Лабораторная работа №1

Web-сервисы. Разработка web-сервиса, реализующего RPC-ориентированное взаимодействие

Цель: научиться работать с технологией web-сервисов, используя RPC-ориентированный подход.

Согласно Gartner Group, web-сервисы – это программные средства, которые динамически взаимодействуют друг с другом посредством стандартных технологий Internet, делая возможным возведение мостов между IT-системами, которое в противном случае потребовало бы громадных усилий.

Можно дать следующее определение:

Web-сервисы – это часть бизнес-логики, размещенная в Internet, доступ к котором обеспечивается по стандартным Интернет протоколам, таким как HTTP или SMTP.

Web-сервисы могут использоваться во многих приложениях. Независимо от того, откуда запускаются Web-сервисы, с настольных компьютеров клиентов или с переносных, они могут использоваться для обращения к таким интернет-приложениям, как система предварительных заказов или контроля выполнения заказов. Web-сервисы пригодны для B2B-интеграции (business-to-business), замыкая приложения, выполняемые различными организациями, в один производственный процесс. Web-сервисы также могут решать более широкую проблему интеграции приложений предприятия (Enterprise Application Integration, EAI), осуществляя связь нескольких приложений одного предприятия с несколькими другими приложениями, размещенными как "до", так и "после" брандмауэра. Во всех перечисленных случаях технологии web-сервисов являются "связующим звеном", объединяющим различные части программного обеспечения.

Интерфейсы web-сервисов получают из сетевой среды стандартные XML-сообщения, преобразуют XML-данные в формат, "понимаемый" конкретной прикладной программной системой, и отправляют ответное сообщение (последнее – не обязательно). Программная реализация web-сервисов (базовое программное обеспечение, нижний уровень) может быть создана на любом языке программирования с использованием любой операционной системы и любого связующего программного обеспечения (middleware).

Главные технологии web-сервисов:

–SOAP (Simple Object Access Protocol) – стандарт «упаковки» XML-документов для транспортировки по протоколам SMTP, HTTP или FTP.

–WSDL (Web Service Description Language) – это XML-технология, описывающая интерфейсы web-сервисов. Помогает клиентам автоматически понять, как работать с сервисом.

–UDDI (Universal Description, Discovery and Integration) – обеспечивает всемирную регистрацию web-сервисов. Используется для обнаружения сервиса путем поиска по имени, категории и др.

Web-сервисы реализуют сервисно-ориентированную архитектуру (*SOA – Service-Oriented Architecture*). SOA – это термин, который появился для описания исполняемых компонентов (таких как Web-сервисы) которые могут вызываться другими программами, выступающими в качестве клиентов или потребителей этих сервисов. Эти сервисы могут быть полностью современными - или даже устаревшими - прикладными программами, которые можно активизировать как черный ящик. От разработчика не требуется знать, как работает программа, необходимо лишь понимать, какие входные и выходные данных нужны, и как вызываются эти программы для исполнения.

В самом общем виде SOA предполагает наличие трех основных участников: поставщика сервиса, потребителя сервиса и реестра сервисов (см. рис. 1). Взаимодействие участников выглядит так:

- Клиент запрашивает у регистра UDDI сервис по его имени или идентификатору
- Клиент получает информацию о размещении WSDL-документа от UDDI-регистра. Он содержит информацию о том, как связаться с сервисом и формат запроса в XML
- Согласно найденной в WSDL информации клиент создает SOAP-сообщение и посылает на хост сервиса

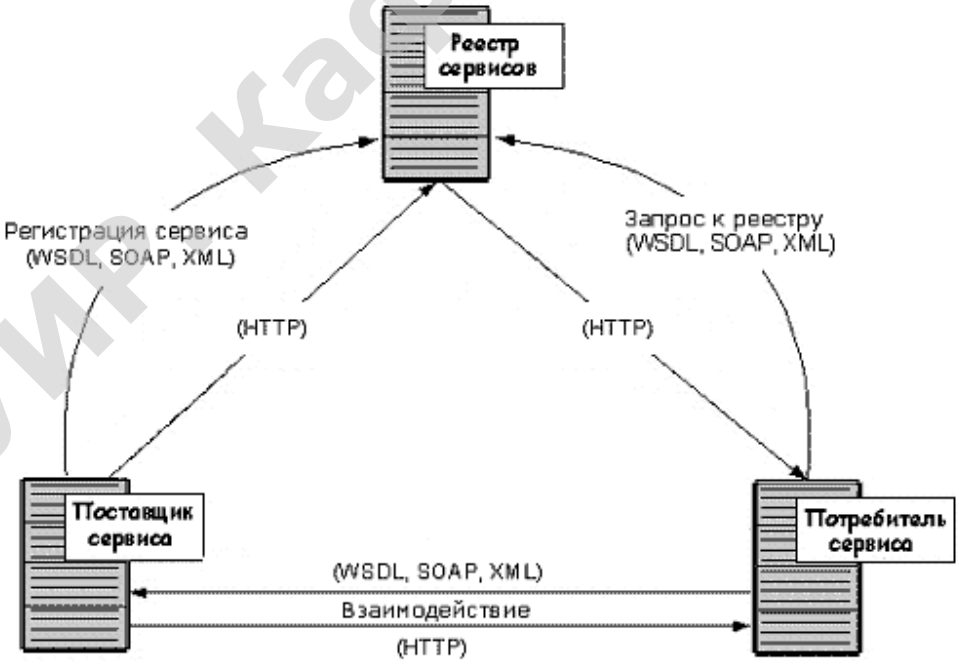

Рис.1. Общая схема SOA

Создание web-служб возможно либо с помощью модели RPC (Remote Procedure Call), либо используя документоориентированную (document-style) модель. В нашей лабораторной работе реализуем RPC, второй модели посвящена вторая лабораторная работа.

В случае с RPC клиент делает вызов удаленного метода web-сервиса, который обрабатывается на сервере. На макроуровне клиент передает SOAP-запрос и получает SOAP-ответ (см. рис.2)

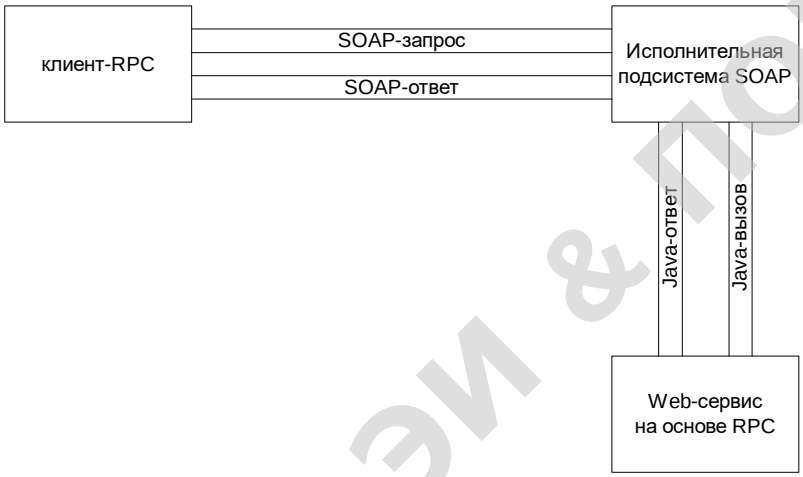

Рис.2. RPC-SOAP-программирование

В качестве «исполнительной системы SOAP» объединение Apache Software Foundation выпустило проект Axis.

Apache Axis (Apache eXtensible Interaction System) – система для конструирования SOAP процессоров, таких как клиенты, сервера, шлюзы и др. SOAP – это механизм для коммуникации приложений посредством Интернет. Однако Axis не просто «движок» SOAP, он также включает:

- простой самостоятельный сервер
- сервер, встраиваемый в контейнеры сервлетов
- расширенную поддержку WSDL
- инструменты, генерирующие Java-классы из WSDL
- примеры программ
- инструмент для отслеживания TCP/IP-пакетов

Axis конвертирует Java-объекты в данные SOAP, когда посылает их по сети, или получая результаты. Все ошибки, генерируемые сервером, Axis преобразовывает в Java-исключения.

Axis реализует JAX-RPC API, один из стандартных способов программирования Java-сервисов.

Axis реализован в *axis.jar*, реализация JAX-RPC API в *jaxrpc.jar* и *saaj.jar*. Также необходимы различные вспомогательные библиотеки для логгирования, обработки WSDL. Для работы Axis'а требуется наличие XML парсера, совместимого с JAXP 1.1 (Java API for XML Processing), предпочтительно Xerces.

Методические указания

В качестве примера рассмотрим задачу:

разработать телефонный справочник, представляющий собой web –сервис и позволяющий проводить поиск по имени или номеру телефона.

Для выполнения задачи воспользуемся Apache Axis 1.4 (хотя подойдет и более ранняя версия) и Tomcat 5.0. Распаковав дистрибутив, скопируем папку axis (axis 1_4\webapps\axis) в папку с приложениями на контейнере Tomcat (Tomcat 5.0\webapps\сюда). Проследите, чтобы в папке Tomcat 5.0\common\lib\ лежали библиотеки *activation.jar*, *mail.jar*, *xmlsec-1.3.0.jar*, *jsse.jar* – это необязательные, опциональные компоненты. Они не пригодятся при выполнении этой работы, однако могут быть необходимыми при выполнении дополнительных заданий от преподавателя, последующих лабораторных работ, курсового проекта.

Теперь запустим Tomcat (Tomcat 5.0\bin\startup.bat). Axis будет работать как сервлет, размещенный в Tomcat. Чтобы проверить, правильно ли настроен Axis, в строке браузера наберем http://localhost:8080/axis (укажите ваш номер порта, по которому запущен Tomcat), после чего должна появиться домашняя страниц Axis (см. рис.3).

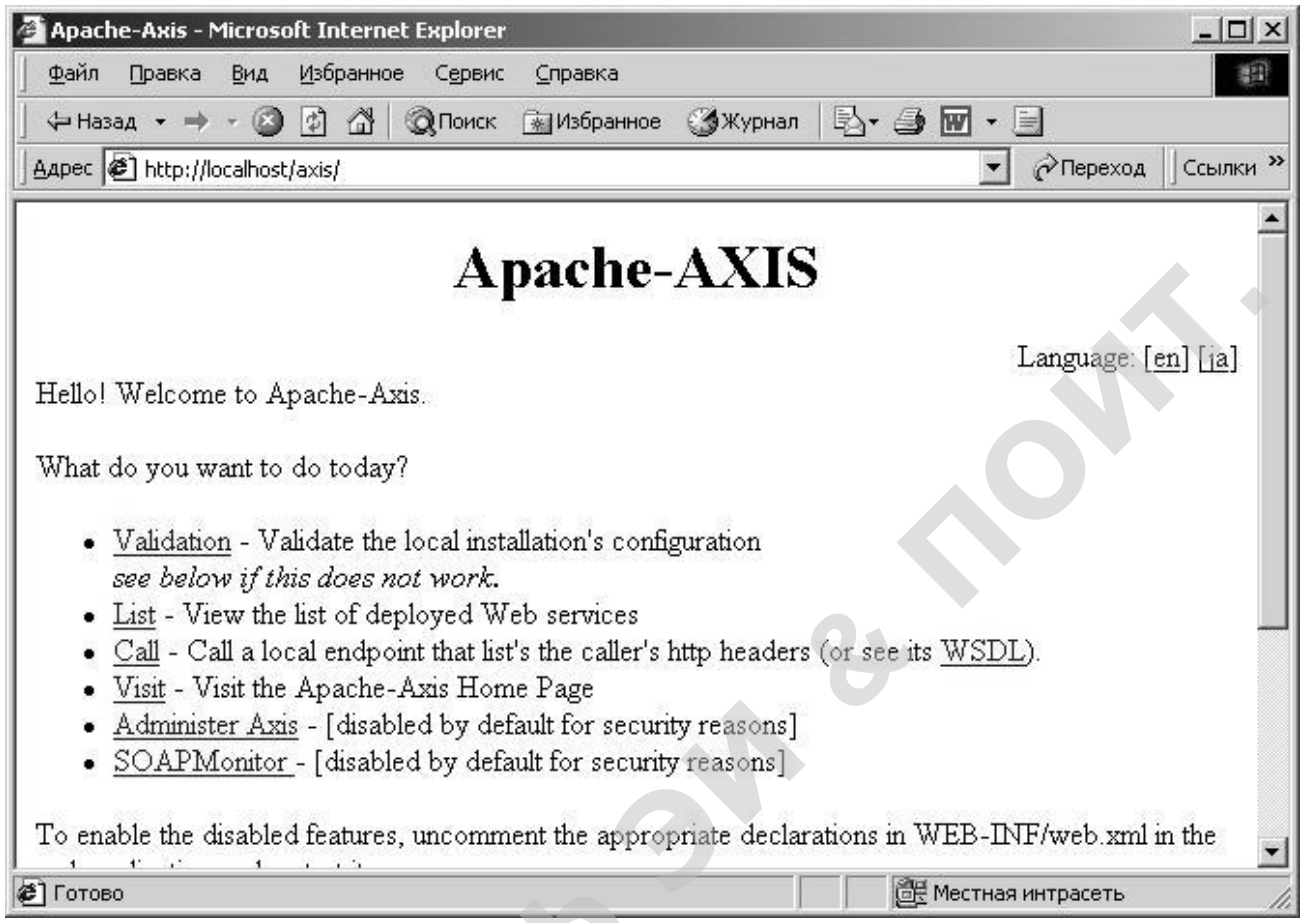

Рис.3. Домашняя страница Axis

По ссылке *Validation* можно проверить, все ли библиотеки нашел Axis. Идеальным является наличие сообщение «The core axis libraries are present. The optional components are present»**.** *List* позволяет просмотреть уже развернутые сервисы.

Создадим класс сервиса – **phone.java**. В нем будут находится две функции – *String phone_name(String phone),* которая по номеру телефона возвращает фамилию и *public String name_phone(String name)*, которая по фамилии возвращает номер телефона.

```
import java.io.*;
import java.util.*;

public class Phone{
```

```
public String phone_name(String phone){

        String name= new String();;
        String line;
```

Создадим файл, в котором в каждой строчке записана пара фамилия-телефон через пробел. В функции FileReader замените путь на свой.

```
        try{
                FileReader fr = new
                        FileReader("D:/work/phone/phone/phones.txt");
                BufferedReader in = new BufferedReader(fr);
```

Предусмотрим ситуацию, при которой соответствующего номера может не оказаться в файле. Для анализа прочитанной с файла строки удобно использовать функцию split, которая расщепляет строку на слова. Аргумент – это разделительный символ. Возвращает функция массив искомых слов.

```
                while ((line=in.readLine())!=null){
                        String[] str = line.split(" ");
                        if (str[0].equals(phone)){ name=str[1]; break;}else
                        name="NO SUCH TELEPHONE";
                }
        }catch(IOException e){
                e.printStackTrace();
        }

        return name;
}
```

Вторая функция (выполняется полностью аналогично).

```
public String name_phone(String name){

        String phone= new String();;
        String line;
        try{
                FileReader fr = new
                        FileReader("D:/work/phone/phone/phones.txt");
                BufferedReader in = new BufferedReader(fr);

                while ((line=in.readLine())!=null){
                        String[] str = line.split(" ");
                        if (str[1].equals(name)){ phone=str[0]; break;}else
                        phone="NO SUCH NAME";
                }
```

```
            }catch(IOException e){
                  e.printStackTrace();
            }
            return phone;
      }
}
```

Данный файл переименуем в *phone.jws* и поместим в папку axis на Tomcat. Теперь Tomcat необходимо перезапустить. Класс должен откомпилироваться и файл с расширением *.class поместится в jwsClasses – папку, которая тоже создастся в папке axis/WEB-INF.

Клиентское приложение

Подключим библиотеки – некоторые из них находятся в *axis.jar*.

```
import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import javax.xml.rpc.ServiceException;
import java.net.URL;
import java.net.MalformedURLException;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
```

Функция main выбрасывает исключения, вызванные вызовом несуществующего сервиса и неправильно сформированным URL.

```
class phoneApp {
   public static void main(String[] args) throws ServiceException,
                                                  MalformedURLException {
```

строка endpoint является строкой URL, по которому размещен сервис. обратите внимание на название хоста, номер порта и название самого сервиса, которые соответствуют нашей реализации лабораторной работы.

```
String endpoint = "http://localhost:8080/axis/Phone.jws";
```

Следующие строки создают объекты сервиса, вызова именно нашего сервиса и устанавливается целевой адрес сервиса через класс URL:

```
Service service = new Service();
```

```
Call call = (Call) service.createCall();
call.setTargetEndpointAddress(new URL(endpoint));
```

на экран пользователя выведется меню. Затем запускаем цикл, который останавливается, как только пользователь нажмет на «3», что в нашем меню означает выход.

```
System.out.println("1 - enter the phone number");
System.out.println("2 - enter the name");
System.out.println("3 - exit");

 try {
        BufferedReader in = new BufferedReader(new
                                                InputStreamReader(System.in));
        String line = null;
        line = in.readLine();
        while(!line.equals("3")){
              if (line.equals("3")) break;
```

при нажатии на «1» прочитаем строку с телефоном с консоли,

```
        if (line.equals("1")){
              String phone = in.readLine();
```

Сформируем массив объектов, состоящий из одного объекта – введенной строки.

```
        Object[] param1 = new Object[]{phone};
```

Вызов сервиса осуществляется с помощью функции *invoke()* с параметрами название функции и массивом передаваемых значений.

```
        String response = (String)call.invoke("phone_name", param1);
```

Выведем возвращенную строку на экран:

```
        System.out.println("PHONE="+phone+"\n"+"NAME="+response);
```

Аналогично обработаем нажатие на кнопку «2».

```
        if (line.equals("2")){
              String name = in.readLine();
              Object[] param2 = new Object[]{name};
              String response = (String)call.invoke("name_phone",  param2);
              System.out.println("NAME="+name+
                                                "\n"+"PHONE="+response);
        };
```

закончим клиентское приложение, повторив вывод меню и ввод значения, которое будет обрабатываться уже на следующей итерации цикла.

```
                System.out.println("1 - enter the phone number");
                System.out.println("2 - enter the name");
                System.out.println("3 - exit");
                line = in.readLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Для запуска всей системы в целом, должен быть запущен Tomcat с axis'ом внутри, затем – клиентский класс. Чтобы компилятор мог использовать подключенные библиотеки, их jar-файлы должны быть прописаны в CLASSPATH.

Контрольные вопросы

1. Что такое web-сервисы?
2. Перечислите и объясните сущность технологий, связанных с web-сервисами.
3. В чем сущность сервис-ориентированной архитектуры?
4. Какова роль каждого из участников SOA?
5. В чем сущность RPC-ориентированного подхода?
6. Что такое Apache Axis?
7. Чем характеризуется и какова роль Axis в разработке web-сервисов?
8. Какие библиотеки необходимо подключить для работоспособности Axis?
9. Как создать простейший web-сервис, используя файл JWS?
10. Как создать объект сервиса и организовать подключение к нему?