Sistemas Distribuídos
Aula 05

Cléver Ricardo Guareis de Farias
DCM/FFCLRP/USP

Conteúdo Programático

- Visão geral Java RMI
- Desenvolvimento de aplicações via RMI

Definições básicas (1)

- Objeto remoto
  - objeto cujos métodos podem ser invocados a partir de outra máquina virtual Java, potencialmente localizada em outro computador
  - descrito por uma ou mais interfaces remotas

Definições básicas (2)

- Interface remota
  - interface escrita em Java que declara os métodos de um objeto remoto
- Remote Method Invocation (RMI)
  - implementação Java de chamada remota de procedimento
  - tem a mesma sintaxe de uma chamada local

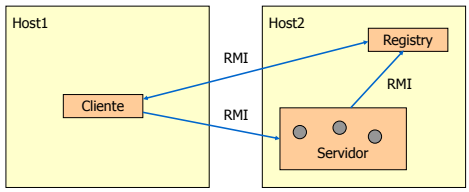
Visão geral (1)

- Tipicamente uma aplicação RMI consiste de um programa servidor e um programa cliente
- Programa servidor cria um conjunto de objetos remotos, disponibiliza as referências para estes objetos e espera pela invocação de operações nestes objetos

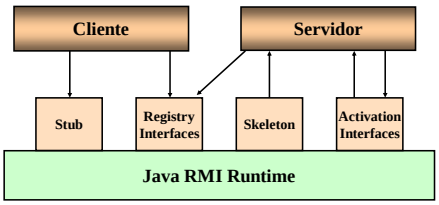
Visão geral (2)

- Programa cliente obtém uma ou mais referências para os objetos servidores e invoca os métodos suportados por estes objetos
- A plataforma RMI provê os mecanismos através dos quais o servidor e o cliente se comunicam e trocam informações

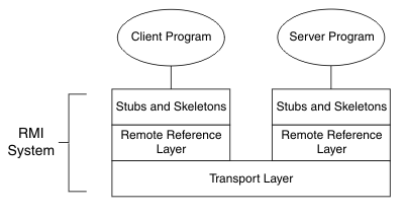
Visão geral (3)



Arquitetura Java RMI (4)



Arquitetura Java RMI (5)



Vantagens de Java RMI (1)

Vantagens de Java RMI (2)

- Simplicidade de uso
- Coleta de lixo (garbage collection) distribuída
  - RMI faz uso da coleta distribuída para remover objetos servidores que não são mais referenciados por objetos clientes
  - uso de contadores na máquina virtual Java para manter o número de referências (“live references”) locais ou remotas

Conteúdo Programático

- Visão geral Java RMI
- Desenvolvimento de aplicações via RMI

Desenvolvimento de aplicações RMI

- Projetar e implementar os componentes (objetos cliente e servidor) da aplicação distribuída
- Compilar arquivos fontes
- Disponibilizar o acesso aos objetos servidores
- Executar a aplicação

Desenvolver obj. servidor (1)

- Projetar interface remota
  - uma interface remota é definida através da especialização da interface java.rmi.Remote

```
public interface HelloInterface extends java.rmi.Remote {
        // métodos da interface
}
```


Desenvolver obj. servidor (2)

- a execução de cada método definido em uma interface remota pode resultar na ocorrência de uma exceção RMI (java.rmi.RemoteException). A ocorrência desta exceção durante a execução de uma chamada indica tanto uma falha de comunicação quanto um erro no protocolo

```
public interface HelloInterface extends java.rmi.Remote {
        String sayHello() throws java.rmi.RemoteException;
}
```


Desenvolver obj. servidor (3)

- Implementar interface remota
  - uma interface remota é implementada em três passos:
    - declaração da interface sendo implementada
    - definição do construtor para o objeto remoto
    - definição do comportamento associado a cada método

Desenvolver obj. servidor (4)

- Declaração da interface sendo implementada
  - para indicar que uma classe implementa uma interface remota, é necessário declarar que esta classe especializa a classe java.rmi.server.UnicastRemoteObject e implementa a referida interface

```
public class HelloInterfaceImpl
        extends java.rmi.server.UnicastRemoteObject
        implements HelloInterface { /... }
```


Desenvolver obj. servidor (5)

- Definição do construtor para o objeto remoto
  - o construtor da classe simplesmente chama o construtor da superclasse

```
public HelloInterfaceImpl() throws java.rmi.RemoteException {
        super();
}
```

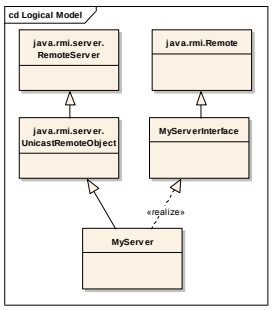
Desenvolver obj. servidor (6)

- Definição do comportamento associado a cada método
  - cada método definido na interface remota deve ser implementado
  - objetos passados como parâmetros podem ser praticamente de qualquer tipo
    - objetos remotos são passados como referência
      - a referência de um objeto remoto é um stub
    - objetos locais são passados como cópia usando a serialização de objetos

Desenvolver obj. servidor (7)

```
public String sayHello() throws java.rmi.RemoteException{
        return "Hello World!";
}
```


Desenvolver obj. servidor (8)



Disponibilizar acesso ao servidor (1)

- Antes que o cliente possa invocar um método em um objeto remoto, é necessário obter a referência do objeto remoto
  -

Disponibilizar acesso ao servidor (2)

- Tipicamente este serviço é utilizado para se obter a referência de um primeiro objeto a ser utilizado. A partir de então, normalmente este objeto provê suporte para a busca da referência de outros objetos que se façam necessários

Disponibilizar acesso ao servidor (3)

- A interface java.rmi.registry.Registry é utilizada para representar o serviço de nomes da plataforma
  - serviço é utilizado para o registro, procura e remoção de referências a objetos remotos
- Servidor de nomes é localizado a partir da invocação do método estático getRegistry da classe java.rmi.registry.LocateRegistry

Disponibilizar acesso ao servidor (4)

- O método getRegistry pode ter como parâmetros as seguintes informações:
  - host (opcional) indica a máquina onde o registry está localizado
    - se omitido assume localhost
  - port (opcional) indica o nro da porta onde o registry espera por requisições
    - se omitido assume a porta 1099

Disponibilizar acesso ao servidor (5)

- O objeto que implementa a interface remota é então instanciado e esta referência é exportada para o servidor de nomes através dos métodos disponíveis na interface Registry
  - ocorre normalmente no corpo do main

Disponibilizar acesso ao servidor (6)

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class HelloServer {
  public static void main(String[] args) {
    try {
      HelloInterfaceImpl obj = new HelloInterfaceImpl();

      // Bind this object instance to the name "HelloServer"
      Registry registry = LocateRegistry.getRegistry("localhost");
      registry.rebind("HelloServer", obj);

      System.out.println("HelloServer bound in registry");
    } catch (RemoteException e) {
      System.out.println("HelloServer err: " + e.getMessage());
    }
  }
}
```


Disponibilizar acesso ao servidor (6)

- Métodos da interface Registry

void	**bind**(String name, Remote obj) Binds the specified name to a remote object.
String[]	**list**(String name) Returns an array of the names bound in the registry.
Remote	**lookup**(String name) Returns a reference, a stub, for the remote object associated with the specified name.
void	**rebind**(String name, Remote obj) Rebinds the specified name to a new remote object.
void	**unbind**(String name) Destroys the binding for the specified name that is associated with a remote object.

Disponibilizar acesso ao servidor (7)

- Por razões de segurança um objeto só pode registrar (bind/rebind) ou remover (unbind) referências de objetos remotos se este estiver executando na mesma máquina do registry

Desenvolver obj. cliente (1)

- O cliente é um objeto que simplesmente obtem a referência do objeto servidor e requisita a execução de serviços no mesmo

Desenvolver obj. cliente (2)

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class HelloClient {
   public static void main(String[] args) {
      HelloInterface obj = null;
      try {
         Registry registry = LocateRegistry.getRegistry();

         // Lookup object reference associated to the name "HelloServer"
         obj = (HelloInterface)registry.lookup("HelloServer");
         String message = obj.sayHello();
         System.out.println(message);
      } catch (Exception e) {
         System.out.println("HelloClient exception: " + e.getMessage());
         e.printStackTrace();
      }
   }
}
```


Compilar arquivos fontes

- Um arquivo fonte é compilado através do compilador javac

  javac nome_do_arquivo.java

- Stub e skeleton são gerados automaticamente

Executar a aplicação

- Inicializar o registry
  - Windows: start rmiregistry
  - Linux: rmiregistry &
- Inicializar o servidor
  - java nome_classe_servidor
- Inicializar o cliente
  - java nome_classe_cliente

Observações

- O servidor de nomes deve ser executado no mesmo diretório da aplicação servidora
  - restrição segurança ???
- Incluir a definição do endereço do servidor como uma propriedade antes de instanciar o objeto remoto no servidor:
  - System.setProperty("java.rmi.server.hostname", "143.107.142.65");
  - restrição sistema operacional Linux ???

Exercícios (1)

- Desenvolva um sistema RMI composto de um cliente e um servidor no qual o servidor retorne quaisquer valores do tipo String que lhe sejam enviados. Os dados enviados para o servidor poderão ser passados para o cliente como parâmetro

```
public interface EchoInterface extends java.rmi.Remote {
        String echo(String word)
           throws java.rmi.RemoteException;
}
```


Exercícios (2)

- Modifique o exercício anterior de forma que o servidor inverta os valores que lhe são passados

```
public interface InversionInterface
        extends java.rmi.Remote {
  String invertText(String text)
        throws java.rmi.RemoteException;
}
```

Exercícios (3)

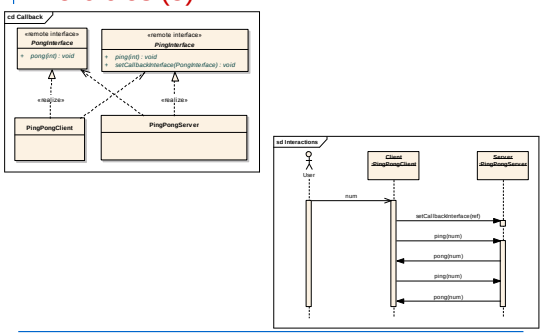
- Desenvolva um sistema de cálculo remoto composto de um cliente e um servidor no qual o servidor implemente a seguinte interface

```
public interface CalculatorInterface extends java.rmi.Remote {
    public int add(int a, int b) throws java.rmi.RemoteException;
    public int sub(int a, int b) throws java.rmi.RemoteException;
    public int times(int a, int b) throws java.rmi.RemoteException;
    public int div(int a, int b) throws java.rmi.RemoteException;
}
```


Exercícios (4)

- Considere as entidades representadas a seguir e seus respectivos comportamentos. Defina as interfaces representadas e implemente o comportamento observado. Considere que este comportamento deverá se repetir um número máximo de vezes.

Exercícios (5)



Exercícios (6)

- Desenvolva uma aplicação de chat usando uma arquitetura centralizada. Considere as definições de interface abaixo:

```
public interface ChatInterface extends java.rmi.Remote {
  int joinGroup(String name, MessageInterface ref)
        throws java.rmi.RemoteException;
  void leaveGroup(int id) throws java.rmi.RemoteException;
  void message(int id, String msg) throws java.rmi.RemoteException;
}

    public interface MessageInterface extends java.rmi.Remote {
      void messageNotification(String sender, String msg)
            throws java.rmi.RemoteException;
      }
```