# Sistemas Distribuídos

## Aula Prática - RMI

Prof. Rodrigo Campiolo

UTFPR - Universidade Tecnológica Federal do Paraná

6 de agosto de 2021

# Introdução

## Objetivos

- Apresentar conceitos básicos para RMI em Python com a API Pyro5
- Desenvolver uma aplicação usando Pyro5.

## Pyro: Características

- suporte ao uso de diferentes serializadores (serpente, json, marshal, msgpack).
- especificação de *timeouts* para as comunicações em rede.
- exceções remotas devolvidas para o cliente.
- três modos para criação de objetos remotos: *single*, *session*, *percall*.
- cliente em Java e .NET com API *Pyrolite*.

## Pyro: Conceitos

- Proxy: realiza a invocação remota (un/marshalling, comunicação remota).
- URI: identifica um objeto Pyro, na forma:
  **PYRO: + object name + @ + server name + : + port number**
- Objeto Pyro: objeto remoto registrado no Pyro Daemon.
- Pyro Daemon: aguarda solicitações remotas, despachante, un/marshalling.
- Pyro Name Server: servidor de nomes para objetos Pyro.
- Configuração: opções para configurar o comportamento do Pyro (**PYRO_**).

# Prática: RMI

## Materiais

- Python 3
- Python 3 APIs
  - Pyro5: biblioteca para RMI em Python.
  - Serpent: biblioteca simples para serialização (utf-8).

# Prática: RMI

## Instalação

- Biblioteca Pyro5 e dependências:

```
pip3 install Pyro5
```

# Prática: RMI

## Iniciando um servidor de nomes (binder)

- Para iniciar o servidor de nomes:

```
python3 -m Pyro5.nameserver -n localhost
```

## Clientes - Localizando um serviço

- Deve-se acessar um **NS** e procurar o serviço:

```
Pyro5.api.locate_ns (host=None, port=None,
                     broadcast=True, hmac_key=None)
```

- Exemplo:

```
nameserver = Pyro5.api.locate_ns()
uri = nameserver.lookup("CalcService")
```

## Clientes - Invocando métodos

- Deve-se usar um objeto **Proxy**:

```
Pyro5.api.Proxy (uri, connected_socket=None)
```

- Exemplo:

```
calc_proxy = Pyro5.api.Proxy(uri)
value = calc_proxy.soma(10, 20)
```

## Objeto Remoto - Implementando e publicando métodos

- Usa-se a anotação `@Pyro5.api.expose` para especificar a interface remota.
- A anotação pode ser usada para classes, atributos e métodos.
- Métodos privados não são publicados.
- Exemplo:

```
@Pyro5.api.expose
class Calculadora:
    def soma(self, a, b):
        return a + b

    def subtrai(self, a, b):
        return a - b
```

## Servidor - Publicando objetos remotos

- Deve-se inicializar o Pyro5 Daemon e registrar o objeto.
- Exemplo:

```
daemon = Pyro5.server.Daemon()
uri = daemon.register(Calculadora)
daemon.requestLoop()
```

## Servidor - Registrando o objeto remoto em um NS

- Localizar o servidor de nomes e registrar a URI.
- Exemplo:

```
uri = daemon.register(Calculadora)
ns = Pyro5.api.locate_ns()
ns.register("CalcService", uri)
```

## Modos de criação e interação com objeto remoto

- Há três modos:
  - *session*: nova instância para cada conexão de um novo proxy.
  - *single*: uma única instância para todas as requisições.
  - *percall*: uma nova instância a cada método invocado.
- Para especificar o modo use a anotação:

```
@Pyro5.api.behavior(instance_mode="session")
```

- O *session* é o modo padrão.

## Notas sobre a API Pyro5

- Todo objeto *Proxy()* tem sua própria conexão com o Daemon.
- Toda conexão tem sua própria thread de processamento.
- A conexão permanece ativa durante o ciclo de vida do *Proxy()*.
- A conexão só é criada após a primeira invocação.
- Invocações *OneWay* podem ser realizadas em métodos com a anotação **@Pyro5.api.oneway**
- Pode-se configurar um *timeout* em segundos para um Proxy:
  `proxy._pyroTimeout = 2.0`
- Pode-se realizar solicitações em lote com um Proxy:

```
batch = Pyro5.api.BatchProxy(proxy)
# ... fazer as chamadas com batch
responses = batch()  #executar as chamadas
```

# Prática: RMI

## Atividades

1. Implementar uma agenda usando RMI em Python.
2. Implementar uma agenda usando Java RMI.

# Referências Bibliográficas

## Referências


- Pyro - Python Remote Objects. Disponível em: `https://pyro5.readthedocs.io/`.