JSON-RPC

JSON-RPC (JavaScript Object Notation – Remote Procedure Call) is a lightweight remote procedure call protocol that uses JSON for encoding messages. It allows a client to call functions (methods) on a remote server as if they were local functions.

Unlike REST, which is resource-oriented, JSON-RPC is procedure-oriented: the client explicitly calls a named method and passes parameters.

JSON-RPC is:

- **transport-independent** (commonly used over HTTP, WebSocket, TCP) - which means the protocol defines only the message format and not the communication channel. It can run over HTTP, HTTPS, WebSocket, raw TCP sockets, message queues, or even serial connections in embedded systems. Because it is transport-agnostic, it can be integrated into very different infrastructures without changing the protocol itself.
- **stateless** - each request contains all the information necessary for execution: the method name, parameters, and request identifier. The server does not need to store session data between calls. This design improves scalability and makes horizontal load balancing easier in distributed systems and microservice architectures.
- **lightweight** - The message structure is minimal and contains only a few required fields. There are no built-in resource descriptions, URL hierarchies, or protocol-specific metadata. As a result, payload sizes are typically small and parsing logic is simple, which is beneficial in high-performance systems, internal APIs, and resource-constrained environments such as IoT devices.
- **simple to implement** - Since JSON parsing libraries are available in nearly every programming language, creating a JSON-RPC server usually requires only a JSON parser, a method-dispatch mechanism that maps method names to functions, and structured error handling.
- **language-independent** - Because JSON is a universal data interchange format, clients and servers can be implemented in different programming languages without compatibility issues. A Python client can communicate with a Java or C++ server as long as both follow the JSON-RPC 2.0 specification.

The current widely used version is JSON-RPC 2.0. https://www.jsonrpc.org/specification

JSON-RPC 2.0 Message Structure

A JSON-RPC request is a JSON object with the following fields:

- **jsonrpc** – must be “2.0”
- **method** – name of the remote method
- **params** – parameters (optional)
- **id** – request identifier (used to match response)

Example request:

```
{ "jsonrpc": "2.0", "method": "add", "params": [5, 7], "id": 1 }
```

Successful response:

```
 { "jsonrpc": "2.0", "result": 12, "id": 1 }
```

Error response:

```
 { "jsonrpc": "2.0", "error": { "code": -32601, "message": "Method not
found" }, "id": 1 }
```

Notifications

If the client sends a request without an id, it becomes a notification.

The server must NOT send a response.

Example:

```
 { "jsonrpc": "2.0", "method": "logEvent", "params": {"event": "started"} }
```

Batch Requests

JSON-RPC supports sending multiple requests in one array:

```
 [ {"jsonrpc": "2.0", "method": "add", "params": [1,2], "id": 1},
{"jsonrpc": "2.0", "method": "subtract", "params": [5,3], "id": 2} ]
```

6. Practical Example in Python

In this section, we implement a simple JSON-RPC server and client using Python.

We will use:

- Flask (for HTTP server)
- requests (for client)

Install dependencies:

```
pip install flask requests
```

JSON-RPC Server (Python + Flask)

Create a file: *server.py*

```
from flask import Flask, request, jsonify

app = Flask(__name__)

# available RPC methods

def add(a, b):
    return a + b


def subtract(a, b):
    return a - b


methods = {
    "add": add,
    "subtract": subtract,
}


@app.route("/rpc", methods=["POST"])
def rpc():
    data = request.get_json()

    if data.get("jsonrpc") != "2.0":
        return jsonify(
            {
                "jsonrpc": "2.0",
                "error": {"code": -32600, "message": "Invalid Request"},
                "id": data.get("id"),
            }
        )

    method_name = data.get("method")
    params = data.get("params", [])
    request_id = data.get("id")

    if method_name not in methods:
        return jsonify(
            {
                "jsonrpc": "2.0",
                "error": {"code": -32601, "message": "Method not found"},
                "id": request_id,
            }
        )

    try:
        result = methods[method_name](*params)
        return jsonify(
            {
                "jsonrpc": "2.0",
                "result": result,
                "id": request_id,
            }
        )
    except Exception as e:
        return jsonify(
            {
                "jsonrpc": "2.0",
                "error": {"code": -32603, "message": str(e)},
                "id": request_id,
            }
        )


if __name__ == "__main__":
    app.run(port=5000)
```

Run:

```
python server.py
```

JSON-RPC Client (Python)

Create a file: *client.py*

```
import requests import json

url = "http://localhost:5000/rpc
"

payload = {
"jsonrpc": "2.0",
"method": "add",
"params": [10, 15],
"id": 1
}

response = requests.post(url, json=payload)

print("Status:", response.status_code)
print("Response:", response.json())
```

Expected output:

```
Status: 200 Response: {'jsonrpc': '2.0', 'result': 25, 'id': 1}
```

Error Codes in JSON-RPC 2.0

Code	Meaning
-32700	Parse error
-32600	Invalid Request
-32601	Method not found
-32602	Invalid params
-32603	Internal error