14

Введение в Web-сервисы и SOAP

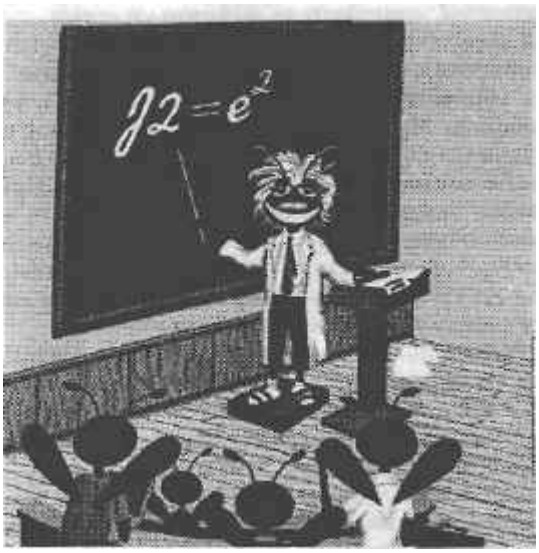
Цели

- Понять, что собой представляет протокол Simple Object Access Protocol (SOAP) и каким образом он использует XML.
- Уяснить структуру сообщений SOAP.
- Научиться создавать приложения Java, которые отправляют и принимают сообщения SOAP.

Ничего не происходит до тех пор, пока что-нибудь не будет продано.
Артур Г. Мотли

Людям пора учиться управлять в мире, состоящем по большей части из множества вождей и одного индейца. Индеец — это, конечно, компьютер.
Томас А. Уислер

...чудеса всегда объясняются просто.
Амелия Барр



Сходство есть повторение внешних свойств объектов...
Чинг Хао

...если вы понимаете, что вам хочет сообщить природа, радуйтесь, ибо душа ваша жива...
Элеоноря Дыос

14.1. Введение

Функциональная совместимость, или беспрепятственное взаимодействие и обмен информацией между различными программными системами, является главной целью предприятий и организаций, которые активно используют в своей деятельности компьютеры и компьютерные сети. Многие приложения используют для передачи данных Internet. Некоторые из этих приложений выполняются на клиентских системах, имеющих небольшую вычислительную мощность, поэтому для обработки данных они обращаются к методам, размещенным на удаленных машинах. Многие приложения используют собственные форматы данных, что затрудняет, или даже делает невозможным взаимодействие с другими приложениями. Большинство приложений также расположено за *брандмауэрами* (или межсетевыми экранами): защитными барьерами, которые ограничивают коммуникационный обмен между сетями. Протокол простого доступа к объектам *Simple Object Access Protocol (SOAP)* решает эти проблемы. Объединяя сильные стороны, присущие HTTP и XML, он обеспечивает полностью расширяемый режим взаимодействия между программными системами.

Web-сервисы — новое слово в технологии распределенных систем. Спецификация *Open Net Environment (ONE)* корпорации Sun Microsystems и инициатива *.NET* корпорации Microsoft обеспечивают инфраструктуры для написания и развертывания Web-сервисов. Имеется несколько определений Web-сервиса. Web-сервисом может быть любое приложение, имеющее доступ к Web, например, Web-страница с динамическим содержимым. В более узком смысле Web-сервис — это приложение, которое предоставляет открытый интерфейс, пригодный для использования другими приложениями в Web. Спецификация ONE Sun требует, чтобы Web-сервисы были доступны через HTTP и другие Web-протоколы, чтобы дать возможность обмениваться информацией посредством XML-сообщений и быть найденными через сервисы поиска. SOAP предоставляет средства взаимодействия на базе XML для многих Web-сервисов. Web-сервисы могут обеспечить высокую степень совместимости между различными системами [1].

Гипотетический Web-сервис, разработанный в соответствии с архитектурой ONE Sun, может принимать форму, в которой реестр сервисов публикует описание Web-сервиса в виде документа *Universal Description. Discovery and Integration (UDDI).* Клиент, например, Web-браузер или клиент GUI Java, ищет службу каталогов для требуемого Web-сервиса. Клиент использует информацию, полученную от сервиса поиска, для отправки XML-сообщения через HTTP Web-серверу, на котором размещен Web-сервис. Сервлет обрабатывает клиентский запрос. После этого сервлет осуществляет доступ к серверу приложений, который предоставляет средства Enterprise JavaBeans. Компоненты EJB, в свою очередь, обращаются к базе данных, которая хранит информацию для Web-сервиса. После обращения к базе данных EJB-компонент отправляет сервлету запрошенную информацию.

Сервлет форматирует информацию для представления ее клиенту (например, создает страницу JavaServer Page). Сервер HTTP отправляет ответ в виде XML обратно клиенту. Клиент осуществляет синтаксический анализ ответа и отображает информацию пользователю [1].

• Огромный потенциал Web-сервисов определяется отнюдь не технологией, используемой для их создания. HTTP, XML и другие протоколы, используемые Web-сервисами, не новы. Функциональная совместимость и масштабируемость Web-сервисов подразумевает, что разработчики могут быстро создавать большие приложения и более крупные Web-сервисы из меньших Web-сервисов. Спецификация Sun Open Net Environment описывает архитектуру для создания *интеллектуальных Web-сервисов.* Интеллектуальные Web-сервисы используют общее операционное окружение. Совместно используя контекст, интеллектуальные Web-сервисы могут выполнять стандартную аутентификацию для финансовых транзакций, предоставлять рекомендации и указания в зависимости от географического местоположения компаний, участвующих в электронном бизнесе. На момент написания этой книги на пути разработки интеллектуальных Web-сервисов имелось два основных препятствия. Во-первых, пока не существует общепринятых стандартов для совместного использования контекста. Web-сервисами. Во-вторых, пока не обеспечивается безопасность и конфиденциальность транзакций, осуществляемых Web-сервисами.

14.2. Простой протокол доступа к объектам (SOAP)

Корпорации IBM, Lotus Development Corporation, Microsoft, Develop-Mentor и Userland Software разработали протокол SOAP, который представляет собой протокол, основанный на HTTP-XML. Он позволяет приложениям взаимодействовать между собой через Internet, используя для этого XML-документы, называемые *сообщениями SOAP.* Протокол SOAP совместим с любой объектной моделью, поскольку он включает только те функции и методы, которые абсолютно необходимы для формирования коммуникационной инфраструктуры. Таким образом, SOAP является независимым от платформы и конкретных приложений, а для его реализации может быть использован любой язык программирования. SOAP поддерживает практически любой транспортный протокол. Например, SOAP привязан к протоколу HTTP и следует модели запрос-ответ HTTP. SOAP также поддерживает любые методы кодирования данных, которые позволяют приложениям, основанным на SOAP, посылать в сообщениях SOAP информацию практически любого типа (например, изображения, объекты, документы и т.д.).

Сообщение SOAP содержит *конверт,* который описывает содержимое, предполагаемого получателя сообщения и требования к обработке сообщения. Необязательный *элемент* ***header*** *(заголовок)* сообщения SOAP содержит инструкции по обработке для приложений, которые принимают сообщение. Например, для реализаций, которые поддерживают транзакции, заголовок может задавать параметры для данной транзакции. Заголовок также может содержать информацию о маршрутизации. С помощью заголовка **header** поверх SOAP могут надстраиваться более сложные протоколы. Записи в заголовке могут модульно расширять сообщение для таких задач, как аутентификация, управление транзакциями и проведение платежей. Тело SOAP-сообщения содержит специфичные для приложения данные, предназначенные для предполагаемого получателя сообщения.

SOAP можно использовать для осуществления *удаленного вызова процедур Remote Procedure Call (RPC),* который представляет собой запрос, посылаемый другому компьютеру для выполнения определенной задачи. RPC использует словарь XML для задания вызываемого метода, передаваемых ему параметров и универсаль-

ного идентификатора ресурса (URI) целевого объекта. Вызов RPC привязывается к HTTP-запросу, так что сообщение отправляется через HTTP-запрос **post**. Сообщение-ответ SOAP представляет собой документ HTTP-ответа, который содержит результаты вызова метода (например, возвращаемые значения, сообщения об ошибках и т.д.). SOAP также поддерживает *асинхронные RPC-вызовы,* при которых программа, инициировавшая RPC-вызов, не ждет ответа от удаленной процедуры.

На момент написания этой книги SOAP еще находится на стадии развития, и разработка многих технологий, основанных на SOAP, только начинается. Чтобы реализовать преимущества, предоставляемые SOAP, необходимо установить высокоуровневые спецификации и стандарты, которые используют эту технологию. Несмотря на это, *SOAP* является перспективным стандартом для XML-распределенных вычислений, предоставляя невиданный ранее уровень расширяемости и функциональной совместимости.

На рис. 14.1—14.4 представлен пример сервиса SOAP, использующий реализацию API SOAP Apache, версия 2.2 (доступна по адресу **xml.apache.org/soap**)[1]. Для RPC SOAP требуется средство работы с сервлетами, например, Tomcat (**jakarta.apache.org**) и синтаксический анализатор Apache Xerces для Java (доступен по адресу **xml.apache.org/xerces-j/index.html**). В документации SOAP (**docs/install/index.html**) содержатся инструкции по установке как для сервера, так и для клиента.

На рис. 14.1 представлен класс SimpleService, который размещается на сервере и содержит метод **getWelcome**. Приложение Java, представленное на рис. 14.4, вызывает этот метод с использованием RPC.

```
1 // Рис. 14.1. SimpleService.Java
2 // Реализация запрашиваемого метода на сервере
3
4 public class SimpleService {
5
6    public String getWelcome( String message ) throws Exception
7    {
8       String text =
9          "Welcome to SOAP!\nHere is your message: " + message;
10
11      return text;   // ответ
12
13
```

Рис. 14.1. Класс **SimpleService**

Метод getWelcome (строки 6-12) возвращает строковые данные. Чтобы сделать этот метод доступным для клиентов (т.е. сделать возможным работу с ним через RPC), необходимо предоставить серверу имя метода, который обрабатывает запрос, т.е. необходимо осуществить *развертывание сервиса.*

Чтобы выполнить развертывание сервиса, сначала скопируйте файл **SimpleService.class** в каталог **jakarta-tomcat/classes**. Если вы создали файл архива Java (JAR), скопируйте JAR-файл в каталог **jakarta-tomcat/lib**. Создайте каталог classes или **lib**, если они не существуют. В Jakarta-Tomcat указание на файлы в этих каталогах включается в описание переменной окружения **CLASSPATH**.

Выполните развертывание сервиса с помощью инструментального средства развертывания XML-SOAP, входящего в состав пакета SOAP (в каталоге **webapps/soap**). Чтобы выполнить приложение, введите URL **localhost:8080/soap/admin**

[1] На момент подготовки к изданию перевода книги актуальной была версия 2.3.1 API Apache SOAP. — *Прим. ред.*

в адресной строке Web-браузере. На рис. 14.2 и 14.3 представлено средство администрирования, которое позволяет развертывать, удалять и получать список сервисов. *Поле* ***ID*** на рис. 14.2 содержит URI **(urn:xml-simple-message),** который идентифицирует сервис клиенту. Этот URI определяется программистом. Если один сервис имеет такой же URI, что и другой сервис, клиент не сможет их различить; как следствие, возможны ошибки. *Поле* ***Scope*** задает границы существования объекта, создаваемого (на сервере) для обработки запроса SOAP. Объект может существовать в пределах запроса **(Request),** сеанса **(Session)** или приложения **(Application).** Значение **Request** подразумевает, что сервер удаляет объект после отправки ответа. Значение Session подразумевает, что объект существует в течение всего сеанса взаимодействия клиента с сервером. Значение **Application** подразумевает, что объект доступен для всех запросов в течение времени жизни приложения. В *поле* ***Method*** (рис. 14.2) задаются методы, доступные для запроса SOAP, в данном случае, метод **getWelcome. В** *поле* ***Provider Type*** задается язык реализации сервиса. Поддерживаются языки Java, JavaScript, Perl и Bean Markup Language (BML). Для примеров, рассматриваемых в этой главе, используется Java. В . поле **Provider Class** указывается класс, который реализует сервис: **SimpleService.** Поля **Script Language, Script File** и **Script** используются только

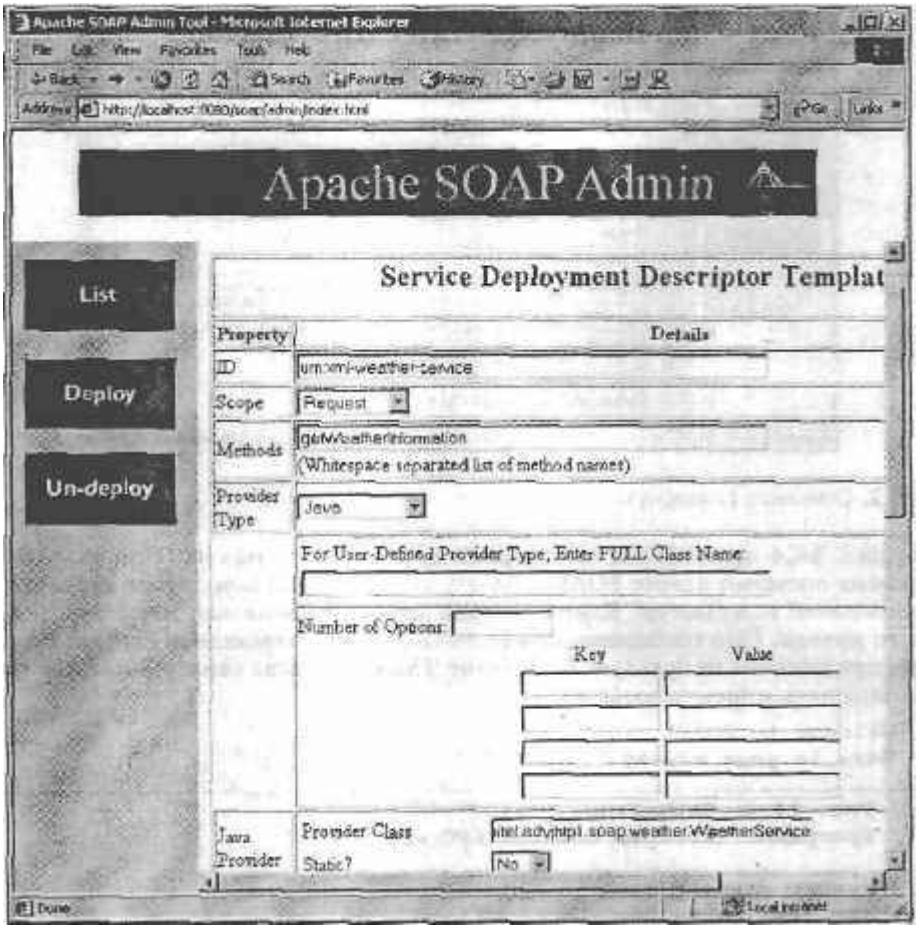


Рис. 14.2. Инструментальное средство администрирования из пакета SOAP

для сервисов, реализованных на поддерживаемых языках написания сценариев. Поле **Type Mapping** позволяет вручную устанавливать карту соответствий между типами Java и XML. Реализация Apache SOAP предоставляет отображения по умолчанию для большинства типов Java и для классов Java, которые следуют паттернам проектирования JavaBeans. Заполнив форму, щелкните на кнопке **Deploy** в нижней части формы, чтобы осуществить развертывание сервиса. Щелкните на кнопке **List,** чтобы получить список сервисов, и убедитесь, что развертывание сервиса завершилось успешно (рис. 14.3). Инструкции относительно других способов развертывания (например, через командную строку) можно найти по адресу **docs\guide\index.html**.

Рис. 14.3. Описание развернутого сервиса

На рис. 14.4 представлен клиентский код для RPC-вызова. При выполнении программа посылает запрос SOAP серверу, которым в данном случае является этот же локальный компьютер. Клиент отправляет сообщение как параметр для удаленного метода. (Это сообщение может быть задано в командной строке; по умолчанию приложение использует сообщение Thanks!) Когда сервер вызывает метод, тот возвращает клиенту сообщение

```
Welcome to SOAP!
Here Is your message: Thanks!
```

```
1 // Рис. 14.4. GetMessage.Java
2 // Программа, которая создает RPC-вызов SOAP
3
4 // импорт пакетов Java
5 import java.io.*;
6 import java.net.*;
7 import java.util.*;
```

```
 8
 9 // импорт пакетов сторонних поставщиков
10 import org.apache.soap.*;
11 import org.apache.soap.rpc.*;
12
13 public class GetHessage {
14
15    // метод main
16    public static void main( String args[] ) {
17       String encodingStyleURI = Constants.NS_URI_SQAP_ENC;
13       String message;
19
20       if ( args.length != 0 )
21         message = args[ 0 ];
22       else
23         message = "Thanks!";
24
25       // попытка удаленного вызова процедуры SOAP
26       try {
27          URL url = new URL(
28             "http://localhost:8080/soap/servlet/rpcrouter" );
29
30          // формирование вызова
31          Call remoteMethod = new Call();
32          remoteMethod.setTargetObjectURI(
33             "urn: jcml-simple-message" ) ;
34
35          // задание имени вызываемого удаленного метода
36          remoteMethod.setMethodName( "getWelcome" );
37          remoteMethod.setEncodingStyleORI( encodingStyleORI );
38
39          // задание параметров для удаленного метода
40          Vector parameters = new Vector();
41
42          parameters.addElement( new Parameter( "message",
43             String.class, message, null ) );
44          remoteMethod.setParams( parameters );
45          Response response;
46
47          // вызов удаленного метода
48          response = remoteMethod.invoke( url, "" );
49
50          // получение ответа
51          if ( response.generatedFault() ) {
52             Fault fault = responee.getFault();
53
54             System.err.println( "CALL FAILED:\nFault Code = "
55                + fault.getFaultCode()+ "\nFault String = "
56                + fault.getFaultString() );
57          }
58
59          else {
60             Parameter result = response.getReturnValue();
61
62             // отображение результатов вызова
63             System.out.println( result.getValue() );
64          }
```

```
65       }
66
67       // перехват исключения при указании неверного URL
68       catch ( MalformedURLException malformedURLException ) {
69          malformedURLException.printStackTrace();
70          System.exit( 1 );
71       }
72
73       // перехват исключения SOAPException
74       catch ( SOAPException soapException ) {
75          System.err.printin( "Error message: " +
76             soapException.getMessage() );
77          System.exit( 1 );
78       }
79    }
80 }
```

```
java getMessage
Welcome ti SOAP!
Here is your message Thanks!
java GetMessage "my message"
Welcome to SOAP!
Here is your message Thanks!
```

Рис. 14.4. Клиент, инициирующий запрос SOAP

В строке 10 импортируется пакет SOAP, который обеспечивает API для реализации SOAP. Пакет ***org.apache.soap.rpc*** в строке 11 предоставляет реализацию RPC, использующую SOAP. В строке 17 задается стиль кодирования, используемый для сообщения. Протокол SOAP, поддерживает множество стилей кодирования, но не имеет стиля кодирования по умолчанию. В данном случае используется стандартная кодировка RPC (**WS_URI_SOAP_ENC**). В строках 27-28 задается URL сервера, которому клиент отправляет содержимое rpcrouter сообщения **message**. Этот документ, который представляет собой сервлет Java, получает конверт SOAP с помощью HTTP-метода **post**. Используя URI, заданный в сообщении SOAP, он ищет сервисы, развернутые на сервере, чтобы реализовать экземпляр соответствующего объекта, в данном случае, объекта **SimpleService**.

Объекты *класса* ***Call*** осуществляют вызовы удаленных методов. В строке 31 реализуется экземпляр объекта **Call**, и ему назначается ссылка **remoteMetkod**. В строках 32—33 устанавливается URI удаленного метода. В строке 36 задается имя вызываемого метода, **getWelcome**. Затем в строке 37 задается стиль кодирования для сообщения. В строках 40-44 формируются параметры, передаваемые удаленному методу для обработки. Каждый параметр должен содержаться в своем собственном объекте, а параметрические объекты должны быть помещены в контейнер **Vector**.

В строках 42-43 формируется новый параметр для метода путем построения *объекта Parameter.* Первым параметром, передаваемым конструктору, является имя переменной или ссылки (**message**), вторым параметром является класс, которому принадлежит объект **Parameter** (String), третьим параметром является значение параметра (объект **message**), а четвертый параметр задает тип кодировки параметра (**null** указывает на использование кодировки, принятой по умолчанию). *Метод* ***setParams*** в строке 44 устанавливает параметры для объекта **remoteMethod**.

Удаленный метод активизируется в строке 48 вызовом метода **invoke.** Метод принимает два параметра: URL сервера, которому будет посылаться сообщение SOAP, и значение заголовка **SOAPAction,** который задает цель запроса. В качестве второго параметра может быть задано **null,** если заголовок **SOAPAction** использоваться не будет. Метод **invoke** возбуждает исключение ***SOAPException*** (строки 74-78), если при отправке запроса SOAP возникает ошибка. После активизации метода на сервере, результат возвращается клиенту и сохраняется в объекте, идентифицируемом ссылкой **response** (строка 48). Этот объект принимает сообщение об ошибке, если на сервере возникает проблема, например, не удается обнаружить соответствующие сервисы. В строках 51-57 определяется, является ли полученное сообщение сообщением об ошибке. Если ошибок не было, в строках 59-64 результат выводится на экран.

14.3. Служба погоды, реализованная посредством SOAP

В этом разделе описан простой Web-сервис, реализованный с помощью Java и SOAP, в котором для отправки информации от сервера к клиенту используется RPC SOAP[1]. Обязательные программные компоненты такие же, что и в примере из предыдущего раздела.

Класс **WeatherService** (рис. 14.5) предоставляет метод **getWeatherInformation,** который класс **WeatherServiceClient** вызывает через RPC SOAP. Чтобы вызов был успешным, класс **WeatherService** должен присутствовать в каталоге **classes** машины сервлетов Tomcat. RMI-версия метода **getWeatherlnformation** возвращает список **(List)** объектов WeatherBean. SOAP не поддерживает прямую передачу этих объектов Java, поэтому версия RPC SOAP метода **getWeatherlnformation** возвращает контейнер (объект **Vector**) строк **(String).**

Метод **updateWeatherConditions** (строки 16-79) прочитывает информацию о погоде с Web-страницы прогноза погоды и сохраняет ее в списке **List** объектов **WeatherBean.** В строках 22-23 создается объект **URL** для Web-страницы прогноза погоды. В строках 26-27 вызывается метод **openStream** класса **URL** для открытия соединения с ресурсом с заданным **URL.** Объект этого соединения помещается в объект-обертку **BufferedRead**.

В строках 30—76 выполняется *скрэпинг HTML* (т.е. извлечение данных с Web-страницы) для получения информации о прогнозе погоды. В строке 30 определяется строковый разделитель — "TAV12" — который задает начальную точку на Web-странице, с которой начинается поиск информации о погоде. В строках 33—34 осуществляют данных чтение с Web-страницы прогноза погоды, пока не будет достигнута сигнальная метка. Это позволяет пропускать информацию, не нужную для этого приложения.

В строках 38-41 определены две строки, которые представляют заголовки столбцов для информации о погоде. В зависимости от времени дня, заголовки столбцов будут иметь вид

```
"CITY            WEA     HI/LO   WEA     HI/LO"
```

после утреннего обновления (обычно около 10:30 по стандартному восточному времени) или

```
"CITY            WEA     LO/HI   WEA     LO/HI"
```

после вечернего обновления (обычно около 22:30 по стандартному восточному времени).

[1] В главе 2 (в оригинале это глава 13) книги *«Технологии программирования на Java 2. Книга 2»* это приложение реализовано с помощью технологии RMI. — *Прим. ред.*

В строках 55-74 осуществляется чтение информацию о погоде в каждом городе и запись этой информации в объекты **WeatherBean**. Каждый объект **WeatherBean** содержит название города, температуру воздуха и описание погоды. В строке 51 создается контейнер **Vector** для хранения объектов **WeatherBean**. В строках 66—71 листинга строки, полученные с Web-страницы прогноза погоды, добавляются в контейнер **weatherInformation** (строка 13). Первые 16 символов в строке **inputLine** относятся к названию города, следующие 6 символов описывают погоду (т.е. содержат прогноз), а следующие 6 символов представляют верхний и нижний предел температуры. Последние два столбца данных относятся к прогнозу погоды на следующий день, и в этом примере игнорируются. В строке 76 закрывается объект **BufferedRead** и связанный с ним поток ввода **InputStream**.

В строке 99 возвращается контейнер **weatherInformation**.

```
1 // WeatherService.java
2 // WeatherService предоставляет метод для извлечения
3 // информации о погоде с сайта National Weather Service.
4 package com.deitel.advjhtp1.soap.weather;
5
6 // Набор базовых пакетов Java
7 import java.io.*;
8 import java.net.URL;
9 import java.util.*;
10
11 public class WeatherService {
12
13    private Vector weatherInformation;   // объекты WeatherBean
14
15    // получение информации о погоде с сайта NWS
16    private void updateWeatherConditions()
17    {
18       try {
19          System.out.println( "Update weather information..." );
20
21          // страница прогноза погоды Национальной метеослужбы США
22          URL url = new URL(
23             "http://iwin.nws.noaa.gov/iwin/us/traveler.html" );
24
25          // задание текстового потока ввода для чтения содержимого
               //Web-страницы
26          BufferedReader in = new BufferedReader(
27             new InputstreamReader( url.openStream() ) );
28
29          // определение начала данных на Web-странице
30          String separator = "TAV12";
31
32          // нахождение разделительного символа на Web- странице
33          while ( !in.readLine().startsWith( separator ) )
34             ;    // отсутствие действий
35
36          // строки, представляющие на Web-странице Travelers
37          // Forecast заголовки для погоды в дневное и ночное время
38          String dayHeader =
39             "CITY            WEA     HI/LO   WEA     HI/LO";
40          String nightReader *
41             "CITY            WEA     LO/HI   WEA     LO/HI";
42
```

```
43          String inputLine = "";
44
45          // нахождение заголовка, которым начинается информация о
               погоде
46          do {
47             inputLine = in.readLine();
48          } while ( !inputLine.equals( dayHeader ) &&
49                    !inputLine.equals( nightHeader ) );
50
51          weatherInformation = new Vector(); // создание контейнера
                                                  Vector
52
53          // создание компонентов WeatherBean, содержащих данные
54          // о погоде, и сохранение их в контейнере weatherInformation
55          inputLine = in.readLine();  // получение данных
                                           для первого города
56
57          // Часть строки ввода inputLine, содержащая нужные данные,
58          // имеет длину в 28 символов. Если длина строки
59          // превосходит 28 символов, выполнить обработку данных.
60          while ( inputLine.length () > 28 ) {
61
62             // Подготовка строк для компонента WeatherBean для
63             // каждого города. Первые 16 символов относятся
64             // к названию города. Далее, шесть символов относятся
65             // к описанию информации о погоде. Следующие шесть
                  символов представляют наибольшую  и наименьшую
                  температуру HI/LO или LO/HI.
66             weatherInformation.add(
67                inputLine.substring( 0, 16 ) );
68             weatherInformation.add(
69                inputLine.substring( 16, 22 ) );
70             weatherInformation.add(
71                inputLine.substring( 23, 29 ) );
72
73             inputLine = in.readLine();  // получение данных для
                                              следующего города
74          }
75
76          in.close();  // закрытие соединения с Web-сервером NWS
77
78          System.out.println( "Heather information updated." );
79
80
81       // обработка неудачного соединения со службой National
            Weather Service
82       catch( java.net.ConnectException connectException ) {
83          connectException.printStackTrace();
84          System.exit( 1 );
85
86
87       // обработка других исключений
88       catch( Exception exception ) {
89          exception.printStackTrace();
90          System.exit( 1 );
91       }
```

```
92    }
93
94    // реализация интерфейсного метода для сервиса WeatherService
95    public Vector getWeatherInformation()
96    {
97       updateWeatherConditions();
98
99       return weatherInformation;
100   }
101 }
```

Рис. 14.5. Реализация класса **WeatherService** с использованием SOAP

Класс **WeatherBean** (рис. 14.6) хранит данные, которые класс **WeatherService** извлекает с Web-сайта прогноза погоды. Этот класс хранит название города, температуру и текстовое описание погоды. В строках 64—85 предоставляются методы *get* для каждого фрагмента информации. В строках 25-45 загружается файл свойств, который содержит имена файлов с условными изображениями характера погоды. Этот статический блок обеспечивает доступность файлов изображений сразу же после того, как виртуальная машина осуществит загрузку класса **WeatherBean** в память.

```
1 // WeatherBean.java
2 // WeatherBean содержит информацию о погоде для одного города.
3 package com.deitel.advjhtp1.rmi.weather;
4
5 // Набор базовых пакетов Java
6 import java.awt.*;
7 import java.io.*;
8 import java.net.*;
9 import java.util.*;
10
11 // Пакеты расширений Java
12 import javax.swing.*;
13
14 public class WeatherBean implements Serializable {
15
16    private String cityName;          // название города
17    private String temperature;       // температура в городе
18    private String description;       // описание погоды
19    private ImageIcon image;          // изображение характера погоды
20
21    private static Properties imageNames;
22
23    // инициализация объекта imageNames при загрузке класса
24    // WeatherInfo в память
25    static {
26       imageNames = new Properties(); // создание таблицы свойств
27
28       // загрузка описаний погоды и имен изображений из
29       // файла свойств
30       try {
31
32          // получение URL для файла свойств
33          URL url = WeatherBean.class.getResource(
34             "imagenames.properties" );
```

```
35
36          // загрузка содержимого файла свойств
37          imageNames.load( new FileInputStream( url.getFile() ) );
38       }
39
40       // обработка исключений при открытии файла
41       catch ( IOException ioException ) {
42          ioException.printStackTrace();
43
44       }
45    } // конец блока static
46
47    // конструктор WeatherBean
48    public WeatherBean( String city, String weatherDescription,
49       String cityTemperature )
50    {
51       cityName = city;
52       temperature = cityTemperature;
53       description = weatherDescription.trim();
54
55       URL url = WeatherBean.class.getResource( "images/" +
56          imageNames.getProperty( description, "noinfo.jpg" ) );
57
58       // получение имени изображения или использование
59       // имени noinfo.jpg, если описание погоды не найдено
60       image = new ImageIcon( url );
61    }
62
63    // получение названия города
64    public String getCityName()
65    {
66       return cityName;
67    }
68
69    // получение температуры
70    public String getTemperature()
71    {
72       return temperature;
73    }
74
75    // получение описания погоды
76    public String getDescription()
77    {
78       return description;
79    )
80
81    // получение условного изображения характера погоды
82    public ImageIcon getImage()
83    {
84       return image;
85    }
86
```

Рис. 14.6. Класс WeatherBean хранит сведения о погоде е одном городе

Класс WeatherServiceClient (рис. 14.7) осуществляет удаленный процедурный вызов SOAP метода getWeatherlnformation класса WeatherService. В строках 31-32 устанавливается URL сервиса SOAP. В строке 35 создается новый объект Call, который хранит информацию, необходимую для выполнения вызова удаленной процедуры. В строках 36—37 устанавливается TJRI, который уникально идентифицирует сервис метеопрогнозов в машине сервлетов. В строках 40—41 задается имя метода для удаленного вызова процедуры. В строках 42-43 устанавливается кодировка, используемая при вызове. В строке 46 создается объект Response и вызывается метод invoke на объекте Call с указанием URL в качестве параметра. Объект Response содержит ответ на вызов удаленной процедуры. В строке 49 определяется, не возникла ли в результате ответа ошибка (объект Fault). В этом случае в строках 52—54 выводится код ошибки. Если ошибок не возникло, в строке 58 извлекается объект, возвращенный вызовом удаленной процедуры. В строках 60-61 осуществляется приведение типа Object к типу Vector. В строках 64-65 создается список List и вызывается метод createBeans (строки 95-107) с передачей контейнером (Vector) строк в качестве параметра. Метод createBeans преобразует контейнер (объект типа Vector) строк в список (List) объектов WeatherBeans. В строках 68-69 создается модель ListModel списка объектов WeatherBean. В строках 73-77 создается компонент JList, содержащий информацию, полученную в результате вызова удаленной процедуры, и список JList отображается в окне JFrame.

```
1 // WeatherServiceClient.java
2 // WeatherServiceClient осуществляет доступ к удаленному объекту
3 // WeatherService через SOAP, чтобы извлечь информацию о погоде.
4 package com.deitel.advjhtp1.soap.weather;
5
6 // Набор базовых пакетов Java
7 import java.util.*;
8 import java.net.*;
9
10 // Пакеты расширений Java
11 import javax.swing.*;
12
13 // Пакеты сторонних поставщиков
14 import org.apache.soap.*;
15 import org.apache.soap.rpc.*;
16
17 // Пакеты Deitel
18
19 import com.deitel.advjhtp1.rmi.weather.*;
20
21 public class WeatherServiceClient extends JFrame {
22
23    // конструктор WeatherServiceClient
24    public WeatherServiceClient( String server )
25    {
26       super( "SOAP WeatherService Client" );
27       // соединение с сервером и получение информации о погоде
28       try {
29
30          // URL удаленного объекта SOAP
31          URL url = new URL( "http://" + server + ":8080/soap/"
32             + "servlet/rpcrouter" ) ;
33
34          // формирование удаленного вызова SOAP
```

```
35          Call remoteMethod = new Call();
36          remoteMethod.setTargetObjectURI{
37             "urn:xml-weather-service" );
38
39          // задание имени вызываемого удаленного метода
40          remoteMethod.setMethodName(
41             "getWeatherInformation" );
42          remoteMethod.setEncodingstyleURI(
43             Constants.NS_URI_SOAP_ENC );
44
45          // вызов удаленного метода
46          Response response = remoteMethod.invoke{ url, "" );
47
48          // получение ответа
49          if ( response.generatedFault() ) {
50             Fault fault = response.getFault();
51
52             System.err.println( "CALL FAILED:\nFault Code = "
53                + fault.getFaultCode() + "\nFault String = "
54                + fault.getFaultString() );
55          }
56
57          else {
58             Parameter result = response.getReturnValue();
59
60             Vector weatherStrings = ( Vector )
61                result.getValue();
62
S3             // получение информации о погоде из объекта результата
64             List weatherInformation = createBeans(
65                weatherStrings );
66
67             // создание модели WeatherListModel для информации
                  о погоде
68             ListModel weatherListModel =
69                new WeatherListModel( weatherInformation );
70
71             // создание списка JList, установка для него
72             // интерфейса CellRenderer и добавление его а макет
73             JList weatherJList = new JList( weatherListModel );
74             weatherJList.setCellRenderer( new
75                WeatherCellRenderer() );
76             getContentPane().add( new
77                JScrollPane( weatherJList ) );
78          }
79
80       } // конец блока try
81
82       // обработка неверного URL
83       catch ( MalformedURLException malformedURLException ) {
            malformedURLException.printstackTrace();
         }

87       // обработка исключения SOAP
88       catch ( SOAPException soapException ) {
89          soapException.printstacJcTrace();
```

```
90          }
91
92       } // конец конструктора WeatherServiceClient.
93
94       // создание списка List объектов WeatherBean из контейнера строк
            weatherStrings
95       public List createBeans( Vector weatherStrings )
96       {
97          List list = new ArrayList();
            for ( int i = 0; ( weatherStrings.size{) - 1 ) > i;
               i += 3 ) {
               list.add( new WeatherBean(
                   ( String ) weatherStrings.elementAt( i ),
102                ( String ) weatherStrings.elementAt( i + 1 ),
103                ( String ) weatherStrings.elementAt( i + 2 ) ) );
104         }
105
106         return list;
107      }
108
109      // выполнение приложения WeatherServiceClient
110      public static void main( String args[] )
111      {
            WeatherServiceClient client = null;

            // если IP-адрес сервера или хост-имя не заданы, использовать
115         // "localhost"; иначе использовать указанное хост-имя
116         if ( args.length == 0 )
117            client = new WeatherServiceClient( "localhost" );
            else
               client = new WeatherServiceClient( args[ 0 ] );
120
121         // настройка и отображение окна приложения
122         client.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
123         client.pack();
            client.setResizable( false );
125         client.setVisible( true );
126      }
127   }
```

Рис. 14.7. Реализация класса **WeatherServiceClient** с использованием SOAP

Класс **WeatherListModel** (рис. 14.8) представляет собой модель типа **ListModel**, которая содержит объекты WeatherBean, отображаемые в списке **JList.** В этом примере мы продолжим применять паттерны проектирования, на этот раз, введя *адаптерный паттерн проектирования Adapter,* который дает возможность взаимодействовать друг с другом компонентам, имеющим несовместимые интерфейсы. Паттерн проектирования Adapter имеет множество аналогий в реальном мире. Например, электрические вилки для бытовых приборов, используемых в США, не совместимы с электрическими розетками, применяемыми в Европе. Чтобы пользоваться американскими электроприборами в Европе, пользователю необходимо вставить переходник-адаптер между электрической розеткой и вилкой. С одной стороны, этот адаптер обеспечивает интерфейс, совместимый с американской электрической вилкой. С другой стороны, адаптер обеспечивает интерфейс, совместимый с европейской электрической розеткой. Класс **WeatherListModel** играет роль

адаптера в паттерне проектирования Adapter. Интерфейс **List** Java не является совместимым с интерфейсом класса **JList,** поскольку компонент **JList** способен извлекать элементы только из модели **ListModel.** В связи с этим мы предоставляем класс **WeatherListModel,** который адаптирует интерфейс **List** к интерфейсу **JList. Когда JList** вызывает метод **getSize** класса **WeatherListModel,** объект **WeatherListModel** вызывает метод **size** интерфейса **List.** Когда объект **JList** вызывает метод **getElementAt** класса **WeatherListModel,** модель **WeatherListModel** вызывает метод **get** класса **JList и т.д.**

```
1 // WeatherListModel.Java
2 // WeatherListModel расширяет класс AbstractListModel, чтобы предос-
3 // тавить модель ListModel для хранения списка объектов HeatherBeans.
4 package com.deitel.advjhtp1.rmi.weather;
5
6 // Набор базовых пакетов Java
7 import java.util.*;
8
9 // Пакеты расширений Java
10 import javax.swing.AbstractListModel;
11
12 public class WeatherListModel extends AbstractListModel {
13
14    // список элементов в модели ListModel
15    private List list;
16
17    // конструктор без параметров WeatherListModel
18    public WeatherListModel()
19    {
20       // создание нового списка для объектов WeatherBean
21       list = new ArrayList();
22    }
23    // конструктор WeatherListModel
24    public WeatherListModel( List itemList )
25    {
26       list = itemList;
27    }
28
29    // получение размера списка
30    public int getSize()
31    {
32       return list.size();
33    }
34
35    // получение ссылки типа Object на элемент с заданным индексом
36    public Object getElementAt( int index )
37    {
38       return list.get( index );
39    }
40
41    // добавление элемента в модель WeatherListModel
42    public void add( Object element )
43    {
44       list.add( element );
45       firelntervalAdded( this, list.size(), list.size() );
46    }
47
```

```
48
49    // удаление элемента из модели WeatherListModel
50    public void remove( Object element )
51    {
52       int index = list.indexOf( element );
53
54       if ( index != -1 ) {
55          list.remove( element );
56          firelntervalRemoved( this, index, index );
57       }
58
59    } // конец метода remove
60
61    // удаление всех элементов из модели WeatherListModel
62    public void clear()
63    {
64    65       // получение исходного размера списка
         int size = list.size();

         // очистка всех элементов в списке
68       list.clear"() ;
69
70       // уведомление слушателей об изменении содержимого
71       fireContentsChanged( this, 0, size );
72
73
```

Рис. 14.8. Класс **WeatherListModel** является реализацией интерфейса **ListModel** и хранит информации о погоде

Класс **JList** использует интерфейс **ListCellRenderer** для воспроизведения каждого элемента, содержащегося в модели **ListModel** объекта **JList**. Класс **WeatherCellRenderer** (рис. 14.9) является подклассом класса **DefaultListCellRenderer** и используется для отображения объектов **WeatherBean** в списке **JList**. Метод **getListCellRendererComponent** создает и возвращает объект **WeatherItem** (рис. 14.10) для заданного объекта **WeatherBean**.

Класс **WeatherItem** (рис. 14.10) является подклассом класса **JPanel** и используется для отображения информации о погоде, хранящейся в объекте **WeatherBean**. Класс **WeatherCellRenderer** использует экземпляры класса **WeatherItem** для отображения информации о погоде в списке **JList**. В блоке **static** (строки 22-29) осуществляется загрузка объекта **backgroundImage** класса **ImageIcon** в память, когда виртуальная машина загружает сам класс **WeatherItem**. Это обеспечивает, что объект **backgroundImage** будет доступен всем экземплярам класса **WeatherItem**. Метод **paintComponent** (строки 38-56) осуществляет вывод фонового изображения **backgroundImage** (строка 43), названия города (строка 50), температуры (строка 51) и изображения **ImageIcon** для объекта **WeatherBean**, которое характеризует погодные условия (строка 54).

```
1 //  WeatherCellRenderer.java
2 // WeatherCellRenderer - специализированный интерфейс
3 // ListCellRenderer для объектов WeatherBean в списке JList.
4 package  com.deitel.advjhtp1.rmi.weather;
5
6 // Набор базовых пакетов Java
7 import  java.awt.*;
```

```
 8
 9 // Пакеты расширении Java
10 import javax.swing.*;
11
12 public class WeatherCellRenderer extends DefaultListCellRenderer {
13
14    // возврат объекта Weatherltem, который отображает информацию о
         погоде в городе
15    public Component getListCellRendererComponent( JList list,
16       Object value, int index, boolean isSelected, boolean focus )
17    {
18       return new Weatherltem( ( WeatherBean ) value );
19    }
20 }
```

Рис. 14.9. Класс WeatherCellRenderer является видоизмененным классом ListCellRenderer для отображения объектов WeatherBean в списке JList

```
 1 // WeatherItem. java
 2 // Weatherltem отображает информацию о погоде в городе в панели
      JPanel.
 3 package com.deitel.advjhtp1.rmi.weather;
 4
 5 // Набор базовых пакетов Java
 6 import java.awt.*;
 7 import java.net.*;
 8 import java.util.*;
 9
10 // Пакеты расширений Java
11 import javax.swing.*;
12
13 public class Weatherltem extends JPanel {
14
15    private WeatherBean weatherBean;  // информация о погоде
16
17    // фоновый рисунок ImageIcon
18    private static ImageIcon backgroundImage;
19
20    // блок инициализации static загружает файл изображения,
21    // когда класс Weatherltem загружается в память
22    static {
23
24       // получение URL для фонового изображения
25       URL url = WeatherItem.class.getResource( "images/back.jpg" );
26
27       // фоновое изображения для информации о погоде в каждом
            из городов
28       backgroundImage = new ImageIcon( url );
29    }
30
31    // инициализация компонента Weatherltem
32    public WeatherItem( WeatherBean bean )
33    {
34       weatherBean = bean;
35    }
36
```

```
37    // отображение информации о погоде в городе
38    public void paintComponent( Graphics g )
39    {
40       super.paintComponent( g );
41
42       // рисование фона
43       backgroundImage.paintIcon( this, g, 0, 0 );
44
45       // задание шрифта и цвета для рисования,
46       // затем отображение названия города и температуры
47       Font font = new Font( "SansSerif", Font.BOLD, 12 );
48       g.setFont( font );
49       g.setColor( Color.white );
50       g.drawstring( weatherBean.getCityNameO, 10, 19 );
51       g.drawstring( weatherBean.getTemperature(), 130, 19 );
52
53       // вывод условного изображения характера погоды
         weatherBean.getlmage().paintIcon( this, g, 253, 1 );

56    } // конец метода paintComponent
57
5S    // установка в качестве желательных размеров окна для
59    // компонента WeatherItem высоты и ширины фонового изображения
60    public Dimension getPreferredSize()
61    {
62       return new Dimension( backgroundImage.getIconWidth(),
63          backgroundlmage.getIconHeight() );
64    }
65 }
```

Рис. 14.10. Класс **WeatherItem** отображает информацию о погоде в одном городе

Развертывание сервиса службы погоды осуществляется точно так же, как и развертывание ранее рассмотренного нами сервиса передачи сообщений. Запустите средство работы с сервлетами Tomcat и откройте инструментальное средство администрирования SOAP (localhost:8080/soap/admin), как показано на рис. 14.11.

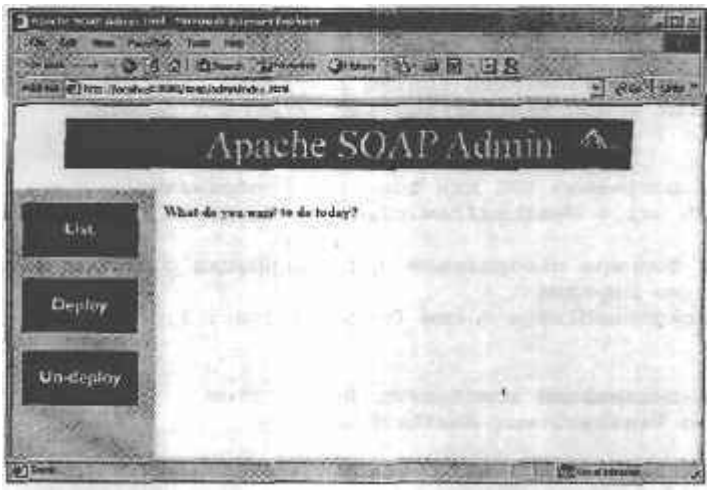


Рис. 14.11. Страница средства администрирования **Apache SOAP Admin**

Щелкните на **Deploy** и введите информацию в форму, как показано на рис. 14.12. Следует вводить полное имя пакета для **WeatherService**. Осуществив развертывание сервиса, выполните класс **WeatherServiceClient**. В результате вызова удаленной процедуры извлекается информацию о погоде, а клиент затем отображает ее (рис. 14.13).

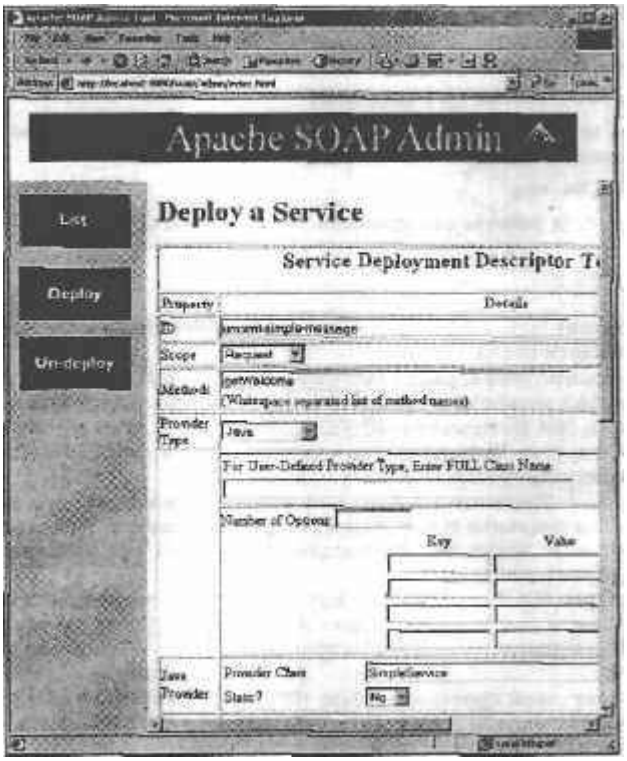


Рис. 14.12. Шаблон дескриптора развертывания **Service Deployment Descriptor Template** SOAP Apache

Рис. 14.13. Клиент **SOAP WeatherService Client**

14.4. Ресурсы в Internet и во Всемирной паутине

www.sun.com/software/sunone/index.html

Сайт корпорации Sun Microsystems, посвященный спецификации Web-сервисов Open Net Environment (ONE).

xml.apache.org/soap

С этого сайта можно загрузить реализацию SOAP Apache. Здесь также можно найти документацию и иную информацию.

xml.apache.org/xerceS~j/index.html

С этого сайта можно загрузить синтаксический анализатор Apache Xerces для Java, а также документацию.

Jakarta.apache.org

Web-сайт средства работы с сервлетами Apache Jakarta Tomcat.

Резюме

- Web-сервисом может быть любое доступное через Web приложение, например, Web-страница с динамическим содержимым.
- В более узком смысле Web-сервис — это приложение, предоставляющее общедоступный интерфейс, который может использоваться другими приложениями в Web.
- Архитектура Open Net Environment (ONE) требует, чтобы Web-сераис был доступен через протокол HTTP и другие Web-протоколы, осуществлял взаимодействие на базе XML-сообщений и был зарегистрирован сервисом поиска.
- Web-сервисы могут обеспечивать высокий уровень совместимости между различными системами. Функциональная совместимость и расширяемость Web-сервисов подразумевает, что разработчики могут быстро создавать большие приложения и более крупные Web-сервисы из более мелких.
- Спецификация Open Net Environment, разработанная корпорацией Sun, описывает архитектуру для создания *интеллектуальных Web-сервисов.* Согласно заявлению Sun, интеллектуальные Web-сервисы совместно используют общее операционное окружение с другими сервисами.
- SOAP представляет собой протокол на базе HTTP-XML, который дает возможность приложениям взаимодействовать через Internet, используя XML-документы, называемые XML-сообщениями.
- SOAP является не зависящим от используемой платформы протоколом, и может быть реализован на любом языке программирования. SOAP поддерживает транспортировку с применением практически любого приемлемого протокола.
- Сообщение SOAP содержит конверт, который описывает содержимое, предполагаемого получателя и требования к обработке сообщения. Необязательный элемент **header** сообщения SOAP содержит дополнительную информацию по обработке для приложений, которые получают сообщение SOAP.
- С помощью заголовка поверх SOAP могут быть надстроены более сложные протоколы. Тело сообщения SOAP содержит специфичные для приложения данные, предназначенные предполагаемому получателю сообщения.
- SOAP может использоваться для удаленного вызова процедур (Remote Procedure Call — RPC), который представляет собой запрос, посылаемый другому компьютеру для выполнения определенной задачи. RPC использует словарь XML для указания метода, подлежащего активизации, передаваемых методу параметров и URL целевого объекта.
- Поскольку различные компании используют различные платформы, приложения и форматы данных, обмен данными может вызвать затруднения. В связи с этим партнеры по бизнесу устанавливают такие протоколы и форматы данных, которые способствуют эффективному ведению электронной коммерции.

Терминология

application-to-application (A2A) integration — интеграция на уровне приложение — приложение
asynchronous RPC — асинхронный удаленный вызов процедур
Call, класс
deploying a service — развертывание сервиса
distributed object architecture — архитектура распределенных объектов
Fault, класс
firewall — брандмауэр, межсетевой экран
Hypertext Transfer Protocol (HTTP)
invoke, метод класса **Call**
loosely coupled messaging — слабосвязанны С обмен сообщениями
messaging — обмен и передача сообщений
org.apache.soap.rpc
Parameter, класс
Remote Procedure Call (RPC) — удаленный вызов процедур
request-response — запрос-ответ
schema — схема
setMethodName, метод класса Call
setParams, метод класса Call
SimpleObjectAccessProtocol (SOAP) — протокол простого доступа К объектам
Sun Open Net Environment — спецификацш открытого сетевого окружения Sun
synchronous RPC — синхронный удаленный вызов процедур
Universal Description, Discovery and Integration (UDDI) — универсальное описание, обнаружение и интеграция
Web services — Web-сервисы
XML-SOAP admin tool — инструментальное средство администрирования XML-SOAP

Упражнения *для самоконтроля*

14.1. Ответьте, является ли каждое из следующих высказываний *истинным* или *ложным.* Если высказывание *ложно,* объясните, почему.

a) SOAP — это технология, которая способствует передаче данных через сеть.

b) Чтобы протокол SOAP работал, он должен быть привязан к HTTP.

c) Чтобы взаимодействовать посредством SOAP, программные системы должны иметь одинаковую распределенную объектную архитектуру.

d) Тело сообщения SOAP может содержать удаленный вызов процедуры.

14.2. Заполните пропуски в следующих высказываниях:

a) Удаленный вызов процедуры RPC SOAP требует указания имени вызываемого метода, его параметров и_____.

b)_____SOAP содержит информацию, которая описывает содержимое, предполагаемого получателя и требования по обработке сообщения SOAP.

c) SOAP допускает передачу через межсетевые экраны, поскольку он использует _____^___ в качестве транспортного механизма.

d) RPC SOAP используют модель_____HTTP.

Ответы на упражнения для самоконтроля

14.1. a) Истинно.

b) Ложно. SOAP может быть привязан к другим протоколам.

c) Ложно. SOAP является не зависящим от используемой платформы протоколом.

d) Истинно.

14.2. a) требований к обработке сообщения.

b) Конверт.

c) HTTP.

d) запрос-ответ.

Упражнения

14.3. Напишите серверный класс, содержащий метод sort, который может осуществлять сортировку заданных чисел. Напишите клиентскую программу, которая может осуществлять удалнный вызов RPC SOAP метода sort, отправляя при этом множество неотсортированных значений. Отобразите на клиенте результаты сортировки.

14.4. Модифицируйте класс WeatherServiceClient, чтобы обновлять информацию в нем через задаваемые пользователем интервалы времени. Измените настройки Tomcat, чтобы сделать объект **WeatherService** постоянно действующим (персистентным). Это позволит более эффективно осуществлять обновления.

14.5. Создайте клиентскую и серверную составляющие приложения с архитектурой, подобной той, которая использовалась для службы погоды, но получающие информацию о ценах с сайта сопоставления цен, такого как shopper.cnet.com.,

14.6. Создайте серверный класс, который способен хранить и извлекать строки. Выполните развертывание класса, чтобы он постоянно присутствовал на сервере (являлся персистентным). После этого создайте клиента, который хранит и извлекает строки С сервера.

14.7. Напишите простой одноранговый сервис для неотложного обмена сообщениями. Создайте серверный класс, содержащий метод, который открывает окно с текстом сообщения при вызове этого метода клиентом. Клиент предоставляет пользователю возможность вводить текст сообщения и вызывать метод серверного класса для отображения сообщения на другом компьютере.

Используемые источники

1. D. Savarese, «ONEWeb to Rule Them ALL». Java Pro August 2001: p. 58.