Лабораторная работа №4
Клиент-серверное взаимодействие
на основе технологии удаленного вызова процедур

Содержание

1. Лабораторная работа 4

1.1. Постановка задачи

В информационной системе, разработанной в рамках лабораторной работы №3, организовать взаимодействие клиента и сервера с использованием механизма удаленного вызова процедур/методов (RPC/RMI).

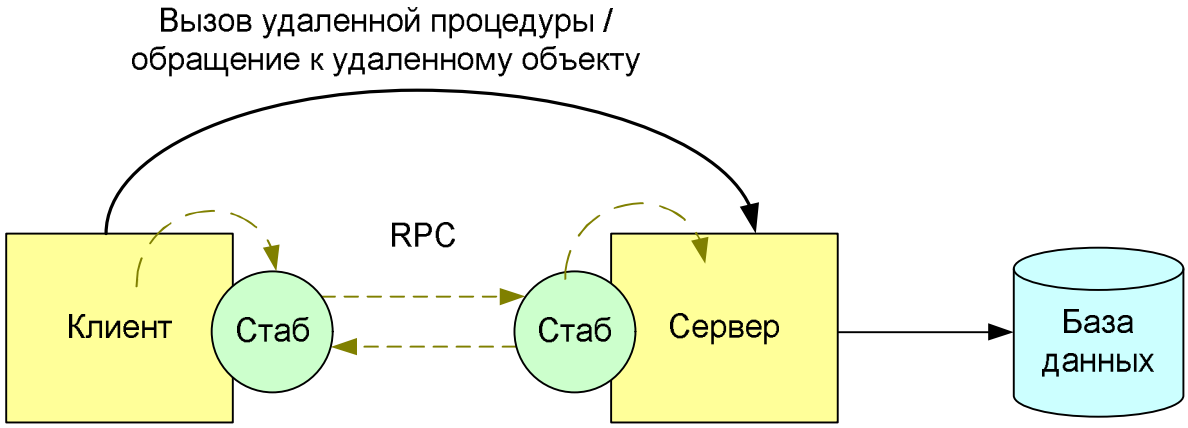

Рис. 1. Клиент-серверное взаимодействие на основе технологии RPC

1.2. 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 вашем варианте задания.

Клиентское приложение вызывает удаленные методы сервера и демонстрирует результаты пользователю. Клиентское приложение не требует создания пользовательского интерфейса. Тестирование работоспособности клиента осуществляется на основе сценариев, демонстрирующих возможности ИС.

Рекомендуемый язык программирования – Java. Рекомендуемый интерфейс взаимодействия – Java RMI.

1.3. Требования к оформлению отчета

Отчет должен содержать:

- титульный лист
- постановку задачи
- исходный код программ
- описание программы (*описание классов, методов, полей, включая ПОДРОБНОЕ описание интерфейса удаленного объекта*)

1.4. 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

2. Рекомендации по выполнению работы

2.1. Технология Java RMI

Технология RMI обеспечивает удаленный вызов методов объектов в распределенных системах и может использоваться для организации синхронного клиент-серверного взаимодействия приложений.

В рамках технологии RMI приложение-сервер создает *объект* и регистрирует его для удаленного доступа. Клиентские приложения получают ссылку на удаленный 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.

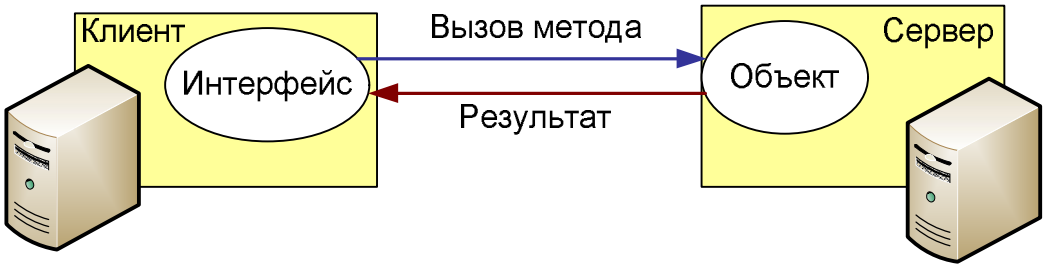

Рис. 2. Вызов удаленного метода

Технология RMI скрывает от разработчиков приложений реализацию процесса сетевого взаимодействия между клиентским и серверным приложениями. За формирование сетевых пакетов и их передачу полностью отвечает промежуточное ПО RMI.

Архитектура RMI представлена схематически на рис. 3. Клиент, вызывая через интерфейс метод удаленного объекта, на самом деле обращается к специализированному локальному объекту-заглушке – *стабу (stub)*. Стаб упаковывает параметры вызова метода в байтовое сообщение. При этом параметры-объекты подвергаются *сериализации*. Процесс упаковывания параметров называется *маршаллингом* (marshaling). Сформированное сообщение передается по сети на сервер. Его получает *скелетон (skeleton)* – серверная заглушка. Задача скелетона – извлечь параметры из полученного сообщения и вызвать необходимый метод серверного объекта (который, заметим, является локальным для скелетона). Результат вызова метода скелетон упаковывает в байтовое сообщение и передает обратно *стабу* клиента. Стаб клиента распаковывает сообщение и возвращает результат вызвавшей его программе.

За стабом и скелетоном скрывается два вспомогательных уровня: *уровень удаленных ссылок* и *транспортный уровень*.

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.

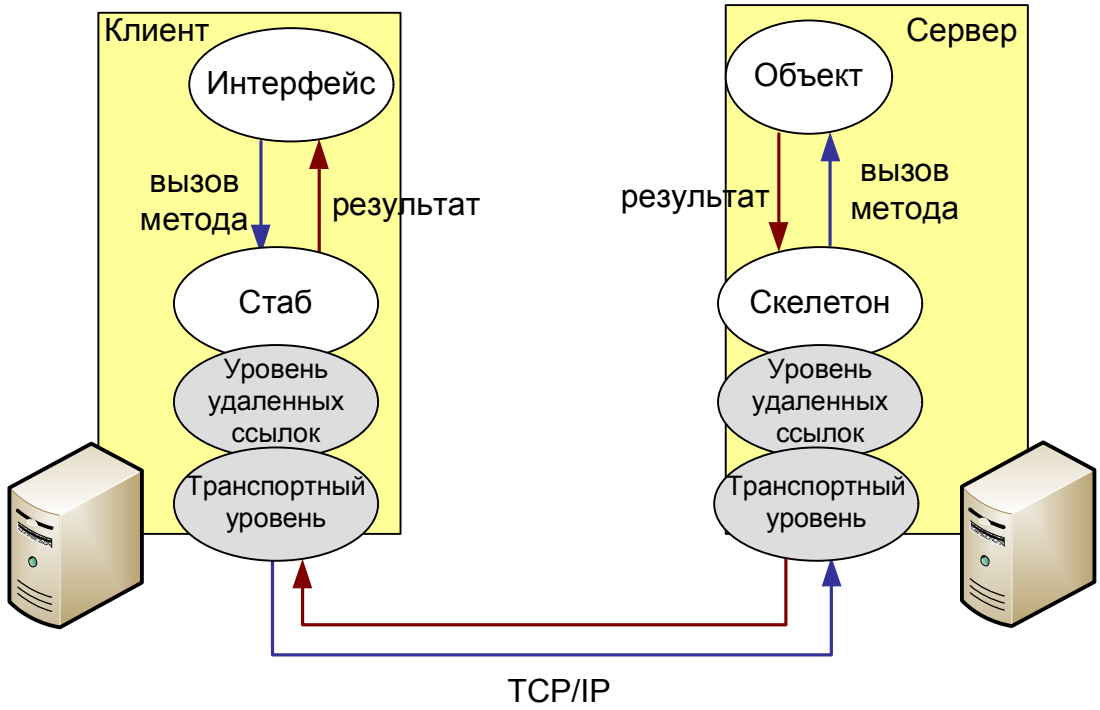

Рис. 3. Архитектура RMI

2.2. Связывание клиента и сервера

Каким образом клиент находит удаленный серверный объект? Связывание клиента и сервера осуществляется через специальную службу – *службу имен и каталогов*. Служба имен или каталогов обычно выполняется на известном всем клиентам хосте и имеет известный номер порта.

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

Реестр RMI работает на серверной машине, содержащей объекты удаленных служб. По умолчанию, реестр использует 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. Клиенты, желающие работать с удаленным объектом, обращаются к реестру RMI и получают ссылку на объект по его имени.

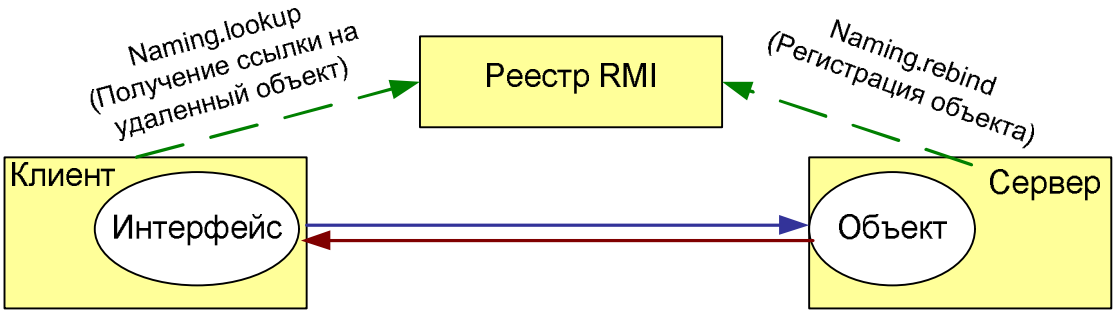

Рис. 4. Связывание клиента и сервера

2.3. Реализация взаимодействия на основе 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.

2.3.1. 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;
}
```

2.3.2. 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) представляет базовые функциональные возможности, которые необходимы удаленным объектам для обслуживания удаленных запросов.

Конструкторы и методы класса UnicastRemoteObject возбуждают контролируемое исключение RemoteException, поэтому наследники класса UnicastRemoteObject должны определять конструкторы, которые также возбуждают исключение RemoteException.

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!"`.

2.3.3. 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.

2.3.3.1 Использование внешнего реестра

При использовании внешнего реестра серверный код может 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 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).

Остановить службу реестра RMI можно нажав клавиши CTRL+C.

Рис. 5. Запуск службы реестра RMI

2.3.3.2 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 полного URL указывается только имя, под которым объект будет зарегистрирован.

!!! В данном случае, запуск приложения rmiregistry не требуется, так как служба RMI создается внутри серверного приложения.

2.3.4. 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.

2.3.5. Создание классов скелетона и стаба

В версии Java 1.1 классы стаба и скелетона создавались вручную при помощи специальной утилиты *rmic (*размещается *в* папке *bin* каталога *java)*.

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

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

2.3.6. Особенности использования RMI

При разработке ПО для лабораторной работы №3 следует обратить внимание на следующие особенности RMI.

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

Передача локальных объектов осуществляется «по значению». При передаче таких объектов RMI использует механизм сериализации. В связи с этим классы передаваемых локальных объектов должны поддерживать интерфейс java.io.Serializable.

2. При вызове клиентом метода удаленного объекта исполняющей частью системы на стороне сервера формируется (или выбирается из числа уже созданных) поток, в котором и происходит вызов метода. Таким образом, становится возможным одновременное выполнение методов сервера несколькими клиентами. Поэтому, возникает необходимость синхронизировать доступ к полям класса из удаленных методов.

2.4. Примеры

2.4.1. Пример 1: Калькулятор

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

Отдельно от клиентского и серверного приложений разрабатывался пакет com.rmi, содержащий описание интерфейса серверного объекта. Клиент и сервер использовали данный пакет как внешнюю библиотеку.

Пакет com.rmi: Интерфейс Calculator

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

public interface Calculator extends Remote {
    public int sum(int x, int y) throws RemoteException;
    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 {
    public CalculatorImpl() throws RemoteException {
        super();
    }
    public int sum(int x, int y) throws RemoteException {
        return x+y;
    }
    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 {
        CalculatorImpl helloImpl = new CalculatorImpl();
        Registry registry = LocateRegistry.createRegistry(123);
        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";
        Calculator Q = (Calculator) Naming.lookup(url);
        System.out.println("RMI object found");
        int x = Q.sum(10, 20);
        int y = Q.sub(10, 4);
        System.out.println(x);
        System.out.println(y);
    }
}
```

2.4.2. Пример 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.

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);
    }
}
```

2.4.3. Пример 3: Удаленный массив

На сервер хранится массив целых чисел.

Сервер предоставляет клиентам удаленные методы, позволяющие добавлять числа в массив и вычислять сумму элементов массива.

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

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.

Пакет com.rmi: Интерфейс MyArray

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

public interface MyArray extends Remote {
    public void Add(Integer v) throws RemoteException;
    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 {
    private ArrayList<Integer> x;

    public MyArrayImpl() throws RemoteException
    {
        x = new ArrayList<Integer>();
    }

    public void Add(Integer v) throws RemoteException
    {
        synchronized (x)
        {
            x.add(v);
        }
    }

    public int Sum() throws RemoteException
    {
        int sum=0;
        synchronized (x)
        {
            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");
        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");
            while (true)
                Q.Add(Integer.valueOf(1));
        }
    }
```