Licenciatura/Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Fundamentos de Sistemas Distribuídos

Ano letivo 2021/2022

Exercício 2 – Invocação de Métodos Remotos (RMI)

Novembro/2021

Início	15 nov.
Fim	04 dez.

Resultados de aprendizagem

- Explicar por palavras próprias o modelo de objetos distribuídos e o paradigma de invocação de métodos remotos (RMI)
- Perante uma arquitetura de referência de um *middleware* RMI descrever o papel desempenhado pelos vários componentes que o constituem, nomeadamente os *stubs* (ou *proxies*), *skeletons*, módulos de comunicações (ou *runtime*), e objeto remoto
- Descrever o papel desempenhado pelas linguagens de IDL (*Interface Definition Language*) na criação de sistemas baseados em objetos distribuídos
- Descrever as implicações para o programador resultantes da invocação de métodos remotos em comparação com a invocação de métodos locais
- Criar pequenas aplicações distribuídas cuja comunicação seja baseada em Java RMI

Descrição

O cenário que serve de base para este exercício é o de um serviço de presenças. Este serviço recebe um indentificador do cliente (neste caso o endereço IP da máquina cliente) e devolve a lista dos identificadores dos clientes que contactaram o serviço nos últimos 3 minutos. Adicionalmente, sempre que há o registo de uma nova presença, este serviço envia uma notificação contendo a identificação da nova presença a todos os clientes que o tenham contactado e que se mantenham ativos. As figuras Figura 1 e Figura 2 apresentam duas representações do sistema (simplificada e completa).

Para construir este sistema é necessário definir as interfaces remotas, implementar os objetos remotos e implementar as aplicações cliente e servidor.

Definir as interfaces remotas

Uma interface remota declara um conjunto de métodos remotos. Neste sistema, tanto o cliente como o servidor declaram interfaces remotas.

PresencesInterface.java – Declara a interface do serviço de presenças. Esta interface inclui um método remoto *getPresences* que recebe como argumento o identificador do cliente (endereço IP) e uma referência remota para

o cliente, e devolve um vetor com a informação (endereços IP) de todos os clientes que contactaram o serviço nos últimos 3 minutos.

NewPresencesInterface.java – Declara a interface remota que o cliente oferece para que o serviço de presenças o possa informar de novas presenças. Esta interface inclui um método remoto *setNewPresence* que recebe como argumento um identificador de cliente (endereço IP) que corresponde à informação da nova presença.

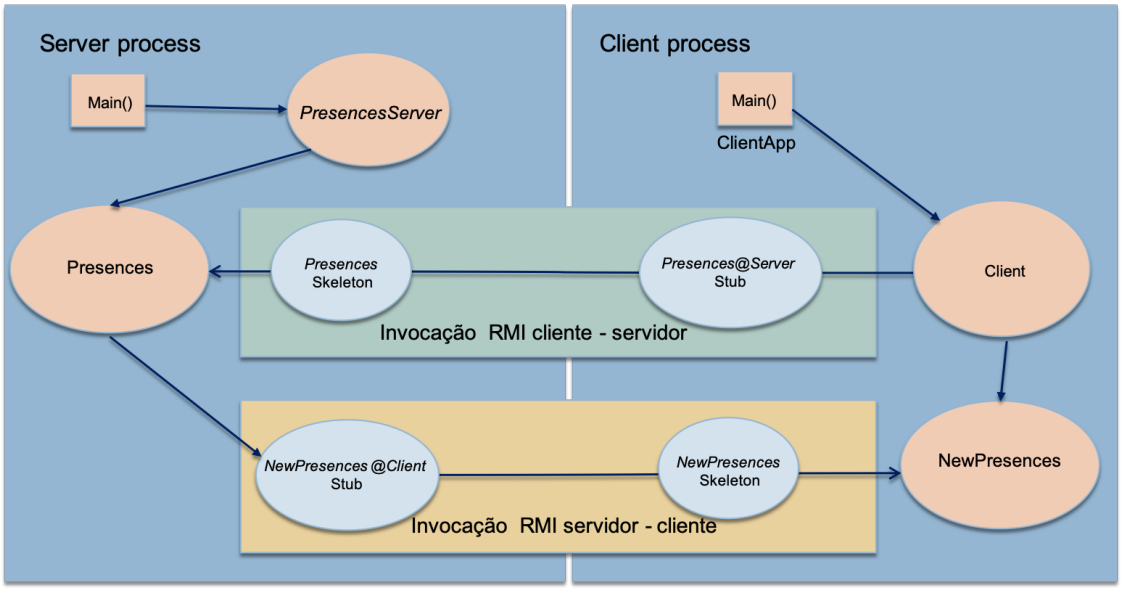

Figura 1 Representação gráfica simplificada do sistema

Implementar os objetos remotos

Um objeto remoto implementa uma interface remota. Neste sistema, precisamos de uma implementação da interface *PresencesInterface* e da interface *NewPresencesInterface*.

Presences.java – Esta classe implementa a interface remota *PresencesInterface.* O método *getPresences* é invocado pelo cliente. Recebe como argumentos o IP do cliente que identifica uma presença, e a referência remota para que o serviço possa invocar o objeto remoto no cliente sempre que recebe informação de uma nova presença.

-> objeto *Presences* na Figura 1.

NewPresences.java – Esta classe implementa a interface remota *NewPresencesInterface*. O método *setNewPresence*, quando invocado, recebe a informação de uma nova presença e imprimi-a no terminal.

-> objeto *NewPresences* na Figura 1.

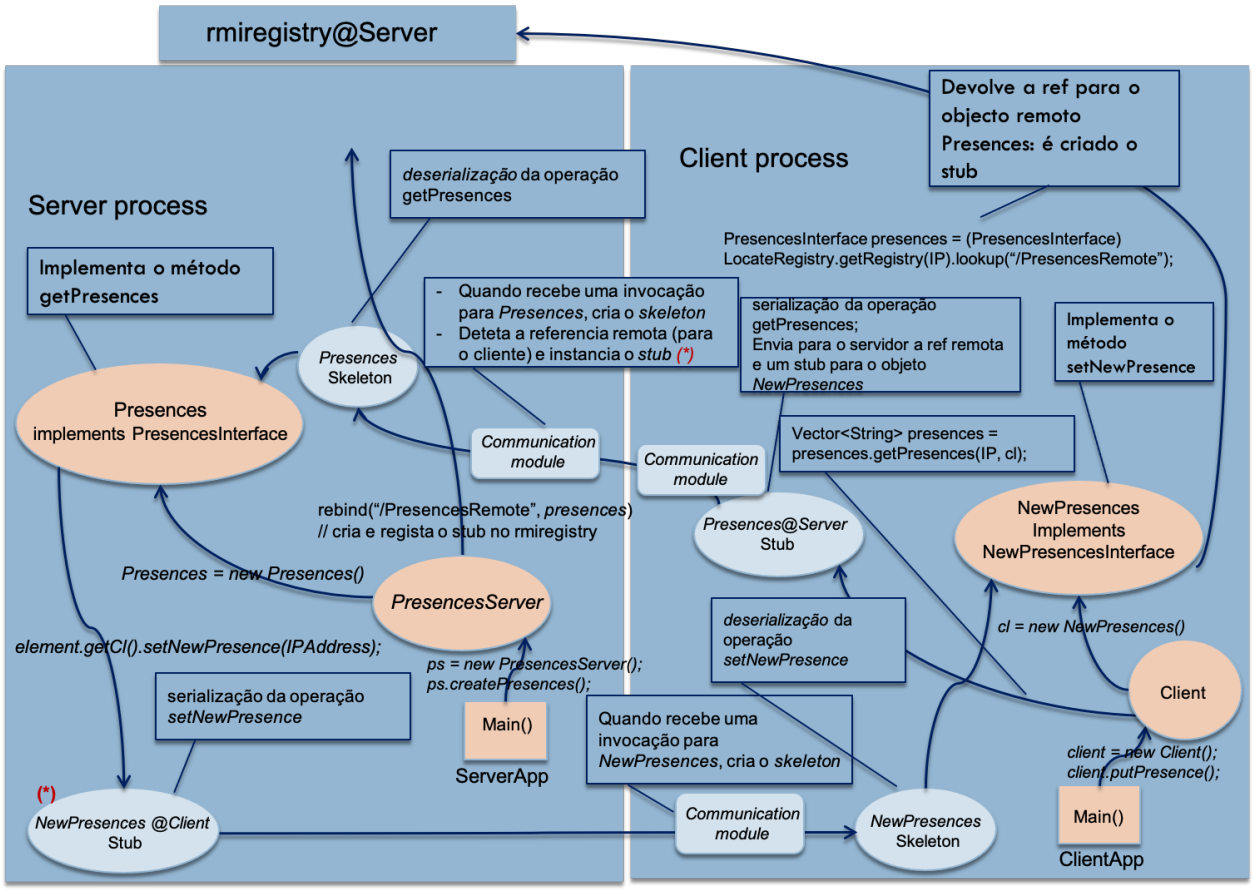

Figura 2 Representação gráfica do sistema

Implementar as aplicações cliente e servidor

Depois de declarar as interfaces remotas e implementar os objetos remotos é necessário implementar as aplicações servidor e cliente responsáveis por implementar a restante lógica do sistema e por construir todo o sistema.

PresencesServer.java – Esta classe cria uma instância da classe *Presences* e regista-a no servidor de nomes Java RMI (*rmiregistry*) com o nome "/PresencesRemote". Esta classe tenta obter uma referência para o *rmiregistry*. Caso não consiga obter essa referência, cria uma instância do *rmiregistry*. Geralmente, o *rmiregistry* é usado apenas para os processos localizarem um primeiro objeto remoto que necessitem de invocar. Em geral, esse primeiro objeto, por sua vez, fornecerá suporte aplicacional específico para a obtenção de mais referências remotas. Por exemplo, a aplicação servidor obtém uma referência para o objeto remoto *NewPresences* quando esta é passada como argumento no método *getPresences*.

-> objeto *PresencesServer* na Figura 1.

ServerApp.java – Implementa o *main*. Cria uma instância da classe *PresencesServer* e invoca o método *creatPresences()*.

-> objeto *ServerApp* na Figura 1.

server.policy – Define a política de segurança para os objetos Java RMI.

Client.java - Implementa o comportamento do cliente do nosso sistema: comunica ao servidor a sua presença e passa-lhe uma referência para o objeto *NewPresences*. O método *putPresence()* faz o *lookup* do objeto que implementa o serviço de presenças (objeto *Presences*), através do nome "/PresencesRemote", sendo-lhe devolvido o *stub* para comunicar a sua presença a esse mesmo objeto remoto *Presences*.

-> objeto *Client* na Figura 1.

ClientApp.java – Implementa o *main*. Cria uma instância da classe *Client* e invoca o método *putPresence()*. A invocação da aplicação cliente deve ser feita da seguinte forma: `java ClientApp <ip Cliente> <ip Servidor de nomes Java RMI>`.

-> objeto *ClientApp* na Figura 1.

Tarefas

T.1 >> Obtenha o código fonte disponível na página da UC (*blackboard*). O código deste sistema está organizado em duas diretorias: *Client* e *Server*. Compile e execute o código de ambos os componentes cliente e servidor e execute o sistema: para executar o serviço deve invocar `java ServerApp` para invocar a aplicação cliente deve invocar `java ClientApp <ip Cliente> <ip Servidor de nomes Java RMI>`.

T.2 >> Explique as linhas de código (*interfaces*):

- Ficheiro *PresencesInterface.java*: 6 e 8.
- Ficheiro *NewPresencesInterface.java*: 4 e 6.

T.3 >> Explique as linhas de código (*servidor*):

- Ficheiro *PresencesServer.java*: 37, 18 e 23.
- Ficheiro *Presences.java*: 9 e 36.

T.4 >> Explique as linhas de código (*Cliente*):

- Ficheiro *NewPresences.java*: 10.
- Ficheiro *Client.java*: 22, 24 e 26.

T.5 >> Compare a implementação do protocolo pedido-resposta sobre as primitivas *Socket* TCP com a implementação Java RMI nas perspetivas:

- Definição protocolo pedido-resposta
- Representação externa de dados
- Transparência de acesso
- Transparência de localização
- Transparência de concorrência

Conclusão

Cada aluno deverá submeter na página da UC um ficheiro em formato *word* com as suas respostas às diferentes tarefas até ao final do dia **5 de dezembro de 2021**. Não devem ser incluídas figuras.

Os alunos cujo o resultado do trabalho (documento *word*) apresente uma percentagem de similaridade inferior a 75% (*SafeAssign*) terão uma bonificação de 0,5 valores na classificação prática final.