Remote Procedure Calls (RPC)

Paskirstitų Sistemų Pagrindai

Elvinas Žukauskas ITFv-22

Kas yra RPC?

- **Remote Procedure Call** (RPC) – architektūrinis modelis, leidžiantis vieno kompiuterio procesui iškviesti funkcijas kitame kompiuteryje, tarsi tai būtu lokali funkcija.
- Slepia tinklo komunikacijos detales nuo developer'io, leidžiant rašyti kodą lyg jis vyktų vienoje mašinoje.
- Pagrindinis tikslas – supaprastinti ir pagreitinti paskirstytų sistemų kūrimą.

RPC architektūra

- Apibrėžia bendrą modelį kaip turėtu veikti RPC implementacija.
- Nurodo pagrindinius principus, kaip RPC veikia, o ne konkrečią technologiją ar įrankius.
- **Pagrindiniai privalumai:**
  - Vientisas API tarp įvairių komponentų.
  - Supaprastina paskirstytų sistemų kūrimą.

gRPC: Įvadas

- gRPC – Google'o RPC implementacija.
- Free and open-source.
- Veikia ant **HTTP/2** ir **TCP**:
  - **HTTP/2**: multiplexing palaikymas.
  - **TCP**: užtikrina patikimą duomenų perdavimą.
- Puikiai tinka mikroservisų architektūrai, tačiau dėl didesnio latency nėra optimalus servisams, kuriems reikalingi itin greiti atnaujinimai – pavyzdžiui, žaidimų serveriams.

`.proto` failas

- Naudoja **proto3** sintaksę – aiškiai apibrėžia duomenų pranešimus ir paslaugas:

```
syntax = "proto3";
package calc;

message Operands {
  double a = 1;
  double b = 2;
}

message Result {
  double value = 1;
}

service Calculator {
  rpc Add (Operands) returns (Result);
}
```

Stub'ų generavimas

```
python -m grpc_tools.protoc \
  --proto_path=. \
  --python_out=./gen \
  --grpc_python_out=./gen \
  calculator.proto
```

- Sugeneruoja `calculator_pb2.py` ir `calculator_pb2_grpc.py`.

Projekto struktūra

```
gRPC-Project-root/
├── calculator.proto
├── client.py # Sukursim veliau
├── server.py # Sukursim veliau
└── gen/
    ├── calculator_pb2.py
    └── calculator_pb2_grpc.py
```

gRPC klientas (`client.py`)

```python
import grpc
import gen.calculator_pb2 as pb2
import gen.calculator_pb2_grpc as pb2_grpc

def run():
    channel = grpc.insecure_channel('localhost:50051')
    stub = pb2_grpc.CalculatorStub(channel)
    req = pb2.Operands(a=3, b=5)
    res = stub.Add(req)
    print(f"3 + 5 = {res.value}")

if __name__ == "__main__":
    run()
```

- **channel**: ryšio kanalas su serveriu.
- **stub**: suteikia paprastą stub objektą metodų kvietimui.
- Argumentai automatiškai serializuojami į protobuf formatą.

gRPC serveris (`server.py`)

```python
import grpc
from concurrent import futures
import gen.calculator_pb2 as pb2
import gen.calculator_pb2_grpc as pb2_grpc

class CalculatorServicer(pb2_grpc.CalculatorServicer):
    def Add(self, request, context):
        return pb2.Result(value = request.a + request.b)

def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=4))
    pb2_grpc.add_CalculatorServicer_to_server(CalculatorServicer(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    print("Server listening on port 50051")
    server.wait_for_termination()

if __name__ == "__main__":
    serve()
```

Projekto struktūra

```
gRPC-Project-root/
├── calculator.proto
├── client.py
├── server.py
└── gen/
    ├── calculator_pb2.py
    └── calculator_pb2_grpc.py
```

Paleidžiamos komandos per CLI:

```
$ python client.py
```

```
$ python server.py
```

RPC vs API

- RPC – API subset
- Naudojamas, kai reikia vientisos logikos tarp darbo stočių.
- Neryškiai, tačiau mažesnis latency nei RESTful.

RPC žaidimuose: Godot

- Godot RPC veikia ant **UDP** – greita komunikacija be patikimumo garantijų.

```
@rpc(mode, sync, transfer_mode)
func hello_world():
    print("Hello, world!")
```

- Funkcijų eksportavimas per `@rpc` anotacijas:
  - **mode**:
    - `"authority"` – kvietimą gali inicijuoti tik serveris (host'inantis mazgas).
    - `"any_peer"` – kvietimą gali inicijuoti bet kuris prisijungęs mazgas.
  - **sync**:
    - `"call_local"` – funkcija bus iškviesta tiek lokaliai, tiek nuotoliniu būdu.
    - `"call_remote"` – funkcija bus iškviesta tik nuotoliniu būdu (kitų mazgų kontekste).
  - **transfer_mode**:
    - `"reliable"` – patikimas paketų perdavimas su patvirtinimu (3-way handshake).
    - `"unreliable"` – patikimumas nekontroliuojamas, mažesnis overhead (greitesnis).

RPC žaidimuose: Godot

Pavyzdinė implementacija.

```
@rpc("any_peer", "reliable", "call_local")
func hello_world():
  print("Hello, world!")

func _process():
  if button_pressed:
    hello_world.rpc()
```

Nesvarbu kuris client paspaus mygtuka, `Hello, world!` žinutė atsiras abiejų client'ų terminaluose.

RPC žaidimuose: Godot

```
extends Node
@export var port: int = 12345
@export var max_clients: int = 8
@export var is_server: bool = false

func _ready():
    var peer = ENetMultiplayerPeer.new()
    if is_server:
        peer.create_server(port, max_clients)
    else:
        peer.create_client("127.0.0.1", port)
    multiplayer.multiplayer_peer = peer
    multiplayer.peer_connected.connect(_on_peer_connected)
    spawn_player()

@rpc("call_local", "reliable")
func spawn_player():
    var p = preload("res://Player.tscn").instantiate()
    add_child(p)
```

RPC žaidimuose: Godot

```
extends CharacterBody2D
@export var speed := 200

func _physics_process(delta):
    if is_multiplayer_authority():
        var inp = Vector2(
            Input.get_action_strength("ui_right") - Input.get_action_strength("ui_left"),
            Input.get_action_strength("ui_down")  - Input.get_action_strength("ui_up")
        )
        velocity = inp.normalized() * speed
        move_and_slide()
        sync_state.rpc(global_position)

@rpc("any_peer", "unreliable", "call_remote")
func sync_state(pos:Vector2):
    global_position = pos
```

Išvados

- **RPC** – architektūrinis modelis, bet ne implementacija.
- **gRPC** – industrijos standartas - implementacija, tačiau nėra toks paprastas, kaip skamba ant popieriaus.
- **RPC vs API** - RPC yra API subset.
- **Godot** – praktinis RPC panaudojimas žaidimuose.

Klausimai?