Professor Hans Noodles

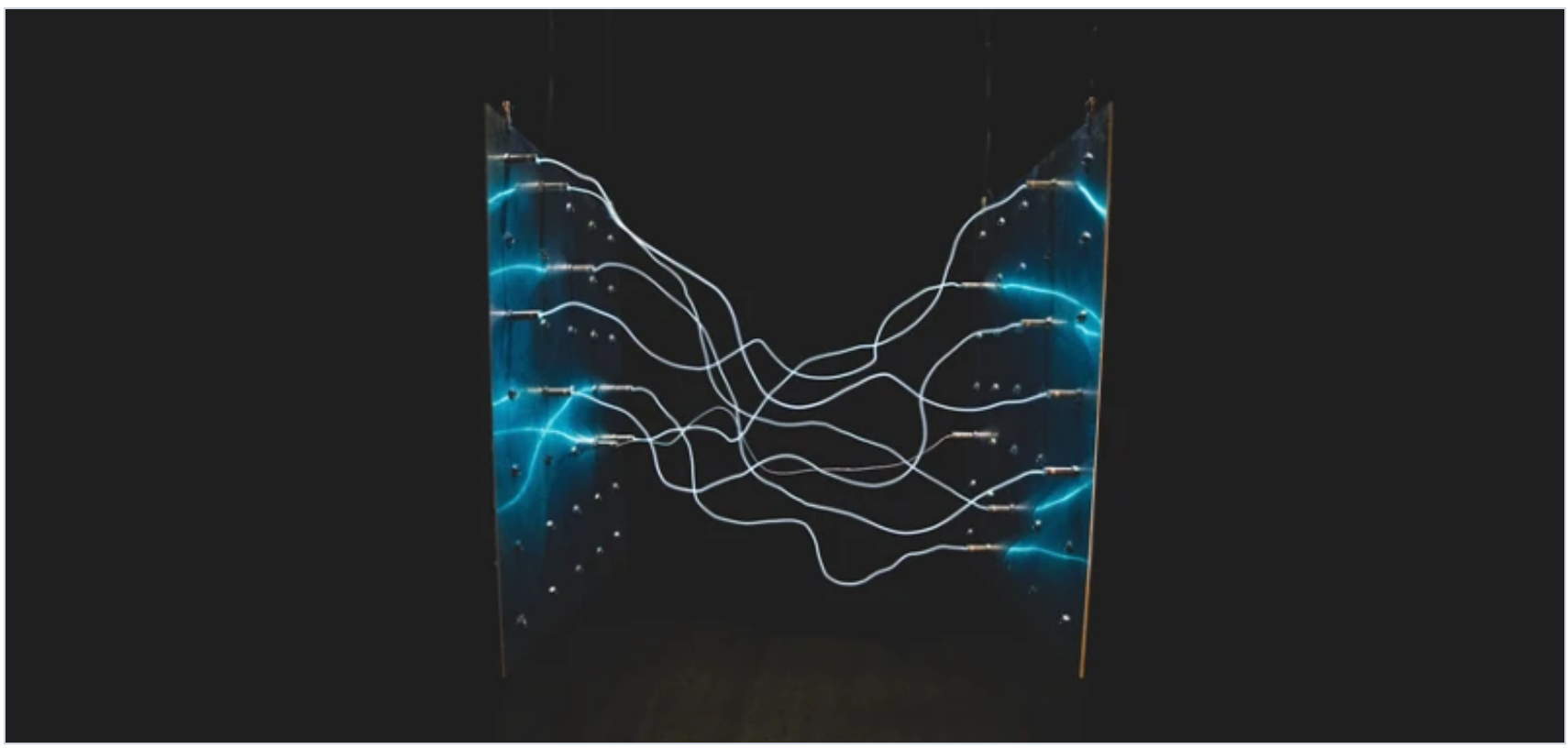
41 уровень

14.05.2019 17990 20

RMI: практика использования

Привет!

Сегодня мы рассмотрим довольно интересную тему — **RMI**. Это расшифровывается как **Remote Method Invocation** — удаленный вызов методов.



При помощи RMI ты можешь научить две программы общаться между собой, даже если они находятся на разных компьютерах.

Звучит круто? :) А ведь это не так уж и сложно сделать!

В сегодняшней лекции мы разберемся, из каких частей состоит RMI-взаимодействие и как его настроить.

Первое, что нам понадобится — это **клиент** и **сервер**.

Можешь особо не углубляться в компьютерную терминологию. В случае с RMI это просто две программы. Одна из них будет содержать какой-то объект, а вторая — вызывать методы этого объекта.

Вызвать в одной программе методы объекта, который находится в другой программе — такого мы еще не делали! Самое время попробовать! :)

Чтобы не утонуть в деталях, пусть наша программа будет простой

Вообще, на серверах обычно происходят какие-то вычисления, которые запрашивает клиент. Так будет и у нас.

В роли сервера у нас будет простая программа-калькулятор. У нее будет всего один метод — `multiply()`. Он будет умножать два числа, которые ему отправила программа-клиент, и возвращать результат.

Прежде всего нам понадобится интерфейс:

```
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3
4  public interface Calculator extends Remote {
5
6     int multiply(int x, int y) throws RemoteException;
7  }
```

Зачем нам интерфейс?

Дело в том, что работа RMI основана на создании прокси, которые ты изучал в одной из прошлых лекций.

А работа с прокси, как ты, наверное, помнишь, ведется именно на уровне интерфейсов, а не классов.

К нашему интерфейсу есть 2 важных требования!

1. Он должен наследовать интерфейс-маркер Remote.
2. Все его методы должны выбрасывать RemoteException (это не делается в IDE автоматически, надо написать руками!).

Теперь нам надо создать класс-сервер, который будет реализовывать наш интерфейс `Calculator`.

Тут тоже все довольно просто:

```
1  import java.rmi.RemoteException;
2
3  public class RemoteCalculationServer implements Calculator {
4
5     @Override
6     public int multiply(int x, int y) throws RemoteException {
7         return x*y;
8     }
9
10 }
```

Здесь даже комментировать особо нечего :)

Теперь нам нужно написать программу-сервер, которая будет настраивать и запускать наш серверный класс-калькулятор.

Она будет выглядеть вот так:

```
1  import java.rmi.AlreadyBoundException;
2  import java.rmi.Remote;
3  import java.rmi.RemoteException;
4  import java.rmi.registry.LocateRegistry;
```

```
7
8  public class ServerMain {
9
10    public static final String UNIQUE_BINDING_NAME = "server.calculator";
11
12    public static void main(String[] args) throws RemoteException, AlreadyBoundException, InterruptedE
13
14        final RemoteCalculationServer server = new RemoteCalculationServer();
15
16        final Registry registry = LocateRegistry.createRegistry(2732);
17
18        Remote stub = UnicastRemoteObject.exportObject(server, 0);
19        registry.bind(UNIQUE_BINDING_NAME, stub);
20
21        Thread.sleep(Integer.MAX_VALUE);
22
23    }
24 }
```

Давай разбираться :)

В первой строке мы создаем какую-то строковую переменную:

```
1  public static final String UNIQUE_BINDING_NAME = "server.calculator";
```

Эта строка — **уникальное имя удаленного объекта**. По этому имени программа-клиент сможет найти наш сервер: ты увидишь это позже.

Далее мы создаем наш объект-калькулятор:

```
1  final RemoteCalculationServer server = new RemoteCalculationServer();
```

Тут все понятно. Далее уже поинтереснее:

```
1  final Registry registry = LocateRegistry.createRegistry(2732);
```

Эта штука под названием Registry — **реестр удаленных объектов**. «Удаленных» не в том смысле, что мы их удалили с компа, а в том, что к объектам из этого регистра возможен удаленный доступ из других программ :)

В метод `LocateRegistry.createRegistry()` мы передали число 2732. Это номер порта. Если не знаешь, что такое порт — можно почитать вот тут, но сейчас тебе достаточно запомнить, что это уникальный номер, по которому другие программы смогут найти наш реестр объектов (это ты тоже увидишь ниже).

Едем дальше. Посмотрим, что у нас происходит в следующей строке:

```
1  Remote stub = UnicastRemoteObject.exportObject(server, 0);
```

В этой строке мы создаем **заглушку**.

Заглушка (stub) инкапсулирует внутри себя весь процесс удаленного вызова. Можно сказать, что это **самый важный элемент** RMI

Что же она делает?

1. Принимает всю информацию об удаленном вызове какого-то метода.
2. **Если у метода есть параметры, заглушка десериализует их.** Обрати внимание на этот пункт! Параметры, которые ты передаешь методам для удаленного вызова, должны быть сериализуемыми (ведь они будут передаваться по сети). У нас такой проблемы нет — мы передаем просто числа. Но если ты будешь передавать объекты, не забудь об этом!
3. После этого она вызывает нужный метод.

Мы передаем в метод `UnicastRemoteObject.exportObject()` наш объект-калькулятор server. Таким образом мы делаем возможным удаленный вызов его методов.

Нам осталось сделать только одно:

```
registry.bind(UNIQUE_BINDING_NAME, stub);
```

Мы «регистрируем» нашу заглушку в реестре удаленных объектов под тем именем, которое придумали в самом начале. Теперь клиент сможет ее найти!

Возможно, ты обратил внимание, что в конце мы усыпили главный поток программы:

```
Thread.sleep(Integer.MAX_VALUE);
```

Нам просто нужно, чтобы сервер работал долгое время. Мы ведь будем запускать в IDEa сразу два метода `main()`: сначала серверный (в классе `ServerMain`, который мы уже написали), а потом — клиентский (в классе `ClientMain`, который мы напишем ниже). Важно, чтобы программа-сервер не вырубилась, пока мы будем запускать клиент, поэтому мы ее просто усыпили на долгое время. Работать она все равно будет :)

Теперь мы можем запустить метод `main()` нашего сервера. Пусть он работает и ждет, когда программа-клиент вызовет какой-нибудь метод :)

Теперь давай напишем программу-клиент! Она будет отправлять нашему серверу числа для умножения.

```
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.calculator";

   public static void main(String[] args) throws RemoteException, NotBoundException {

       final Registry registry = LocateRegistry.getRegistry(2732);

       Calculator calculator = (Calculator) registry.lookup(UNIQUE_BINDING_NAME);

       int multiplyResult = calculator.multiply(20, 30);

       System.out.println(multiplyResult);
   }
}
```

Она выглядит несложно. Что же здесь происходит?

Во-первых, клиент должен быть в курсе уникального имени объекта, методы которого он будет вызывать удаленно. Поэтому в программе-клиенте мы тоже создали переменную `public static final String UNIQUE_BINDING_NAME = "server.calculator";`

Далее, в методе `main()` мы получаем доступ к регистру удаленных объектов.

Для этого нам нужно вызвать метод **`LocateRegistry.getRegistry()`** и передать туда номер порта, на котором создавали наш регистр в программе ServerMain — порт 2732 (этот номер был выбран для примера, ты можешь попробовать использовать другой):

```
1  final Registry registry = LocateRegistry.getRegistry(2732);
```

Теперь нам осталось только получить из регистра нужный объект! Это легко, ведь мы знаем его уникальное имя!

```
1  Calculator calculator = (Calculator) registry.lookup(UNIQUE_BINDING_NAME);
```

Обрати внимание на приведение типов. Мы приводим полученный объект к **интерфейсу `Calculator`, а не к конкретному классу** `RemoteCalculationServer`. Как мы и говорили в начале лекции, работа RMI основана на использовании прокси, поэтому удаленный вызов доступен только для методов интерфейсов, а не классов.

В конце мы удаленно вызываем метод `multiply()` у нашего объекта и выводим результат в консоль.

```
1  int multiplyResult = calculator.multiply(20, 30);
2  System.out.println(multiplyResult);
```

Метод `main()` в классе `ServerMain` мы уже давно запустили, самое время запустить метод `main()` в программе-клиенте `ClientMain`!

Вывод в консоль:

600

Вот и все! Наша программа (даже две!) успешно выполнила свою функцию :)

Если у тебя есть время и желание, можешь немного разнообразить ее. Например, сделать так, чтобы калькулятор поддерживал все четыре стандартные операции, а в качестве параметров передавались не числа, а объект **`CalculationInstance`**`(int x, int y)`.

В качестве дополнительного материала можешь посмотреть статьи и примеры — вот здесь:

- Использование Java RMI
- RMI: Основы на примере
- RMI что это

Увидимся на следующих занятиях! :)

НАЧАТЬ ОБУЧЕНИЕ

ЯЗЫК ИНТЕРФЕЙСА

Русский