|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**Отчёт по лабораторной работе №2 «Концепция удаленного вызова метода (RMI) в JAVA» по дисциплине «Архитектура клиент-серверных приложений»**

**Студент группы** ИКБО-10-19 Полетаев Владимир Викторович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** преподаватель Степанов П.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Москва 2021

**Цель работы:**

Знакомство с механизмами удаленного вызова процедур в JAVA.

**Теоретическое введение:**

RMI - это объектно-ориентированный эквивалент RPC (удаленного вызова процедур). Это называется вызовом удаленного метода. Вызов удаленного метода (RMI) позволяет объекту Java вызывать метод объекта, запущенного на другом компьютере. RMI обеспечивает удаленную связь между Java-программой.

Архитектура RMI основана на одном важном принципе: определение поведения и реализация этого поведения считаются разными понятиями. RMI дает возможность разделить и выполнить на разных JVM код, определяющий поведение, и код, реализующий поведение.

*Работа приложения RMI*

Обобщённо, работу распределённого приложения RMI можно описать следующими шагами:

1. Когда клиент делает вызов удаленному объекту, он принимается заглушкой, которая в конечном итоге передает этот запрос в RRL.

2. Когда клиентский RRL получает запрос, он вызывает метод invoke () объекта remoteRef. Он передает запрос в RRL на стороне сервера.

3. RRL на стороне сервера передает запрос скелету (прокси на сервере), который, наконец, вызывает требуемый объект на сервере.

4. Результат полностью передается клиенту.

**Постановка задачи:**

Используя информацию из описания данной практической работы, необходимо реализовать удалённый метод решения квадратных уравнений общего вида ax2 + bx + c = 0. При этом, условие уравнения передавать на сервер, а клиентская часть должна получать результат в виде объектов пользовательского класса. Клиент и сервер должны работать на одном хосте.

**Код программы:**

Создаём интерфейс, который обязательно должен быть наследован от Remote:

package com.example.demo.Pr2.server;  
  
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface TheRootFinder extends Remote {  
  
 String find(double a, double b, double c) throws RemoteException;  
  
}

Далее нужен класс, имплементирующий наш интерфейс и реализующий его метод:

package com.example.demo.Pr2.server;  
  
import java.rmi.RemoteException;  
  
public class Finder implements TheRootFinder{  
 @Override  
 public String find(double a, double b, double c) throws RemoteException {  
  
 double d = Math.*pow*(b, 2) - 4 \* a \* c;  
 System.*out*.println(d);  
 System.*out*.println(Math.*sqrt*(d));  
 if(d<0)  
 return "Корней нет";  
 Double x1 = (-b + Math.*abs*(d)) / (2 \* a);  
 Double x2 = (-b - Math.*abs*(d)) / (2 \* a);  
 if(x1.equals(x2))  
 return "x = " + x1;  
 return String.*format*("x1 = %s, x2 = %s", x1, x2);  
 }  
}

Код серверной части, предоставляющий объект для удалённого доступа:

package com.example.demo.Pr2.server;  
  
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 ServerMain {  
  
 public static final String *UNIQUE\_BINDING\_NAME* = "server.finder";  
  
 public static void main(String[] args) throws RemoteException, AlreadyBoundException, InterruptedException {  
  
 Finder finder = new Finder();  
 Registry registry = LocateRegistry.*createRegistry*(7777);  
 Remote stub = UnicastRemoteObject.*exportObject*(finder, 0);  
 registry.bind(*UNIQUE\_BINDING\_NAME*, stub);  
  
 Thread.*sleep*(Integer.*MAX\_VALUE*);  
  
 }  
  
}

Код клиентской части, где устанавливается соединение с сервером и куда передаются параметры, установленные именно на клиенте:

package com.example.demo.Pr2.client;  
  
import com.example.demo.Pr2.server.TheRootFinder;  
  
import java.rmi.NotBoundException;  
import java.rmi.RemoteException;  
import java.rmi.registry.LocateRegistry;  
import java.rmi.registry.Registry;  
  
public class ClientMain {  
  
 public static final String *UNIQUE\_BINDING\_NAME* = "server.finder";  
  
 public static void main(String[] args) throws RemoteException, NotBoundException {  
  
 Registry registry = LocateRegistry.*getRegistry*(7777);  
 TheRootFinder rootFinder = (TheRootFinder) registry.lookup(*UNIQUE\_BINDING\_NAME*);  
 String result = rootFinder.find(15, 55, 5);  
 System.*out*.println(result);  
  
 }  
  
}

**Результат работы программы**

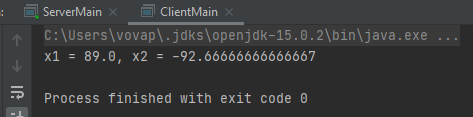


Рис. 1 Скриншот результатов работы программы

**Выводы:**

В работе была изучена и применена на практике информация о механизмах удаленного вызова процедур в JAVA.