

Chamadas de Procedimentos Remotos em Python

A. Zamberlan e G. Kurtz

UFN

www.lapinf.com.br



Agenda

- 1 Contexto
- 2 Definições e Conceitos
- 3 Exemplo de código Java RMI
- 4 Exemplo de código RPC Python
- 5 Considerações

Agenda

- 1 Contexto
- 2 Definições e Conceitos
- 3 Exemplo de código Java RMI
- 4 Exemplo de código RPC Python
- 5 Considerações

- Apresentar a dinâmica de funcionamento de programação RPC
- Discutir implementações:
 - RPC protocolo
 - RMI Java API
 - XML RPC
 - RPC Python sistema

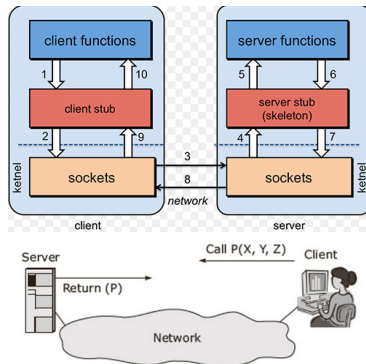
Agenda

- 1 Contexto
- 2 Definições e Conceitos
- 3 Exemplo de código Java RMI
- 4 Exemplo de código RPC Python
- 5 Considerações

RPC - Remote Procedure Call

Protocolo comunicação

... que um programa utiliza para requisitar/solicitar serviço de um programa localizado em outro computador na rede sem ter que se preocupar com detalhes de rede.



RPC - Remote Procedure Call

Conceitos atrelados

- Sistemas Distribuídos + Sistema Operacional + Redes
- Arquitetura Cliente-Servidor
- Camada Transporte:
 - TCP (síncrono): Sistema bloqueante
 - servidor processa...
 - ...cliente espera
 - UDP (assíncrono): Não bloqueante
- Sockets
- Threads
- Programação Orientada a Objetos
- Serialização de objetos
- Bruce Jay Nelson cunhou **Remote Procedure Call** (1981)

Application Program Interface

- ... fornece mecanismos para criar aplicações distribuídas em Java
- Permite um objeto chamar/invocar métodos de outro objeto rodando em outra JVM

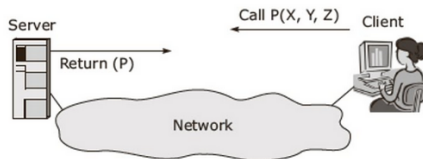
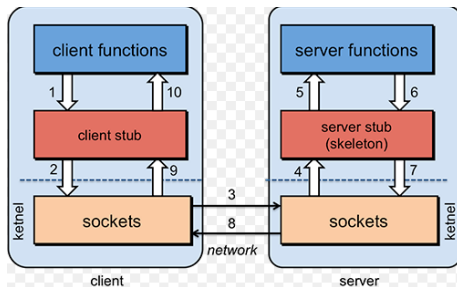
Microsoft

- business-to-business (e-commerce)
- ideia de "human-readable-and-writable"
- script padrão baseado em HTML capaz de ser 'parseado' (análise sintática)

Sistema

- ... apresenta mesma funcionalidade RPC tradicional e RMI Java.
- CONTUDO, é mais "simples tratar tudo isso!!!
- MAS, e o desempenho?????

Remote Procedure Call



Agenda

- 1 Contexto
- 2 Definições e Conceitos
- 3 Exemplo de código Java RMI**
- 4 Exemplo de código RPC Python
- 5 Considerações

RMI Java

```
1 import java.rmi.Remote;  
2 import java.rmi.RemoteException;  
3  
4 public interface IHoje_eh extends Remote{  
5     public DataHora pegaDataHora() throws RemoteException;  
6 }  
7
```

```
1 import java.rmi.RemoteException;  
2 import java.rmi.server.UnicastRemoteObject;  
3 import java.util.Date;  
4  
5 public class Hoje_eh extends UnicastRemoteObject implements IHoje_eh {  
6  
7     public Hoje_eh() throws RemoteException{  
8  
9     }  
10  
11     @Override  
12     public DataHora pegaDataHora() throws RemoteException {  
13         DataHora obj = new DataHora();  
14         return obj;  
15     }  
16 }  
17
```

```
1 import java.io.Serializable;
2 import java.sql.Time;
3 import java.util.Date;
4
5 public class DataHora implements Serializable {
6     Date data;
7     Time hora;
8
9     public DataHora() {
10         this.data = new Date();
11         this.hora = new Time(1000);
12     }
13 }
```

RMI Java

```
1 import java.rmi.Naming;
2 import java.rmi.registry.LocateRegistry;
3 import java.rmi.registry.Registry;
4
5 public class Server {
6     String HOST_URL = "rmi://localhost/Hoje_eh";
7
8     public Server() {
9         try {
10             LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
11             Hoje_eh objetoRemoto = new Hoje_eh();
12             System.out.println("servidor retornando a data/hora...");
13             Naming.bind(HOST_URL, objetoRemoto);
14         } catch (Exception e) {
15             System.out.println("Error: " + e);
16         }
17     }
18
19     public static void main(String args[]) {
20         new Server();
21     }
22 }
```

```
1 import java.rmi.Naming;
2
3 public class Client {
4     public static void main(String args[]) {
5         try {
6             IHoje_eh d = (IHoje_eh) Naming.lookup("rmi://localhost/Hoje_eh");
7             System.out.println("Data e hora vinda do servidor!!");
8             System.out.println("Data: " + d.pegaDataHora().data);
9             //System.out.println("Hora: " + d.pegaDataHora().hora);
10         } catch (Exception e) {
11             System.out.println("Error: " + e);
12         }
13     }
14 }
15 }
```

RMI Java - Saída

Cmd Prompt de Comando - java Server

```
C:\Users\usrlab25\Downloads\sistemasDistribuidos\7-RMI-Java\exemplo\src>java Server  
servidor retornando a data/hora...
```

Cmd Prompt de Comando

```
C:\Users\usrlab25\Downloads\sistemasDistribuidos\7-RMI-Java\exemplo\src>java Client  
Data e hora vinda do servidor!!  
Data: Fri May 31 21:19:48 BRT 2019
```

```
C:\Users\usrlab25\Downloads\sistemasDistribuidos\7-RMI-Java\exemplo\src>java Client  
Data e hora vinda do servidor!!  
Data: Fri May 31 21:19:53 BRT 2019
```

```
C:\Users\usrlab25\Downloads\sistemasDistribuidos\7-RMI-Java\exemplo\src>java Client  
Data e hora vinda do servidor!!  
Data: Fri May 31 21:19:56 BRT 2019
```

```
C:\Users\usrlab25\Downloads\sistemasDistribuidos\7-RMI-Java\exemplo\src>java Client  
Data e hora vinda do servidor!!  
Data: Fri May 31 21:19:59 BRT 2019
```


Agenda

- 1 Contexto
- 2 Definições e Conceitos
- 3 Exemplo de código Java RMI
- 4 Exemplo de código RPC Python**
- 5 Considerações

Python RPC

```
servidor.py
1 from xmlrpc.server import SimpleXMLRPCServer
2 import datetime, xmlrpc.client
3
4 def hoje_eh():
5     hoje = datetime.datetime.today()
6     return xmlrpc.client.DateTime(hoje)
7
8 servidor = SimpleXMLRPCServer(("localhost", 8000))
9 print("Ouvindo a porta 8000...")
10 servidor.register_function(hoje_eh, "hoje")
11 servidor.serve_forever()
```

```
cliente.py
1 import xmlrpc.client, datetime
2
3 servidor = xmlrpc.client.ServerProxy("http://localhost:8000/")
4
5 hoje = servidor.hoje()
6
7 # convert the ISO8601 string to a datetime object
8 data_hora_convertida = datetime.datetime.strptime(hoje.value, "%Y%m%dT%H:%M:%S")
9 print("Hoje é: %s" % data_hora_convertida.strftime("%d.%m.%Y, %H:%M:%S"))
```

Python RPC - Saída

```

C:\Users\usrlab25\Downloads\istemasDistribuidos\6-RPC_Python>py servidor.py
Ouvindo a porta 8000...
127.0.0.1 - - [31/May/2019 21:06:49] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [31/May/2019 21:06:55] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [31/May/2019 21:06:58] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [31/May/2019 21:07:00] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [31/May/2019 21:07:09] "POST / HTTP/1.1" 200 -

```

Prompt de Comando

```

C:\Users\usrlab25\Downloads\istemasDistribuidos\6-RPC_Python>py cliente.py
Hoje é: 31.05.2019, 21:06:49

C:\Users\usrlab25\Downloads\istemasDistribuidos\6-RPC_Python>py cliente.py
Hoje é: 31.05.2019, 21:06:55

C:\Users\usrlab25\Downloads\istemasDistribuidos\6-RPC_Python>py cliente.py
Hoje é: 31.05.2019, 21:06:58

C:\Users\usrlab25\Downloads\istemasDistribuidos\6-RPC_Python>py cliente.py
Hoje é: 31.05.2019, 21:07:00

C:\Users\usrlab25\Downloads\istemasDistribuidos\6-RPC_Python>py cliente.py
Hoje é: 31.05.2019, 21:07:09

```

Agenda

- 1 Contexto
- 2 Definições e Conceitos
- 3 Exemplo de código Java RMI
- 4 Exemplo de código RPC Python
- 5 Considerações

Considerações

- Simples e rápido para 'produção' e 'entrega'
- Desempenho é pior do que sockets
 - simplificar o desenvolvimento X melhorar desempenho
 - 'serviços' simples não faz diferença

Abstração para o programador

- Sistemas Distribuídos + Sistema Operacional + Redes
- Arquitetura Cliente-Servidor
- Camada Transporte:
 - TCP (síncrono): Sistema bloqueante - server processa... cliente espera
 - UDP (assíncrono): Não bloqueante
- Sockets
- Threads
- Programação Orientada a Objetos
- Serialização de objetos

Alexandre Zamberlan e Guilherme Kurtz
alexz@ufn.edu.br e guilhermechagaskurtz@ufn.edu.br

<https://github.com/alexandrezamberlan/sistemasDistribuidos>

www.lapinf.com.br