

Curso Java.

Todo sobre RMI en Java.

(Programación distribuida)

Lo primero que debemos saber es qué es un RPC. Un RPC, o también conocido como Llamada a Procedimientos Remotos la cual consiste en comunicarse y sincronizarse con otros procesos que están en otro equipo.

Cuando llamamos a un proceso se produce lo siguiente:

* *Se crean los canales de comunicación de forma automática.*
* *Se envía la información que hayamos introducido en la llamada.*
* *Gestión automática de bloqueos por si hay varias llamadas a un mismo proceso y está ocupado atendiendo anteriores.*
* *Recepción de la respuesta y preparación de los parámetros de salida*

Cuando un proceso es llamado se produce lo siguiente:

* *Se crean los canales de comunicación de forma automática.*
* *Se espera a recibir la información, si no está disponible, bloqueamos el proceso.*
* *Si recibimos la llamada, se cargan los parámetros y se ejecuta código.*
* *Enviamos la respuesta como resultado de dicha ejecución de código.*

La comunicación es síncrona, por lo que los procesos deben «esperarse». Hay varios tipos de RPC, pero que el usaremos es el RMI (Invocación de métodos remotos en Java).

**RMI**

Permite pasar objetos por referencia, recolección de basura distribuída y paso de tipos arbitrarios. Cuando un determinado programa exporta un objeto, estarán accesibles a través de la red mediante un servidor de registro (rmiregistry), dicho programa permanece a la espera de peticiones mediante el protocolo TCP (gestionado internamente por RMI, no debemos hacer nada). Una vez que lo hemos exportado, un programa puede usarlo e invocar los métodos que proporciona dicho objeto. La invocación de objetos es totalmente transparente, y se realiza como si dicha invocación hubiera sido de forma local (en la misma máquina).

RMI posee una arquitectura en cuatro capas:

* *Capa de aplicación: se encarga de implementar las aplicaciones cliente-servidor, se realizan llamadas de alto* *nivel y los métodos se declaran extendiendo java.rmi.Remote*
* *Capa proxy: está muy relacionada con la capa de aplicación, se encarga de preparar las llamadas a objetos remotos.*
* *Capa de referencia remota: Es la responsable de la semántica de las invocaciones remotas así como gestionar la replicación de los objetos.*
* *Capa de transporte: Se encarga de realizar la conexión, se realiza mediante el protocolo JRMP, que sólo entiende Java.*

Cualquier programa RMI es cliente-servidor. En el servidor RMI se realiza la creación de objetos remotos, las referencias a ellos para que sean accesibles y se espera a que accedan a ellos. En el cliente RMI invocamos los métodos del servidor y obtenemos una referencia a objetos remotos en el servidor.

**Pasos para la creación de un servidor RMI**:

Lo primero que debemos hacer es definir la interfaz remota la cual debe ser public y heredar de java.rmi.Remote y lanzará java.rmi.RemoteException.

A continuación debemos hacer una clase que implemente la interfaz remota y que heredará de UnicastRemoteObject. El constructor lanzará UnicastRemoteException.

Hacer visible el objeto reoto mediante Naming.rebind()

Compilar y ejecutar.

**Pasos para la creación de un cliente RMI:**

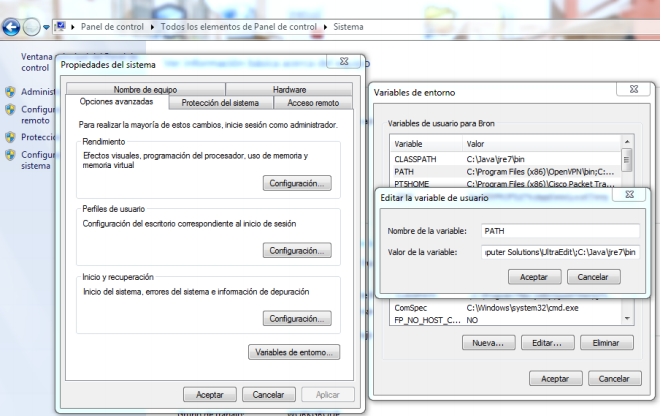
Definir una clase para obtener los objetos remotos mediante Naming.lookup()

Compilar y ejecutar

Hasta aquí hemos llegado con la teoría, ahora omenzamos la práctica.

Lo primero que debemos hacer es ejecutar correctamente el servidor de registro RMI, para ello debemos editar las variables de entorno de Windows tal y como está en la captura de pantalla.

En el CLASSPATH ponemos C:Javajre7bin y en PATH lo mismo (en ambos).



Ahora deberéis abrir una consola en CMD y ejecutar: rmiregistry para iniciar el servidor de registro, sino, RMI no funcionará y dará error al ejecutar. Una vez que lo hemos ejecutado, nos aparecerá el cursor parpadeando pero sin poner nada.

Ahora vamos a hacer un programa muy sencillo con RMI:

Primero vamos a crear la interfaz, nos fijamos que hereda de Remote y que lanza la excepción RemoteException:

*[java]*

*package rmi;*

*import java.rmi.Remote;*

*import java.rmi.RemoteException;*

*public interface InterfaceRMI extends Remote {*

*String PrimerMetodo () throws RemoteException;*

*}*

*[/java]*

Ahora vamos a crear una clase que haga uso de esta interfaz, simplemente lo que hará es devolver una cadena cuando invoquemos el método.

*[java]*

*package rmi;*

*import java.rmi.RemoteException;*

*import java.rmi.server.UnicastRemoteObject;*

*public class UsarInterfaceRMI extends*

*UnicastRemoteObject implements InterfaceRMI {*

*public UsarInterfaceRMI() throws RemoteException {*

*}*

*@Override*

*public String PrimerMetodo() throws RemoteException { return("Hola Primer Método");*

*}*

*}*

*[/java]*

Creamos el servidor RMI:

*[java]*

*package rmi;*

*import java.net.MalformedURLException;*

*import java.rmi.Naming;*

*import java.rmi.RemoteException;*

*public class ServidorRMI {*

*public static void main(String[] args) throws RemoteException, MalformedURLException { UsarInterfaceRMI uso = new UsarInterfaceRMI(); //Creamos el objeto usoNaming.rebind("//127.0.0.1/Objeto", uso); //Hacemos visible el objeto remoto, mucho cuidado con /Objeto, debe*ser igual en el cliente RMI.

System.out.println("El servidor ha arrancadocorrectamente");

}

}

[/java]

Por último, creamos el cliente RMI:

*[java]*

*package rmi;*

*import java.net.MalformedURLException;*

*import java.rmi.Naming;*

*import java.rmi.NotBoundException;*

*import java.rmi.RemoteException;*

*public class ClienteRMI {*

*public static void main(String[] args) throws NotBoundException, MalformedURLException,*

*RemoteException {*

*InterfaceRMI inter = (InterfaceRMI)*

*Naming.lookup("//127.0.0.1/Objeto"); //Creamos el objeto remoto mediante la orden, acordaos de poner /Objeto, o si queréis ponéis /loquesea, pero debe ser igual en el servidor y en el cliente*

*inter.PrimerMetodo(); //Llamamos al método (por si hiciera alguna acción, en este caso devuelve un string por lo que**no hará nada aquí).*

*System.out.println(""+inter.PrimerMetodo()); //Imprimimos el string que hemos "return" en el servidor RMI*

*}*

*}*

*[/java]*

RMI es una herramienta muy potente, el poder ejecutar métodos remotos permite que tengamos un gran rendimiento en nuestras aplicaciones.

Ejemplo bastante simple de RMI

* [Java Standard Edition](https://www.javamexico.org/categorias/java_standard_edition)

He aquí un ejemplo bastante simple de RMI (Servidor y Cliente).

Primeramente escribimos la interfaz remota con los métodos requeridos. Por ejemplo, un método para saludar:

**package** com.company.app;

**import** java.rmi.Remote;  
**import** java.rmi.RemoteException;

**public** **interface** TestRemote **extends** [**Remote**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/Remote.html) {

[**String**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/String.html) sayHello(**[String](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/String.html)** name) **throws** **[RemoteException](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/RemoteException.html)**;

}

A continuación, implementamos esta interfaz en una clase. Para mayor simplicidad, he utilizado una clase anónima. Y ya que estamos en esto, escribimos también el programa Servidor, el cual registrará la instancia generada a partir de la clase anónima:

**package** com.company.app;

**import** java.rmi.AlreadyBoundException;  
**import** java.rmi.Remote;  
**import** java.rmi.RemoteException;  
**import** java.rmi.registry.LocateRegistry;  
**import** java.rmi.registry.Registry;  
**import** java.rmi.server.UnicastRemoteObject;

**public** **class** Server {

**public** **static** **void** main([**String**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/String.html)[] args) **throws** [**RemoteException**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/RemoteException.html), [**AlreadyBoundException**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/AlreadyBoundException.html) {

[**Remote**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/Remote.html) stub = **[UnicastRemoteObject](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/server/UnicastRemoteObject.html)**.exportObject(**new** TestRemote() {  
            @[**Override**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/Override.html)  
            **public** [**String**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/String.html) sayHello([**String**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/String.html) name) **throws** **[RemoteException](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/RemoteException.html)** {  
                **return** "Hello, " + name;  
            }  
        }, 0);

[**Registry**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/registry/Registry.html) registry = [**LocateRegistry**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/registry/LocateRegistry.html).createRegistry(**[Registry](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/registry/Registry.html)**.REGISTRY\_PORT);  
        registry.bind("Test", stub);

    }

}

Por último, escribimos el cliente, el cual accederá al objeto remoto en el servidor.

**package** com.company.app;

**import** java.rmi.NotBoundException;  
**import** java.rmi.RemoteException;  
**import** java.rmi.registry.LocateRegistry;  
**import** java.rmi.registry.Registry;

**public** **class** Client {

**public** **static** **void** main([**String**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/String.html)[] args) **throws** [**RemoteException**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/RemoteException.html), [**NotBoundException**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/NotBoundException.html) {  
        [**Registry**](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/registry/Registry.html) registry = **[LocateRegistry](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/rmi/registry/LocateRegistry.html)**.getRegistry();  
        TestRemote testRemote = (TestRemote) registry.lookup("Test");  
        **[System](http://java.sun.com/j2se/1%2E5%2E0/docs/api/java/lang/System.html)**.out.println(testRemote.sayHello("JavaMexico"));  
    }

}

Una vez que corremos el servidor, este se quedará esperando por peticiones. Y por cada vez que corramos el cliente, éste imprimirá «Hello, JavaMexico».