Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Московский государственный
университет путей сообщения»

Кафедра «Автоматизированные системы управления»

А.В. ВАРФОЛОМЕЕВ

КЛИЕНТ-СЕРВЕРНОЕ ВЗАИМОДЕЙСТВИЕ НА БАЗЕ ТЕХНОЛОГИИ УДАЛЕННОГО ВЫЗОВА ПРОЦЕДУР

Рекомендовано редакционно-издательским советом
университета в качестве методических указаний
для студентов специальностей 230201 и 230102

МОСКВА – 2013

УДК 004
В 18

Варфоломеев А.В. Клиент-серверное взаимодействие на базе технологии удаленного вызова процедур: Методические указания. -- М.: МИИТ, 2013, — 41с.

В методических указаниях рассказывается об организации взаимодействия приложений с использованием технологии удаленного вызова процедур, приводятся рекомендации по реализации клиент-серверного взаимодействия с использованием интерфейса Java RMI.

Методические указания ориентированы на студентов, владеющих технологией объектно-ориентированного программирования на платформе Java, понимающих механизмы сетевого взаимодействия по протоколу TCP/IP, имеющих опыт параллельного программирования и синхронизации потоков.

СОДЕРЖАНИЕ

1. ТЕХНОЛОГИЯ УДАЛЕННОГО ВЫЗОВА ПРОЦЕДУР

1.1. Введение

Удаленный вызов процедур (remote procedure call, RPC) – технология, позволяющая программам вызывать процедуры (функции, методы) в другом адресном пространстве.

С помощью средств RPC, прикладная программа может обратиться к процедуре, выполняющейся под управлением другой программы на другом компьютере. При этом вызов процедуры будет осуществляться таким образом, как будто она является локальной для вызывающей программы.

Основное применение технологии RPC – организация клиент-серверного взаимодействия приложений в распределенных системах. Также RPC часто используется для взаимодействия нескольких процессов на одном компьютере.

Технология RPC ориентирована на *синхронное* взаимодействие приложений. В момент вызова удаленной процедуры программа блокируется и переходит в режим ожидания результатов выполнения процедуры. После того как результаты получены, программа продолжает свою работу.

На сегодняшний день существует большое количество разнообразных программных средств и протоколов, позволяющих реализовывать взаимодействие приложений на основе удаленного вызова процедур: ONC RPC, XML-RPC, Java RMI, COM/DCOM, CORBA, SOAP, и др. Средства RPC принято относить к категории *промежуточного программного обеспечения* (*middleware*).

1.2. Принципы работы

В процессе удаленного вызова процедур участвуют следующие основные объекты:

- ***Сервер*** – приложение, обладающее ***процедурой***, к которой предполагается удаленное обращение.
- ***Клиент*** – приложение, осуществляющее вызов удаленной процедуры.
- ***Стаб*** (***stub***) – программный компонент, управляющий процессом вызова удаленной процедуры. Стаб входит в состав клиентского и серверного приложений. Он отвечает за формирование сетевых сообщений, в частности, за представление параметров вызова процедуры и результатов ее выполнения.
- ***Ядро (kernel)*** – программный компонент, управляющий передачей данных по сети между клиентом и сервером.

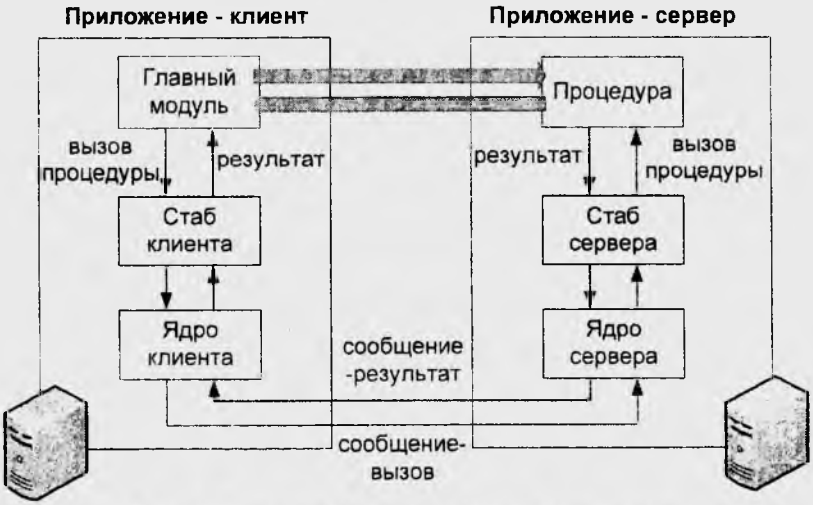

Рис. 1. Процесс вызова удаленной процедуры

Процесс вызова удаленной процедуры происходит следующим образом (рис. 1):

1. В момент вызова удаленной процедуры, *клиент* обращается к своему локальному *стабу*, и передает ему необходимые параметры.
2. *Клиентский стаб* упаковывает параметры вызова процедуры в сообщение. и обращается к *ядру* для передачи сообщении по сети.
3. *Ядро клиента* отправляет сообщение по сети на удаленную серверную машину. *Ядро сервера* принимает сообщение по сети и передает его *серверному стабу.*
4. *Серверный стаб* распаковывает сообщение и вызывает необходимую *процедуру* с заданными параметрами.
5. По завершению работы *процедуры*, *серверный стаб* формирует ответное сообщение с результатами работы процедуры, и обращается к ядру, для передачи сообщения клиенту.
6. *Ядро сервера* отправляет сообщение по сети на удаленную клиентскую машину. *Ядро клиента* принимает сообщение и передает его *клиентскому стабу.*
7. *Клиентский стаб* распаковывает сообщение, и отдает результаты *клиентскому модулю*, выполнившему вызов удаленной процедуры.

Разработчику клиент-серверной системы, базирующейся на технологии RPC, в большинстве случаев не требуется самостоятельно программировать серверные и клиентские *стабы* и *ядра*. Создание и подключение нужных *стабов* и *ядер* осуществляется средствами RPC.

Таким образом, использование средств RPC избавляет разработчика приложений от необходимости программировать сетевое взаимодействие и

самостоятельно управлять процессом передачи информации между программами.

1.3. Связывание

Для вызова удаленной процедуры клиенту необходимо знать месторасположение сервера.

Один из способов решения данной проблемы – указание сетевого адреса сервера непосредственно в клиентской программе. Такой подход принято называть ***статическим связыванием*** клиента и сервера.

Существенным недостатком статического связывания является отсутствие гибкости и масштабируемости клиент-серверной системы. При изменении адреса сервера или увеличении количества серверов, потребуется перекомпилировать или перенастраивать все клиентские приложения в распределенной системе, использующие статическое связывания.

Альтернативным способом является ***динамическое связывание*** клиента и сервера. При использовании данного подхода клиент не привязывается изначально к конкретному серверу, а узнает о местоположении сервера в процессе своего выполнения. Сведения о местоположении сервера предоставляются клиенту специальной службой-посредником, например, ***службой каталогов и имен*** (***naming and directory service***). В некоторых реализациях RPC в процессе динамического связывания клиенту передается непосредственно *стаб*, учитывающий особенности вызова текущей версии процедуры и содержащий адрес нужного сервера.

Основными недостатками динамического связывания являются сложность в реализации и затраты на сопровождение дополнительной службы-посредника.

2. JAVA RMI

2.1. Java RMI как средство RPC

Одной из реализаций технологии удаленного вызова процедур для платформы Java является интерфейс ***Java RMI*** (***Remote Method Invocation, удаленный вызов методов***).

RMI является объектно-ориентированной реализацией RPC. В рамках RMI приложения обращаются не к самостоятельным удаленным процедурам, а к *методам* определенного *удаленного объекта*.

Клиент-серверное взаимодействие на основе RMI осуществляется следующим образом. Приложение-сервер создает некоторый *объект* и регистрирует его для удаленного доступа. Клиентские приложения получают ссылку на этот объект и работают с ним таким образом, как будто он является для них обычным локальным объектом.

Обращение к серверному объекту клиент осуществляет с использованием *интерфейса объекта* (рис. 2). ***Интерфейс*** описывает серверные методы и их параметры, но не содержит реализации методов.

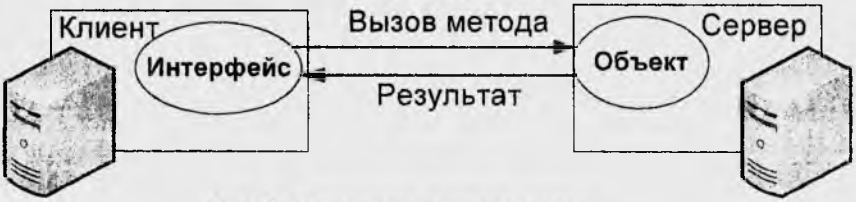

Рис. 2. Вызов удаленного метода

Средства RMI скрывают от разработчиков программ реализацию процесса сетевого взаимодействия между клиентским и серверным приложениями. За формирование сетевых пакетов и их передачу полностью отвечает промежуточное программное обеспечение RMI.

2.2. Архитектура Java RMI

Архитектура RMI представлена схематически на рис. 3.

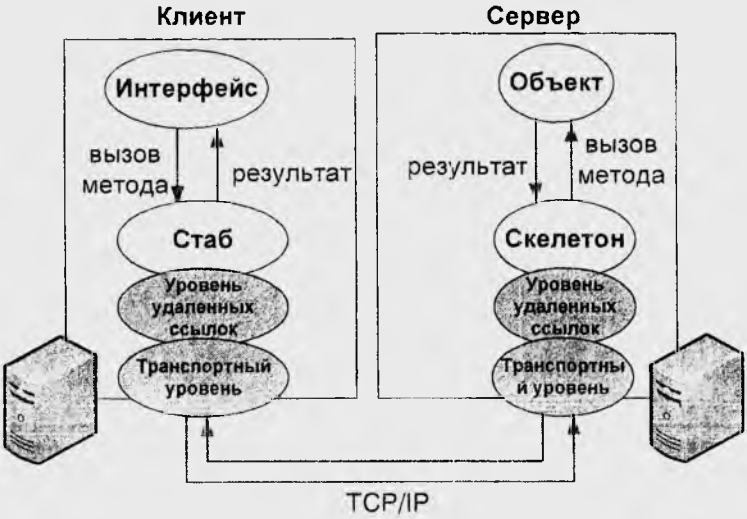

Рис. 3. Вызов удаленного метода

Клиент, вызывая через *интерфейс* метод *удаленного объекта*, на самом деле обращается к специализированному локальному объекту-заглушке – *стабу*. Клиентский стаб упаковывает параметры вызова метода в двоичное сообщение. Сформированное сообщение передается по сети на сервер. Его получает *серверный стаб*, который в терминологии RMI принято назвать ***скелетоном*** (***skeleton***). Задача скелетона – извлечь параметры из полученного сообщения и вызвать необходимый метод серверного объекта. Результат вызова метода скелетон упаковывает в двоичное сообщение и передает обратно стабу клиента. Стаб клиента распаковывает сообщение и возвращает результат вызвавшей его программе.

За стабом и скелетоном скрывается два вспомогательных уровня, соответствующих уровню *ядра (kernel)* RPC: ***уровень удаленных ссылок (remote reference layer)*** и ***транспортный уровень (transport layer)***.

Уровень удаленных ссылок реализует протокол взаимодействия, независимый от стаба и скелетона. Данный уровень знает, как нужно управлять ссылками клиента на удаленные объекты, и какую модель передачи данных использовать при взаимодействии клиента и сервера.

Транспортный уровень отвечает за доставку сетевых пакетов между клиентом и сервером.

2.3. Реестр RMI и связывание

Связывание клиента и сервера RMI осуществляется *динамически* через специальную службу – ***службу имен и каталогов***. Служба имен и каталогов обычно выполняется на общедоступном сервере, о расположении которого знают все клиенты в системе.

RMI может использовать много различных служб каталогов, включая службу ***Java Naming and Directory Interface*** (***JNDI***). В то же время, RMI включает в себя простую стандартную службу, называемую ***реестром RMI*** (***RMI registry***).

Реестр RMI работает на серверной машине, содержащей объекты удаленных служб. По умолчанию, реестр использует TCP-порт 1099. Основная функция реестра RMI – предоставление клиентам ссылок на удаленные серверные объекты.

Сервер, желающий предоставить свой объект удаленным приложениям, предварительно регистрирует его в реестре RMI под некоторым общеизвестным именем (рис. 4). Клиенты, желающие работать с удаленным

объектом, обращаются к реестру RMI и получают ссылку на объект по его имени.

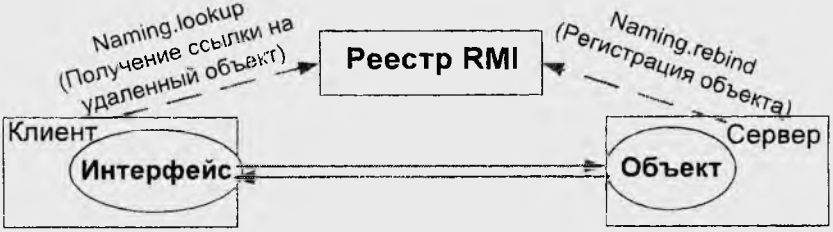

Рис. 4. Связывание клиента и сервера

3. РАЗРАБОТКА ПРИЛОЖЕНИЙ НА БАЗЕ JAVA RMI

Разработка клиент-серверной системы на базе RMI в самом простом случае предполагает выполнение следующих шагов:

1. Разработка интерфейса для удаленного (серверного) объекта. Интерфейс в дальнейшем должен быть доступен как клиентскому, так и серверному приложениям.
2. Разработка класса серверного объекта, поддерживающего данный интерфейс.
3. Разработка серверного приложения (приложения, управляющего серверным объектом).
4. Разработка клиентского приложения (приложения, обращающегося к удаленному серверному объекту).
5. Создание классов скелетона и стаба (начиная с J2SE 5.0, данный шаг не обязателен).

При работе клиент-серверной системы должен соблюдаться следующий порядок действий:

1. Запускается служба реестра RMI (это можно сделать как в самом серверном приложении, так и в отдельном процессе).

2. Создается серверный объект и регистрируется в реестре RMI.
3. Запускаются клиентские приложения, получают из реестра RMI ссылку на удаленный объект, и вызывают методы объекта.

3.1. Разработка интерфейса удаленного объекта

Методы серверного объекта, которые планируется предоставлять удаленным клиентам через RMI, должны быть описаны при помощи *интерфейса удаленного объекта*. Такой интерфейс должен расширять базовый интерфейс *java.rmi.Remote*. Кроме того, все методы интерфейса должны быть объявлены с возможностью генерации специального исключения *RemoteException*. Данное исключение сигнализирует о проблемах сетевого взаимодействия при вызове удаленного метода.

В следующем примере приводится интерфейс *RMIServer* для некоторого серверного объекта с методом *hello* (метод возвращает строку).

```
public interface RMIServer extends Remote
{
    public String hello() throws RemoteException;
}
```

3.2. Разработка класса серверного объекта

Серверный объект реализует разработанный выше интерфейс. При этом класс серверного объекта должен наследоваться от класса *UnicastRemoteObject*. Класс *UnicastRemoteObject* (пакет *java.rmi.server*) предоставляет базовые функциональные возможности, которые необходимы удаленным объектам для обслуживания удаленных запросов.

Конструкторы и методы класса *UnicastRemoteObject* могут генерировать исключение *RemoteException*, поэтому наследники класса *UnicastRemoteObject* должны

определять конструкторы, которые также могут генерировать данное исключение.

В следующем примере мы реализуем интерфейс *RMIServer* в классе *RMIServerImpl*.

```
public class RMIServerImpl
     extends UnicastRemoteObject
     implements RMIServer
{
    public RMIServerImpl() throws RemoteException
    {
    }
    public String hello() throws RemoteException
    {
       return "Hello from server!";
    }
}
```

Метод *hello* класса *RMIServerImpl* возвращает клиенту строку с приветствием от сервера: `"Hello from server!"`.

3.3. Разработка серверного приложения

Приложение-сервер должно создавать серверный объект и регистрировать его в реестре RMI под определенным именем.

При этом можно использовать внешний реестр *RMI* (приложение *rmiregistry*, в папке *java\bin*), а можно создать локальный реестр непосредственно из самого серверного приложения.

Использование внешнего реестра

При использовании внешнего реестра серверный код должен выглядеть следующим образом:

```
public class Server {
  public static void main(String[] args) {
    try
    {
        // Создаю серверный объект
        RMIServerImpl server = new RMIServerImpl();
        // Регистрирую его в реестре
        Naming.rebind("//127.0.0.1/SayHello",
                      server);
    }
    catch(RemoteException e)
    {
        e.printStackTrace();
    }
    catch(MalformedURLException e)
    {
        e.printStackTrace();
    }
  }
}
```

В начале программы мы создаем новый серверный объект:

```
RMIServerImpl server = new RMIServerImpl();
```

Затем мы регистрируем данный объект под именем *SayHello* в реестре RMI, служба которого выполняется по адресу 127.0.0.1 (localhost) и прослушивает порт по умолчанию (1099):

```
Naming.rebind("//127.0.0.1/SayHello", server);
```

При необходимости, порт может быть задан явно в строке URL:

```
Naming.rebind("//127.0.0.1:1099/SayHello", server);
```

Здесь *Naming* – это класс пакета java.rmi, позволяющий обращаться к реестру RMI. Статический метод *rebind* связывает объект с заданным *url*.

Перед выполнением приложения-сервера, использующего внешний реестр, необходимо запустить

службу реестра RMI. Сделать это можно из командной строки, выполнив программу *rmiregistry*, размещающееся в папке *bin* каталога *java*. Программа принимает на вход в качестве параметра командной строки номер порта, который будет прослушивать служба RMI. Если данный параметр не задан, будет прослушиваться порт RMI по умолчанию (1099). Для остановки службы реестра RMI следует использовать комбинацию клавиш *CTRL+C*.

Создание внутреннего реестра

Ниже представлен пример исходного кода сервера, создающего свой локальный реестр:

```
public class Server {
  public static void main(String[] args) {
    try
    {
        // Создаю службу реестра RMI на локальном
        // хосте, прослушивающую порт 1099
        Registry r =
              LocateRegistry.createRegistry(1099);
        // Создаю серверный объект
        RMIServerImpl server = new RMIServerImpl();
        // Регистрирую его в реестре
        r.rebind("SayHello", server);
    }
    catch(RemoteException e)
    {
        e.printStackTrace();
    }
  }
}
```

Статический метод *createRegistry* класса *java.rmi.registry.LocateRegistry* запускает службу реестра RMI и возвращает ссылку на нее. Метод принимает в качестве параметра номер порта TCP, который будет прослушивать служба.

Для работы с реестром используется интерфейс *java.rmi.registry.Registry*. Для регистрации объекта в реестре используется метод *rebind* интерфейса *Registry*. В данном случае, вместо полной строки URL указывается только имя, под которым объект будет зарегистрирован.

Следует отметить, что при работе с внутренним реестром запуск приложения rmiregistry не требуется.

3.4. Разработка клиентского приложения

Клиентское приложение обращается к удаленному реестру RMI и получает ссылку на серверный объект. Для работы с объектом (для вызова удаленных методов) клиентское приложение использует интерфейс серверного объекта.

Ниже представлен пример клиента, вызывающего метод *hello* удаленного серверного объекта.

```
public class Client {
  public static void main(String[] args) throws
       MalformedURLException, RemoteException,
       NotBoundException
  {
    RMIServer s =
       (RMIServer)Naming.lookup
                          ("//127.0.0.1/SayHello");
    String message = s.hello();
    System.out.println(message);
  }
}
```

Метод *lookup* класса *Naming* возвращает ссылку на серверный объект, зарегистрированный под именем *SayHello* в удаленном реестре RMI, служба которого выполняется по адресу 127.0.0.1 (*localhost*) и прослушивает порт по умолчанию (1099). Для работы с серверным объектом клиент использует интерфейс *RMIServer*.

3.5. Создание классов скелетона и стаба

В Java версии 1.1 классы стаба и скелетона необходимо было создавать самостоятельно при помощи специальной утилиты *rmic* (размещается *в* папке *bin* каталога *java)*.

Начиная с Java версии 1.2, необходимость создавать скелетоны отпала, однако стабы по-прежнему приходилось создавать вручную.

В J2SE 5.0 появилась поддержка динамической генерации класса стаба в процессе выполнения программы. Таким образом, использование утилиты *rmic* стало необязательным.

3.6. Особенности использования RMI

При разработке программного обеспечения по заданию, представленному в разделе 5, следует обратить внимание на следующие особенности RMI.

1. Передаваемые в удаленный метод параметры, а также возвращаемые значения могут относиться к примитивным типам java, а также являться локальными или удаленными объектами.

Передача локальных объектов осуществляется «по значению» (т.е. в метод передается копия объекта). При передаче таких объектов RMI использует механизм *сериализации.* В связи с этим классы передаваемых локальных объектов должны поддерживать интерфейс *java.io.Serializable.*

2. В момент вызова клиентом метода удаленного объекта на стороне сервера средствами RMI создается отдельный поток, в котором и происходит выполнение метода. Таким образом, обеспечивается возможность одновременного выполнения методов сервера несколькими клиентами. В связи с этим возникает необходимость синхронизировать работу методов серверного объекта.

4. ПРИМЕРЫ ПРОГРАММ JAVA RMI

4.1. Калькулятор

Система состоит из клиентского и серверного приложений. Сервер управляет объектом, предоставляющим клиентам удаленные методы, вычисляющие сумму (*sum*) и разность (*sub*) чисел.

Отдельно от клиентского и серверного приложений разрабатывается пакет *com.myrmi*, содержащий описание интерфейса серверного объекта. Клиент и сервер используют данный пакет как внешнюю библиотеку.

Пакет com.rmi: Интерфейс Calculator

```
package com.myrmi;
import java.rmi.*;

public interface Calculator extends Remote
{
    // вычисление суммы чисел x и y
    public int sum(int x, int y)
                    throws RemoteException;
    // вычисление разности чисел x и y
    public int sub(int x, int y)
                    throws RemoteException;
}
```

Приложение-сервер: Класс серверного объекта CalculatorImpl

```
import com.myrmi.Calculator;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

public class CalculatorImpl
                    extends UnicastRemoteObject
                    implements Calculator
{
```

```
    // конструктор
    public CalculatorImpl() throws RemoteException
    {
        super();
    }
    // реализация метода, вычисляющего сумму
    public int sum(int x, int y)
                        throws RemoteException
    {
        return x+y;
    }
    // реализация метода, вычисляющего разность
    public int sub(int x, int y)
                        throws RemoteException
    {
        return x-y;
    }
}
```

Приложение-сервер: Главный класс

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Server
{
    public static void main(String[] args)
      throws RemoteException, MalformedURLException
    {
        // создаю объект
        CalculatorImpl helloImpl =
                          new CalculatorImpl();
        // создаю реестр RMI
        Registry registry =
              LocateRegistry.createRegistry(123);
        // регистрирую объект в реестре
        registry.rebind("Calc", helloImpl);
        System.out.println("Server started!");
    }
}
```

Приложение-клиент:

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import com.myrmi.Calculator;

public class Client {

    public static void main(String[] args)
        throws NotBoundException, RemoteException,
               MalformedURLException
    {
        String url = "//localhost:123/Calc";
        // получаю ссылку на объект
        Calculator Q =
               (Calculator) Naming.lookup(url);
        System.out.println("RMI object found");
        // вызываю удаленные методы
        int x = Q.sum(10, 20);
        int y = Q.sub(10, 4);
        System.out.println(x);
        System.out.println(y);
    }
}
```

4.2. Калькулятор (с передачей объекта в качестве параметра)

Сервер предоставляет клиентам удаленные методы, вычисляющие сумму и разность чисел. Операнды для вычислений передаются через объект класса *Operands*.

Чтобы в удаленные методы могли передаваться объекты класса *Operands*, объявляем класс как *сериализуемый*. Чтобы класс *Operands* был доступен как серверному. так и клиентскому приложениям, помещаем его в общий пакет *com.myrmi*.

Пакет com.rmi: Класс Operands

```
package com.myrmi;
import java.io.Serializable;

public class Operands implements Serializable
{
    public Operands(int v1, int v2)
    {
        x = v1;
        y = v2;
    }
    public int x;
    public int y;
}
```

Пакет com.rmi: Интерфейс Calculator

```
package com.myrmi;
import java.rmi.*;

public interface Calculator extends Remote
{
    public int sum(Operands o)
                        throws RemoteException;
    public int sub(Operands o)
                        throws RemoteException;
}
```

Приложение-сервер: Класс серверного объекта CalculatorImpl

```
import com.myrmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
public class CalculatorImpl
                        extends UnicastRemoteObject
                        implements Calculator
{
    public CalculatorImpl() throws RemoteException
    {   super();
    }
```

```
    public int sum(Operands o)
                        throws RemoteException
    {
        return o.x+o.y;
    }
    public int sub(Operands o)
                        throws RemoteException
    {
        return o.x-o.y;
    }
}
```

Приложение-сервер: Главный класс

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server
{
    public static void main(String[] args)
      throws RemoteException, MalformedURLException
    {
        CalculatorImpl helloImpl =
                                new CalculatorImpl();
        Registry registry =
               LocateRegistry.createRegistry(123);
        registry.rebind("Calc", helloImpl);
        System.out.println("Server started!");
    }
}
```

Приложение-клиент:

```
import com.myrmi.*;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
```

```
public class Client
{
    public static void main(String[] args)
      throws NotBoundException, RemoteException,
                          MalformedURLException
    {
        String url = "//localhost:123/Calc";
        Calculator Q =
               (Calculator) Naming.lookup(url);
        System.out.println("RMI object found");
        Operands q = new Operands(20, 10);
        int x = Q.sum(q);
        int y = Q.sub(q);
        System.out.println(x);
        System.out.println(y);
    }
}
```

4.3. Удаленный массив

На сервер хранится массив целых чисел.

Сервер предоставляет клиентам удаленные методы, позволяющие добавлять числа в массив (*Add*) и вычислять сумму элементов массива (*Sum*).

Необходимо предотвратить одновременный доступ нескольких потоков к серверному массиву (такая ситуация возможна в случае, если несколько клиентов одновременно будут вызывать методы сервера). Для этого все обращения к массиву помещаются в блок синхронизации.

В данном примере реализовано два клиента. Клиент 1 в бесконечном цикле запрашивает у сервера сумму элементов массива и выводит ее на экран. Клиент 2 в бесконечном цикле запрашивает у сервера добавление в массив нового элемента.

Клиент 1 и клиент 2 могут быть функционировать одновременно. При этом они не будут мешать работе друг друга и не нарушат целостность серверного объекта.

Пакет com.rmi: Интерфейс MyArray

```
package com.myrmi;
import java.rmi.*;

public interface MyArray extends Remote
{
    // добавление элемента v в массив
    public void Add(Integer v)
              throws RemoteException;
    // вычисление суммы элементов массива
    public int Sum() throws RemoteException;
}
```

Приложение-сервер: Класс серверного объекта MyArrayImpl

```
import com.myrmi.*;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

public class MyArrayImpl
   extends UnicastRemoteObject implements MyArray
{
    // массив
    private ArrayList<Integer> x;
    // конструктор объекта
    public MyArrayImpl() throws RemoteException
    {
        x = new ArrayList<Integer>();
    }
    // добавление элемента в массив
    public void Add(Integer v)
                        throws RemoteException
    {
        synchronized (x) // блок синхронизации
        {
            x.add(v);
        }
    }
```

```
    // вычисление суммы элементов массива
    public int Sum() throws RemoteException
    {
        int sum=0;
        synchronized (x) // блок синхронизации
        {
            for (Object i : x)
                sum+=(Integer)i;
        }
        return sum;
    }
}
```

Приложение-сервер: Главный класс

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server
{
    public static void main(String[] args)
     throws RemoteException, MalformedURLException
    {
        MyArrayImpl helloImpl = new MyArrayImpl();
        Registry registry =
             LocateRegistry.createRegistry(123);
        registry.rebind("Array", helloImpl);
        System.out.println("Server started!");
    }
}
```

Приложение-клиент 1: Главный класс

```
import com.myrmi.*;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
```

```
public class Client1
{
    public static void main(String[] args)
        throws NotBoundException, RemoteException,
               MalformedURLException
    {
        String url = "//localhost:123/Array";
        MyArray Q = (MyArray) Naming.lookup(url);
        System.out.println("RMI object found");
        // непрерывно считаем сумму
        // элементов удаленного массива
        while (true)
            System.out.println(Q.Sum());
    }
}
```

Приложение-клиент 2: Главный класс

```
import com.myrmi.*;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class Client2
{
    public static void main(String[] args)
        throws NotBoundException, RemoteException,
               MalformedURLException
    {
        String url = "//localhost:123/Array";
        MyArray Q = (MyArray) Naming.lookup(url);
        System.out.println("RMI object found");
        // непрерывно добавляем в массив элементы
        while (true)
            Q.Add(Integer.valueOf(1));
    }
}
```

5.ЛАБОРАТОРНАЯ РАБОТА

5.1. Постановка задачи

Разработать клиент-серверную систему.

Взаимодействие клиента и сервера осуществляется в синхронном режиме на основе технологии удаленного вызова процедур (рис. 5).

Серверное приложение предоставляет удаленным клиентам процедуры/методы по управлению некоторыми объектами. Сведения об объектах и их характеристиках хранятся на стороне сервера в базе данных или файле XML (на усмотрение студента). Перечень необходимых операций над объектами и тип объектов определяется в соответствии с номером варианта студента.

Клиентское приложение выполняет удаленные процедуры, размещающиеся на сервере, и демонстрирует ответы пользователю.

Рис. 5. Клиент-серверная система

5.2. Требования к работе

Сервер запускается в фоновом режиме и не имеет пользовательского интерфейса. Сервер должен поддерживать одновременную работу с несколькими клиентами. Сервер должен обеспечивать выполнение операций, представленных в варианте задания студента.

Клиентское приложение не требует создания пользовательского интерфейса. Тестирование работоспособности системы может осуществляться на

основе сценариев, реализованных в программном коде клиента.

Характеристики объектов определяются студентами самостоятельно. Обязательной характеристикой для объектов всех типов является *идентификатор объекта*. Идентификатор требуется для того, чтобы клиент мог в запросе обозначить объект, над которым будут выполняться действия. Сервер должен обеспечивать уникальность идентификаторов в рамках множества однотипных объектов и выдавать клиенту ошибки при нарушении уникальности идентификатора (например, при попытке добавить новый объект с идентификатором, совпадающим с существующим объектом).

Например, для варианта №1, объект «Страна» может иметь характеристики:

- *код страны (уникальный идентификатор)*
- *название страны*

а объект «Город» – характеристики:

- *код города (уникальный идентификатор)*
- *ссылка на страну*
- *название города*
- *количество жителей*
- *признак столицы*

Обратите внимание, что во всех вариантах заданий объекты различных категорий находятся в иерархической зависимости. *Например, в варианте №1 у каждой страны может быть несколько городов, при этом один город принадлежит только одной стране.*

Рекомендуемый язык программирования – ***Java***. Рекомендуемое средство удаленного вызова процедур – ***Java RMI***.

5.3. Требования к оформлению отчета

Отчет должен содержать:

- титульный лист;
- постановку задачи;
- исходный код программ клиента и сервера;
- описание программы (краткое описание модулей, классов, методов, полей);
- подробное описание удаленных процедур или интерфейса удаленного объекта.

5.4. Варианты заданий

Вариант №1

Предметная область	Карта мира
Объекты	Страны, Города
Примечание	Карта мира содержит множество *стран*. Для каждой *страны* определено множество *городов*.
Требуемые операции	1. Добавление новой страны 2. Удаление страны 3. Добавление нового города в заданную страну 4. Удаление города 5. Редактирование города 6. Подсчет количества городов в стране 7. Выдача полного списка городов с указанием названия страны 8. Выдача списка городов для заданной страны 9. Выдача полного списка стран

Вариант №2

Предметная область	Библиотека
Объекты	Авторы, Книги
Примечание	Книги в библиотеке сгруппированы по *авторам*. У каждого *автора* имеется множество *книг*.
Требуемые операции	1. Добавление нового автора 2. Удаление автора 3. Добавление новой книги для автора

	4. Удаление книги 5. Редактирование книги 6. Подсчет общего количества книг 7. Выдача полного списка книг с указанием ФИО автора 8. Выдача книг заданного автора 9. Выдача списка авторов

Вариант №3

Предметная область	Отдел кадров
Объекты	Подразделения, Сотрудники
Примечание	Имеется множество *подразделений* предприятия. В каждом *подразделении* работает множество *сотрудников*.
Требуемые операции	1. Добавление нового подразделения 2. Удаление подразделения 3. Прием на работу сотрудника в заданное подразделение 4. Увольнение сотрудника 5. Редактирование личных данных сотрудника 6. Перевод сотрудника из одного подразделения в другое 7. Подсчет количества сотрудников в подразделении 8. Выдача списка сотрудников для заданного подразделения 9. Выдача списка подразделений

Вариант №4

Предметная область	Учебный отдел
Объекты	Группы, Студенты
Примечание	Имеется множество учебных *групп*. Каждая группа включает в себя множество *студентов*.
Требуемые операции	1. Добавление новой группы 2. Удаление группы 3. Зачисление студента в заданную группу 4. Отчисление студента 5. Перевод студента в заданную группу

	6. Редактирование личных данных студента 7. Выдача списка студентов для заданной группы 8. Выдача списка групп 9. Выдача полного списка студентов с указанием названия группы

Вариант №5

Предметная область	Автосалон
Объекты	Производители автомобилей, Марки
Примечание	*Марки* автомобилей сгруппированы по производителям. У каждого *производителя* имеется множество *марок*.
Требуемые операции	1. Добавление производителя 2. Удаление производителя 3. Добавление новой марки заданного производителя 4. Удаление марки 5. Редактирование марки 6. Подсчет количества марок у производителя 7. Выдача полного списка марок с названием производителя 8. Выдача полного списка производителей 9. Выдача списка марок заданного производителя

Вариант №6

Предметная область	Агентство новостей
Объекты	Категории новостей, Новости
Примечание	Новости сгруппированы по *категориям*. У каждой *категории* имеется множество *новостей*.
Требуемые операции	1. Добавление новой категории 2. Удаление категории 3. Добавление новости заданной категории 4. Удаление новости 5. Редактирование новости 6. Подсчет количества новостей в категории

	7. Выдача новости по идентификатору 8. Выдача полного списка новостей для заданной категории 9. Выдача полного списка категорий

Вариант №7

Предметная область	Продуктовый магазин
Объекты	Категория продукта, Продукт
Примечание	*Продукты* в магазине сгруппированы по *категориям*. Для каждой *категории* определено множество *продуктов*.
Требуемые операции	1. Добавление новой категории 2. Удаление категории 3. Добавление продукта заданной категории 4. Удаление продукта 5. Редактирование продукта 6. Подсчет количества продуктов в категории 7. Поиск продуктов по названию 8. Выдача списка продуктов заданной категории 9. Выдача списка категорий

Вариант №8

Предметная область	Футбол
Объекты	Команды, Игроки
Примечание	Имеется множество футбольных *команд*. Для каждой *команды* определено множество *игроков*.
Требуемые операции	1. Добавление новой команды 2. Удаление команды 3. Добавление нового игрока в команду 4. Удаление игрока 5. Редактирование сведений об игроке 6. Перевод игрока из одной команды в другую 7. Выдача полного списка игроков с указанием названия команды 8. Выдача полного списка команд 9. Выдача списка игроков заданной команды

Вариант №9

Предметная область	Музыкальный магазин
Объекты	Исполнители, Альбомы
Примечание	В музыкальном магазине *альбомы* сгруппированы по *исполнителям*. Для каждого *исполнителя* задано множество *альбомов*.
Требуемые операции	1. Добавление нового исполнителя 2. Удаление исполнителя 3. Добавление нового альбома заданного исполнителя 4. Удаление альбома 5. Редактирование данных об исполнителе 6. Подсчет количества альбомов исполнителя 7. Выдача полного списка альбомов с указанием исполнителя 8. Выдача списка альбомов заданного исполнителя 9. Выдача полного списка исполнителей

Вариант №10

Предметная область	Аэропорт
Объекты	Авиакомпании, Рейсы
Примечание	Имеется множество *авиакомпаний*. Для каждой *авиакомпании* определены ее *рейсы*.
Требуемые операции	1. Добавление новой компании 2. Удаление компании 3. Добавление нового рейса 4. Отмена (удаление) рейса 5. Редактирование рейса 6. Поиск рейса по номеру рейса (идентификатору) 7. Выдача полного списка рейсов с указанием названия авиакомпании 8. Выдача полного списка авиакомпаний 9. Выдача списка рейсов заданной авиакомпании

Вариант №11

Предметная область	Файловая система
Объекты	Папки, Файлы
Примечание	Имеется множество *папок* (независимых друг от друга). Для каждой *папки* определено множество *файлов*.
Требуемые операции	1. Добавление новой папки 2. Удаление папки 3. Добавление нового файла в заданную папку 4. Удаление файла 5. Редактирование файла 6. Перенос файла в заданную папку 7. Копирование файла в заданную папку 8. Выдача полного списка папок 9. Выдача списка файлов для заданной папки

Вариант №12

Предметная область	Расписание занятий
Объекты	Дни недели, Занятия
Примечание	Имеется множество *дней*. Для каждого *дня* определен перечень *занятий*.
Требуемые операции	1. Добавление нового дня 2. Удаление дня 3. Добавление нового занятия 4. Удаление занятия 5. Редактирование занятия 6. Запрос количества занятий в заданный день 7. Выдача полного списка занятий с указанием дня 8. Выдача списка занятий для заданного дня 9. Выдача списка дней

Вариант №13

Предметная область	Записная книжка
Объекты	Календарные дни, Мероприятия
Примечание	Имеется множество *дней*. Для каждого *дня* определен перечень *мероприятий*.
Требуемые операции	1. Добавление нового дня 2. Удаление дня 3. Добавление нового мероприятия 4. Удаление мероприятия 5. Перенос заданного мероприятия на заданный день 6. Запрос количества мероприятий в заданный день 7. Выдача полного списка мероприятий с указанием дня 8. Выдача полного списка дней 9. Выдача списка мероприятий для заданного дня

Вариант №14

Предметная область	Видеомагазин
Объекты	Жанры, Фильмы
Примечание	Имеется множество *жаров*. Для каждого *жанра* определен перечень *фильмов*.
Требуемые операции	1. Добавление нового жанра 2. Удаление жанра 3. Добавление нового фильма 4. Удаление фильма 5. Редактирование фильма 6. Запрос количества фильмов заданного жанра 7. Поиск фильма по названию 8. Выдача списка фильмов заданного жанра 9. Выдача полного списка жанров

Вариант №15

Предметная область	Железная дорога
Объекты	Дороги, Станции
Примечание	Имеется множество *железных дорог*. В ведомстве каждой дороги находится множество *станций*.
Требуемые операции	1. Добавление новой дороги 2. Удаление дороги 3. Добавление новой станции 4. Удаление станции 5. Редактирование станции 6. Запрос количества станции на заданной дороге 7. Поиск станции по названию 8. Выдача полного списка дорог 9. Выдача списка станций для заданной дороги

Вариант №16

Предметная область	Склад
Объекты	Секции, Товары
Примечание	*Товары* на складе сгруппированы по *секциям*. Для каждой *секции* задано множество *товаров*.
Требуемые операции	1. Добавление новой секции 2. Удаление секции 3. Добавление нового товара в заданную секцию 4. Удаление товара 5. Редактирование товара 6. Поиск товара по названию 7. Запрос количества товаров в заданной секции 8. Выдача полного списка секций 9. Выдача списка товаров для заданной секции

Вариант №17

Предметная область	Кафедра университета
Объекты	Преподаватели, Дисциплины
Примечание	На кафедре имеется множество *преподавателей*. Для каждого *преподавателя* задано множество *дисциплин*.
Требуемые операции	1. Прием на работу (добавление) нового преподавателя 2. Увольнение (удаление) преподавателя 3. Добавление новой дисциплины 4. Удаление дисциплины 5. Редактирование личных данных преподавателя 6. Запрос количества дисциплин у преподавателя 7. Поиск дисциплины по названию 8. Выдача полного списка преподавателей 9. Выдача списка дисциплин для заданного преподавателя

Вариант №18

Предметная область	Программное обеспечение
Объекты	Производители, Программные продукты
Примечание	Программные *продукты* сгруппированы по *производителям*. Для каждого *производителя* задано множество *продуктов*.
Требуемые операции	1. Добавление нового производителя 2. Удаление производителя 3. Добавление нового продукта 4. Удаление продукта 5. Редактирование продукта 6. Поиск продукта по названию 7. Запрос количества продуктов у производителя 8. Выдача полного списка производителей 9. Выдача списка продуктов заданного производителя

Вариант №19

Предметная область	Геометрия
Объекты	Многоугольники, Вершины
Примечание	Имеется множество *многоугольников*. Каждый *многоугольник* состоит из произвольного числа *вершин*.
Требуемые операции	1. Добавление нового многоугольника 2. Удаление многоугольника 3. Добавление новой вершины 4. Удаление вершины 5. Редактирование координат вершины 6. Запрос количества вершин у многоугольника 7. Выдача полного списка многоугольников 8. Выдача полного списка вершин заданного многоугольника 9. Сравнение двух заданных многоугольников на равенство

Вариант №20

Предметная область	Схема метро
Объекты	Линии, Станции
Примечание	Имеется множество *линий* метрополитена. Каждая *линия* состоит из последовательности *станций*.
Требуемые операции	1. Добавление новой линии 2. Удаление линии 3. Добавление новой станции 4. Удаление станции 5. Редактирование станции 6. Поиск станции по названию 7. Запрос количества станций на линии 8. Выдача списка станций для заданной линии. 9. Выдача полного списка линий

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. На какой тип взаимодействия (синхронный, асинхронный) ориентирована технология RPC? Обоснуйте свой ответ.
2. Где фактически выполняется удаленная процедура: в адресном пространстве клиента или сервера?
3. Какие функции выполняют клиентский и серверный стабы?
4. Какими преимуществами и недостатками обладает статическое связывание?
5. Какими преимуществами и недостатками обладает динамическое связывание?
6. Какие функции выполняет реестр RMI?
7. В каком порядке следует запускать программы (клиент, сервер, реестр RMI) в клиент-серверной системе, взаимодействующей через интерфейс RMI? Обоснуйте свой ответ.
8. Что такое интерфейс удаленного объекта и для чего он нужен?
9. Могут ли объекты классов Java являться параметрами удаленных методов при использовании RMI? В каком случае?
10. Могут ли два клиента одновременно вызвать один и тот же метод удаленного объекта при использовании RMI?
11. Могут ли два клиента одновременно вызвать различные методы удаленного объекта при использовании RMI?
12. В каком случае требуется синхронизировать методы удаленного объекта при использовании RMI?

7. ЛИТЕРАТУРА

1. Свистунов А.Н. Построение распределенных систем на Java. – М.: ИНТУИТ, 2010 *(ISBN 978-5-9963-0444-8)*
2. Вольфганг Эммерих. Конструирование распределенных объектов. Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft/COM и Java/RMI. – М.: Мир, 2002 *(ISBN 5-03-003405-6, 0-471-98657-7)*
3. Дэвид Хеффельфингер. Разработка приложений Java EE 6 в NetBeans 7. – М.: ДМК Пресс, 2013 *(ISBN 978-5-94074-914-1, 978-5-94074-914-1)*

Учебно-методическое издание

Варфоломеев Алексей Викторович

Клиент-серверное взаимодействие на базе технологии удаленного вызова процедур

Методические указания к лабораторным работам по дисциплине «Корпоративные информационные системы» для студентов специальностей 230102 и 230201

Подписано к печати 15.10

Формат - *60x84/16* Усл. печ. л. – Тираж – 100 экз.

Заказ № 191/14 Изд. № *124-13*