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

Институт информационных технологий

Кафедра Корпоративных Информационных Систем

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6**

**по дисциплине**

«Разработка клиент-серверных приложений»

**Тема: «Удаленный вызов процедур (RPC)»**

|  |  |  |
| --- | --- | --- |
| Выполнил студент группы ИКБО-08-18 ИНБО-01-17 |  | Валяев Д.А. |
| Принял старший преподаватель |  | Мирзоян Д.И. |
| Лабораторная работа выполнена | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2020г. | *(подпись студента)* |
| «Зачтено» | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2020г. | *(подпись руководителя)* |

Москва 2020

# Задание

Создать серверное приложение для веб-сервера Apache, осуществляющее исполнение удалённых вызовов процедур.

Создать клиентское веб-приложение, осуществляющее удалённый вызов процедур в соответствии с составленной спецификацией и вывод результатов.

# Теоретическая часть

Удалённый вызов процедур, реже Вызов удалённых процедур (RPC) — класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве (на удалённых компьютерах, либо в независимой сторонней системе на том же устройстве). Обычно реализация RPC-технологии включает в себя два компонента: сетевой протокол для обмена в режиме клиент-сервер и язык сериализации объектов (или структур, для необъектных RPC). Различные реализации RPC имеют очень отличающуюся друг от друга архитектуру и разнятся в своих возможностях: одни реализуют архитектуру SOA, другие — CORBA или DCOM. На транспортном уровне RPC используют в основном протоколы TCP и UDP, однако, некоторые построены на основе HTTP (что нарушает архитектуру ISO/OSI, так как HTTP — изначально не транспортный протокол). Apache thrift - это язык описания интерфейсов, который используется для определения и создания служб под разные языки программирования. Также является фреймворком к удалённому вызову процедур.

# Практическая часть

Воспользуемся фреймворком Apache Thrift для создания класса, позволяющего воспользоваться механизмом RPC.

Необходимо определить методы сервиса, а также необходимые структуры данных в файле с расширением .thrift.

В настоящей работе сервис предоставляет данные о клиентах, которые хранятся на сервере. Определена структура данных пользователя и методы для получения данных по разным признакам.

namespace java laba

typedef i32 int

typedef string String

struct User {

1:int id,

2:String firstName,

3:String lastName,

4:String email

}

service UserService

{

User getUser(1:int id),

String getUserEmail(1:int id),

list<User> getUsersByFirstName(1:String firstName),

list<User> getAllUsers()

}

Сгенерируем класс сервиса с помощью команды:

thrift --gen java multi.thrift

В создавшейся папке определены два класса: User и UserService.

Создадим проект и подключим необходимые библиотеки для работы с сервисом.

В классе запуска сервера необходимо определить класс, реализующий интерфейс Iface, который содержит реализацию заданных процедур.

Класс клиента подключается к серверу и вызывает процедуры.

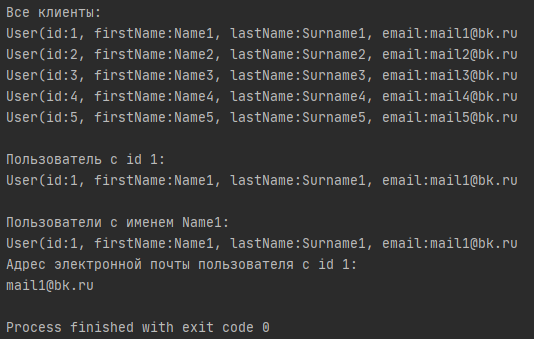


рис 1 - Результат работы программы

**Вывод**

В данной лабораторной работе была рассмотрена технология RPC. Реализованы серверное и клиентское приложения на языке Java с использованием фреймворка Apache Thrift, реализующие технологию RPC.

# Исходный код

*/\* Server.java \*/*  
import org.apache.thrift.server.TServer;  
import org.apache.thrift.server.TSimpleServer;  
import org.apache.thrift.transport.TServerSocket;  
import org.apache.thrift.transport.TServerTransport;  
  
public class Server {  
  
 private static UserService.Processor *processor*;  
  
 private static CommandHandler *handler*;  
  
 public static void main(String[] args) {  
 *handler* = new CommandHandler();  
 *processor* = new UserService.Processor(*handler*);  
  
 try {  
 TServerTransport serverTransport = new TServerSocket(9000);  
 TServer server = new TSimpleServer(new TServer.Args(serverTransport).processor(*processor*));  
  
 System.*out*.println("Start server");  
 server.serve();  
 } catch (Exception ignored) {}  
 }  
}

*/\* Client.java \*/*  
import org.apache.thrift.protocol.TBinaryProtocol;  
import org.apache.thrift.protocol.TProtocol;  
import org.apache.thrift.transport.TSocket;  
import org.apache.thrift.transport.TTransport;  
  
import java.util.List;  
  
public class Client {  
  
 public static void main(String[] args) throws Exception {  
 TTransport transport = new TSocket("localhost", 9000);  
 transport.open();  
  
 TProtocol protocol = new TBinaryProtocol(transport);  
 UserService.Client client = new UserService.Client(protocol);  
  
 System.*out*.println("Все клиенты:");  
 List<User> users = client.getAllUsers();  
 for(User user : users) {  
 System.*out*.println(user.toString());  
 }  
  
 System.*out*.println("\n\nПользователь c id 1:");  
  
 User usr = client.getUser(1);  
 System.*out*.println(usr.toString());  
  
 System.*out*.println("\n\nПользователи с именем Name1:");  
  
 users = client.getUsersByFirstName("Name1");  
 for(User user : users) {  
 System.*out*.println(user.toString());  
 }  
  
 System.*out*.println("\n\nАдрес электронной почты пользователя с id 1:");  
  
 String email = client.getUserEmail(1);  
 System.*out*.println(email);  
  
 transport.close();  
 }  
}

*/\* CommandHandler.java \*/*  
import org.apache.thrift.TException;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class CommandHandler implements UserService.Iface {  
  
 private static List<User> *users* = null;  
  
 public CommandHandler() {  
 *users* = new ArrayList<User>();  
 *users*.add(new User(1, "Name1", "Surname1", "mail1@bk.ru"));  
 *users*.add(new User(2, "Name2", "Surname2", "mail2@bk.ru"));  
 *users*.add(new User(3, "Name3", "Surname3", "mail3@bk.ru"));  
 *users*.add(new User(4, "Name4", "Surname4", "mail4@bk.ru"));  
 *users*.add(new User(5, "Name5", "Surname5", "mail5@bk.ru"));  
 }  
  
 @Override  
 public User getUser(int id) throws TException {  
 for(User user : *users*) {  
 if(user.getId() == id)  
 return user;  
 }  
 return new User();  
 }  
  
 @Override  
 public String getUserEmail(int id) throws TException {  
 for(User user : *users*) {  
 if(user.getId() == id)  
 return user.getEmail();  
 }  
 return "";  
 }  
  
 @Override  
 public List<User> getUsersByFirstName(String firstName) throws TException {  
 List<User> result = new ArrayList<User>();  
 for(User user : *users*) {  
 if(user.getFirstName().equals(firstName))  
 result.add(user);  
 }  
 return result;  
 }  
  
 @Override  
 public List<User> getAllUsers() throws TException {  
 return *users*;  
 }  
}