument is true and the instance does not have a `_dispatch()` method, then if the requested method name contains periods, each component of the method name is searched for individually, with the effect that a simple hierarchical search is performed. The value found from this search is then called with the parameters from the request, and the return value is passed back to the client.

> **Warning:** Enabling the *allow_dotted_names* option allows intruders to access your module's global variables and may allow intruders to execute arbitrary code on your machine. Only use this option on a secure, closed network.

`SimpleXMLRPCServer.`**`register_introspection_functions`**()

Registers the XML-RPC introspection functions `system.listMethods`, `system.methodHelp` and `system.methodSignature`.

`SimpleXMLRPCServer.`**`register_multicall_functions`**()

Registers the XML-RPC multicall function system.multicall.

`SimpleXMLRPCRequestHandler.`**`rpc_paths`**

An attribute value that must be a tuple listing valid path portions of the URL for receiving XML-RPC requests. Requests posted to other paths will result in a 404 "no such page" HTTP error. If this tuple is empty, all paths will be considered valid. The default value is `('/', '/RPC2')`.

21.27.1.1. SimpleXMLRPCServer Example

Server code:

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler

# Restrict to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

# Create server
with SimpleXMLRPCServer(("localhost", 8000),
                        requestHandler=RequestHandler) as server:
    server.register_introspection_functions()

    # Register pow() function; this will use the value of
    # pow.__name__ as the name, which is just 'pow'.
    server.register_function(pow)

    # Register a function under a different name
    def adder_function(x,y):
        return x + y
    server.register_function(adder_function, 'add')

    # Register an instance; all the methods of the instance are
    # published as XML-RPC methods (in this case, just 'mul').
    class MyFuncs:
        def mul(self, x, y):
            return x * y
```

```
    server.register_instance(MyFuncs())

    # Run the server's main loop
    server.serve_forever()
```

The following client code will call the methods made available by the preceding server:

```
import xmlrpc.client

s = xmlrpc.client.ServerProxy('http://localhost:8000')
print(s.pow(2,3))  # Returns 2**3 = 8
print(s.add(2,3))  # Returns 5
print(s.mul(5,2))  # Returns 5*2 = 10

# Print list of available methods
print(s.system.listMethods())
```

The following example included in the `Lib/xmlrpc/server.py` module shows a server allowing dotted names and registering a multicall function.

Warning: Enabling the *allow_dotted_names* option allows intruders to access your module's global variables and may allow intruders to execute arbitrary code on your machine. Only use this example only within a secure, closed network.

```
import datetime

class ExampleService:
    def getData(self):
        return '42'

    class currentTime:
        @staticmethod
        def getCurrentTime():
            return datetime.datetime.now()

with SimpleXMLRPCServer(("localhost", 8000)) as server:
    server.register_function(pow)
    server.register_function(lambda x,y: x+y, 'add')
    server.register_instance(ExampleService(), allow_dotted_names=True
    server.register_multicall_functions()
    print('Serving XML-RPC on localhost port 8000')
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print("\nKeyboard interrupt received, exiting.")
        sys.exit(0)
```

This ExampleService demo can be invoked from the command line:

```
python -m xmlrpc.server
```

The client that interacts with the above server is included in *Lib/xmlrpc/client.py*:

```
server = ServerProxy("http://localhost:8000")

try:
    print(server.currentTime.getCurrentTime())
except Error as v:
    print("ERROR", v)

multi = MultiCall(server)
multi.getData()
multi.pow(2,9)
multi.add(1,2)
try:
    for response in multi():
        print(response)
except Error as v:
    print("ERROR", v)
```

This client which interacts with the demo XMLRPC server can be invoked as:

```
python -m xmlrpc.client
```

21.27.2. CGIXMLRPCRequestHandler

The `CGIXMLRPCRequestHandler` class can be used to handle XML-RPC requests sent to Python CGI scripts.

`CGIXMLRPCRequestHandler.`**`register_function`**(*function*, *name=None*)

Register a function that can respond to XML-RPC requests. If *name* is given, it will be the method name associated with function, otherwise *function.__name__* will be used. *name* can be either a normal or Unicode string, and may contain characters not legal in Python identifiers, including the period character.

`CGIXMLRPCRequestHandler.`**`register_instance`**(*instance*)

Register an object which is used to expose method names which have not been registered using `register_function()`. If instance contains a `_dispatch()` method, it is called with the requested method name and the parameters from the request; the return value is returned to the client as the result. If instance does not have a `_dispatch()` method, it is searched for an attribute matching the name of the requested method; if the requested method name contains periods, each component of the method name is searched for individually, with the

effect that a simple hierarchical search is performed. The value found from this search is then called with the parameters from the request, and the return value is passed back to the client.

`CGIXMLRPCRequestHandler.`**`register_introspection_functions`**()

Register the XML-RPC introspection functions `system.listMethods`, `system.methodHelp` and `system.methodSignature`.

`CGIXMLRPCRequestHandler.`**`register_multicall_functions`**()

Register the XML-RPC multicall function `system.multicall`.

`CGIXMLRPCRequestHandler.`**`handle_request`**(*request_text=None*)

Handle an XML-RPC request. If *request_text* is given, it should be the POST data provided by the HTTP server, otherwise the contents of stdin will be used.

Example:

```
class MyFuncs:
    def mul(self, x, y):
        return x * y


handler = CGIXMLRPCRequestHandler()
handler.register_function(pow)
handler.register_function(lambda x,y: x+y, 'add')
handler.register_introspection_functions()
handler.register_instance(MyFuncs())
handler.handle_request()
```

21.27.3. Documenting XMLRPC server

These classes extend the above classes to serve HTML documentation in response to HTTP GET requests. Servers can either be free standing, using `DocXMLRPCServer`, or embedded in a CGI environment, using `DocCGIXMLRPCRequestHandler`.

class `xmlrpc.server.`**`DocXMLRPCServer`**(*addr*, *requestHandler=DocXMLRPCRequestHandler*, *logRequests=True*, *allow_none=False*, *encoding=None*, *bind_and_activate=True*, *use_builtin_types=True*)

Create a new server instance. All parameters have the same meaning as for `SimpleXMLRPCServer`; *requestHandler* defaults to `DocXMLRPCRequestHandler`.

Changed in version 3.3: The *use_builtin_types* flag was added.

class `xmlrpc.server.`**`DocCGIXMLRPCRequestHandler`**

Create a new instance to handle XML-RPC requests in a CGI environment.

class `xmlrpc.server.`**`DocXMLRPCRequestHandler`**

Create a new request handler instance. This request handler supports XML-RPC POST requests, documentation GET requests, and modifies logging so that the *logRequests* parameter to the `DocXMLRPCServer` constructor parameter is honored.

21.27.4. DocXMLRPCServer Objects

The `DocXMLRPCServer` class is derived from `SimpleXMLRPCServer` and provides a means of creating self-documenting, stand alone XML-RPC servers. HTTP POST requests are handled as XML-RPC method calls. HTTP GET requests are handled by generating pydoc-style HTML documentation. This allows a server to provide its own web-based documentation.

`DocXMLRPCServer.`**`set_server_title`**(*server_title*)

Set the title used in the generated HTML documentation. This title will be used inside the HTML “title” element.

`DocXMLRPCServer.`**`set_server_name`**(*server_name*)

Set the name used in the generated HTML documentation. This name will appear at the top of the generated documentation inside a “h1” element.

`DocXMLRPCServer.`**`set_server_documentation`**(*server_documentation*)

Set the description used in the generated HTML documentation. This description will appear as a paragraph, below the server name, in the documentation.

21.27.5. DocCGIXMLRPCRequestHandler

The `DocCGIXMLRPCRequestHandler` class is derived from `CGIXMLRPCRequestHandler` and provides a means of creating self-documenting, XML-RPC CGI scripts. HTTP POST requests are handled as XML-RPC method calls. HTTP GET requests are handled by generating pydoc-style HTML documentation. This allows a server to provide its own web-based documentation.

`DocCGIXMLRPCRequestHandler.`**`set_server_title`**(*server_title*)

Set the title used in the generated HTML documentation. This title will be used inside the HTML “title” element.

`DocCGIXMLRPCRequestHandler.`**`set_server_name`**(*server_name*)

Set the name used in the generated HTML documentation. This name will appear at the top of the generated documentation inside a “h1” element.

DocCGIXMLRPCRequestHandler.**set_server_documentation**(*server_documentation*)

Set the description used in the generated HTML documentation. This description will appear as a paragraph, below the server name, in the documentation.