Universidade Federal de Viçosa
Campus Florestal
CCF 355

Trabalho 04 - FindGamers

Vinícius Teixeira - 3057
Jonathan Lopes - 2666
Matheus S. Alves - 2649

Florestal
2023

Universidade Federal de Viçosa
Campus Florestal
CCF 355

Trabalho prático 04

Usando o middleware RMI

Quarta parte do trabalho prático da disciplina de Sistemas Distribuídos.

Professor: Thais Regina de M. B. Silva

Florestal
2023

Conteúdo

Trabalho 03

1. Introdução

Nessa parte, damos continuidade ao projeto iniciado esse semestre FindGamers. Agora vamos reimplementar o software criado utilizado o middleware Pyro5. Com o uso de um muiddleware, os elementos do software passam a ser vistos como objetos e podemos 'chama-los' como tal no nosso cliente.

2. Objetivo

Reimplementar a camada de comunicação usando o middleware Pyro5.

3. Desenvolvimento

Para desenvolver essa parte, utilizamos a classe **Router** criada na parte 3. Essa classe foi criada para gerenciar as chamadas do cliente para o servidor, criando rotas para endereçar as funções adequadas de cada chamada.

Utilizamos o middleware para expor o Router para o cliente, e no cliente criamos serviços que encapsulam as chamadas para o servidor. Assim podemos reutilizar esses serviços durante nossa aplicação sem que o cliente se preocupe em como as chamadas para os servidor estão sendo feitas.

Abaixo podemos ver um exemplo do que foi feito, primeiro vemos o código antigo de como era feito para fazer uma chamada ao servidor usando a classe **API**, implementada na parte 3 usando sockets, e como fazemos agora utilizando o middleware Pyro5 implementado na classe **Server**.

```
import api

class GamesService:
  def __init__(self) -> None:
      self.__api = api.API()
      pass

  def getAllGames(self):
      try:
        response =  self.__api.GET('/games')
        print('from server', response)
        return response
```

```
      except:
        print('Erro ao tentar buscar jogos!')
```

```
import pyro

class GamesService:
  def __init__(self) -> None:
      self.__api = pyro.Server()
      pass

  def getAllGames(self):
      try:
        response =  self.__api.exec('GET', '/games')
        print('from server', response)
        return response
      except:
        print('Erro ao tentar buscar jogos!')
```

4. Pyro5

Primeiramente é necessário definir o que é um RMI: Um RMI (Remote Method Invocation) é uma interface de programação que permite ao programador invocar metodos de objetos remotos de uma forma muito semelhante a invocações locais, possibilitanto e facilitando a criação de sistemas distribuidos, o RMI utilizado ao longo do projeto é o Pyro5. O Pyro é uma biblioteca destinada a projetos em Python, ele permite que os objetos se comuniquem por meio de uma rede, seja ela a Internet ou uma rede local, com esforço reduzido. O front-end do projeto não precisou ser alterado para a mudança Socket-Pyro Diferente de sockets no Pyro não é necessário atribuir todos os detalhes da conexão, essas informações são abstraidas para tentar facilitar a vida do programador. Para utilização do Pyro, foi necessário instalar o venv um suporte para ambiente virtuais, ele permite que o nosso projeto seja executado em diversos terminais na mesma máquina de forma independente.

O RMI tem o arquivo próprio chamado pyro.py, neste arquivo está contido todos os imports necessários para o pyro, são eles: json, pyro5.api e pyro5.errors, esses imports são os utilizados ao logno do sistema FindGamers além de duas funções dentro da class server:

```
class Server:
def __init__(self) -> None:
    # We create a proxy with a PYRONAME uri.
    # That allows Pyro to look up the object again in the NS when
    # it needs to reconnect later.
    self.__server = Pyro5.api.Proxy("PYRONAME:server.router")
    pass

def exec(self, method, url, payload):
    response = self.__server.exec(method, url, payload)
    return json.loads(response)
```

A função init define o server em pyro e a função exec é uma chamada genérica de apis, e cada uma das API's é definida na pasta service dentro de cada um dos respectivos arquivos

Para execução do projeto, primeiramente é necessário a criação do server no ambiente virtual utilizando o comando:

```
python -m Pyro5.nameserver
```

Em seguida, a execução normal do server e do client:

```
python ./server/pyro.py  python ./client/client.py
```

5. Comparando Pyro - Sockets

A implementação utilizando pyro demonstra-se bem mais simples em comparação a sockets onde é necessário declarar cada conexão, cada abertura e fechamento, todavia a utilização do RMI se demonstrou mais lenta, o sistema após adesão do pyro perdeu, e muito em tempo de resposta, os services demoraram bem mais a responder e o sistema travava bastante, isso na utilização do projeto em Windows, utilizando o projeto com WSL ou em um computador nativo com sistema Unix como o ubuntu 20.04 ou o Pop.os (baseado em ubuntu 22.04) essa lentidão era bem menor, mas ainda sim, mais lento que o projeto em socket.

Além da facilidade de programação do sistema, o RMI também possibilita a diminuição do projeto como um todo, com sockets, todas as rotas de todas as chamadas utilizadas pelo sistema deviam ser criadas, mesmo que elas possuissem semelhanças:

```
from router import Router as RouterClass
```

```
from controllers import sessionController, usersController,
lobbyController, gamesController, matchController

Router = RouterClass()
Router.get(url='/users', callback=usersController.GetUsers)
Router.post(url='/users', callback=usersController.CreateUser)
Router.post(url='/session', callback=sessionController.PostSession)
Router.get(url='/lobby', callback=lobbyController.GetAllLobbies)
Router.post(url='/lobby', callback=lobbyController.CreateLobby)
Router.post(url='/lobby-by-id', callback=lobbyController.GetLobbyById)
Router.post(url='/lobby-enter', callback=lobbyController.EnterLobby)
Router.post(url='/lobby-leave', callback=lobbyController.LeaveLobby)
Router.post(url='/lobby-by-page', callback=lobbyController.GetAllLobbies)
Router.get(url='/games', callback=gamesController.GetAllGames)
Router.post(url='/match', callback=matchController.Create)
Router.post(url='/match-by-id', callback=matchController.GetMatch)
Router.post(url='/challenges', callback=matchController.GetAllChallenges)
Router.post(url='/accept-challenge', callback=matchController.AcceptChallenge)
Router.post(url='/reject-challente', callback=matchController.RejectChallenge)
Router.post(url='/check-for-challenges', callback=matchController.CheckForLobbie
```

Já com o pyro, a chamada genérica permite que o projeto utilize-se apenas dos services criados pelo programador em cada arquivo. Por fim manifesta-se com clareza que o uso do RMI Pyro5, troca velocidade por facilidade, o sistema se torna mais facilmente criado pelo programador em troca de ficar mais lento para o usuário, porém essa lentidão pode ser amenizada caso o programador informe ao usuário por exemplo o sistema operacional ideal.

6. Dificuldades encontradas

- Sincronização e concorrência: Se o sistema distribuído permitir múltiplas conexões e operações concorrentes, é importante garantir a sincronização adequada dos recursos compartilhados.
- Lidar com erros e exceções → Durante a troca de mensagens tratar problemas de conexão, timeouts, mensagens malformadas e outros cenários imprevistos que possam ocorrer.

Nessa parte, o maior problema está sendo em manter os clientes atualizados com as informações do banco à medida que elas mudam. Manter todos os clientes sincronizados, é uma tarefa trabalhosa, para resolver seria

importante criar algum sistema de mensagens em que cada cliente conectado ao servidor poderia se juntar e assim, o servidor, gerenciar as mensagens trocadas. Porém, por questões de tempo, o grupo decidiu tentar resolver esse problema na parte 5 utilizando ferramentas web.