Sistemas Distribuídos

11 Java RMI

- **Exemplo de Aplicação Distribuída**
- **Serviço de Nomes**
- **Gerente de Segurança**
- ***Callbacks***

Profª Ana Cristina B. Kochem Vendramin
DAINF / UTFPR

Introdução

- Estende o modelo de objeto Java para fornecer suporte a objetos distribuídos.
- Utiliza a mesma sintaxe para invocações remotas e locais.
- Interfaces remotas estendem a **interface Remote** (pacote java.rmi).
- Métodos remotos devem lidar com ***RemoteExceptions***.

Exemplo de Aplicação Distribuída em Java RMI

- ***Whiteboard*** compartilhado
  - Permite a um grupo de usuários compartilhar uma visão comum de uma superfície para desenhar objetos gráficos, tais como, retângulos, linhas e círculos.
  - O servidor mantém o estado corrente dos desenhos.
  - O servidor permite aos clientes recuperarem o último desenho do quadro através de *polling*.
  - O programa servidor possui um número de versão (inteiro) que é incrementado a cada novo desenho.

Exemplo de Aplicação Distribuída em Java RMI

- Duas interfaces remotas: **Shape** e **ShapeList**.
- GraphicalObject é uma classe que mantém o estado de um objeto gráfico (tipo, posição, cor da linha e cor de preenchimento) e fornece métodos para acessar e atualizar seu estado.
- GraphicalObject deve implementar a interface Serializable.
- Na interface ShapeList, o método newShape é utilizado por um cliente para passar uma instância de GraphicalObject ao servidor.
- O servidor cria um objeto remoto do tipo Shape contendo o estado do GraphicalObject e retorna uma referência de objeto remoto como resultado → interface remota Shape.

Exemplo de Aplicação Distribuída em Java RMI

```
import java.rmi.*;
import java.util.Vector;

public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject  getAllState() throws RemoteException;        1
}
public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException;  2
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```


Passagem de Referências de Objetos

- Um parâmetro será passado como uma referência de um objeto remoto quando seu tipo for definido como uma interface remota.
- Exemplo: o valor de retorno do método newShape da interface ShapeList é definido como Shape - a interface remota.
  - Quando uma referência de objeto remoto é recebida, esta pode ser utilizada para realizar invocações no objeto remoto a qual se refere.

Passagem de Objetos

- Na interface ShapeList, o argumento do método newShape e o valor de retorno de getAllState são serializados e passados por valor.
  - Ambos do tipo GraphicalObject.
- Quando um objeto é passado por valor, um novo objeto é criado no processo receptor.
- Os métodos do novo objeto podem ser invocados localmente, possivelmente causando um estado diferente do estado do objeto original no emissor.

Serviço de Nomes – RMI Registry

- Uma instância do RMI registry deve ser executada em cada computador que hospeda objetos remotos.
- Mantém uma tabela que mapeia nomes em referências de objetos remotos hospedados naquele computador.
- Acessado por métodos da classe Naming, cujos argumentos são da forma:
  - //NomedoComputador:porta/NomedoObjeto
  - Se o computador e porta forem omitidos assume-se o computador local e a porta default 1099

Classe Naming - *RMI Registry*

- ***void rebind (String name, Remote obj)***
  - Usado pelo servidor para registrar uma referência de objeto remoto.
- ***void bind (String name, Remote obj)***
  - Igual ao anterior, porém uma exceção será gerada caso o nome já exista.
- ***void unbind (String name, Remote obj)***
  - Remove um registro.
- ***Remote lookup(String name)***
  - Usado pelos clientes para obter uma referência de objeto remoto.
- ***String [] list()***
  - Retorna um array de strings contendo os nomes registrados.

Exemplo de Aplicação Distribuída em Java RMI

Versão completa disponível em **www.cdk4.net/rmi**

Programa Servidor *Whiteboard*

- Servidor representa cada forma (desenho) como um objeto gráfico remoto que implementa a interface *Shape* mantendo o seu estado e seu número de versão.
- Servidor representa uma coleção de formas que implementa a interface *ShapeList* mantendo-as em um vetor.

Programa Servidor

```
import java.rmi.*;
public class ShapeListServer{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
         try{
            ShapeList aShapeList = new ShapeListServant();
        Naming.rebind("Shape List", aShapeList );
            System.out.println("ShapeList server ready");
            }catch(Exception e) {
            System.out.println("ShapeList server main " + e.getMessage());}
    }
}
```


Classes Serventes

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;
public class ShapeListServant extends UnicastRemoteObject implements ShapeList
{
     private Vector theList;              // contains the list of Shapes
     private int version;
    public ShapeListServant() throws RemoteException{
        theList = new Vector();
        version = 0; }
    public Shape newShape(GraphicalObject g) throws RemoteException {
            version++;
            Shape s = new ShapeServant( g, version);
            theList.addElement(s);
            return s; }
    public  Vector allShapes() throws RemoteException { return theList; }
    public int getVersion() throws RemoteException { return version; }
}
```

Classes Serventes

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class ShapeServant extends UnicastRemoteObject implements Shape
{
        int myVersion;
        GraphicalObject theG;
  public ShapeServant(GraphicalObject g, int version) throws RemoteException {
        theG = g;
        myVersion = version;
  }
  public int getVersion() throws RemoteException {
        return myVersion;
  }
  public GraphicalObject getAllState() throws RemoteException {
        return theG;
  }}
```


Programa Clientes

- Procura uma referência do objeto remoto invocando o método *lookup* do serviço de nomes.
- Uma vez obtida a referência, continua enviando RMIs para o objeto remoto.
- No exemplo, o cliente invoca o método *allShapes* e recebe um vetor de referências de objetos remotos para todos os desenhos armazenados no servidor.

Programa Clientes

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
   public static void main(String args[]){
     System.setSecurityManager(new RMISecurityManager());
     ShapeList aShapeList = null;
     try{
          aShapeList  = (ShapeList) Naming.lookup("//cdk3.net/ShapeList");
          Vector sList = aShapeList.allShapes();
     } catch(RemoteException e) {System.out.println(e.getMessage());
     }catch(Exception e) {System.out.println("Client: " + e.getMessage());}
   }
}
```


Exemplo de política de segurança

```
grant {
    permission java.net.SocketPermission "*:1024-65535","connect,accept";
    permission java.lang.RuntimePermission "accessClassInPackage.sun.*";
   permission java.lang.RuntimePermission "createSecurityManager";
   permission java.lang.RuntimePermission "setSecurityManager";
};
```


Gerente de Segurança

- *System.setSecurityManager(new **RMISecurityManager**());*
- *Estabelecendo as propriedades do sistema no código:*
  - *Properties **props** = new Properties();*
  - ***props**.put("**java**.security.**policy**","**arquivo_politica**");*
- *ou via linha de comando:*
  - *Java -Djava.security.policy=arquivo_politica aplicacao*

Implementação de *callback*

- Cliente cria um objeto remoto - ***callback object*** - que implementa uma interface com um método para o servidor invocar.
- Servidor fornece um método para permitir que clientes interessados informem as referências remotas de seus objetos *callback*.
- Servidor registra os clientes interessados em uma lista.
- Sempre que um evento ocorre, o servidor notifica os clientes interessados.
  - Por exemplo, o servidor *Whiteboard* chamaria seus clientes sempre que um novo objeto gráfico fosse adicionado.

Profa. Ana Cristina B. Kochem Vendramin
DAINF/UTFPR

Implementação de *callback*

- Interface *WhiteboardCallback* poderia ser definida como:

public interface WhiteboardCallback extends Remote {
void callback (int version) throws RemoteException;
};

- Esta interface é implementada pelo cliente.
- Habilita o servidor a enviar um número de versão sempre que um novo objeto for adicionado.
- Para que isto aconteça, o cliente deve ter se registrado com o servidor.

Implementação de *callback*

- A interface *ShapeList* requer dois métodos adicionais:

int register (WhiteboardCallback callback) throws RemoteException;
int deregister (int callbackId) throws RemoteException;

- Após o cliente ter obtido uma referência do objeto remoto contendo a interface *ShapeList* e ter criado seu objeto *callback*, ele invoca o método *register* que retorna um inteiro (*callbackId*).
- Quando o cliente terminar, ele deverá chamar *deregister* para informar ao servidor que não requer mais *callbacks*.

Referências Bibliográficas

- Coulouris, George; Dollimore, Jean; Kindberg, Tim. Distributed Systems: concepts and design. Third Edition. Addison-Wesley 2001.
- Coulouris, George; Dollimore, Jean; Kindberg, Tim; tradução João Tortello. Sistemas Distribuídos: conceitos e projeto. 4. ed. Bookman 2007.