

Sistemas Distribuídos

Sistemas de Objetos Distribuídos
Prof. Tales Viegas

Evolução dos Sistemas Cliente/Servidor

- ▶ O modelo clássico (request-response) começou a ser implementado por instruções de baixo nível
 - Ex: Sockets
- ▶ Com o aparecimento das linguagens procedurais, passou-se à utilização do modelo RPC (Remote Procedure Calling)
- ▶ O sucesso das linguagens OO motivou o aparecimento de Objetos Distribuídos e o RMI (Remote Method Invocation)

Modelo de Objetos

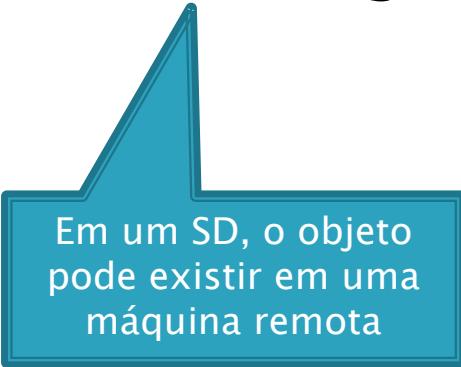
Pessoa cliente;

...

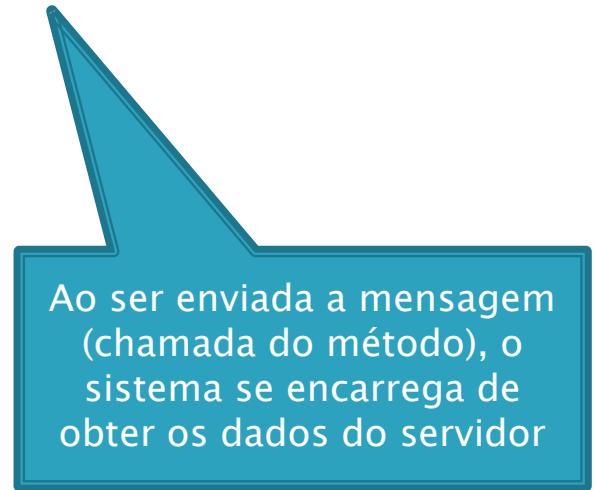
String endereco = cliente.getEndereco();



Esta variável está na máquina cliente



Em um SD, o objeto pode existir em uma máquina remota



Ao ser enviada a mensagem (chamada do método), o sistema se encarrega de obter os dados do servidor

Objetos Locais vs Objetos Remotos

- ▶ Em um sistema OO, os objetos são acessados através de suas referências
- ▶ Para invocar um método precisamos da referência do objeto, do nome do método e dos argumentos correspondentes
- ▶ Referências podem ser atribuídas a variáveis, passadas como argumentos e devolvidas como resultado de métodos

Referência Remota

- ▶ No âmbito de um SD, é um identificador que pode ser usado para se referir a um particular e único objeto em todo o sistema.

Internet Addr.	Port Number	Time	Object ID	Interface of Remote Object
32 bits	32 bits	32bits	32 bits	

Referência Remota

- ▶ São idênticas a referências locais nos seguintes aspectos:
 - O objeto que recebe a invocação do método é especificado como o objeto local
 - Referências remotas podem ser passadas como argumentos e ser retornadas como invocação de métodos remotos

Objetos Remotos

- ▶ São passados por referência (são objetos de acesso global)
- ▶ Na prática, uma referência remota “aponta” para um objeto local, que funciona como um “proxy” do objeto remoto
- ▶ Na passagem de um objeto remoto, é passado um proxy que é guardado na máquina receptora, e para o qual serão passadas as futuras invocações ao objeto

Objetos Locais

- ▶ Implementam a interface `java.io.Serializable`
- ▶ São passados por cópia (a interface não faz sentido na máquina remota)
- ▶ Do lado do receptor, é criado um novo objeto que pode ser acessado localmente

Interface

- ▶ Especificação de um conjunto de métodos
- ▶ Uma classe que implemente a interface deverá, obrigatoriamente, implementar todos os métodos desta interface

Interface Remota

- ▶ Cada objeto remoto tem uma Interface Remota, que especifica quais de seus métodos podem ser invocados remotamente

Métodos Locais

- ▶ A invocação de um método local resulta na execução do código correspondente do objeto
- ▶ No final do fluxo de execução, retorna ao objeto invocador
- ▶ A invocação pode resultar em:
 - O estado do objeto é alterado
 - Outras invocações a outros objetos

Métodos Remotos

- ▶ A invocação de um objeto remoto pode resultar na invocação de outros objetos (assim como em métodos locais)
- ▶ Eventualmente os objetos envolvidos podem estar em máquinas diferentes
- ▶ Quando uma invocação ultrapassa a barreira de um processo ou computador, tem lugar uma invocação de método remoto (RMI)
- ▶ Para um objeto invocar um método remoto, ele deve possuir a referência ao objeto remoto

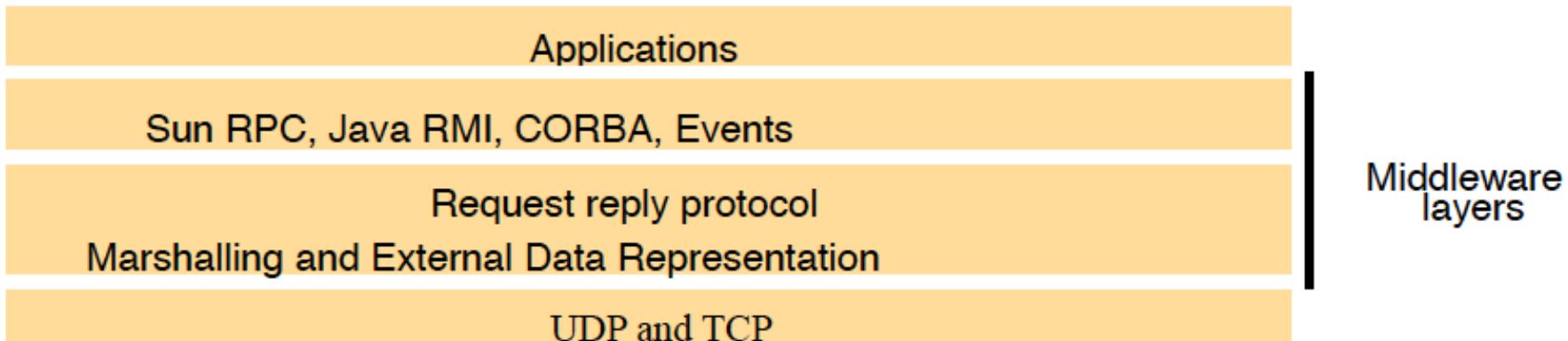
Exceções Locais

- ▶ Quando ocorre uma situação de erro (recuperável), é gerada uma Exceção
- ▶ É possível capturar uma exceção e transferir a execução para o bloco de código que irá tratar a condição de erro

Exceções Remotas

- ▶ Uma invocação remota, além das exceções ocorridas no processo receptor, também pode gerar exceções devido a:
 - Erro na transmissão dos argumentos
 - O processo que contém o objeto remoto “falhou”
 - Crash
 - Está tão ocupado que não consegue responder
 - O resultado da invocação se perdeu

Invocação Remota de Objetos



Middlewares

- ▶ CORBA – Common Object Request Broker Architecture
 - Object Management Group (OMG)
- ▶ DCOM – Distributed Component Object Model
 - Microsoft
- ▶ RPC – Remote Procedure Call
 - Sun/Oracle
- ▶ RMI – Remote Method Invocation
 - Java RMI (Sun/Oracle)
 - CORBA é uma forma de RMI
 - .Net Remoting
- ▶ SOAP – Simple Access Object Protocol
 - Microsoft, Sun/Oracle, ...

Medidas de Tolerância a Falhas

Medidas de Tolerância a Falhas			Semântica de Invocação
Retransmissão da Mensagem de Requisição	Filtro de Duplicidade	Reexecução do procedimento ou retransmissão da Resposta	
Não	Não aplicável	Não aplicável	Talvez
Sim	Sim	Reexecução do Procedimento	Pelo menos uma vez
Sim	Sim	Retransmissão da Resposta	No máximo uma vez

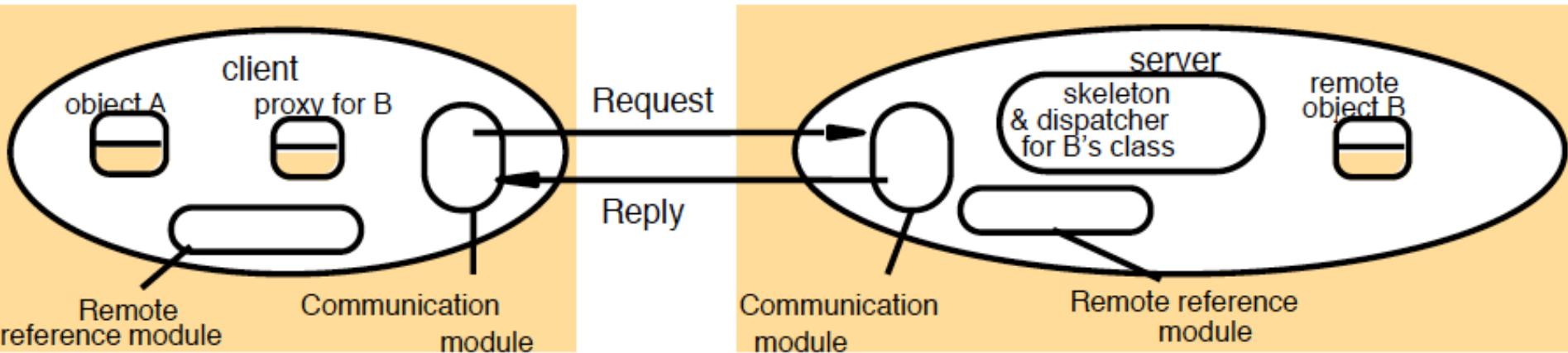
Medidas de Tolerância a Falhas

- ▶ Java RMI e CORBA fornecem “no máximo uma vez” (At-most-once)
 - O invocador recebe uma resposta e, nesse caso, sabe que o método foi executado uma única vez, ou recebe uma Exceção, o que significa que o método ou foi executado uma vez ou não foi executado
- ▶ CORBA permite “talvez” (maybe) para métodos que não devolvem resultado
 - O invocador não sabe se o método foi ou não executado

Medidas de Tolerância a Falhas

- ▶ Sun RPC fornece “pelo menos uma vez” (at-least-once)
 - O invocador ou recebe uma resposta e, nesse caso, o método foi executado pelo menos uma vez ou...
 - Recebe uma Exceção informando que não foi recebido resultado e, neste caso, o servidor pode ter falhado.
 - Em qualquer dos casos, execuções repetidas podem ter originado valores errados

Implementação do RMI



Objeto A invoca um método no Objeto B

Implementação do RMI

- ▶ Módulo de Comunicação
 - Implementa o protocolo pedido-resposta
- ▶ Módulo de Referência Remota
 - Mantém uma tabela de referências remotas, que contém a correspondência entre as referências locais e remotas
 - A tabela do servidor contém uma entrada para o objeto B
 - A tabela do cliente contém uma entrada para o proxy de B

Implementação do RMI

- ▶ O software RMI
 - Camada de Software entre a aplicação e os módulos de comunicação e referência
- ▶ Proxy
 - Torna transparente a invocação do método remoto
 - Recebe a invocação, serializa os argumentos e envia a invocação através do módulo de comunicação
 - Quando recebe o resultado, desserializa e o envia ao objeto cliente

Implementação do RMI

- ▶ **Dispatcher**
 - Recebe o pedido do módulo de comunicação e encaminha para um skeleton
- ▶ **Skeleton**
 - Dessaializa os argumentos
 - Invoca o método no objeto local
 - Recebe o resultado
 - Encaminha-o no sentido reverso

Java RMI

- ▶ Suponhamos a construção de um objeto servidor que implementa uma calculadora simples
- ▶ Passo 1: Definir a interface do objeto remoto
 - Tem de ser definida como uma interface filha de **java.rmi.Remote**
 - Todos os métodos devem lançar (throws) a Exceção **java.rmi.RemoteException**

Java RMI

- ▶ **Passo 2: Implementar a Interface Remota**
 - O objeto remoto deve ser uma classe filha de **java.rmi.server.UnicastRemoteObject** e implementar a interface remota definida no Passo 1
 - Nota: Qualquer classe usada como parâmetro ou resultado deve ser serializável

Java RMI

- ▶ **Passo 3: Implementar o servidor**
 - Criar uma instância de nomes e ligá-la ao serviço de nomes (RMI Registry)
- ▶ **Passo 4: Desenvolver o cliente que usa a interface remota**
- ▶ **Passo 5: Gerar stubs (proxies) e skeletons**
- ▶ **Passo 6: Iniciar o RMI Registry (se não foi criado no servidor)**
- ▶ **Passo 7: Criar um arquivo com política de segurança**
- ▶ **Passo 8: Executar o servidor e o cliente**

Interface do Objeto Remoto

```
public interface Calculator extends java.rmi.Remote{  
  
    public double soma(double a, double b) throws  
        java.rmi.RemoteException;  
    public double subtracao(double a, double b) throws  
        java.rmi.RemoteException;  
    public double multiplicacao(double a, double b) throws  
        java.rmi.RemoteException;  
    public double divisao(double a, double b) throws  
        java.rmi.RemoteException;  
}
```

Implementar a Interface Remota

```
import java.rmi.RemoteException;

public class CalculatorImpl extends java.rmi.server.UnicastRemoteObject implements Calculator {

    // Necessita criar um construtor invocando o construtor da
    // classe pai
    CalculatorImpl() throws RemoteException{
        super();
    }

    @Override
    public double soma(double a, double b) throws RemoteException {
        return a + b;
    }

    @Override
    public double subtracao(double a, double b) throws RemoteException {
        return a - b;
    }

    @Override
    public double multiplicacao(double a, double b) throws RemoteException {
        return a * b;
    }

    @Override
    public double divisao(double a, double b) throws RemoteException {
        return a / b;
    }
}
```

Implementar o Servidor

- ▶ Criar um gestor de segurança e instalá-lo
- ▶ Criar uma instância do objeto remoto
- ▶ Registrar o objeto remoto no serviço de nomes

Implementar o Servidor

```
import java.net.MalformedURLException;
import java.rmi.*;

public class CalculatorServer {

    public CalculatorServer(){

        // Cria um gestor de segurança e instala-o
        System.setProperty("java.security.policy", "secutiry.policy");
        System.setSecurityManager(new RMI SecurityManager());
        try {
            // Inicializa o Registry
            java.rmi.registry.LocateRegistry.createRegistry(2099);
            // Cria o objeto remoto. Detalhe para o tipo
            // da variável do objeto instanciado
            Calculator calc = new CalculatorImpl();
            // Registra o objeto no registro de nomes
            Naming.rebind("rmi://localhost:2099/CalculatorService", calc);
        } catch (RemoteException | MalformedURLException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args){
        new CalculatorServer();
    }
}
```

Implementar o Servidor

- ▶ Os servidores unicast são o tipo mais simples de servidor
- ▶ Referência para ele são válidas apenas enquanto o processo que instanciou o objeto está executando
- ▶ Comunicação cliente/servidor usa o protocolo TCP

Implementar o Cliente

- ▶ Obter a referência do objeto a partir do serviço de nomes (RMI Registry)
 - É necessário saber o nome da máquina e do objeto remoto
- ▶ Executar as invocações remotas (executar os métodos no objeto remoto)

Implementar o Cliente

```
import java.net.MalformedURLException;
import java.rmi.*;

public class CalculatorClient {

    public static void main(String[] args){
        try {
            Calculator c = (Calculator)Naming.lookup
                ("rmi://localhost:2099/CalculatorService");
            System.out.println(c.soma(3, 4));
            System.out.println(c.subtracao(3, 4));
            System.out.println(c.multiplicacao(3, 4));
            System.out.println(c.divisao(3, 4));
        } catch (RemoteException | 
                 NotBoundException | 
                 MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```

Iniciar o RMI Registry

- ▶ Um cliente precisa de um meio para obter a referência ao objeto remoto
- ▶ Um serviço de nomes (“binder”) mantém uma tabela com a correspondência entre a referência remota do objeto e um nome textual (URL-style)
- ▶ O serviço de nomes é usado pelo servidor para registrar os seus objetos remotos por um nome e pelo cliente para pesquisar se o servidor contém o objeto desejado

RMI Registry

- ▶ O serviço de nomes do RMI é o RMI Registry, baseado na interface `java.rmi.registry.Registry`, que define as operações
- ▶ `void rebind(String nomeObjeto, Remote objeto)`
 - Substitui o nome do objeto remoto
- ▶ `void unbind(String nomeObjeto)`
 - Remove a referência do objeto remoto
- ▶ `Remote lookup(String nomeObjeto)`
 - Retorna a referência do objeto remoto

RMI Registry

- ▶ Para acessar o endereço de um objeto remoto a partir do serviço de nomes precisamos:
 - O endereço da máquina que executa o RMIRegistry no qual o objeto remoto está registrado
 - A porta onde o serviço RMIRegistry está recebendo requisições
 - O nome que o objeto remoto tem no RMIRegistry
- ▶ `rmi://nomeMaquina:porta/nomeObjeto`

Iniciar o RMIRegistry

- ▶ Para podermos executar o servidor RMIRegistry é necessário executá-lo:
 - `rmiregistry <porta>`
- ▶ Para iniciar via código, é necessário executar o comando:
 - `java.rmi.registry.LocateRegistry.createRegistry(<porta>)`

Política de Segurança

- ▶ Em Java, através de um gestor de segurança (Security Manager), é possível controlar os acessos a um servidor
- ▶ Um servidor só precisa de um Security Manager se houver transferência de arquivos de uma máquina para outra
- ▶ As permissões de um programa Java são especificadas em um arquivo de policy, por exemplo:
 - `java -Djava.security.policy=file:c:\teste\Calculator.policy`

Política de Segurança

- ▶ Exemplo de um arquivo de policy:

```
grant {  
    permission java.security.AllPermission;  
};
```

- ▶ Clientes precisam de permissão de connect
- ▶ Servidores precisam de permissão de accept e de connect para acessar o registro de nomes

Executar

- ▶ Executar o servidor e o cliente

Modelo de Threading

- ▶ A especificação do Java RMI não faz garantias sobre o modelo de threading no momento de servir as requisições remotas
- ▶ Uma única Thread pode servir a várias requisições
- ▶ Pode ser usada uma Thread por requisição
- ▶ Na prática, invocações remotas que chegarem concorrentemente em um mesmo objeto são despachadas para Threads diferentes