Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

В. В. ГОЛЕНКОВ, Д. Г. КОЛБ, В. П. ИВАШЕНКО

АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СЕТЕЙ И ОСНОВЫ ЗАЩИТЫ ИНФОРМАЦИИ

Рекомендовано УМО вузов Республики Беларусь по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для студентов учреждений, обеспечивающих получение высшего образования по специальности «Искусственный интеллект»

Минск БГУИР 2010

УДК 004.4:004.7(076)
ББК 32.973.202-018.2я73
Г60

Р е ц е н з е н т ы:

кафедра автоматизированных систем управления войсками
Военной академии Республики Беларусь;
заведующий лабораторией Объединённого института проблем информатики
Национальной академии наук Беларуси,
доцент кафедры «Интеллектуальные системы»
Белорусского национального технического университета,
кандидат технических наук Н. И. Мурашко

Голенков, В. В.

Г60 Аппаратное и программное обеспечение сетей и основы защиты информации : учеб.-метод. пособие / В. В. Голенков, Д. Г. Колб, В. П. Ивашенко. – Минск : БГУИР, 2010. – 111 с. : ил.
ISBN 978-985-488-523-0.

Содержит теоретические сведения, необходимые для выполнения лабораторного практикума по дисциплине «Аппаратное и программное обеспечение сетей и основы защиты информации». Пособие рассчитано на студентов, имеющих базовые навыки программирования на языках C++ и Java.

Предназначено для студентов специальности «Искусственный интеллект» всех форм обучения.

УДК 004.4:004.7(076)
ББК 32.973.202-018.2я73

ISBN 978-985-488-523-0

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Пособие состоит из разделов, соответствующих темам лабораторных занятий. Каждый раздел сопровождается краткими теоретическими сведениями и примерами, позволяющими на практике закрепить теоретические сведения. После группы тем приведены варианты заданий для выполнения лабораторных работ. Лабораторные работы подобраны таким образом, чтобы студент мог ознакомиться с различными аспектами программирования компьютерных сетей, такими, как программирование для сетевых протоколов, программирование широко распространенных протоколов прикладного уровня, программирование с использованием сервисно-ориентированных технологий, а также разработка интернет-приложений.

В процессе лабораторной работы студентам необходимо:

1. Изучить теоретическую часть (тема данного практикума, конспект лекций, дополнительная литература).

2. Осуществить программную реализацию задания средствами языка высокого уровня.

3. Подготовить и защитить отчет по лабораторной работе.

4. Ответить на контрольные вопросы.

В отчете необходимо отразить цель лабораторной работы; описать этапы жизненного цикла приложения, разработанного в результате реализации индивидуального задания. Сформулировать выводы по работе.

1. СТЕКИ ПРОТОКОЛОВ TCP/IP И NETBIOS/SMB

1.1. Стек протоколов TCP/IP

Стек протоколов Transmission Control Protocol/Internet Protocol (TCP/IP)– это стандартный промышленный набор протоколов, разработанный для глобальных вычислительных сетей (Wide Area Networks, WAN). Протокол TCP/IP использует IP-адрес, маску подсети и шлюз по умолчанию для соединения с узлами сети. Узлы TCP/IP, работающие в глобальной сети, требуют задания всех трех параметров в конфигурации. Каждая плата сетевого адаптера в компьютере, использующем TCP/IP, нуждается в этих параметрах.

IP-адрес – 4-байтное (для 4-й версии протокола IP) или 16-байтное (для 6-й версии протокола IP) целое число, позволяющее однозначно определить узел сети TCP/IP. Каждому сетевому устройству, использующему TCP/IP, требуется уникальный IP-адрес. IP-адрес состоит из двух частей: идентификатора сети и идентификатора узла. Первый служит для обозначения всех узлов в одной физической сети; второй обозначает конкретный узел сети. Возможна адресация IP-адресом группы узлов сети.

Маска подсети выделяет часть IP-адреса и позволяет TCP/IP отличить идентификатор сети от идентификатора узла. Для связи узлы TCP/IP используют маску подсети (например 255.255.255.0), чтобы определить, находится узел-получатель в локальной или удаленной сети.

Для того чтобы установить соединение с узлом из другой сети, необходимо сконфигурировать IP-адрес ***шлюза по умолчанию***. TCP/IP посылает пакеты, предназначенные для удаленных сетей, на шлюз по умолчанию только в том случае, если на локальном узле не сконфигурирован другой маршрут к сети получателя. Если не сконфигурирован шлюз по умолчанию, то связь может быть ограничена локальной сетью.

1.2. Стек протоколов NetBIOS/SMB

Стандарт NetBIOS, разработанный для компании IBM фирмой Sytek Corporation в 1983 г., позволяет приложениям взаимодействовать по сети. Этот стандарт определяет интерфейс сеансового уровня и протокол передачи данных

и управления сеансом. Сам протокол NetBIOS (Network Basic Input/Output System) был реализован в 1984 г. как сетевое расширение стандартных функций базовой системы ввода/вывода (BIOS) IBM PC для сетевой программы PC Network компании IBM.

В дальнейшем этот протокол был заменен так называемым протоколом расширенного пользовательского интерфейса NetBEUI – NetBIOS Extended User Interface. Для обеспечения совместимости приложений в качестве интерфейса к протоколу NetBEUI был сохранен интерфейс NetBIOS. Протокол NetBEUI разрабатывался как эффективный протокол, использующий немного ресурсов и предназначенный для сетей, насчитывающих не более 200 рабочих станций. Однако отсутствие в стандарте протокола функций поддержки маршрутизации пакетов ограничивает его применение локальными сетями, не разделенными на подсети, и делает невозможным его использование в составных сетях. Некоторые ограничения NetBEUI снимаются реализацией этого протокола – NBF (NetBEUI Frame), которая была включена в операционную систему Microsoft Windows NT.

Протокол SMB (Server Message Block) выполняет функции сеансового, представительного и прикладного уровней. На основе SMB реализуется файловая служба, а также службы печати и передачи сообщений между приложениями.

NetBIOS поддерживает следующие команды и функции:

- регистрация и проверка сетевых имен;
- запуск и завершение сеанса;
- надежная передача данных сеанса, ориентированного на соединение;
- ненадежная передача дейтаграмм (datagram) без установки соединения;
- мониторинг и управления протоколом (драйвером) и адаптером.

Имя NetBIOS – это уникальный 16-байтный адрес, используемый для идентификации в сети ресурса NetBIOS. Имена могут быть эксклюзивными (exclusive) или групповыми – не эксклюзивными (non-exclusive). Первые, как правило, используют для взаимодействия с некоторым процессом на компьютере, вторые – для передачи информации нескольким компьютерам одновременно. Все сетевые службы операционной системы Windows регистрируют свои имена NetBIOS. Все сетевые команды операционной системы Windows используют имена NetBIOS для доступа к этим службам. Имена NetBIOS также

применяются в других системах, основанных на NetBIOS, например LAN Manager для операционной системы UNIX.

Все узлы, использующие NetBIOS поверх TCP/IP, взаимодействуя с узлами NetBIOS, например с Windows NT, применяют регистрацию, обнаружение и освобождение имен.

Начиная работу, узел NetBIOS поверх TCP/IP ***регистрирует имя*** NetBIOS при помощи запроса регистрации имени (name registration request) – широковещательного или направленного только к серверу имен NetBIOS. Если какой-то узел пытается зарегистрировать уже зарезервированное имя NetBIOS, то либо узел с таким именем, либо сервер имен NetBIOS посылает отказ в регистрации имени (negative name registration response). Начинающий работу узел получает в качестве ответа сообщение об ошибке инициализации.

Обнаружение имени (name discovery) в локальной сети осуществляется при помощи механизма широковещания или сервера имен NetBIOS. Когда узел Windows NT хочет связаться с другим узлом TCP/IP, он посылает запрос, содержащий искомое имя NetBIOS, на определение имени (name query request), используя широковещательный пакет или адресуя запрос только серверу имен NetBIOS. Узел, которому принадлежит искомое имя или сервер имен NetBIOS, отправляет обратно положительный ответ об определении имени (positive name query response).

Освобождение имени (name release) происходит, если приложение или служба NetBIOS прекращает работу. Например, когда на узле останавливается служба Workstation, этот узел перестает посылать отказы в регистрации имен при попытке использовать зарезервированное им имя. В таком случае говорят, что имя NetBIOS освобождено (released) и его могут использовать другие узлы.

1.3. Работа со стеком протоколов TCP/IP с помощью команд операционной системы

В операционной системе Windows предусмотрен целый набор команд, позволяющих тестировать и настраивать сеть TCP/IP. Рассмотрим их подробнее.

Команда ipconfig. Для проверки параметров конфигурации узла, включая IP-адрес, маску подсети и шлюз по умолчанию можно использовать утилиту

ipconfig. Она служит для отображения всех текущих параметров сети TCP/IP и обновления параметров DHCP и DNS.

Данная команда используется при выяснении, успешно ли прошла инициализация TCP/IP и не дублируется ли IP-адрес, указанный в конфигурации.

Синтаксис команды:

ipconfig [/x],

где *x* –параметр команды. Полный список параметров *ipconfig* можно увидеть, набрав в консоли команду *ipconfig /?*

При вызове команды *ipconfig* без параметров выводятся только IP-адрес, маска подсети и основной шлюз для каждого сетевого адаптера. Если адрес, заданный в конфигурации уже используется другим узлом в сети на том же сегменте, то отобразится заданный IP-адрес, но маска подсети будет равна 0.0.0.0.

Команда ping. После проверки конфигурации командой *ipconfig* можно запустить команду *ping* (**p**acket **i**nter**n**et **g**roper) для тестирования соединений. Это диагностическое средство тестирует сеть TCP/IP и позволяет определить неисправности соединения. Команда *ping* использует пакеты эхозапроса (echo request) и эхоответа (echo reply) протокола ICMP (Internet Control Message Protocol). *Ping* – это основная TCP/IP-команда, используемая для устранения неполадки в соединении, проверки возможности доступа и разрешения имен.

Синтаксис команды:

ping IP_адрес [–x] …,

где *x* – параметр команды. Полный список параметров *ping* можно увидеть, набрав в консоли команду *ping /?*

Если проверка прошла успешно, то отобразится сообщение типа:

```
Pinging IP адрес with 32 bytes of data:
Reply from IP_адрес: bytes= m time<10ms TTL= n
Reply from IP_адрес: bytes= m time<10ms TTL= n
Reply from IP_адрес: bytes= m time<10ms TTL= n
Reply from IP_адрес: bytes= m time<10ms TTL= n.
```

Команда arp. Команда *arp* используется для просмотра, добавления или удаления записей в таблицах трансляции адресов IP в физические адреса. Эти записи используются при работе протокола ARP (Address Resolution Protocol)

Синтаксис команды:

arp -a [inet_addr] [-N [if_addr]]

arp -d inet_addr [if_addr]

arp -s inet_addr ether_addr [if_addr]

Команда netstat. Производит отображение активных подключений TCP-портов, прослушиваемых компьютером, статистики протокола Ethernet, таблицы маршрутизации IP, статистики семейства протоколов IPv4 (для протоколов IP, ICMP, TCP и UDP) и семейства протоколов IPv6 (для протоколов IPv6, ICMPv6, TCP через IPv6 и UDP через IPv6). Запущенная без параметров команда *netstat* отображает подключения TCP.

Синтаксис команды:

netstat [-a] [-e] [-n] [-o] [-p протокол] [-r] [-s] [интервал]

Чтобы вывести все активные подключения, отсортированные по возрастанию номера порта, необходимо набрать *netstat –n.*

Команда nslookup. Предоставляет сведения, предназначенные для диагностики инфраструктуры DNS. Для использования этой команды необходимо быть знакомым с принципами работы системы DNS. Средство командной строки *nslookup* доступно, только если установлен протокол TCP/IP.

Синтаксис команды:

nslookup [-подкоманда ...] [{искомый_компьютер| [-сервер]}]

Команда hostname. Утилита командной строки *hostname* выводит имя системы, в которой была запущена эта команда.

У данной команды нет параметров.

Команда traceroute. Определяет путь до точки назначения с помощью посылки в точку назначения эхосообщений протокола Internet Control Message Protocol (ICMP) с постоянным увеличением значений срока жизни (Time to Live, TTL). Выведенный путь – это список ближайших адресов маршрутизаторов, находящихся на пути между узлом источника и точкой назначения. Ближайший адрес представляет собой адрес маршрутизатора, который является ближайшим к узлу отправителя на пути. Например, чтобы вывести трассу маршрута к *http://www.google.com*, нужно выполнить команду

tracert www.google.com

Запущенная без параметров команда *tracert* выводит справку.

Команда route. Эта команда используется для редактирования или просмотра таблицы маршрутов IP из командной строки. Параметр /? выводит все доступные параметры команды.

Команда nbtstat. Эта команда служит для отображения статистики протокола NetBIOS через TCP/IP (NetBT), таблиц имен NetBIOS для локального и удаленного компьютеров, а также кэша имен NetBIOS. Команда *nbtstat* позволяет обновить кэш имен NetBIOS и имена, зарегистрированные в службе имен Интернета для Windows (WINS). Запущенная без параметров команда *nbtstat* выводит справку. Синтаксис команды следующий:

nbtstat [-a удаленное_имя*] [-A IP-адрес] [-c] [-n] [-r] [-R] [-RR] [-s] [-S] [интервал]*

1.4. Задание для лабораторной работы

Разработать приложение, выполняющее следующие функции:

1. Отображение MAC-адреса компьютера.
2. Отображение IP-адреса.
3. Отображение имени NetBIOS.
4. Отображение всех рабочих групп, компьютеров в сети и их ресурсов (папок, открытых для общего доступа, принтеров).

1.5. Контрольные вопросы

1. Дайте определение протокола. Что такое стек протоколов ?
2. Назовите известные вам стеки протоколов.
3. Поясните понятие IP-адреса. Для чего и каким образом используется IP-адрес.
4. Поясните понятие шлюза. Что такое «шлюз по умолчанию»?
5. Что такое маска? Для чего используется маска подсети?
6. Какие утилиты для тестирования сетей TCP/IP вы знаете. Поясните основные принципы работы с ними.
7. Дайте описание стека протоколов NETBIOS/SMB.
8. Поясните назначение и использование команд ***arp***, ***netstat***, ***nslookup***.
9. Поясните назначение и использование команд ***hostname***, ***traceroute***, ***route***, ***nbstat***.

2. ПРОТОКОЛЫ ПРИКЛАДНОГО УРОВНЯ

2.1. Простейшие сетевые протоколы прикладного уровня

Протокол Echo. Эхосервис используется для отладки и выполнения измерения времени выполнения запроса между клиентом и сервером. Этот сервис просто возвращает отправителю любые данные, полученные от него. Полное описание протокола можно найти в документе RFC 862.

Один из вариантов эхосервиса определен как использующий для организации соединения протокол TCP. Эхосервер прослушивает соединения TCP на седьмом порту. После организации соединения все полученные данные возвращаются отправителю. Процесс возврата полученных данных отправителю продолжается до тех пор, пока инициатор соединения не разорвет его.

Другой вариант эхосервиса не использует прямых соединений и основан на передаче дейтаграмм UDP. Эхосервер прослушивает седьмой порт (UDP) и возвращает отправителю все принятые через этот порт дейтаграммы.

Протокол Time. Данный протокол предназначен для синхронизации времени. В сети, как правило, всегда работают time-серверы, у которых можно запросить точное время. Следует заметить, что на данный момент для синхронизации времени в глобальных сетях используется более сложный протокол – NTP (Network Time Protocol). В ответ на запрос клиента сервер возвращает время в секундах (32-битное двоичное число), прошедшее с 00:00:00 1 января 1900 г.

Этот протокол может использовать в качестве транспортной службы как UDP-протокол, так и TCP-протокол. Стандартный порт протокола – 37.

Если в качестве транспортной службы используется TCP, взаимодействие осуществляется так:

```
SERVER: прослушивает 37 порт, ожидая соединений
CLIENT: запрашивает соединение с портом 37 сервера
SERVER: посылает время в виде двоичного 32-битного числа
CLIENT: получает время
SERVER: закрывает соединение
CLIENT: закрывает соединение
```

Если сервер по каким-либо причинам не может определить время на своей стороне, он отказывается от соединения, в ответ ничего не посылая.

Если в качестве транспортной службы используется UDP, взаимодействие осуществляется так:

SERVER: прослушивает 37 порт, ожидая соединений

CLIENT: посылает серверу пустой UDP-пакет, номер порта = 37

SERVER: получает пустой UDP-пакет

SERVER: посылает UDP-пакет, содержащий время в виде двоичного 32-битного числа

CLIENT: получает UDP-пакет, содержащий время

Если сервер по каким-либо причинам не может определить время на своей стороне, он отбрасывает полученный пустой UDP-пакет, в ответ ничего не посылая.

Протокол DayTime. Протокол DayTime определен в документе RFC 867. Протокол может использовать в качестве транспортного протоколы UDP и TCP. В случае использования UDP сервер занимает 13-й порт и ожидает поступления дейтаграммы. После получения дейтаграммы он отправляет по обратному адресу пакет, содержащий текущие дату и время в виде текстовой строки. Дополнительно сервер выводит информацию о поступившем запросе в поток стандартного вывода. В случае использования TCP, как и при UDP, сервер занимает 13-й порт и ожидает поступления запросов на установление соединения. После установления соединения сервер отправляет клиенту строку, содержащую дату и время, и закрывает соединение.

Протокол Whols. Протокол Whois – это информационный сервис. Несмотря на то что любой узел может предоставить Whois-сервис, наиболее широко используется ресурс организации InterNIC – http://rs.internic.net. Этот сервер содержит информацию обо всех зарегистрированных DNS-доменах и о большинстве системных администраторов, которые ответственны за системы, подключенные к Internet. К сожалению, не все серверы, предоставляющие Whois-сервис, дают возможность получить полную информацию о DNS-доменах. Описан протокол Whois в документе RFC 954.

Сервер, работающий по протоколу Whois, взаимодействует с клиентом через 43-й порт протокола TCP. Сервер принимает от клиента запрос на соединение, после чего клиент отправляет на сервер запрос длиной в одну строку. Сервер выдает информацию и закрывает соединение. Запросы и отклики передаются

в формате NVT ASCII. Протокол Whois практически идентичен протоколу Finger за исключением того, что запросы и отклики содержат разную информацию.

В Unix-средах для взаимодействия с Whois-сервером широко используется клиент-программа *whois*, однако можно использовать и утилиту *telnet*, вводя команды протокола Whois самостоятельно.

Протокол Finger. Протокол Finger позволяет получить информацию об одном или нескольких пользователях на указанном хосте. Приложение, работающее по этому протоколу, обычно используется для того, чтобы посмотреть, находится ли конкретный пользователь в настоящее время в системе, или чтобы получить имя какого-либо пользователя и послать ему почту. Документируется протокол Finger RFC 1288.

Многие узлы не запускают Finger-сервер по двум причинам. Во-первых, ошибки в программировании в ранних версиях сервера были одной из точек входа «червя» в Internet в 1988 г. Во-вторых, протокол Finger может предоставить подробную информацию о пользователях (имя входа в систему, телефонные номера, время последнего входа и так далее), а эту информацию большинство администраторов считают частной. Раздел 3 RFC 1288 детально описывает аспекты секретности, соответствующие серверам, предоставляющим сервис Finger.

Сервер Finger использует заранее известный порт 79. Клиент осуществляет активное открытие на этот порт и отправляет запрос длиной в одну строку. Сервер обрабатывает запрос, посылает назад отклик и закрывает соединение. Запрос и отклик для протокола Finger представляются в формате NVT ASCII.

Обычно большинство пользователей Unix получают доступ к серверу Finger с использованием клиента *finger*, однако можно воспользоваться *telnet* клиентом. Если запрос клиента состоит из пустой строки (которая в NVT ASCII передается как CR, за которой следует LF), это воспринимается как запрос на информацию обо всех текущих пользователях.

Протокол Rlogin. Rlogin появился в BSD 4.2 и был предназначен для работы в качестве средства, обеспечивающего доступ удаленного терминала к Unix-хосту. Эти особенности позволили разработать протокол, который был проще, чем протокол Telnet, так как он не требовал определения параметров, которые для одной и той же операционной системы известны заранее как для

клиента, так и для сервера. В настоящее время Rlogin используется также и для не-Unix систем.

Спецификация протокола Rlogin содержится в документе RFC 1282. Однако, как и в случае с RFC, посвященными RIP (Routing Information Protocol), RFC 1282 был разработан через несколько лет после начала использования протокола Rlogin. Rlogin использует одно TCP-соединение между клиентом и сервером. После того как TCP-соединение установлено, между клиентом и сервером осуществляется следующая последовательность действий.

Клиент отправляет серверу четыре строки: нулевой байт; имя пользователя на хосте клиента, заканчивающееся нулевым байтом; имя пользователя на хосте сервера, заканчивающееся нулевым байтом; тип терминала пользователя, за которым следует слэш, затем следует скорость терминала, и все это заканчивается нулевым байтом. Необходимо отправить именно два имени пользователя, потому что пользователи могут не иметь одинаковых имен на разных хостах.

Тип терминала передается от клиента к серверу, потому что эта информация необходима для большинства полноэкранных приложений. Скорость терминала передается, так как некоторые приложения работают по-разному в зависимости от скорости.

Сервер отвечает нулевым байтом.

У сервера есть опция, с помощью которой он просит ввести пароль. Ввод пароля осуществляется как обычный обмен данными по Rlogin-соединению – специальные протоколы не применяются. Сервер отправляет клиенту строку, которую клиент отображает на терминале. Чаще всего это строка «Password:». Если клиент не вводит пароль в течение определенного времени (обычно 60 с), сервер закрывает соединение. Все, что вводится в ответ на приглашение сервера ввести пароль, передается в виде открытого текста. Символы введенного пароля посылаются так, как они есть. Каждый, кто может прочитать пакеты в сети, может прочитать любой пароль.

Сервер обычно отправляет запрос клиенту, спрашивая размер окна терминала.

Клиент посылает за один раз серверу один байт, каждый байт сервер отражает эхоотклико м. Функционально все довольно просто: то, что вводит пользователь, отправляется на сервер, а то, что сервер отправляет клиенту, отображается на терминале.

Протокол Telnet. Telnet был разработан для того, чтобы осуществлять соединение между хостами, работающими под управлением любых операционных систем, а также между любыми удаленными терминалами. Его спецификация приведена в RFC 854. В этом документе определяется терминал, который может являться наиболее общим и который называется виртуальным сетевым терминалом (NVT – Network Virtual Terminal). NVT – это воображаемое устройство, находящееся на обоих концах соединения (у клиента и сервера), с помощью которого устанавливается соответствие между их реальными терминалами. Таким образом, операционная система клиента должна определять соответствие между тем типом терминала, за которым работает пользователь, и NVT. В свою очередь сервер должен устанавливать соответствие между NVT и теми типами терминалов, которые он (сервер) поддерживает.

NVT – это символьное устройство с клавиатурой и принтером. Данные, введенные пользователем с клавиатуры, отправляются серверу, а данные, полученные от сервера, поступают на принтер. По умолчанию клиент отражает эхом на принтер все, что ввел пользователь, однако существуют опции, которые позволяют изменить подобное поведение.

2.2. Задания для лабораторной работы

Лабораторная работа выполняется на любом языке программирования, возможно использование библиотек работы с сокетами.

Разработать программное приложение, реализующее функции клиента и сервера одного из предложенных протоколов:

Вариант 1: протокол Echo.
Вариант 2: протокол Rlogin.
Вариант 3: протокол Time.
Вариант 4: протокол DayTime.
Вариант 5: протокол Telnet.
Вариант 6: протокол Whols.
Вариант 7: протокол Finger.

Для вариантов 2 и 5 допускается реализация неполного набора команд.

В лабораторной работе должна быть продемонстрирована совместимость разработанного клиента (сервера) с сервером (клиентом) стороннего производителя.

2.3. Контрольные вопросы

1. Дайте определение понятию порт.
2. Поясните, что такое RFC.
3. Какие основные характеристики протокола Echo.
4. Назовите основные характеристики протокола Time.
5. Приведите основные характеристики протокола DayTime.
6. Перечислите основные характеристики протокола Whois.
7. Приведите основные характеристики протокола Finger.
8. Какие основные характеристики протокола Rlogin.
9. Назовите основные характеристики протокола Telnet.
10. Поясните разницу между протоколами Time и DayTime.
11. Раскройте понятие Network Virtual Terminal.
12. Для чего используется протокол NTP?

2.4. Интернет-протоколы

Протокол HTTP. Протокол пересылки гипертекста (HTTP, Hypertext Transfer Protocol) – это язык, которым клиенты и серверы World Wide Web пользуются для общения между собой. Он по сути является основой Web. Несмотря на то что HTTP в большей степени относится к сфере программирования серверов и клиентов, знание этого протокола важно и для CGI-программирования. Кроме того, иногда HTTP фильтрует информацию и передает ее обратно пользователям. Это происходит, например, когда в окне браузера отображаются коды ошибок сервера.

Все HTTP-транзакции имеют один общий формат. Каждый запрос клиента и ответ сервера состоит из трех частей: строки запроса (ответа), раздела заголовка и тела.

Клиент инициирует транзакцию следующим образом:

1. Устанавливает связь с сервером по назначенному номеру порта (по умол-

чанию – 80). Затем посылает запрос документа, указав HTTP-команду, называемую методом, адрес документа и номер версии HTTP. Например, в запросе

```
GET /index.html HTTP/1.0
```

используется метод GET, которым с помощью версии 1.0 HTTP запрашивается документ *index.html*.

2. Посылает информацию заголовка (необязательную), чтобы сообщить серверу о своей конфигурации и данные о форматах документов, которые он может принимать. Вся информация заголовка указывается построчно, при этом в каждой строке приводятся имя и значение. Например, приведенный ниже заголовок, посланный клиентом, содержит его имя и номер версии клиента, а также информацию о некоторых предпочтительных для клиента типах документов:

```
User-Agent: Mozilla/4.05 (WinNT; 1)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/jpeg, */*
```

Завершается заголовок пустой строкой.

3. Послав запрос и заголовки, клиент может отправить и дополнительные данные. Эти данные используются главным образом теми CGI-программами, которые применяют метод POST. Клиенты также могут использовать их для помещения отредактированной страницы обратно на web-сервер.

Сервер отвечает на запрос клиента следующим образом:

1. Первая часть ответа сервера – строка состояния, содержащая три поля: версию HTTP, код состояния и описание. Поле версии содержит номер версии HTTP, которой данный сервер пользуется для передачи ответа.

Код состояния – это трехразрядное число, обозначающее результат обработки сервером запроса клиента. Описание, следующее за кодом состояния, представляет собой просто понятный для человека текст, поясняющий код состояния. Например, строка состояния

```
HTTP/1.0 200 OK
```

говорит о том, что сервер для ответа использует версию HTTP 1.0. Код состояния 200 означает, что запрос клиента был успешным и затребованные данные будут переданы после заголовков.

2. После строки состояния сервер передает клиенту информацию заголовка, содержащую данные о самом сервере и затребованном документе. Ниже приведен пример заголовка:

```
Date: Fri, 10 April 2009 09:45:28 GMT
```

```
Server: Apache/2.2.11
Last-modified: Fri, 16 Jun 2009 20:54:48 GMT
Content-type: text/html
Content-length: 2482
```

Завершают заголовок две пустые строки.

3. Если запрос клиента успешен, то посылаются затребованные данные. Это может быть копия файла или результат выполнения CGI-программы. Если запрос клиента удовлетворить нельзя, передаются дополнительные данные в виде текстового сообщения для пользователя, в котором приводятся разъяснения причин, по которым сервер не смог выполнить данный запрос и код ошибки.

В HTTP 1.0 за передачей затребованных сервером данных следует разъединение с клиентом, и транзакция считается завершенной, если не передан заголовок *Connection: Keep Alive*.

Запросы клиента разбиваются на три раздела. Первая строка сообщения всегда содержит HTTP-команду, называемую методом; URI, который обозначает запрашиваемый клиентом файл или ресурс; и номер версии HTTP. Следующие строки запроса клиента содержат информацию заголовка. Строка заголовка содержит сведения о клиенте и информационном объекте, который он посылает серверу. Третья часть клиентского запроса представляет собой тело содержимого – собственно данные, посылаемые серверу.

Метод – это HTTP-команда, с которой начинается первая строка запроса клиента. Метод сообщает серверу о цели запроса. Для HTTP определены три основных метода: GET, HEAD и POST. Определены и другие методы, но они не так широко поддерживаются серверами, как три перечисленных. При задании имен методов учитывается регистр, поэтому «GET» и «get» различаются.

Метод GET – это запрос информации, расположенной на сервере по указанному URL. *GET* – наиболее распространенный метод поиска документов для визуализации с помощью браузеров. *GET* примерно означает следующее: «Дайте мне этот файл».Результат запроса *GET* может представлять собой, например, файл, доступный для сервера, результат выполнения программы или CGI-сценария, выходную информацию аппаратного устройства и т. д. Если клиент пользуется в своем запросе методом *GET*, сервер отвечает строкой состояния, заголовками и затребованными данными. Если сервер не может обработать запрос вследствие ошибки или отсутствия полномочий, то, как правило, сервер посылает в информационном разделе ответа текстовое пояснение. Тело

информационного содержимого запроса *GET* всегда пустое. Для идентификации данных, указанных в запросе клиента, файла или программы обычно используется полное имя этого объекта на сервере.

Ниже приведен пример успешного запроса GET на получение файла. Клиент посылает запрос:

```
GET /index.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.05 (WinNT; 1)
Host: www.belarus.fmjd.org
Accept: image/gif, image/x-bitmap, image/jpeg, image/jpeg, */*
```

Сервер отвечает:

```
HTTP/1.0 200 Document follows
Date: Fri, 10 Apr 2009 09:45:58 GMT
Server: Apache/2.2.11
Last-modified: Fri, 10 Apr 2009 21:53:08 GMT
Content-type: text/html
Content-length: 2482
(далее следует тело документа)
```

Метод POST позволяет посылать на сервер данные о запросе клиента. Эти данные направляются в программу обработки данных, к которой сервер имеет доступ (например в CGI-сценарий). Метод POST может использоваться во многих приложениях. Например, его можно применять для передачи входных данных в сетевых службах (таких как телеконференции); программах с интерфейсом в виде командной строки; для аннотирования документов на сервере и выполнения операций в базах данных.

Данные, посылаемые на сервер, находятся в теле содержимого запроса клиента. По завершении обработки запроса POST и заголовков сервер передает тело содержимого в программу, заданную URL. В качестве схемы кодирования с методом POST используется Base64-кодирование, которое позволяет преобразовывать данные форм в список переменных и значений для CGI-обработки.

Приведем пример запроса клиента с использованием метода POST. Клиент посылает на сервер регистрационные данные, введенные в форму:

```
POST /cgi-bin/table.php HTTP/1.0
User-Agent; Mozilla/4.05 (WinNT; 1)
Accept: image/gif, image/x-bitmap, image/jpeg, image/jpeg, */*
Host: www.belarus.fmjd.org
Content-type: application/x-www-form-urlencoded
Content-Length: 20
month=april&date=01&fio=KolbDG
```

Ответ сервера на запрос клиента состоит из трех частей. Первая строка – это строка ответа сервера, которая содержит номер версии HTTP; вторая строка – число, обозначающее состояние запроса; и третья строка – краткое описание состояния. После строки ответа следует информация заголовка и тело содержимого.

Протокол SMTP. Основная задача протокола SMTP (Simple Mail Transfer Protocol) заключается в том, чтобы обеспечивать передачу электронных сообщений (почту). Для работы через протокол SMTP-клиент создает TCP-соединение с сервером через порт 25. Затем клиент и SMTP-сервер обмениваются информацией, пока соединение не будет закрыто или прервано.

Основной процедурой в SMTP является передача почты (Mail Procedure). Далее идут процедуры форвардинга почты (Mail Forwarding), проверка имен почтового ящика и вывод списков почтовых групп. Самой первой процедурой является открытие канала передачи, а последней – его закрытие.

При этом отправитель инициирует соединение и посылает запросы на обслуживание, а получатель отвечает на эти запросы. Фактически отправитель выступает в роли клиента, а получатель – сервера. Канал связи устанавливается непосредственно между отправителем и получателем сообщения. При таком взаимодействии почта достигает абонента в течение нескольких секунд после отправки.

Команды SMTP указывают серверу, какую операцию хочет произвести клиент. Команды состоят из ключевых слов, за которыми следует один или более параметров. Ключевое слово состоит из четырех символов и отделено от аргумента одним или несколькими пробелами. Каждая командная строка заканчивается символами перевода строки (CRLF). Ниже приведен синтаксис всех команд протокола SMTP (SPACE – пробел):

```
HELO <SPACE> <domain> <CRLF>
MAIL <SPACE> FROM:<reverse-path> <CRLF>
RCPT <SPACE> TO:<forward-path> <CRLF>
DATA <CRLF>
RSET <CRLF>
SEND <SPACE> FROM:<reverse-path> <CRLF>
SOML <SPACE> FROM:<reverse-path> <CRLF>
SAML <SPACE> FROM:<reverse-path> <CRLF>
VRFY <SPACE> <string> <CRLF>
EXPN <SPACE> <string> <CRLF>
HELP <SPACE> <string> <CRLF>
```

NOOP <CRLF>
QUIT <CRLF>

Обычный ответ SMTP-сервера состоит из номера ответа, за которым через пробел следует дополнительный текст. Номер ответа служит индикатором состояния сервера.

Первым шагом в процессе ***отправки почты*** является подключение к SMTP-серверу через порт 25. Затем серверу передается команда HELLO и IP-адрес компьютера пользователя (здесь и далее символически обозначены: C: – запрос клиента, S: – ответ сервера):

C: HELLO 94.100.177.1 S: 250 smtp.mail.ru is ready

При отправке почты передаются необходимые для отправки данные (отправитель, получатель и само письмо):

C: MAIL FROM: <отправитель> S: 250 OK
C: RCPT TO: <получатель>
S: 250 OK

Таким образом серверу указывается, что будет передаваться содержание письма (заголовок и тело письма):

C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
<само письмо>
передачу письма необходимо завершить символами CRLF.CRLF
S: 250 OK

Работа завершается путем команды

QUIT: S: QUIT C: 221 smtp.mail.ru is closing transmission channel

Пример:

C: From: test <test@tut.by>
C: To: test1 <test1@mail.ru>
C: Subject: Hello
C: Hello test!
<CRLF.CRLF>
S: 250 OK
S: QUIT
C: 221 smtp.mail.ru is closing transmission channel

Протокол POP3. Простейший протокол для работы пользователя с содержимым своего почтового ящика. Он позволяет только забрать почту из почтового ящика сервера на рабочую станцию клиента и удалить ее из почтового ящика на сервере. Всю дальнейшую обработку почтовое сообщение проходит на компьютере клиента.

POP3-сервер не отвечает за отправку почты, он работает только как универсальный почтовый ящик для группы пользователей. Когда пользователю

необходимо отправить сообщение, он должен установить соединение с каким-либо SMTP-сервером и отправить свое сообщение по SMTP. Этот SMTP-сервер может быть тем же хостом, где работает POP3-сервер, а может располагаться совсем в другом месте.

POP3-сервер, как правило, занимает 110-й TCP-порт сервера, который будет находиться в режиме ожидания входящего соединения. Когда клиент хочет воспользоваться POP3-сервером, он просто устанавливает TCP-соединение через 110 TCP-порт этого хоста. После установления соединения сервер POP3 отправляет подсоединившемуся клиенту приветственное сообщение. После этого клиент и сервер начинают обмен командами и данными. По окончании обмена POP3-канал закрывается.

Команды POP3 состоят из ключевых слов (группы ASCII-символов) с одним или несколькими параметрами, отделяемыми друг от друга символом «пробел» – <SPACE>. Все команды заканчиваются символами «возврат каретки» и «перевод строки» – <CRLF>. Длина ключевых слов не превышает четырех символов, а каждого из аргументов – 40 символов.

Ответы POP3-сервера на команды состоят из строки статус-индикатора, ключевого слова, строки дополнительной информации и символов завершения строки <CRLF>. Длина строки ответа может достигать 512 символов. Строка статус-индикатора принимает два значения: положительное («+OK») и отрицательное («-ERR»). Любой сервер POP3 обязан отправлять строки статус-индикатора в верхнем регистре, тогда как другие команды и данные могут приниматься или отправляться как в нижнем, так и в верхнем регистрах.

Ответы POP3-сервера на отдельные команды могут составлять несколько строк. В этом случае строки разделены символами <CRLF>. Последнюю строку информационной группы завершает строка, состоящая из символа «.» (код 046) и <CRLF>, т. е. последовательность «CRLF.CRLF».

POP3-сессия состоит из нескольких частей. Как только открывается TCP-соединение и POP3-сервер отправляет приветствие, сессия должна быть зарегистрирована – состояние аутентификации (AUTHORIZATION state). Клиент должен зарегистрироваться в POP3-сервере, т. е. ввести свой идентификатор и пароль.

После этого сервер предоставляет клиенту его почтовый ящик и открывает для данного клиента транзакцию – состояние начала транзакции обмена

(TRANSACTION state). На этой стадии клиент может считать и удалить почту из своего почтового ящика.

После того как клиент заканчивает работу (передает команду QUIT), сессия переходит в состояние UPDATE – завершение транзакции. В этом состоянии POP3-сервер закрывает транзакцию данного клиента и закрывает TCP-соединение.

В случае получения неизвестной, неиспользуемой или неправильной команды POP3-сервер должен ответить отрицательным состоянием индикатора.

При открытии TCP-соединения POP3-клиентом POP3-сервер отправляет сообщение приветствия (далее во всех примерах POP3-протокола используются следующие обозначения: C: – клиент, S: – сервер POP3): ***S: +OK POP3 server ready***.

Теперь POP3-сессия находится в состоянии авторизации (AUTHORIZATION) и клиент должен зарегистрировать себя на POP3-сервере. Регистрация может быть выполнена либо с помощью команд USER и PASS – ввод открытого пользовательского идентификатора и пароля (именно этот способ используется чаще), либо командой APOP – авторизация цифровой подписью на базе секретного ключа. Любой POP3-сервер должен поддерживать хотя бы один из механизмов авторизации.

Команда USER имеет следующий формат:

USER name

Аргументом (name) является строка, идентифицирующая почтовый ящик системы. Этот идентификатор должен быть уникальным в данной почтовой системе POP3-сервера. Если ответом на эту команду является строка индикатора «+OK», клиент может отправлять команду PASS – ввод пароля или QUIT – завершить сессию. Если ответом является строка «–ERR», клиент может либо повторить команду USER, либо закрыть сессию. Примеры использования команды:

```
C: USER test
S: -ERR sorry, no mailbox for test here или
C: USER test1
S: +OK test1 is a real hoopy frood
```

Команда PASS используется только после положительного ответа на команду USER:

PASS string

Аргументом команды PASS является строка пароля данного почтового ящика. После получения команды PASS POP3-сервер на основании аргументов команд USER и PASS определяет возможность доступа к заданному почтовому ящику. Если POP3-сервер ответил «+OK», то авторизация клиента прошла успешно и он может работать со своим почтовым ящиком, т. е. сессия переходит в состояние TRANSACTION. Если POP3-сервер ответил «-ERR», то либо был введен неверный пароль, либо не найден указанный почтовый ящик:

```
C: USER test1
S: +OK test1 is a real hoopy frood
C: PASS test
S: +OK test1's maildrop has 2 messages (320 octets)
```

Команда закрытия POP3-сессии: QUIT – отправляется без аргументов и всегда имеет единственный ответ «+OK», например:

```
C: QUIT
S: +OK dewey POP3 server signing off
```

После того как клиент успешно прошел процедуру авторизации в POP3-сервере и POP3-сервер «открыл» определенный почтовый ящик только для использования данным клиентом, POP3-сессия переходит в режим TRANSACTION, и клиент может начать работу со своей почтой.

Команда STAT (без аргументов) используется для просмотра состояния текущего почтового ящика. В ответ на команду STAT POP3-сервер возвращает строку, содержащую количество и общий размер в байтах сообщений, которые клиент может получить с POP3-сервера. Сообщения, помеченные на удаление, не учитываются. Формат ответа: «+OK nn mm», где nn – количество сообщений, mm – их общий объем:

```
C: STAT
S:+OK2 320
```

Команда RETR – используется для передачи клиенту запрашиваемого сообщения:

```
RETR msg
```

Аргумент команды – номер сообщения. Если запрашиваемого сообщения нет, возвращается отрицательный индикатор «–ERR»:

```
C: RETR 1
S: +OK 120 octets
S: <text message>
S:.
```

После получения сообщение, как правило, помечается на удаление из почтового ящика, при этом используется команда DELE:

```
DELE msg
```

Аргумент команды – номер сообщения. Сообщения, помеченные на удаление, реально удаляются только после закрытия транзакции, на стадии UPDATE:

```
C: DELE 1
S: +OK message I deleted
```

Или

```
C: DELE 2
S: -ERR message 2 already deleted
```

Для проверки состояния соединения с POP3-сервером используется команда NOOP. При активном соединении ответом на нее будет положительный индикатор «+ОК»:

```
C: NOOP
S:+OK
```

Ниже приведена стандартная сессия работы с POP3-протоколом:

```
S: <ждем соединения с 110 TCP-портом>
C: <открываем соединение>
S: +OK POP3 server ready
C: USER test1
S: +OK test1 is a real hoopy frood
C: PASS test
S: +OK test1's maildrop has 2 messages (320 octets)
C: STAT S: +OK 2 320 C: LIST
S: +0K 2 messages (320 octets)
S: 1 120
S: 2 200
S:.
C: RETR 1
S: +OK 120 octets
S: <POP3-сервер отсылает первое сообщение>
S:.
C: DELE 1
S: +OK message 1 deleted
C: RETR 2
S: +OK 200 octets
S: < POP3-сервер отсылает второе сообщение >
S:.
C: DELE 2
S: +OK message 2 deleted
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <закрываем соединение>
S: <ждем следующего соединения>
```

Простота протокола POP3, которая послужила росту его популярности вначале, обернулась затем отсутствием гибкости и невозможностью выполнять другие операции управления почтовыми ящиками.

Протокол IMAP. На смену POP3 пришло новое поколение протоколов работы с электронной почтой – протокол IMAP(RFC 3501). Протокол IMAP имеет следующие преимущества по сравнению с протоколом POP3:

– письма хранятся на сервере, а не на клиенте. Возможен доступ к одному и тому же почтовому ящику с разных клиентов. Поддерживается также одновременный доступ нескольких клиентов. В протоколе есть механизмы, с помощью которых клиент может быть проинформирован об изменениях, сделанных другими клиентами;

– поддержка нескольких почтовых ящиков (или папок). Клиент может создавать, удалять и переименовывать почтовые ящики на сервере, а также перемещать письма из одного почтового ящика в другой;

– возможно создание общих папок, к которым могут иметь доступ несколько пользователей;

– информация о состоянии писем хранится на сервере и доступна всем клиентам. Письма могут быть помечены как прочитанные, важные и т. п.;

– поддержка поиска на сервере. Нет необходимости скачивать с сервера множество сообщений для того, чтобы найти одно нужное;

– поддержка онлайн-работы. Клиент может поддерживать с сервером постоянное соединение, при этом сервер в реальном времени информирует клиента об изменениях в почтовых ящиках, в том числе о новых письмах;

– предусмотрен механизм расширения возможностей протокола;

– протокол поддерживает передачу пароля пользователя в зашифрованном виде. Кроме того, IMAP-трафик можно зашифровать с помощью SSL.

Протокол FTP. Протокол FTP (RFC 959) обеспечивает файловый обмен между удаленными пользователями.

Работа FTP на пользовательском уровне состоит из нескольких этапов:

1. Идентификация (ввод имени пользователя и пароля).
2. Выбор каталога.
3. Определение режима обмена (поблочный, поточный, ASCII или двоичный).

4. Выполнение команд обмена (GET, MGET, DIR, MDEL, MPUT или PUT).

5. Завершение процедуры (QUIT или CLOSE). Команды FTP приведены в табл. 2.1.

Таблица 2.1

Команды FTP-протокола

Команда <параметр>	Описание
ABOR	Прервать предыдущую команду FTP и любую передачу данных
LIST <список файлов>	Список файлов или директорий
PASS <пароль>	Пароль на сервере
PORT n1, n2, n3, n4, n5, n6	IP-адрес клиента (nl.n2.n3.n4) и порт (n5*256+n6)
QUIT	Закрыть бюджет на сервере
RETR <имя файла>	Получить (GET) файл
STOR <имя файла>	Положить (PUT) файл
SYST	Сервер возвращает тип системы
TYPE <тип>	Указать тип файла: A для ASCII, I для двоичного
USER <имя пользователя>	Имя пользователя на сервере

Команды и отклики передаются по управляющему соединению между клиентом и сервером в формате NVT ASCII. В конце каждой строки команды или отклика присутствует пара CR, LF. Команды состоят из трех или четырех байт, а именно, из заглавных ASCII символов (некоторые из символов с необязательными аргументами). Клиент может отправить серверу более чем 30 различных FTP-команд.

FTP отклики состоят из 3-циферных значений в формате ASCII и необязательных сообщений, которые следуют за числами. Подобное представление откликов объясняется тем, что программному обеспечению необходимо «посмотреть» только цифровые значения, чтобы понять, что ответил процесс, а дополнительную строку может прочитать человек. Поэтому пользователю достаточно просто прочитать сообщение (причем нет необходимости запоминать все цифровые коды откликов).

Первая и вторая цифра отклика может принимать значения в диапазоне от одного до пяти. В табл. 2.2 представлены группы откликов в виде формулы xyz.

В каждой ячейке «Отклик» в табл. 2.2 указана цифра вместо кода отклика. Семантика первых двух цифр отклика поясняется в соответствующей ячейке «Описание».

Таблица 2.2

Отклики FTP-протокола

Отклик	Описание
lyz	Положительный предварительный отклик. Действие началось, однако необходимо дождаться еще одного отклика перед отправкой следующей команды
2yz	Положительный отклик о завершении. Может быть отправлена новая команда
3yz	Положительный промежуточный отклик. Команда принята, однако необходимо отправить еще одну команду
4yz	Временный отрицательный отклик о завершении. Требуемое действие не произошло, однако ошибка временная, поэтому команду необходимо повторить позже
5yz	Постоянный отрицательный отклик о завершении. Команда не была воспринята и повторять ее не стоит
x0z	Синтаксическая ошибка
x1z	Информация
x2z	Соединения. Отклики имеют отношение либо к управляющему, либо к соединению данных
x3z	Аутентификация и бюджет. Отклик имеет отношение к логированию или командам, связанным с бюджетом
x4z	Не определено
x5z	Состояние файловой системы

Третья цифра дает дополнительное объяснение сообщению об ошибке. Ниже приведены некоторые типичные отклики с возможными объясняющими строками:

- 125 – Соединение данных уже открыто; начало передачи.
- 200 – Команда исполнена.
- 214 – Сообщение о помощи (для пользователя).
- 331 – Имя пользователя принято, требуется пароль.

- 425 – Невозможно открыть соединение данных.
- 452 – Ошибка записи файла.
- 500 – Синтаксическая ошибка (неизвестная команда).
- 501 – Синтаксическая ошибка (неверные аргументы).
- 502 – Нереализованный тип MODE.

Использовать соединение данных можно тремя способами.

1. Отправка файлов от клиента к серверу.
2. Отправка файлов от сервера к клиенту.
3. Отправка списка файлов или директорий от сервера к клиенту.

2.5. Протоколы для защищенной передачи данных

Протокол PPTP (англ. Point-to-Point Tunneling Protocol) – туннельный протокол типа «точка–точка», позволяющий компьютеру устанавливать защищённое соединение с сервером за счет создания специального туннеля в стандартной незащищенной сети. PPTP помещает (инкапсулирует) кадры PPP в IP-пакеты для передачи по глобальной IP-сети. PPTP может также использоваться для организации туннеля между двумя локальными сетями. Протокол использует дополнительное TCP-соединение для обслуживания туннеля. Спецификация протокола была опубликована как «информационный» RFC 2637 в 1999 г. Она не была ратифицирована IETF. Протокол считается менее безопасным, чем другие VPN-протоколы, например IPSec. PPTP-трафик может быть зашифрован с помощью протокола Microsoft Point-to-Point Encryption (MPPE). Для аутентификации клиентов могут использоваться различные механизмы, наиболее безопасные из них – протокол аутентификации Microsoft Challenge Handshake Authentication Protocol (MSCHAP) и протокол аутентификации Extensible Authentication Protocol for Transport Layer Security (EAP-TLS).

Протокол PPPoE. Протокол PPPoE (англ. Point-to-point protocol over Ethernet) – это туннелирующий протокол, который позволяет настраивать (или инкапсулировать) IP-протокол или другие протоколы, которые наслаиваются на протокол PPP, через соединения Ethernet, но с программными возможностями PPP-соединений. Используется для виртуальных «звонков» на соседнюю Ethernet-машину и устанавливает соединение «точка–точка», которое используется для транспортировки IP-пакетов и работает с возможностями протокола PPP. В основном используется xDSL-сервисами. Предоставляет дополнитель-

ные возможности (аутентификация, сжатие, шифрование). Протокол PPPoE разработан организациями UUNET, Redback Network и RouterWare и описан в RFC 2516.

Протокол PPPoE позволяет применять традиционное PPP-ориентированное программное обеспечение для настройки соединения, которое использует не последовательный канал, а пакетно-ориентированную сеть (как Ethernet), чтобы организовать классическое соединение с логином, паролем для интернет-соединений. IP-адрес по другую сторону соединения назначается, только когда PPPoE-соединение открыто, что позволяет использовать динамические IP-адреса.

Работа PPPoE осуществляется следующим образом. Существует Ethernet-среда, т. е. несколько соединенных сетевых карт, которые адресуются MAC-адресами. Заголовки Ethernet-кадров содержат адрес отправителя кадра, адрес получателя кадра и тип кадра. Одну из карт слушает PPPoE-сервер. Клиент посылает широковещательный Ethernet-кадр, на который должен ответить PPPoE-сервер (адрес отправителя кадра – свой MAC-адрес, адрес получателя кадра – FF:FF:FF:FF:FF:FF, тип кадра – PPPoE Discovery). PPPoE-сервер посылает клиенту ответ (адрес отправителя, адрес получателя кадра, тип кадра). Если в сети несколько PPPoE-серверов, то все они посылают ответ. Клиент выбирает подходящий сервер и посылает ему запрос на соединение. Сервер посылает клиенту подтверждение с уникальным идентификатором сессии (все последующие кадры в сессии будут иметь этот идентификатор). Таким образом, между сервером и клиентом создается виртуальный канал, который идентифицируется идентификатором сессии и MAC-адресами клиента и сервера. Затем в этом канале поднимается PPP-соединение, а уже в PPP-пакеты упаковывается IP-трафик.

Протокол SSL. SSL (англ. Secure Sockets Layer – уровень защищённых сокетов) – криптографический протокол, обеспечивающий безопасную передачу данных по сети Интернет. При его использовании создается защищенное соединение между клиентом и сервером. SSL изначально разработан компанией Netscape Communications. Впоследствии на основании протокола SSL 3.0 был разработан и принят стандарт RFC, получивший имя Transport Layer Security (TLS).

Протокол использует шифрование с открытым ключом для подтверждения подлинности передатчика и получателя, поддерживает надёжность передачи данных за счёт использования корректирующих кодов и безопасных хэш-функций.

Протокол SSL состоит из двух уровней. На нижнем уровне многоуровневого транспортного протокола, такого как TCP, он является протоколом записи и используется для инкапсуляции (т. е. формирования пакета) различных протоколов (SSL работает совместно с таким протоколами, как POP3, SMTP и HTTP). Для каждого инкапсулированного протокола SSL обеспечивает условия, при которых сервер и клиент могут подтверждать друг другу свою подлинность, выполнять алгоритмы шифрования и производить обмен криптографическими ключами, прежде чем протокол прикладной программы начнет передавать и получать данные.

Для доступа к web-страницам, защищённым протоколом SSL, в URL вместо обычного префикса HTTP, как правило, применяется префикс HTTPS, указывающий на то, что будет использоваться SSL-соединение. Стандартный TCP-порт для соединения по протоколу HTTPS – 443.

Для работы SSL требуется, чтобы на сервере имелся SSL-сертификат.

Протокол TLS. TLS – криптографический протокол, обеспечивающий защищённую передачу данных между узлами в сети Интернет.

TLS-протокол основан на Netscape SSL-протоколе версии 3.0 и состоит из двух частей – TLS Record Protocol и TLS Handshake Protocol. Различие между SSL 3.0 и TLS 1.0 незначительные, поэтому далее в тексте термин «SSL» будет относиться к ним обоим.

TLS предоставляет возможности аутентификации и безопасной передачи данных через Интернет с использованием криптографических средств. Часто происходит лишь аутентификация сервера, в то время как клиент остается неаутентифицированным. Для взаимной аутентификации каждая из сторон должна поддерживать инфраструктуру открытого ключа (PKI), которая позволяет защитить клиент-серверные приложения от перехвата сообщений, редактирования существующих сообщений и создания поддельных.

SSL включает в себя три основные фазы:

- диалог между сторонами, целью которого является выбор алгоритма шифрования;

• обмен ключами на основе криптосистем с открытым ключом или аутентификация на основе сертификатов;

• передача данных, шифруемых при помощи симметричных алгоритмов шифрования.

Клиент и сервер, работающие по протоколу TLS, устанавливают соединение, используя процедуру handshake («рукопожатие»). В течение этой процедуры клиент и сервер принимают соглашение относительно параметров, используемых для установления защищенного соединения.

Последовательность действий при установлении TLS-соединения:

• клиент подключается к TLS-поддерживаемому серверу и запрашивает защищенное соединение;

• клиент предоставляет список поддерживаемых алгоритмов шифрования и хеш-функций;

• сервер выбирает из списка, предоставленного клиентом, наиболее устойчивые алгоритмы, которые также поддерживаются сервером, и сообщает о своем выборе клиенту;

• сервер отправляет клиенту цифровой сертификат для собственной идентификации. Обычно цифровой сертификат содержит имя сервера, имя доверенного центра сертификации и открытый ключ сервера;

• клиент может связаться с сервером доверенного центра сертификации и подтвердить аутентичность переданного сертификата до начала передачи данных;

• для того чтобы сгенерировать сеансовый ключ для защищенного соединения, клиент шифрует случайно сгенерированную цифровую последовательность открытым ключом сервера и посылает результат на сервер с учетом специфики алгоритма асимметричного шифрования, используемого для установления соединения; только сервер может расшифровать полученную последовательность с помощью своего закрытого ключа.

Рассмотрим процедуру handshake более подробно. Согласно протоколу TLS приложения обмениваются записями, инкапсулирующими (хранящими внутри себя) информацию, которая должна быть передана. Каждая из записей может быть сжата, дополнена, зашифрована или идентифицирована MAC в зависимости от текущего состояния соединения (состояния протокола). Каждая запись в TLS содержит следующие поля: content type (определяет тип содержи-

мого записи), поле, указывающее длину пакета, и поле, указывающее версию протокола TLS.

Когда соединение только устанавливается, взаимодействие идет по протоколу TLS handshake, content type которого 22.

Ниже описан простой пример установления соединения.

1. Клиент посылает сообщение *ClientHello*, указывая последнюю версию поддерживаемого TLS-протокола, случайное число и список поддерживаемых методов шифрования и сжатия, подходящих для работы с TLS.

2. Сервер отвечает сообщением *ServerHello*, содержащим: выбранную сервером версию протокола; случайное число, посланное клиентом; подходящий алгоритм шифрования и сжатия из списка предоставленного клиентом.

3. Сервер посылает сообщение *Certificate*, которое содержит цифровой сертификат сервера (в зависимости от алгоритма шифрования этот этап может быть пропущен).

4. Сервер может запросить сертификат у клиента, в таком случае соединение будет взаимно аутентифицировано.

5. Сервер отсылает сообщение *ServerHelloDone*, идентифицирующее окончание handshake.

6. Клиент отвечает сообщением *ClientKeyExchange*, которое содержит открытый PreMasterSecret-ключ или ничего (опять же зависит от алгоритма шифрования).

7. Клиент и сервер, используя PreMasterSecret-ключ и случайно сгенерированные числа, вычисляют общий секретный ключ. Вся остальная информация о ключе будет получена из общего секретного ключа (и сгенерированных клиентом, и сгенерированных сервером случайных значений).

8. Клиент посылает *ChangeCipherSpec* сообщение, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе handshake алгоритмом с использованием общего секретного ключа.

9. Клиент посылает сообщение *Finished*, которое содержит хеш и MAC (код аутентификации сообщения), сгенерированные на основе предыдущих сообщений handshake.

10. Сервер пытается расшифровать *Finished*-сообщение клиента и проверить хеш и MAC. Если процесс расшифровки или проверки не удается, handshake считается неудавшимся и соединение должно быть оборвано.

11. Сервер посылает *ChangeCipherSpec*-сообщение и зашифрованное *Finished*-сообщение, в свою очередь клиент тоже выполняет расшифровку и проверку.

С этого момента handshake считается завершенным, протокол – установленным. Все последующее содержимое пакетов идет с типом 23, а все данные будут зашифрованы.

Алгоритмы, использующиеся в TLS. В текущей версии протокола доступны следующие алгоритмы:

- для обмена ключами и проверки их подлинности применяются комбинации алгоритмов: RSA (асимметричный шифр), алгоритм Диффи–Хелмана (безопасный обмен ключами), DSA (алгоритм цифровой подписи) и алгоритмы технологии Fortezza;
- для симметричного шифрования: RC2, RC4, IDEA, DES, Triple DES или AES;
- для хеш-функций: MD5 или SHA.

Алгоритмы могут дополняться в зависимости от версии протокола.

Протокол SSH. SSH (англ. Secure Shell – «безопасная оболочка») – сетевой протокол прикладного уровня, позволяющий производить удаленное управление операционной системой и туннелирование TCP-соединений (например для передачи файлов).

Первая версия протокола, SSH-1, была разработана в 1995 г. исследователем Тату Улёнен из Технологического университета Хельсинки, Финляндия. SSH-1 был написан для обеспечения большей конфиденциальности, чем протоколы Rlogin, Telnet и Rsh. В 1996 г. была разработана более безопасная версия протокола, SSH-2, несовместимая с SSH-1. Протокол приобрел ещё большую популярность, и к 2000 г. его использовало уже порядка двух миллионов пользователей. В 2006 г. протокол был утвержден рабочей группой IETF в качестве Интернет-стандарта.

Протокол сходен по функциональности с протоколами Telnet и Rlogin, но в отличие от них шифрует весь трафик, включая и передаваемые пароли. SSH-клиенты и SSH-серверы имеются для большинства сетевых операционных систем. SSH допускает выбор различных алгоритмов шифрования.

Протокол SSH позволяет безопасно передавать в незащищенной среде практически любой другой сетевой протокол, таким образом можно не только

удаленно работать на компьютере через командную оболочку, но и передавать по шифрованному каналу звуковой поток или видео (например с web-камеры). Также SSH может использовать сжатие передаваемых данных для последующего их шифрования, что удобно, например, для удаленного запуска клиентов XWindow System.

Большинство хостинг-провайдеров за определенную плату предоставляют клиентам доступ к их домашнему каталогу по SSH. Это может быть удобно как для работы в командной строке, так и для удаленного запуска программ (в том числе графических приложений).

Распространены две реализации SSH: собственническая коммерческая и бесплатная свободная. Свободная реализация называется OpenSSH. К 2006 г. 80 % компьютеров сети Интернет использовало именно OpenSSH. Собственническая реализация разрабатывается организацией SSH Inc., она бесплатна для некоммерческого использования. Эти реализации содержат практически одинаковый набор команд.

Существуют две версии протокола SSH: SSH-1 и SSH-2. В первой версии протокола есть существенные недостатки, поэтому в настоящее время SSH-1 практически нигде не применяется.

Многие взломщики сканируют сеть в поиске открытого порта SSH, особенно адреса хостинг-провайдеров, обычно пытаясь подобрать пароль суперпользователя.

Протокол SSH-2 в отличие от протокола Telnet устойчив к атакам прослушивания трафика («снифинг»), т. е. прослушивание трафика ничего не дает злоумышленнику. Но неустойчив к атакам «man-in-middle», когда атакующий способен читать и видоизменять по своей воле сообщения, которыми обмениваются корреспонденты, причём ни один из последних не может догадаться о его присутствии в канале. Протокол SSH-2 также устойчив к атакам путем присоединения «посредине» (англ. session hijacking) – невозможно включиться в или перехватить уже установленную сессию.

Для предотвращения атак «man-in-middle» при подключении к хосту, ключ которого еще не известен клиенту, клиентское ПО показывает пользователю «слепок ключа» (key fingerprint). Рекомендуется тщательно сверять показываемый клиентским программным обеспечением «слепок ключа» (key fingerprint)

со слепком ключа сервера, желательно, полученным по надежным каналам связи или лично.

Поддержка SSH реализована во всех UNIX-подобных системах, и на большинстве из них в числе стандартных утилит присутствуют SSH-клиент и SSH-сервер. Существует множество реализаций SSH-клиентов и для не-UNIX операционных систем.

SSH – это протокол прикладного уровня (или уровня приложений). SSH-сервер обычно прослушивает соединения на TCP-порту 22. Спецификация протокола SSH-2 содержится в RFC 4251. Для аутентификации сервера SSH использует алгоритм Диффи–Хеллмана для аутентификации клиента – шифрование с открытым ключом (оно сравнительно медленное), для шифрования передаваемых данных – более быстрое симметричное шифрование. Среди алгоритмов шифрования с открытым ключом чаще всего используются RSA и DSA, из симметричных алгоритмов – AES, BlowFish и 3DES. Целостность переданных данных проверяется с помощью CRC32 в SSH1 или HMAC-SHA1/HMAC-MD5 в SSH2.

Для сжатия шифруемых данных может использоваться алгоритм LempelZiv, который обеспечивает такой же уровень сжатия, что и архиватор ZIP. Сжатие SSH включается лишь по запросу клиента и на практике используется редко.

2.6. Задания для лабораторной работы

Вариант 1. Разработать программу, реализующую функции HTTP-сервера версии 1.0. В обязательном порядке должны поддерживаться следующие виды запросов: GET, POST, HEAD, а также наиболее распространенные коды ответов. Сервер конфигурируется на определенный каталог, где расположены html-файлы и другие подкаталоги. В главном окне сервера расположена текстовая область, в которой отображается весь протокол общения HTTP-клиента с HTTP-сервером.

Тестирование и подача программы-сервера производится при помощи web-браузера.

Вариант 2. Разработать программу, реализующую функции HTTP-клиента версии 1.0. В обязательном порядке должны поддерживаться следующие виды запросов: GET, POST, HEAD, а также наиболее распространенные коды ответов. Отображение полученных данных в форматированном виде необязательно

(можно в виде plain text). В окне клиента должна быть расположена текстовая область, в которой отображается весь протокол общения HTTP-клиента с HTTP-сервером. Тестирование и подача HTTP-клиента производится при помощи запроса к реальному web-серверу, расположенному в Интернет или установленному в локальной сети.

Вариант 3. Разработать программу, реализующую функции POP3-сервера. В главном окне сервера расположена текстовая область, в которой отображается весь протокол общения клиента с сервером.

Тестирование и подача программы-сервера производится при помощи любого стандартного почтового клиента.

Вариант 4. Разработать программу, реализующую функции POP3-клиента. В главном окне клиента расположена текстовая область, в которой отображается весь протокол общения клиента с сервером.

Тестирование и подача программы-клиента производится при помощи любого стандартного почтового сервера, расположенного в сети Интернет или локальной сети. В качестве сервера может использоваться программа, написанная к варианту 3.

Вариант 5. Разработать программу, реализующую функции SMTP-сервера. В главном окне сервера расположена текстовая область, в которой отображается весь протокол общения клиента с сервером.

Тестирование и подача программы-сервера производится при помощи любого стандартного почтового клиента.

Вариант 6. Разработать программу, реализующую функции SMTP-клиента. В главном окне клиента расположена текстовая область, в которой отображается весь протокол общения клиента с сервером.

Тестирование и подача программы-клиента производится при помощи любого стандартного почтового сервера, расположенного в сети Интернет или локальной сети. В качестве сервера может использоваться программа, написанная к варианту 5.

Вариант 7. Разработать программу, реализующую функции FTP-сервера. В главном окне сервера расположена текстовая область, в которой отображается весь протокол общения клиента с сервером.

Тестирование и подача программы-сервера производится при помощи любого стандартного FTP-клиента.

Вариант 8. Разработать программу, реализующую функции FTP-клиента. В главном окне клиента расположена текстовая область, в которой отображается весь протокол общения клиента с сервером.

Тестирование и подача программы-клиента производится при помощи любого стандартного FTP сервера, расположенного в сети Интернет или локальной сети. В качестве сервера может использоваться программа, написанная к варианту 7.

2.7. Контрольные вопросы

1. Приведите основные технические характеристики протокола HTTP.
2. Каковы основные технические характеристики протокола FTP.
3. Приведите основные технические характеристики протокола SMTP.
4. Приведите основные технические характеристики протокола POP3.
5. Перечислите основные технические характеристики протокола PPTP.
6. Приведите основные технические характеристики протокола PPoE.
7. Перечислите основные технические характеристики протокола SSL.
8. Назовите основные технические характеристики протокола TLS.
9. Приведите основные технические характеристики протокола SSH.

3. СЕРВИСНО-ОРИЕНТИРОВАННЫЕ АРХИТЕКТУРЫ

3.1. Сервисно-ориентированная архитектура. Определение web-сервиса

Сервисно-ориентированная архитектура (СОА) – это подход к организации и использованию распределенных информационных ресурсов (таких как приложения и данные), находящихся в сфере ответственности разных владельцев, для достижения желаемых результатов потребителем, которым может быть конечный пользователь или другое приложение.

Концепция проектирования программного обеспечения как службы, независимой от пользовательского интерфейса, была одним из подходов к разработке программного обеспечения на протяжении почти десятилетия. Ранние попытки в распределенных вычислениях привели к пониманию того, что бизнес-логика приложения необязательно связана с логикой представления. Она может быть полностью отдельной и предлагаться в виде службы с определенным API (программный интерфейс приложения).

В основе технологии СОА лежат принципы многократного использования функциональных элементов, ликвидации дублирования функциональности в программном обеспечении, унификации типовых операционных процессов, обеспечения перевода операционной модели компании на централизованные процессы и функциональную организацию на основе промышленной платформы интеграции.

Компоненты программы могут быть распределены по разным узлам сети и предлагаются как независимые, слабосвязанные, заменяемые сервисы-приложения. Программные комплексы, разработанные в соответствии с СОА, часто реализуются как набор web-служб, интегрированных при помощи известных стандартных протоколов.

Web-служба, или ***web-сервис*** (англ. web service), – это программная система, идентифицируемая строкой URI (Uniform Resource Identifier), общедоступные интерфейсы которой определены на языке XML (eXtensible Markup Language). Описание этой программной системы может быть найдено другими программными системами, которые могут взаимодействовать с ней согласно этому описанию посредством сообщений, основанных на XML и передаваемых

с помощью интернет-протоколов. Web-служба является единицей модульности при использовании сервисно-ориентированной архитектуры приложения.

Один из способов определения web-служб – это перечисление стандартов, по которым можно судить, является ли конкретное приложение web-службой. Это сложнее, чем может показаться на первый взгляд, так как для web-служб нет ни одной спецификации стандартов. Тем не менее стандартное определение включает в себя следующее:

- XML – для представления данных в независимом стиле. XML используется не только для определения передаваемых данных, специфических для web-служб, но также в определениях данных в других протоколах.
- Протокол HTTP – для передачи по Интернет или внутри Интранет. Он предоставляет механизм как для передачи данных, так и для обеспечения связей типа запрос/ответ.
- Simple Object Access Protocol (SOAP), который предоставляет механизм определения запросов/ответов и допускает диалог в формате Remote Procedure Call (RPC).
- Universal Description, Discovery and Integration (UDDI), которая предоставляет средства локализации служб через общий реестр поставщиков и служб.
- Web Services Description Language (WSDL), который специфицирует правило записи информации, необходимой для активизации конкретной службы. WSDL и UDDI в некоторой степени дополняют друг друга, а также перекрываются в некоторых областях.

3.2. Web-сервисы, реализующие RPC-ориентированное взаимодействие

Один из способов создания web-служб – создание с помощью модели RPC (Remote Procedure Call). Клиент делает вызов удаленного метода web-сервиса, который обрабатывается на сервере. На макроуровне клиент передает SOAP-запрос и получает SOAP-ответ (рис. 3.1). В RPC-SOAP-программировании клиент и сервер должны придерживаться строго определенной программной модели – клиент вызывает метод с возможными аргументами, а сервер в ответ возвращает одно значение. RPC-метод разработки SOAP эффективен, если проблему можно легко смоделировать с помощью вызовов методов или если существует функциональный API для решения поставленной задачи.

Для приложений, которые манипулируют документами XML с помощью XSLT, документ XML более предпочтителен, чем структуры данных Java, так как процессор XSLT может автоматически преобразовывать этот документ.

В качестве «исполнительной системы SOAP» используется большой набор программных средств. Для выполнения лабораторной рекомендуется использовать проект Axis компании Apache Software Foundation .

Axis (Apache eXtensible Interaction System) – система для конструирования SOAP-процессоров, таких, как клиенты, серверы, 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;

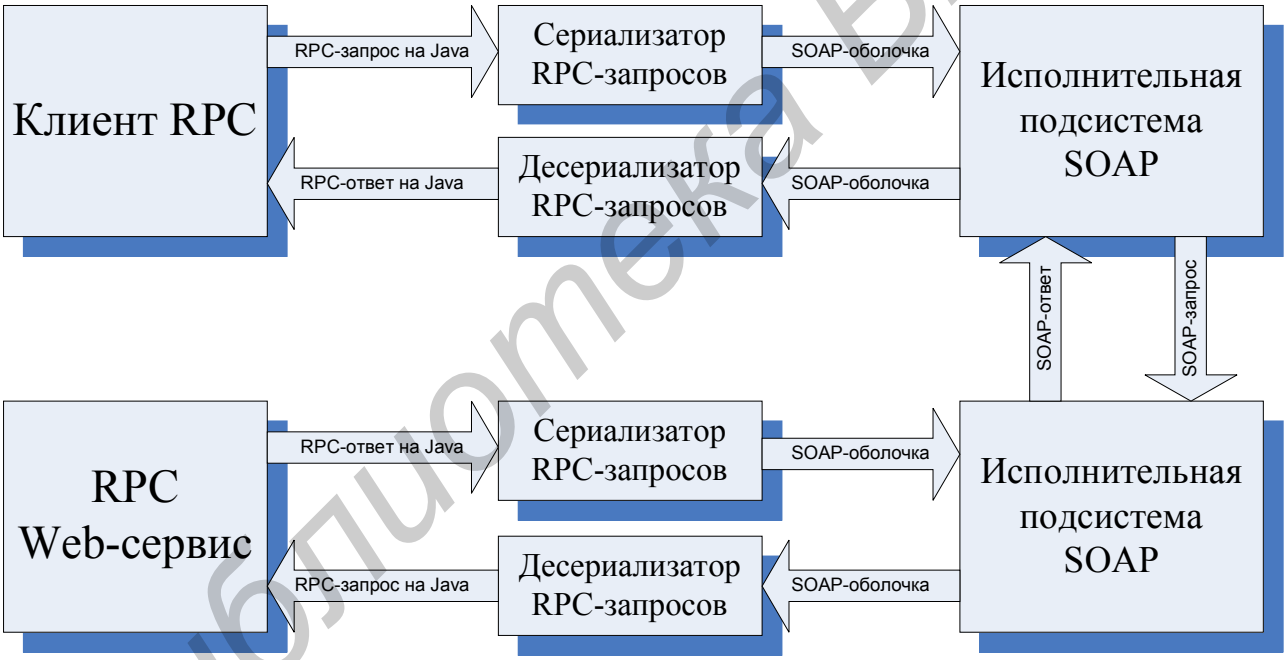


Рис. 3.1. Схема взаимодействия при RPC-SOAP-программировании

- 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 получает 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 логирования, обработки WSDL. Для работы Axis требуется наличие XML-парсера, совместимого с JAXP 1.1 (Java API for XML Processing), предпочтительно Xerces.

В качестве примера использования Axis 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 (подойдет и более ранняя версия) и Tomcat 5.0 (или более поздней версии). 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 страница Axis (рис. 3.2).

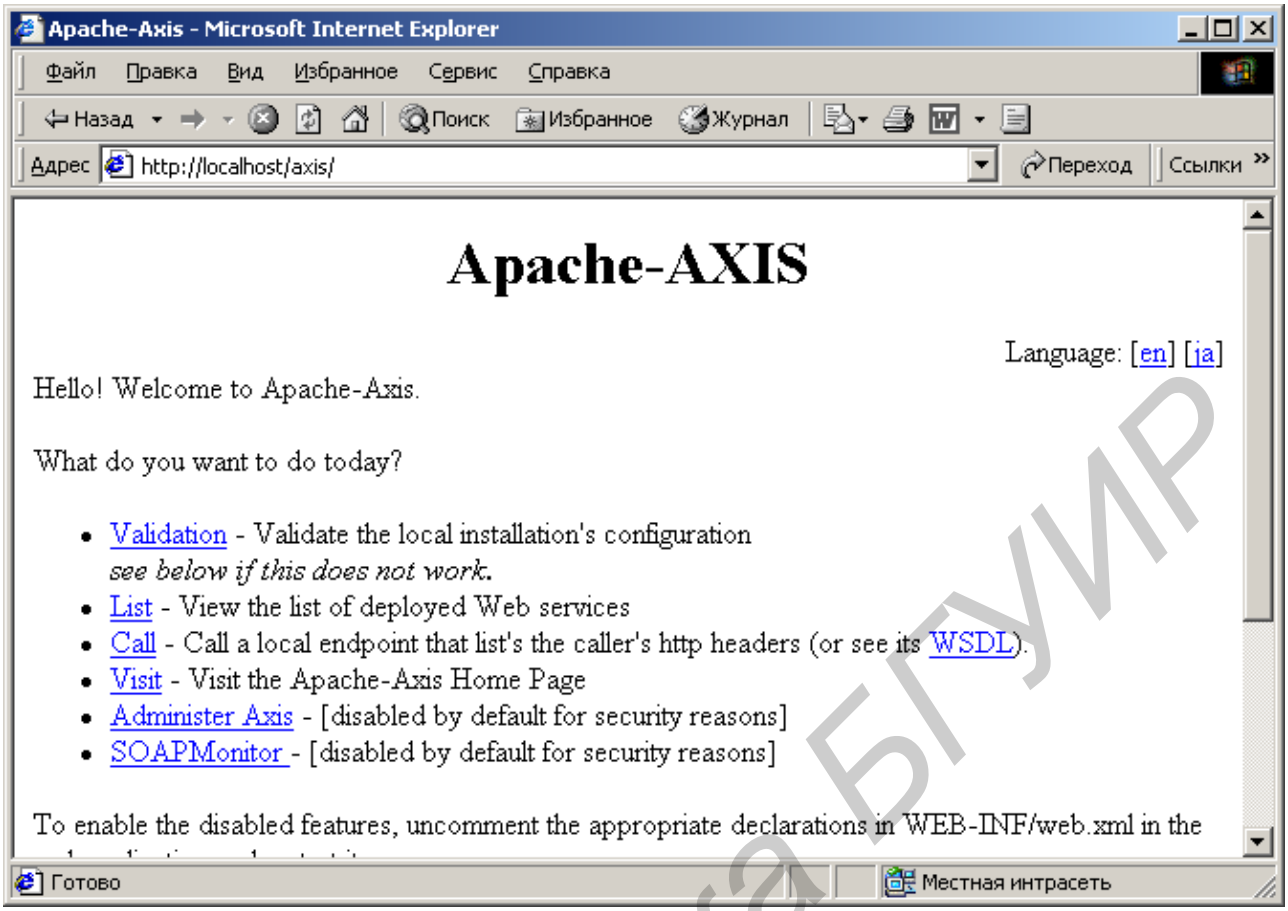


Рис. 3.2. Стартовая страница Axis

По ссылке ***Validation*** можно проверить, все ли библиотеки нашел Axis. Идеальным является наличие сообщения «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 находиться 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 каждой строчке которого записана пара: фамилия – телефон через пробел. В методе 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. Массив искомых слов возвращает функция split:

```
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 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.

Рассмотрим 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, так как был неправильно сформированным URL произведен вызов несуществующего сервиса.

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

Строка *endpoint* является строкой URL, по которому размещен сервис. Обратите внимание на название хоста, номер порта и название самого сервиса, которые соответствуют вашей реализации лабораторной работы:

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

Следующие строки создают объекты сервиса, вызова требуемого сервиса. Устанавливается целевой адрес сервиса через класс URL:

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

На экран пользователя выводится меню. Затем запускаем цикл, который останавливается, как только пользователь нажмет на «3», что в реализуемом 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 с консоли строку с телефоном:

```
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 «массив 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' внутри, затем – клиентский класс. Чтобы компилятор мог использовать подключенные библиотеки, их jar-файлы должны быть прописаны в переменной среде окружения CLASSPATH.

3.3. Web-сервисы, реализующие документно-ориентированное взаимодействие

В случае документно-ориентированного программирования клиент передает web-сервису документ XML, который обрабатывается на сервере. Здесь также (как и в случае RPC-взаимодействия) на макроуровне клиент передает запрос SOAP и получает ответ SOAP (рис. 3.3).

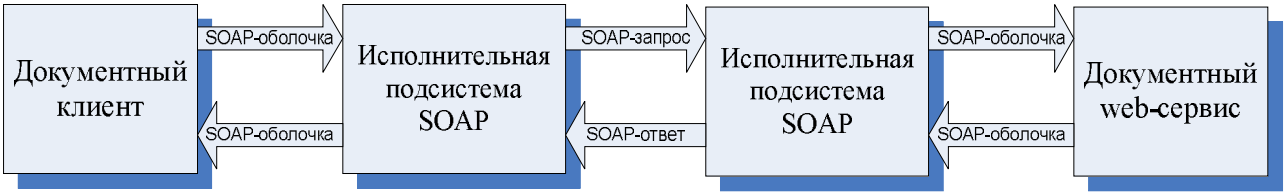


Рис. 3.3. Схема взаимодействия при документно-ориентированном программировании web-сервисов

Чтобы лучше увидеть различия между двумя подходами, рассмотрим принцип их действия подробнее.

Основными различиями являются сериализация/десериализация и семантика.

В документно-ориентированном SOAP-программировании клиенту не нужно сериализовать вызов Java и его аргументы в документ XML. И наоборот, серверу не нужно десериализовать документ XML в вызов Java и типы данных. В документно-ориентированном программировании в обмене участвует документ XML, и значение каждого элемента интерпретируется участвующими во взаимодействии сторонами. Вдобавок к этому клиент и web-сервис могут объединить в запрос и ответ несколько документов. Так как процесс сериализации/десериализации отсутствует, приложения, разработанные посредством использования документно-ориентированного программирования, должны быть более быстрыми, чем их RPC-аналоги. Тем не менее практика показывает, что приложения, использующие документно-ориентированное программирование, необязательно будут работать быстрее, чем их RPC-аналоги; это можно аргументировать тем, что при документно-ориентированном программировании сериализация зависит от разработчика, так как в некоторый момент программисту может понадобиться преобразовать данные Java в XML и наоборот. А также, задействованные документы XML потенциально могут быть намного больше, чем простые типы при взаимодействии «запрос-ответ».

Документно-ориентированное SOAP-программирование используется, когда необходимо осуществлять обмен данными между двумя и более сторонами. Такое использование особенно уместно в случае, когда у программиста уже есть документ XML, представляющий данные, так как можно обмениваться самим документом XML в исходном виде без необходимости преобразования его структуры данных Java. Отсутствие шагов сериализации/десериализации упрощает разработку и ускоряет обработку данных.

В качестве примера использования Axis рассмотрим задачу, поставленную в подразд. 3.2.

Для реализации сервиса будем использовать сервер Tomcat (для развертывания web-служб Axis2, входящих в комплект поставки среды IDE NetBeans версии 6.8) и собственно саму среду. Параметры настройки IDE NetBeans для работы с Axis2 доступны на сайте разработчиков NetBeans.

В среде IDE NetBeans web-службу Axis2 можно создать из класса Java только для проекта приложения на Java или библиотеки Java. Следующий пример будет посвящен созданию проекта библиотеки Java, созданию web-службы Axis2 и развертыванию web-службы Axis2 на сервере.

Создание и развертывание web-службы Axis2:

1. Щелкните по значку ***New Project*** или выберите в меню ***File -> New Project***. Откроется мастер создания проекта. Выберите проект библиотеки классов Java в категории ***Java***. Нажмите кнопку ***Next***.

2. Присвойте проекту имя ***AxisPhone***. Убедитесь, что используется правильное имя и местоположение папки проекта. Решение о необходимости совместного использования проекта принимается разработчиком. Нажмите кнопку ***Finish*** для создания проекта.

3. Щелкните правой кнопкой мыши по узлу проекта. Откроется контекстное меню. Выберите в контекстном меню ***New -> Other***. Откроется мастер создания файла. Выберите ***Axis2 Service from Java*** из категории ***Web Services*** и нажмите кнопку ***Next*** (рис. 3.4).

4. Откроется страница мастера создания файла ***Service Type Selection***. В проекте отсутствуют классы Java, поэтому выберите ***Create an Empty Web Service***. Если класс Java уже разработан, можно установить переключатель ***Create a Web Service from an Existing Java Class***.

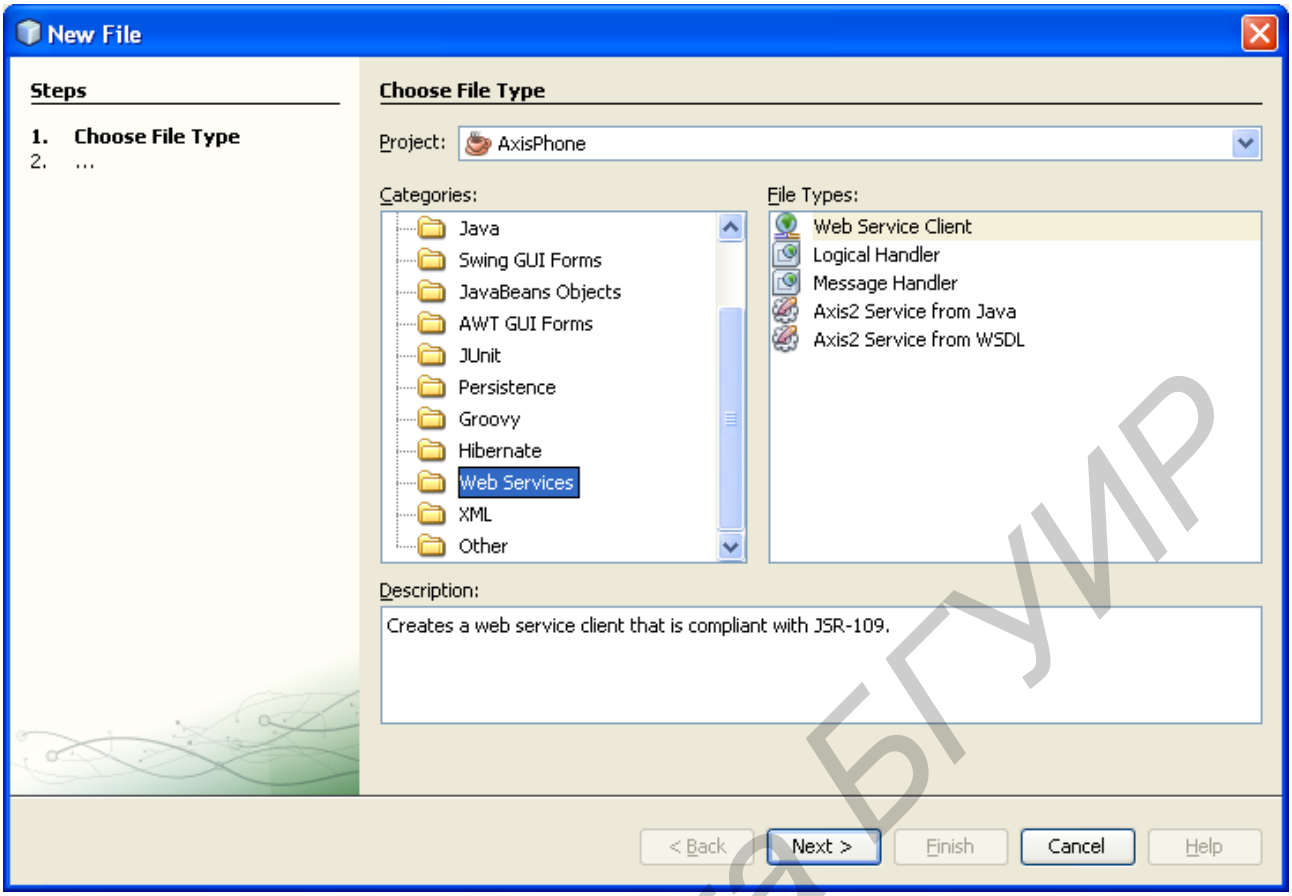


Рис. 3.4. Окно создания шаблона web-сервиса

Если требуется изменить WSDL web-службы, например, добавить или изменить пространства имен, следует установить переключатель ***Generate a WSDL from Java Source Code***. Изменение WSDL выходит за рамки данного примера, поэтому устанавливать этот переключатель здесь не требуется. Мастер должен выглядеть следующим образом (рис. 3.5).

5. Нажмите кнопку ***Next***. Откроется страница ***Name and Location***. Назовите класс Java ***AxisPhone***. Присвойте пакету имя ***axisphone***. Параметр ***Generate Sample Method*** должен быть установлен. В результате в классе Java создается метод, возвращающий строку «***Hello, World***».

6. Нажмите кнопку ***Finish***. В среде IDE создастся класс ***AxisPhone.java*** в пакете исходного кода ***AxisPhone*** и web-служба Axis2 ***AxisPhone***, на которую отображен этот класс Java. Можно заметить, что и в классе Java, и в web-службе Axis2 имеется операция ***hello:String***, показанная на вкладке ***Navigator*** и в виде узла в web-службе Axis2 соответственно.

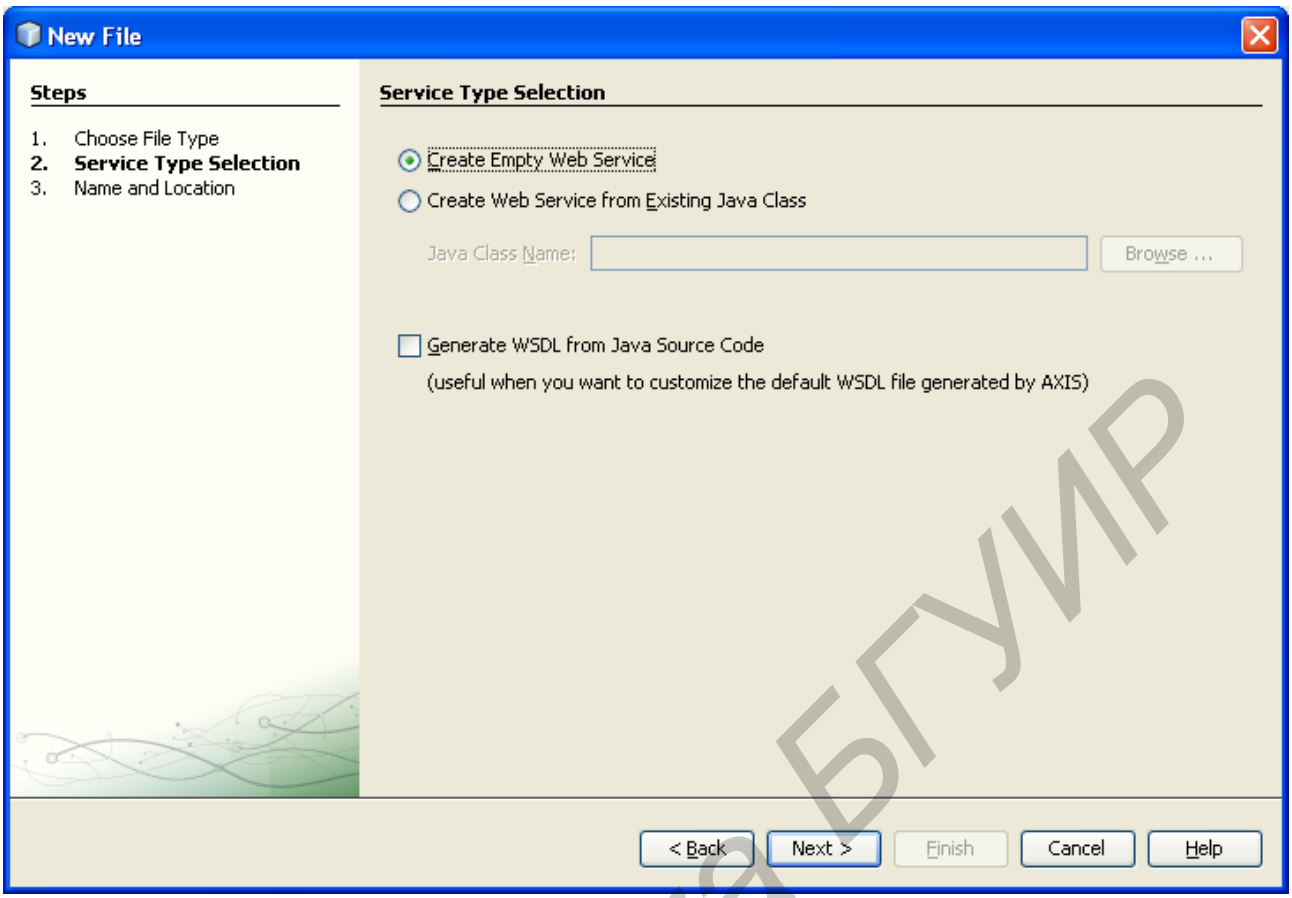


Рис. 3.5. Окно создания скелета web-сервиса

Рассмотрим развертывание и тестирование web-службы Axis2.

Созданную web-службу Axis2 необходимо развернуть на сервере. Фактически развертывание на сервере сводится к двум действиям:

1. Копирование web-службы Axis2 в файл ***axis2.war*** используемый сервером. Это действие может выполняться при неработающем сервере.

2. Повторное развертывание на сервере обновленного файла ***axis2.war***.

Если развертывание выполняется на сервере Tomcat с включенным параметром ***Use Tomcat Manager for Deployment,*** обновленный файл ***axis2.war*** автоматически повторно развертывается на сервере. В противном случае повторное развертывание файла ***axis2.war*** выполняется вручную с помощью средств сервера приложений. В данном учебном проекте предполагается настройка параметров Axis2, так что повторное развертывание выполняется автоматически.

Развертывание web-службы Axis2 на сервере:

1. Щелкните по узлу web-службы правой кнопкой мыши. Откроется контекстное меню. Выберите ***Deploy to Server***. Файл AAR Axis2 компилируется и копируется в файл ***axis2.war***, используемый сервером приложений.

2. Если включено автоматическое развертывание, web-служба развертывается на сервере. Если сервер не запущен, автоматическое развертывание выполняется при запуске.

3. Для тестирования службы разверните узел web-службы и соответствующие операции (рис. 3.6). Щелкните правой кнопкой мыши по узлу ***hello:String*** и выберите ***Test Operation in Browser***.

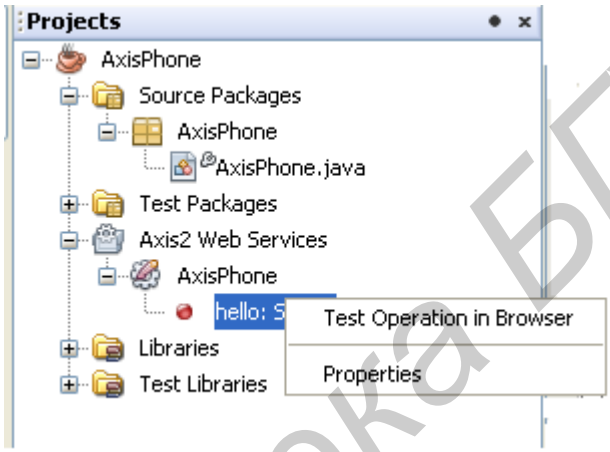


Рис. 3.6. Тестирование шаблона web-сервиса

4. В результате открывается обозреватель с тестовыми значениями переменных (рис. 3.7). Тестовое значение добавляется к URL-адресу.

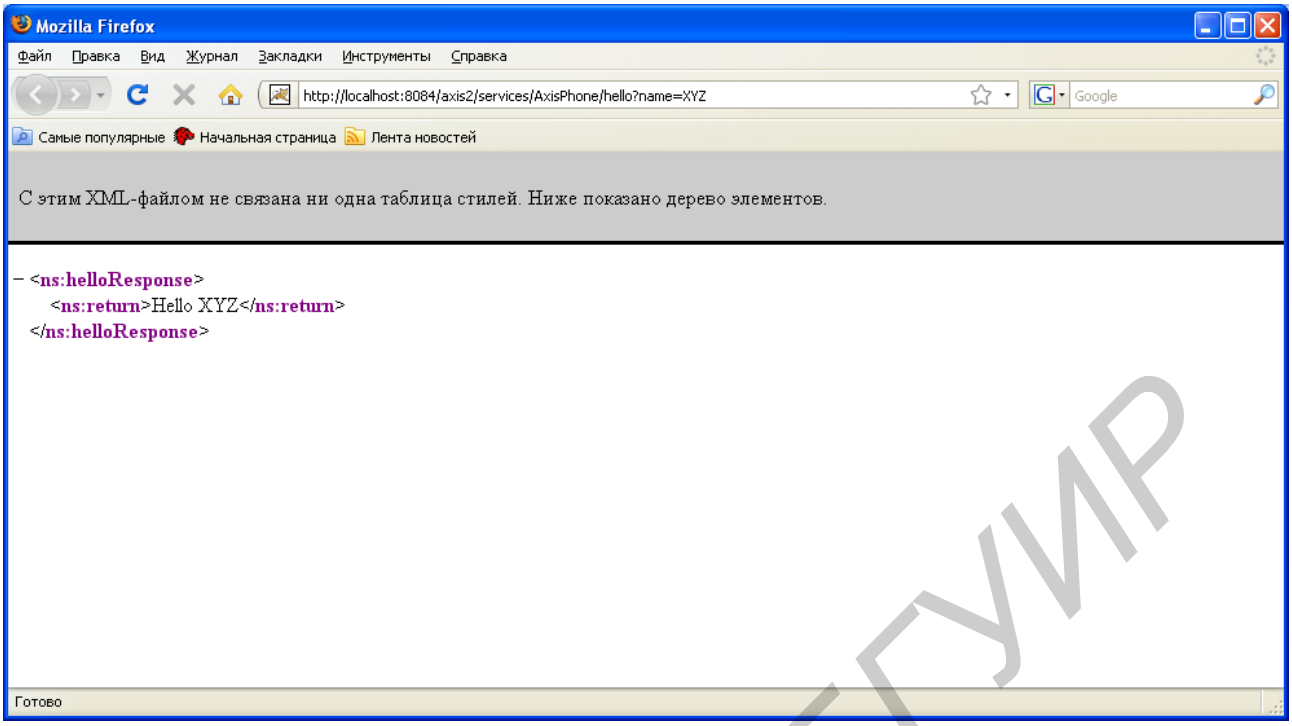


Рис. 3.7. Результат вызова тестового метода web-сервиса

5. Измените значение переменной в URL-адресе и нажмите клавишу ***Enter***. При этом изменяется и результат тестирования (рис. 3.8).

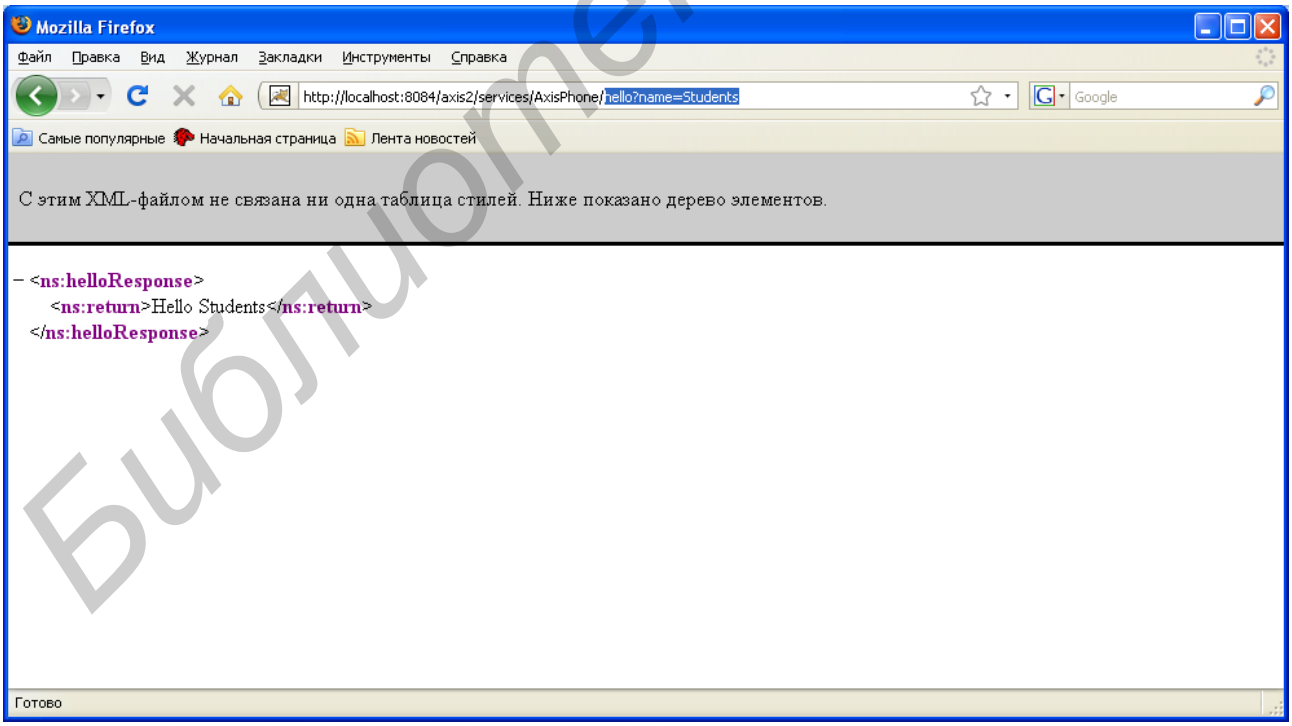


Рис. 3.8. Изменение параметров web-сервиса через строку браузера

Для изменения операций web-службы необходимо изменить файл Java в проекте. Одновременно с файлом изменятся и операции web-службы. Добавьте к ***AxisPhone.java*** два метода – ***phone_name*** и ***name_phone*** :

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class AxisPhone {
public String hello(String name) {
return "Hello "+name;
}
public String phone_name(String phone) {
String name = 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[0].equals(phone)) {
name = str[1];
break;
} else {
name = "NO SUCH TELEPHONE";
}
}
} catch (IOException e) {
e.printStackTrace();
}
return name;
}
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;
}
}
```

После сохранения файла Java, операции выводятся в качестве подузлов web-службы (рис. 3.9).

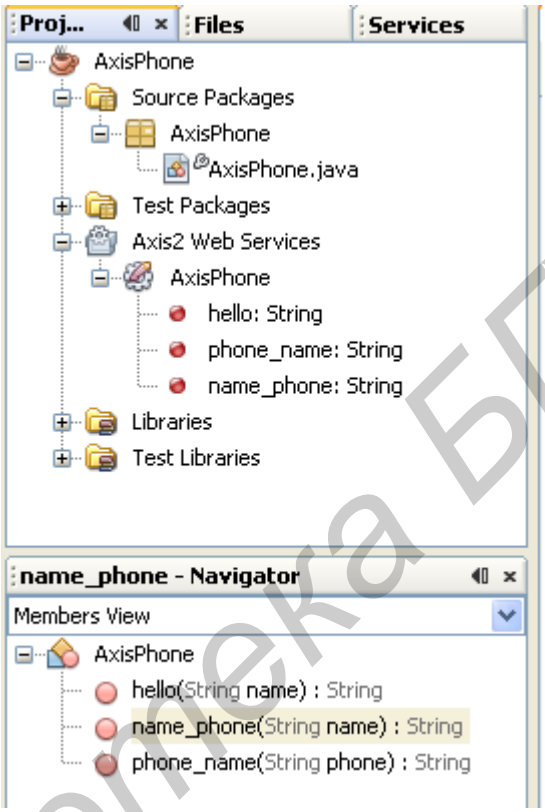


Рис. 3.9. Вид окна проекта после модификации шаблона web-сервиса

Теперь перейдем к реализации клиента. Дистрибутив Axis2 включает два файла ***axis2.war*** и ***axis2-<version>-bin.zip***, которые необходимо распаковать в папку, например ***C:/Program Files/axis2-1.4.1***. Затем необходимо указать в файлах ***java2wsdl.bat***, ***axis2.bat***, ***axis2server.bat***, ***wsdl2java.bat***, находящихся в данном случае в каталоге ***C:/Program Files/axis2-1.4.1/bin/,*** переменную среды окружения ***JAVA_HOME***, например:

```
set JAVA_HOME=c:\Java\jre6
```

Далее необходимо получить спецификацию данного web-сервиса в виде WSDL-документа. Для этого следует повторно развернуть обновленный сервис ***AxisPhone*** на сервере.

Затем нужно запустить страницу Axis2, где отображается список сервисов, для чего следует правой кнопкой мыши щелкнуть в дереве проекта по узлу ***Axis2 Web Services***, чтобы появилось раскрывающееся меню (рис. 3.10). В этом меню необходимо выбрать пункт ***Show Services in Browser***.

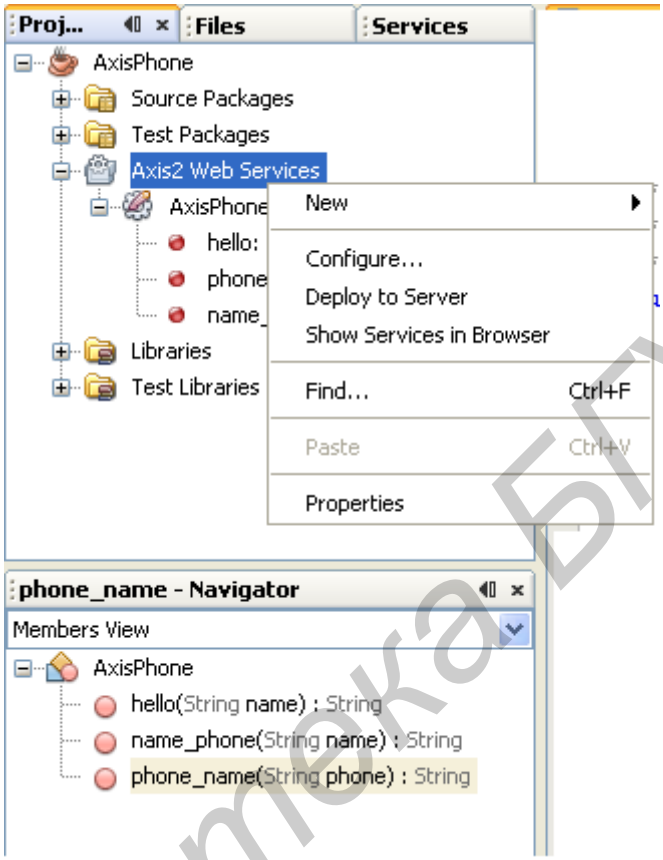


Рис. 3.10. Окно запуска просмотра списка web-сервисов в браузере

Результатом таких действий будет появление окна, показанного на рис. 3.11. В списке web-сервисов Axis2 найдите ссылку на разрабатываемый web-сервис ***AxisPhone*** и щелкните на ней. В результате вы должны увидеть окно браузера со спецификацией в виде WSDL-документа разрабатываемого web-сервиса. Для удобства сохраните содержимое окна в файл ***AxisPhone.wsdl*** в каталог ***C:/Program Files/axis2-1.4.1/bin***. Затем запустите консоль командой операционной системы ***cmd*** и перейдите в каталог ***C:/Program Files/axis2-1.4.1/bin*** и наберите в консоли

```
wsdl2java.bat -uri AxisPhone.wsdl
```

В результате таких действий в каталоге C***:/Program Files/axis2-1.4.1/bin*** должен появиться каталог ***src*** со следующим содержимым:

```
src/
```

```
axisphone/
                AxisPhoneCallbackHandler.java
                AxisPhoneStub.java
```

Полученные Java-файлы будут содержать код proxy-заглушки, который упрощает работу с web-сервисом.

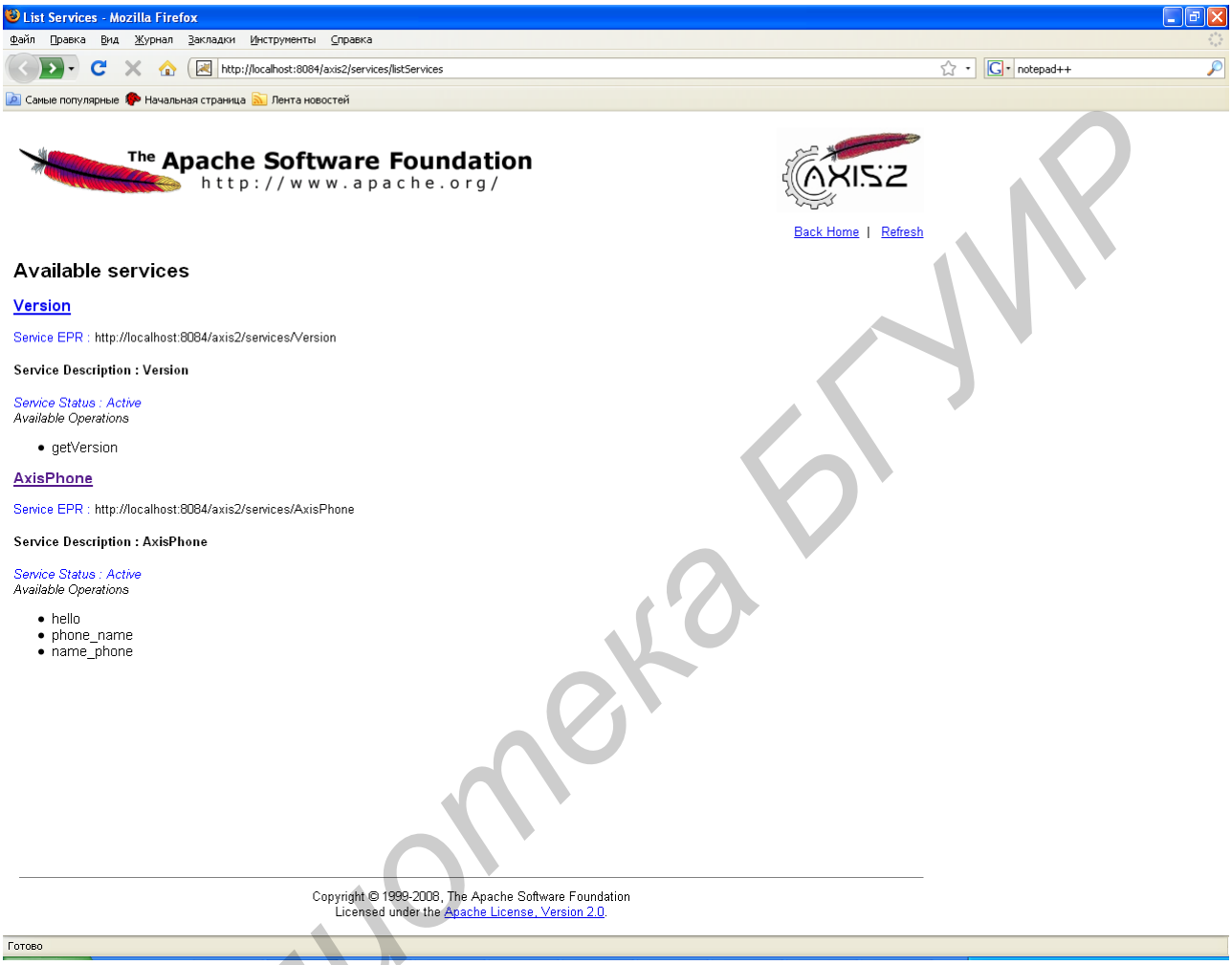


Рис. 3.11. Список доступных web-сервисов в окне браузера

Далее создадим проект Java Application в среде NetBeans IDE. Назовем его ***AxisPhoneClient***. Скопируем в каталог ***src*** созданного проекта каталог **axisphone** из каталога ***C:/Program Files/axis2-1.4.1/bin/src***.

Перейдем в окно NetBeans IDE: среда указывает на то, что в приложении допущены ошибки, а также, что она определила в проекте ***AxisPhoneClient*** новый пакет ***axisphone***.

Ошибки в проекте, на которые указывает NetBeans IDE, произошли из-за того, что в переменную среды окружения **CLASSPATH** не были добавлены пу-

ти к библиотекам классов Axis2, которые использует proxy-заглушка разрабатываемого web-сервиса ***AxisPhone***.

Для того чтобы устранить ошибки в проекте, перейдем к пункту меню NetBeans IDE ***Tools/Libraries***, который содержит перечень всех Java-библиотек, предлагаемых NetBeans IDE для разработки. Создадим новую библиотеку с именем ***axis2-1.4.1*** и добавим, воспользовавшись кнопкой ***Add JAR/Folder…***, перечень Java-библиотек из каталога ***C:/Program Files/axis2-1.4.1/lib***. В результате увидим следующее окно (рис. 3.12).

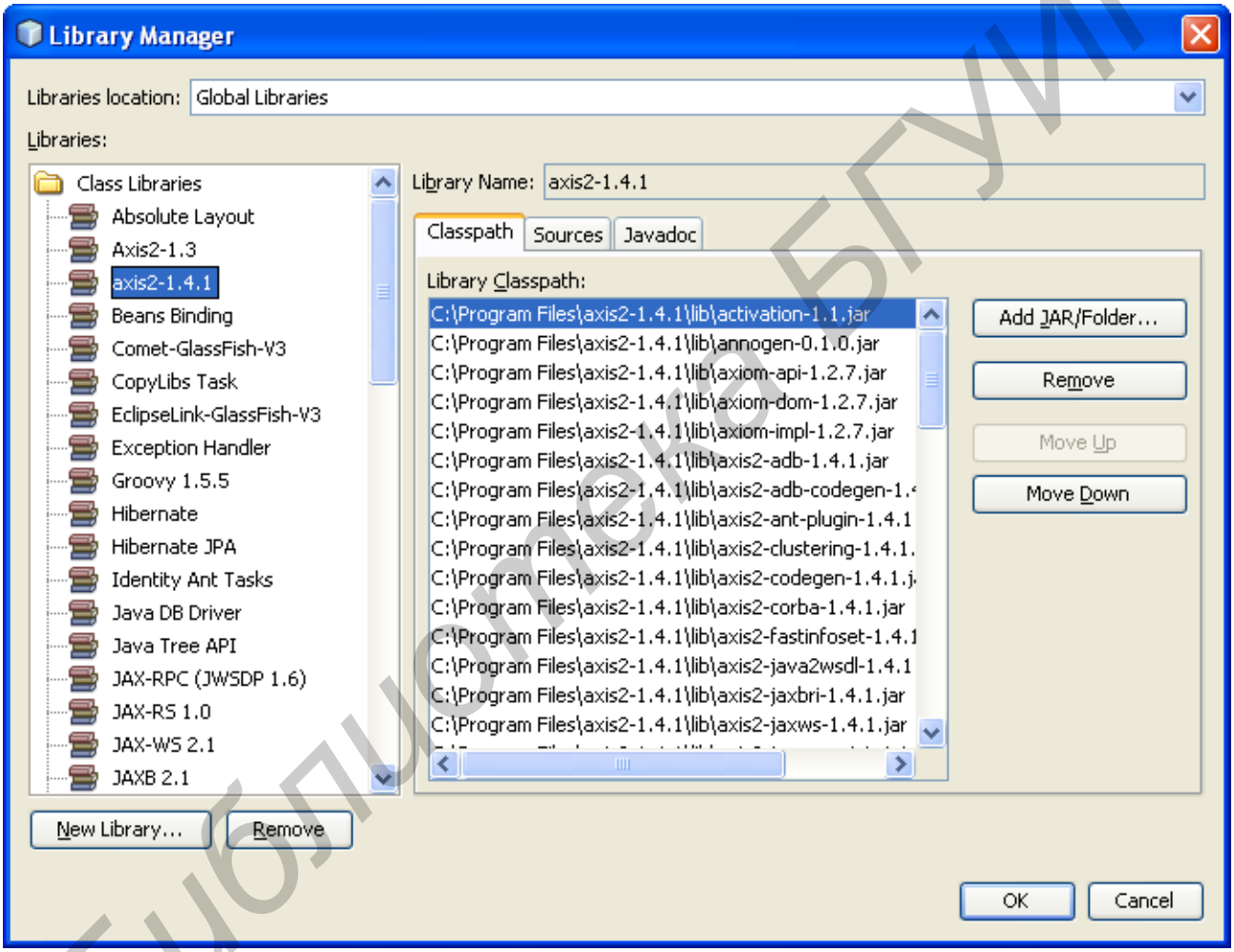


Рис. 3.12. Окно подключения Java-библиотек Axis2 в среде NetBeans IDE 6.5

После настройки NetBeans IDE библиотеки Axis2 необходимо подключить к проекту ***AxisPhoneClient.*** Для этого зайдем в настройки проекта ***AxisPhoneClient***, щелкнув правой клавиши мыши по имени проекта в окне ***Projects NetBeans IDE*** и выбрав пункт всплывающего меню ***Properties.***

Выберем в появившемся окне ***Project Properties -AxisPhoneClient*** категорию ***Libraries***, затем, воспользовавшись кнопкой ***Add Library…***, откроем окно доступных NetBeans IDE Java-библиотек и выберем библиотеку axis2-1.4.1 (рис. 3.13).

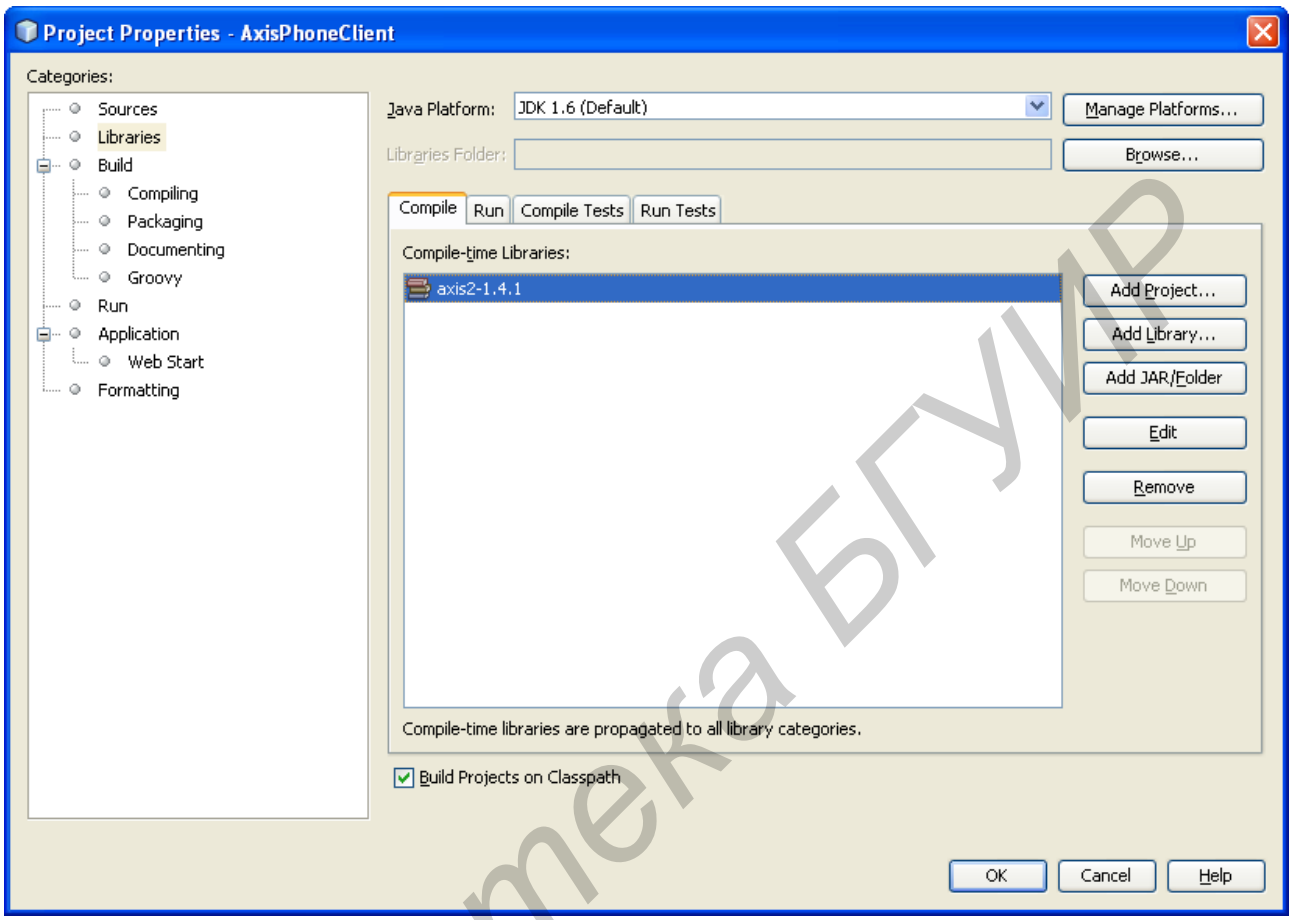


Рис. 3.13. Настройка библиотек проекта AxisPhoneClient

После сканирования добавленных в проект библиотек Axis2 NetBeans IDE сообщит, что ошибки в проекте устранены.

Добавим в класс ***Main*** проекта ***AxisPhoneClient*** следующий код:

```
package axisphoneclient;

import axisphone.AxisPhoneStub;
import java.rmi.RemoteException;
import javax.swing.JOptionPane;
import org.apache.axis2.AxisFault;
public class Main {
public static void main(String[] args) throws AxisFault, RemoteException
{
        AxisPhoneStub test  = new AxisPhoneStub();
        AxisPhoneStub.Hello h = new AxisPhoneStub.Hello();
        h.setName("AxisPhone Tester");
```

```
        AxisPhoneStub.HelloResponse hr = test.hello(h);
        JOptionPane.showMessageDialog(null,hr.get_return());
    }

    }
```

Запустим проект, предварительно запустив сервер Tomcat, на котором развернут Axis2 с web-сервисом ***AxisPhone***. В результате работы клиентского приложения должно появиться окно с сообщением ***«Hello AxisPhone Tester»*** (рис. 3.14).

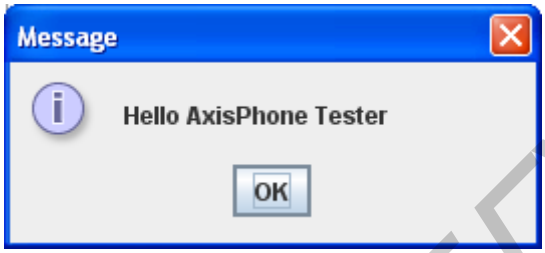


Рис. 3.14. Результат выполнения проекта AxisPhoneClient

Рассмотрим подробнее код, который был добавлен в приложение ***AxisPhoneClient.***

Первой строкой мы создаем объект класса proxy-заглушки ***AxisPhoneStub***:

```
AxisPhoneStub test  = new AxisPhoneStub();
```

Следующий шаг – это создание объекта параметра ***Hello*** метода ***Hello*** web-сервиса ***AxisPhone***:

```
AxisPhoneStub.Hello h = new AxisPhoneStub.Hello();
```

Затем инициализируем созданный объект параметра строкой ***«AxisPhone Tester»***:

```
h.setName("AxisPhone Tester");
```

Далее осуществляем вызов метода ***Hello*** web-сервиса ***AxisPhone***, передавая ему предварительно созданный объект параметра переменной ***h***:

```
AxisPhoneStub.HelloResponse hr = test.hello(h);
```

И наконец передаем полученные данные от web-сервиса ***AxisPhone*** в окно сообщений:

```
JOptionPane.showMessageDialog(null,hr.get_return
```

Задание для лабораторной работы

1. Установить и настроить web-сервер Apache Tomcat.

2. **Серверная часть:** необходимо создать сервис, выполняющий функции информационного справочника согласно варианту.

Клиентская часть: клиент должен уметь взаимодействовать с web-сервисом как с помощью механизма RPC, так и с помощью документно-ориентированного подхода.

Данные, полученные с web-сервиса, необходимо отобразить в окне программы. Для каждого информационного справочника следует реализовать функции редактирования информации (добавление, изменение, удаление) и просмотра.

Вариант 1. Информационный справочник по группе студентов.

Вариант 2. Информационный библиотечный справочник.

Вариант 3. Информационный справочник по web-сервисам J2EE.

Вариант 4. Информационный справочник по технологиям локальных сетей.

Вариант 5. Информационный справочник по языку программирования Java.

Вариант 6. Информационный справочник по языку программирования C++.

Вариант 7. Информационный справочник по функциям WinAPI.

Вариант 8. Информационный справочник по технологии JavaBeans.

Вариант 9. Информационный справочник по паттернам проектирования, используемым в web-сервисах.

Вариант 10. Информационный справочник по языку JavaScript.

Вариант 11. Информационный справочник по языку VBScript.

Вариант 12. Информационный справочник по языку программирования Pascal.

Вариант 13. База данных для учета компьютерной техники.

Вариант 14. Справочник по видам цветов.

Вариант 15. База данных по спортсменам по виду (на выбор) спорта.

Контрольные вопросы

1. Поясните понятие сервисно-ориентированная архитектура.
2. Как осуществляется RPC-ориентированное взаимодействие с web-сервисом?
3. Для чего используется система Axis?
4. Для чего используется JAX-RPC API?
5. Поясните, как осуществляется документно-ориентированное взаимодействие с web-сервисом.
6. Расскажите о различиях между документно-ориентированным и RPC-ориентированным взаимодействием с web-сервисом.
7. Для чего используется язык WSDL?
8. Для чего необходим протокол SOAP?
9. Для чего используется UDDI?

4. СЕРВЕРНЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ WEB-ИНТЕРФЕЙСОВ

4.1. Основы технологии CORBA

С развитием сетевого программного обеспечения в начале 1990-х гг. появились проблемы, связанные с обеспечением возможности общения программ, выполняемых на разных машинах, особенно при использовании разных аппаратных средств, операционных систем и языков программирования: либо программисты использовали сокеты и сами реализовывали весь стек протоколов, либо их программы вовсе не взаимодействовали.

Технология CORBA (Common Object Request Broker Architecture), предложенная консорциумом OMG (Object Management Group), была призвана решить возникшие проблемы. Однако первая спецификация CORBA 1.0 не поддерживала интероперабельность и обеспечивала отображение только для языка C. Только в 1997 г. с появлением спецификации CORBA 2.0 и отображением её на язык C++, а с 1998 г. и на язык Java CORBA начинает использоваться для разработки сложных программных систем.

В технологиях и спецификациях CORBA OMG реализует свое видение концепций объектно-ориентированного программирования.

Применительно к CORBA универсальным является решение, которое не зависит:

- от языка программирования;
- от аппаратной платформы;
- от операционной системы;
- от сетевого протокола передачи данных;
- от двоичных стандартов и структур.

Стандарт CORBA 2.0 образует структуру объектной системы, так называемую ссылочную модель (Reference Model), и содержит следующие компоненты.

Object Request Broker (ORB, брокер запросов объекта) – позволяет объектам строить и отправлять запросы и ответы в распределенной системе. Является основанием для приложений с распределенными объектами и для интероперабельности между приложениями в одно- и разнородной средах. Архитектура и спецификации ORB описываются в CORBA.

Object Services (OS, сервисы объекта) – набор сервисов (интерфейсов и объектов), которые поддерживают основные функции использования и применения объектов. Сервисы являются необходимой частью при конструировании распределенных приложений и всегда остаются независимыми от них. Например, Life Cycle Service (сервис жизненного цикла) определяет конструкции для создания, удаления, копирования и перемещения объектов. Но не определяет, как именно объекты реализуются. Информация по сервисам объектов рассматривается в спецификациях CORBA-services.

Common Facilities (CF, общие свойства) – набор сервисов, которые многие приложения могут использовать совместно (не являются такими фундаментальными, как OS). Например, сервис электронной почты может рассматриваться как CF. Информация по общим свойствам рассматривается в спецификациях CORBA facilities.

Application Objects (AO, объекты приложений) – являются продуктами индивидуального разработчика, реализующими собственные интерфейсы. Application Objects соответствуют обычным приложениям, поэтому не стандартизуются OMG. Они являются самым верхним уровнем ссылочной модели.

Спецификации CORBA создаются на специальном языке – OMG IDL (Interface Definition Language). Универсальность технологии главным образом обеспечивается базированием исключительно на IDL и использованием утвержденных OMG стандартов отображения IDL-объявлений на конкретные языки программирования. В связи с отсутствием реализации собственной компонентной модели CORBA – Corba Beans, разработка которой ведется, очевидное преимущество имеют реализации технологий CORBA на Java в силу универсальности Java и переносимости его кода на другие платформы. Ярким примером является компонентная модель EJB, которая будет рассмотрена в подразд. 4.2.

Задачи поиска, отношений между объектами, сохранения их состояний, управления транзакциями и безопасностью и другие возложены на элементы архитектуры CORBA – **сервисы CORBA** (Object Services). Спецификация сервисов состоит из множества интерфейсов, описывающих поведение сервисов на языке IDL.

4.2. Технология RMI. Технология Enterprise JavaBeans

На технологию CORBA оказала сильное влияние **технология RMI,** требующая использования языка программирования Java. Главная задача, которую

призвана решать RMI, – обеспечить передачу сообщений от объекта, существующего в контексте одной виртуальной машины Java (JVM), к объекту, существующему в контексте другой.

RMI использует два стандартных протокола обмена: собственный неуниверсальный протокол обмена RMP и стандартный для CORBA протокол IIOP. Поддержка IIOP позволяет RMI-приложениям получить доступ ко всем сервисам CORBA и CORBA-серверам, написанным на любом языке, поддерживающим CORBA. В свою очередь интеграция с RMI позволяет CORBA компенсировать отсутствие в настоящий момент собственной компонентной модели CORBA и соответственно универсального монитора транзакций. В основу RMI заложены средства сериализации Java.

Enterprise JavaBeans (EJB). Спецификация EJB составляет значительную часть платформы Java 2 Enterprise Edition (J2EE) и описывает модель компонентов серверной стороны, ориентированную на создание масштабируемых, устойчивых и надежных серверных приложений. EJB служит для использования ресурсов серверов, управления правами доступа, безопасностью, транзакциями и взаимодействием с базами данных различного типа. При программировании EJB-компонента программист получает возможность «избавиться» от написания низкоуровневого кода и полностью сфокусироваться на реализации бизнес-логики.

Составной частью EJB может быть только то, что соответствует стандартам как RMI, так и CORBA. Такое подмножество называется RMI/IDL.

Технология создания серверных объектов EJB базируется на Java и использует технологию RMI для организации удаленных вызовов между виртуальными машинами JVM. Протоколом взаимодействия в EJB является стандартный протокол CORBA IIOP. Схема управления транзакциями JTS (Java Transaction Service) – это реализованный на Java сервис транзакций CORBA (OTS, Object Transaction Service). Существует стандарт отображения EJB на CORBA, касающийся управления транзакциями, безопасностью и службой имен (Naming Service).

EJB используются интерфейсы, определяемые стандартной семантикой RMI. На базе этих Java-объявлений с помощью компилятора, разработанного OMG (rmi2idl), можно сгенерировать IDL-файл, который может быть использован CORBA-программистами для создания на любом поддерживающем

CORBA языке клиентских приложений, взаимодействующих с EJB как с серверными объектами CORBA. То есть любой компонент EJB является одновременно полноценным CORBA-объектом.

Сервер EJB – это среда выполнения, в которой функционируют компоненты EJB. Задачей сервера является обеспечение доступа к системным сервисам, необходимым для работы компонентов. Важнейшим из сервисов является сервис распределенных транзакций JTS. Сервер не взаимодействует непосредственно с компонентами, а реализует взаимодействие через некий логический уровень управления – контейнер EJB.

Контейнер EJB взаимодействует с сервером, когда компонентам, находящимся под управлением контейнера, необходим доступ к системным ресурсам.

Контейнер обеспечивает:

- управление циклом жизни компонента: его созданием, инициализацией, сохранением его состояния в базе данных, если это необходимо;
- поиск компонента;
- гарантию того, что вызов методов происходит в контексте нужной транзакции;
- базовый уровень обеспечения безопасности.

В EJB используется спецификация управления транзакциями CORBA, реализованная на Java (JTS). Управлять транзакциями – начинать транзакцию, завершать или откатывать транзакцию – может как компонент, так и контейнер.

Обычно сервер и контейнер поставляются в готовом виде, например, Inprise Application Server, WebLogic Application Server, GlassFish Application Server, JBoss Application Server, Sun Application Server и многие другие.

Компонент EJB –это класс Java, который и реализует всю необходимую функциональность. Компонент находится под управлением контейнера и состоит из нескольких частей – класс компонента, реализация некоторых интерфейсов и информационный файл. Все это запаковано вместе в специальный jar-файл. Все объекты в реализациях компонента должны быть действительными в RMI/IDL.

Класс компонента – реализует интерфейс Enterprise Bean и обеспечивает реализацию бизнес-методов, которые выполняет компонент. Класс не реализует

никакую авторизацию, многопоточность или код транзакции, реализации этих ролей берет на себя поставщик EJB.

При реализации EJB-класса необходимо следовать нескольким требованиям (полный список требований содержится в спецификации Enterprise JavaBeans), основными из которых являются следующие:

- класс должен быть публичным (public);
- класс должен реализовывать EJB-интерфейс (либо javax.ejb.SessionBean, либо javax.ejb.EntityBean);
- класс должен определять методы, которые напрямую связываются с методами внешнего интерфейса. Но класс не реализует удаленный интерфейс, он лишь отражает методы внешнего интерфейса и не обрабатывает java.rmi.RemoteException;
- в классе должен быть определен один или несколько методов ejbCreate() для инициализации компонента EJB.

Все запросы к компоненту EJB передаются контейнеру компонентов посредством внутреннего и удаленного интерфейсов.

Клиент может обращаться к компонентам через специальный интерфейсный объект-посредник (EJBObject), определяемый контейнером EJB. EJBObject реализует **удаленный интерфейс** (Remote-интерфейс). Удаленный интерфейс играет ту же роль, что и IDL интерфейс в CORBA. При создании удаленного интерфейса необходимо следовать следующим принципам:

- удаленный интерфейс должен быть публичным (public);
- удаленный интерфейс должен расширять интерфейс javax.ejb.EJBObject.

Каждый метод удаленного интерфейса должен декларировать java.rmi.RemoteException в предложении throws, помимо всех исключений, специфичных для приложения.

Внутренний интерфейс – используется для создания компонента и реализуется объектом HomeObject. Клиент использует внутренний интерфейс для нахождения экземпляра данного EJB и создания нового экземпляра данного EJB. При создании внутреннего интерфейса необходимо следовать следующим принципам:

- внутренний интерфейс должен быть публичным (public);
- внутренний интерфейс должен расширять интерфейс javax.ejb.EJBHome;

- каждый метод create внутреннего интерфейса должен декларировать java.rmi.RemoteException в предложении throws наряду с javax.ejb.CreateException;
- возвращаемое значение метода create должно быть удаленным интерфейсом.

Метод finder используется только для компонентов с данными и должен иметь в качестве возвращаемого значения удаленный интерфейс, или java.util.Enumeration, или java.util.Collection. Стандартное соглашение об именах внутренних интерфейсов состоит в прибавлении слова «Home» в конец имени удаленного интерфейса.

Описатель развертывания (deployment descriptor) содержит всю информацию о компоненте EJB и является XML-файлом. Использование XML позволяет программисту легко изменять атрибуты данного EJB. Конфигурационные атрибуты, определенные в описателе развертывания, включают:

- имена внутреннего и внешнего интерфейса;
- имя для публикации в JNDI для внутреннего интерфейса;
- транзакционные атрибуты для каждого метода EJB;
- контрольный Список Доступа для авторизации;
- инструменты разработки компонентов EJB, которые могут автоматически генерировать описатель развертывания.

EJB-jar файл – это обычный jar-файл, который содержит класс EJB, внутренний и удаленный интерфейсы и описатель развертывания.

Рассмотрим типы компонентов, которые обладают различными характеристиками и свойствами. Существует два типа компонентов – сессионный компонент (***Session Bean***) и компонент с данными (***Entity Bean***). В каждом из них имеются свои разновидности.

Сессионный компонент используется для представления событий в потоке работы с клиентом, его создавшим. Такие компоненты представляют операции с постоянными данными, но не сами постоянные данные. Сессионный компонент может иметь состояние (***Stateful Session Bean***) и может не иметь (***Stateless Session Bean***). Все сессионные компоненты должны реализовывать интерфейс ***javax.ejb.SessionBean***. EJB-контейнер управляет жизнью сессионного компонента.

Сессионный компонент без состояния не содержит никаких значимых состояний для клиента между вызовами методов, а вся информация о состоянии компонента должна храниться за пределами самого компонента.

Сессионный компонент с состоянием сохраняет состояние между вызовами. Каждый компонент взаимно однозначно соответствует определенному клиенту и может содержать состояние в себе. EJB-контейнер может управлять такими компонентами, используя активизацию (**activation**) и деактивизацию (**passivation**). При деактивизации контейнер временно удаляет компонент из памяти, помещая его состояние в объектную или реляционную базу данных. Обратный процесс – это активизация, размещение объекта в памяти с восстановлением его предыдущего состояния. Естественно, компонент без состояния может быть просто уничтожен контейнером, а потом заново создан в нужный момент.

Если работа EJB-контейнера нарушается, данные всех EJB-сессионных компонентов с состоянием могут быть потеряны.

Компоненты с данными являются компонентами, представляющими постоянные данные и их поведение, и управляются EJB-контейнером. Они могут быть разделены между многими клиентами точно так же, как могут разделяться данные в базе данных. Существование компонента с данными определяется EJB-контейнером, так что если работа EJB-контейнера нарушается, то данные не теряются; сам компонент будет доступен только тогда, когда будет доступен EJB-контейнер. Есть два типа компонент с данными: существующие с управлением контейнером (***Container-managed persistence beans***) и существующие с управлением компонентами (***Bean-Managed persistence beans***).

Container-Managed Persistence (CMP) через указанные в описании развертывания спецификации EJB-контейнер связывает атрибуты компонента с некоторым постоянным хранилищем, обычно это база данных. CMP снижает время разработки требуемого для EJB кода, так же как и значительно снижается объём требуемого кода.

Bean-Managed Persistence (BMP) – BMP-компоненты реализуются поставщиком Enterprise Bean. Поставщик Enterprise Bean отвечает за реализацию логики, требуемой для создания новых EJB, изменения некоторых атрибутов EJB, удаление EJB и нахождение EJB в постоянном хранилище. Обычно для

этого требуется написание JDBC-кода для взаимодействия с базой данных или другим постоянным хранилищем. С помощью BMP разработчик полностью контролирует управление существованием компонента. BMP также дает гибкость в тех местах, где реализация CMP не может быть использована. Например, если необходимо создать EJB, который включает в себя код, реализующий некие сервисы CORBA.

4.3. Пример создания библиотеки EJB-компонентов

Закрепим полученные сведения на практике. Разработаем библиотеку EJB-компонентов, которые будут взаимодействовать с базой данных. Клиент сможет взаимодействовать с EJB-компонентами посредством web-сервиса, который тоже будет разработан в данном примере.

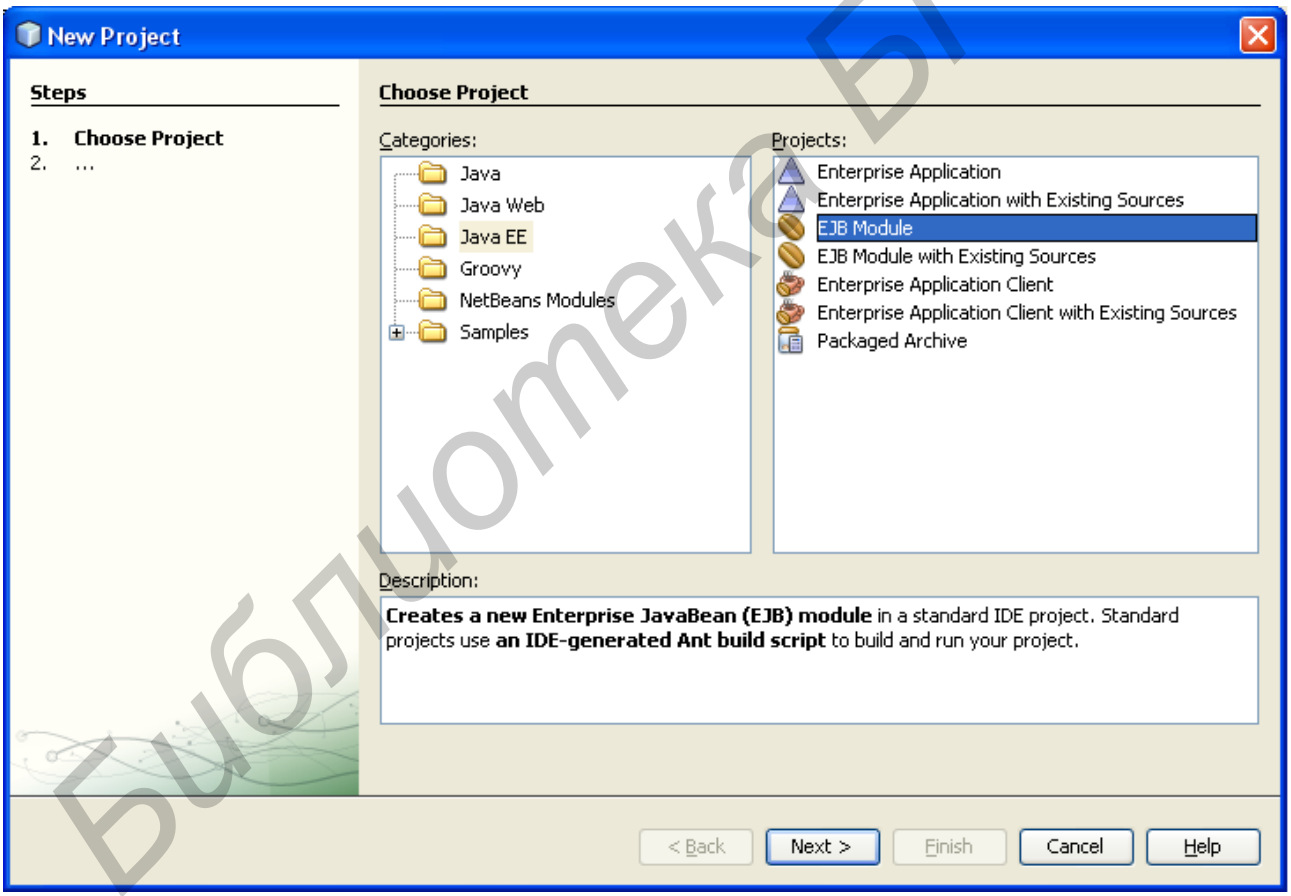


Рис. 4.1. Создание проекта для библиотеки EJB-компонент

Создадим проект в IDE NetBeans, для чего выберем пункт меню ***File->New Project***. В появившемся окне мастера создания проекта выберем категорию ***Java EE*** и тип проекта ***EJB Module*** (рис. 4.1). Далее щелкнем по кнопке ***Next***.

На следующем шаге мастер создания проектов предложит выбрать место положения проекта и ввести название. Введем название проекта ***EJBCustomerslib***, а местоположение укажем наиболее удобным. Выполнив данные действия, щелкнем по кнопке ***Next***.

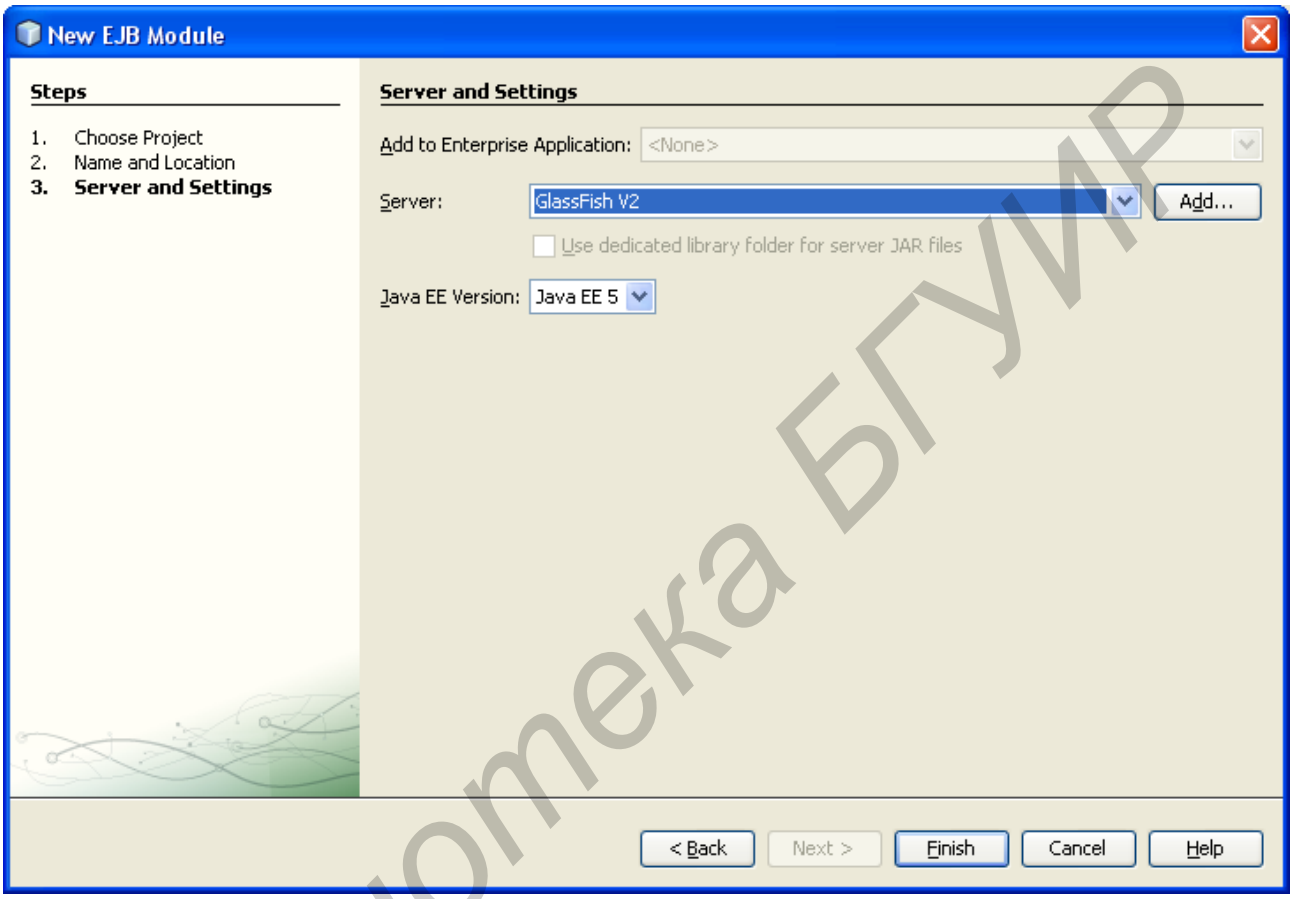


Рис. 4.2. Окно настройки сервера приложений и версии Java EE

На следующем шаге предложено будет выбрать сервер приложений, на котором будет размещаться библиотека EJB-компонентов, и версию Java EE, с помощью которой будем разрабатывать EJB-компоненты. Для данного примера выберем сервер приложений ***GlassFish V2***, который устанавливается вместе со средой NetBeans IDE 6.8. Версию Java EE оставим 5, как предложило само IDE (рис. 4.2). Затем нажмем кнопку ***Finish.*** Проект создан.

Далее необходимо создать подключения к базе данных. В данном примере не рассматривается процесс создания базы данных ввиду того, что это выходит за пределы рассматриваемого примера. Воспользуемся готовой базой данных,

которая разработана в СУБД Java DB и входит в состав jdk 1.6. База данных, с которой мы будем работать, является демонстрационным примером.

Перейдите во вкладку ***Services*** NetBeans IDE и щелкните левой клавишей мыши по узлу Databases.

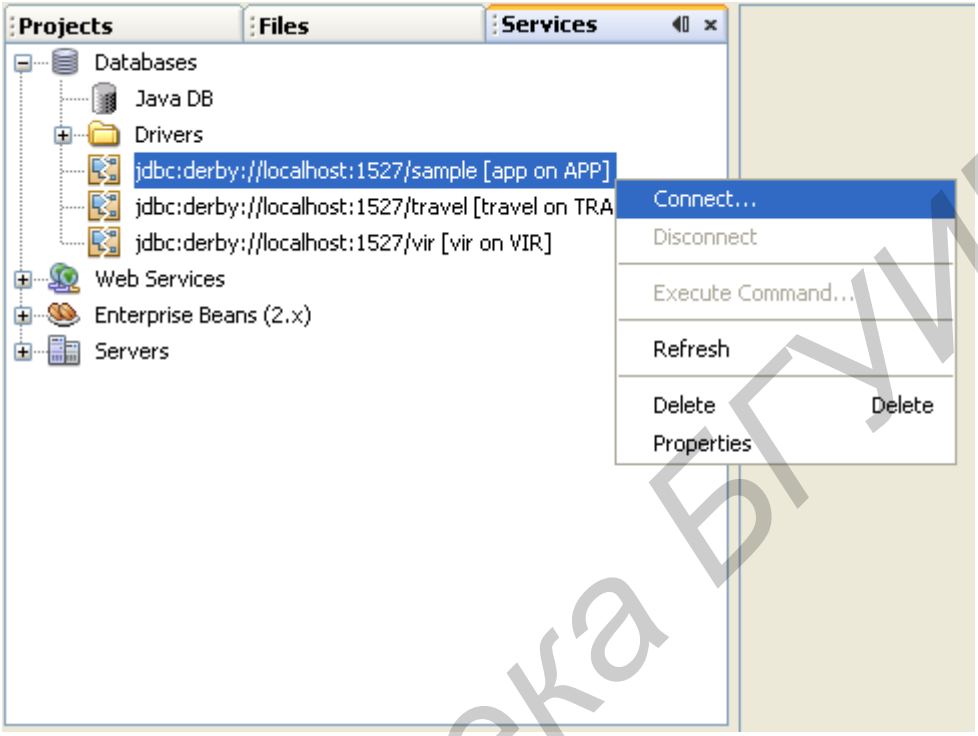


Рис. 4.3. Создание подключения к базе данных sample

Далее щелкнем правой кнопкой мыши по узлу с названием ***jdbc:derby://localhost:1527/sample***(рис. 4.3) и в появившемся всплывающем меню выберем пункт меню ***Connect…*** . После этих действий NetBeans IDE подключится к СУБД Java DB, и можно работать с базой данных ***sample.***

Создадим компонент с данными (Entity Bean), который будет взаимодействовать с базой данных **sample.** Для этого перейдем на вкладку **Projects** NetBeans IDE, правой кнопкой мыши щелкнем по имени проекта и выберем ***"New->Entity Classes from Database…"*** (рис. 4.4), в появившемся окне мастера в поле ***Data Source*** необходимо выбрать источник данных с именем JNDI ***jdbc/sample*** или ***New Data Source …***, если нужно создать новый источник данных на основе существующего подключения.

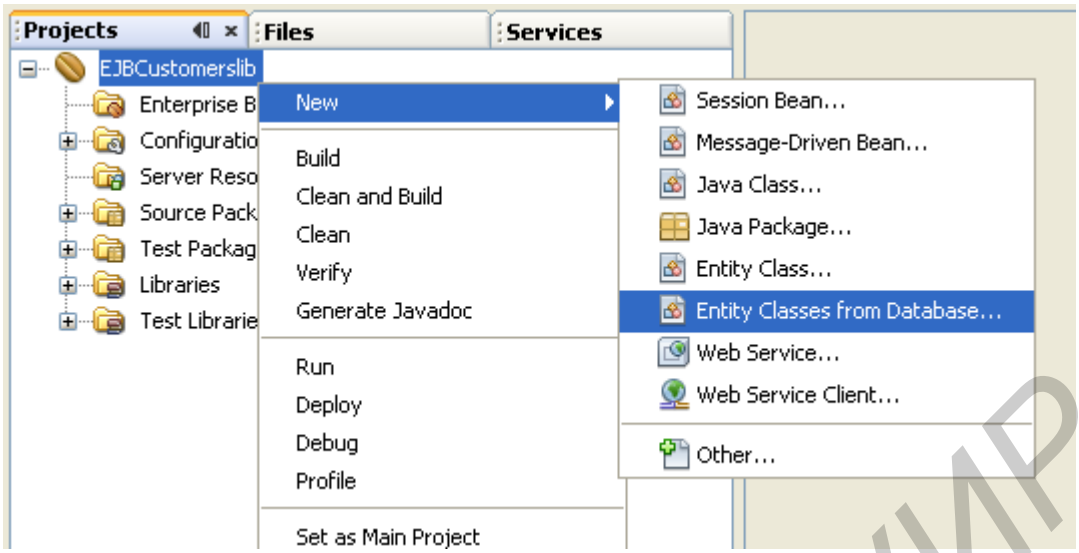


Рис. 4.4. Создание компонента с данными (Entity Bean)

Воспользовавшись источником данных ***jdbc/sample***, увидим в столбце ***Available Tables*** (доступные таблицы) перечень таблиц, с которыми можно работать. Выберем в списке таблицу ***Customer*** и нажмем на кнопку ***Add*** > – результат ваших действий должен быть таким же, как на рис. 4.5. Далее нажмем кнопку ***Next.***

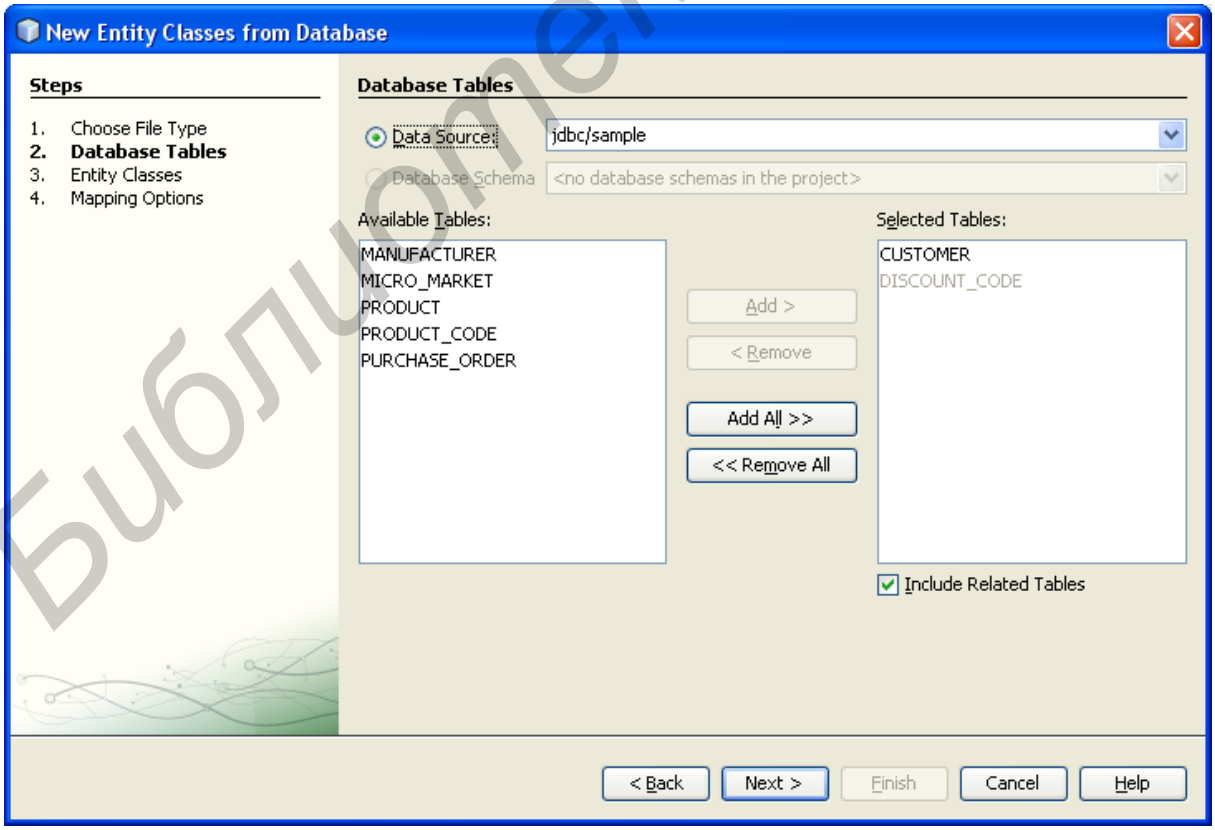


Рис. 4.5. Выбор таблицы из источника данных

Следующее появившееся окно мастера позволит задать имя пакета, в котором будут размещаться компонент с данными. Укажем имя пакета ***ejbbeanspackage.***

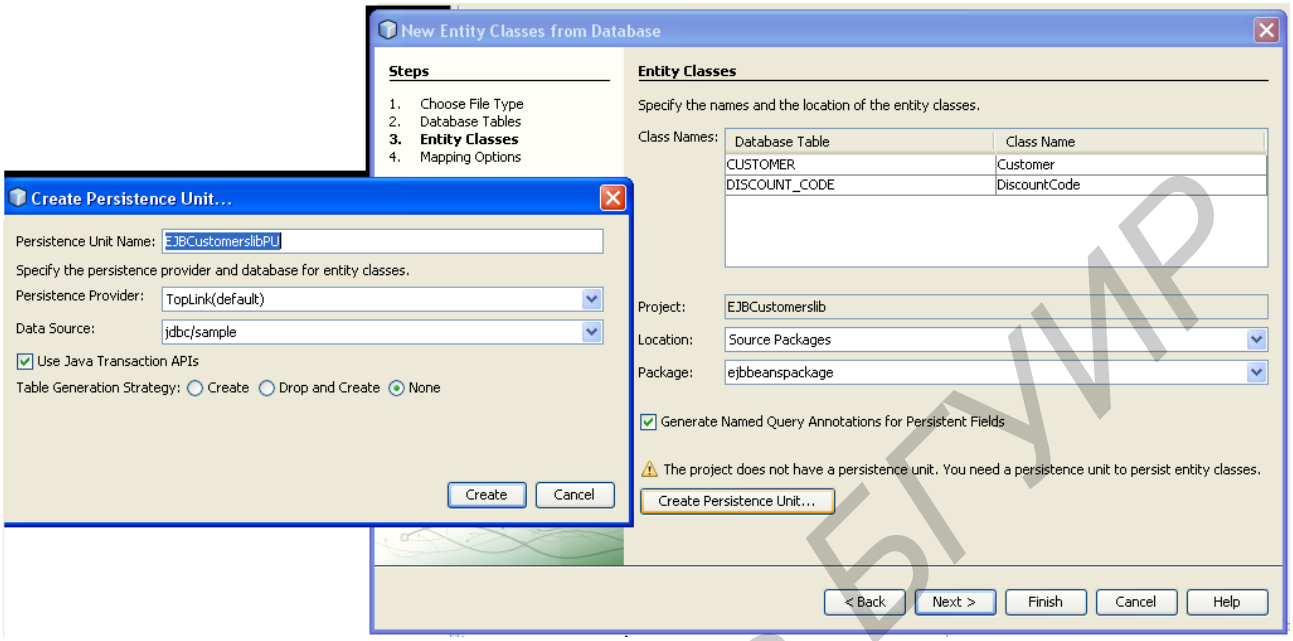


Рис. 4.6. Создание классов ORM-провайдера для базы данных Customer

Убедимся, что на странице мастера установлен флажок ***«Generate Named Query Annotations for Persistent Fields»*** (рис. 4.6).

Затем нажмем кнопку ***Create Persistent Unit*** для того, чтобы в появившемся окне выбрать ORM-провайдера для работы с базой данных **sample.** По умолчанию NetBeans IDE предлагает использовать ***TopLink*** (ORM-провайдер компании Oracle), остановимся на нем.

В поле ***Persistent Unit Name*** предлагается указать имя, под которым будут храниться настройки, указывающие серверу приложений как подключаться к источнику данных, оставим его без изменений. Поставим галочку ***Use Java Transaction APIs.***

Установим флажок ***Table Generation Strategy*** (стратегии генерации таблиц) в режим ***None***, который позволит нам восстановить компонент данных на основе таблицы **Customer.** Два других режима не подходят: режим **Create** дает возможность создать новый объект данных (т. е. на основании классов объектов при развертывании приложения создаются таблицы), а режим ***Drop and Create*** позволяет пересоздать уже созданные когда-либо компоненты данных. Затем нажмем кнопку **Create**, после этого в окне мастера нажмем кнопку ***Finish.*** В

результате этих действий в пакете ***ejbbeanspackage*** должны были сгенерироваться два класса – ***Customer***, соответствующий таблице ***Customer*****,** и ***Discount-Code***, соответствующий таблице ***DiscountCode***, от которой таблица ***Customer*** зависит**.**

Следующим шагом создадим сессионный компонент на основе созданного компонента данных. Для этого правой кнопкой мыши щелкнем по имени проекта и в появившемся выпадающем меню выберем ***New->Other…***; в появившемся окне мастера выберем категорию **Persistence** и тип **файла Session Beans For Entity Classes** (рис. 4.7). Нажмем ***Next>.***

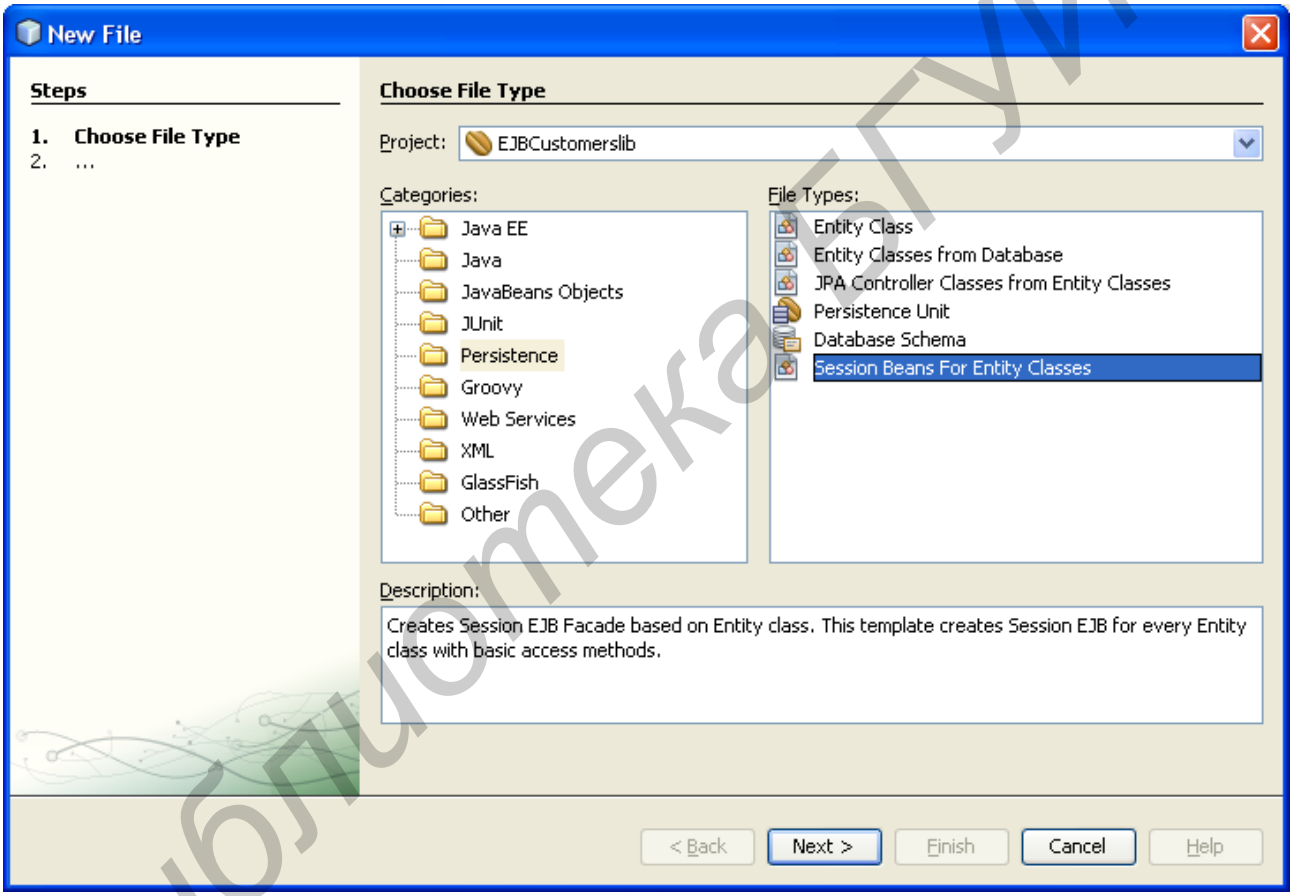


Рис. 4.7. Окно создания сессионного компонента

Далее в появившемся окне мастера необходимо выбрать классы компонент с данными, для которых будут создаваться сессионные компоненты. Нажмем на кнопку **Add All >>** для того, чтобы добавить все классы, которые есть в проекте. Затем нажмем клавишу **Next>**.

В появившемся окне мастера (рис. 4.8) необходимо указать название пакета, в котором будут храниться класс и тип сессионного компонента. Укажем

имя пакета ***ejbbeanspackage.*** Установим флажок ***Local*** для того, чтобы для сессионного компонента был сгенерирован внутренний интерфейс. В удаленном интерфейсе (флажок ***Remote***) нет необходимости, так как создаваемый сессионный компонент будет использоваться в рамках одного сервера приложений. Нажмем кнопку ***Finish.***

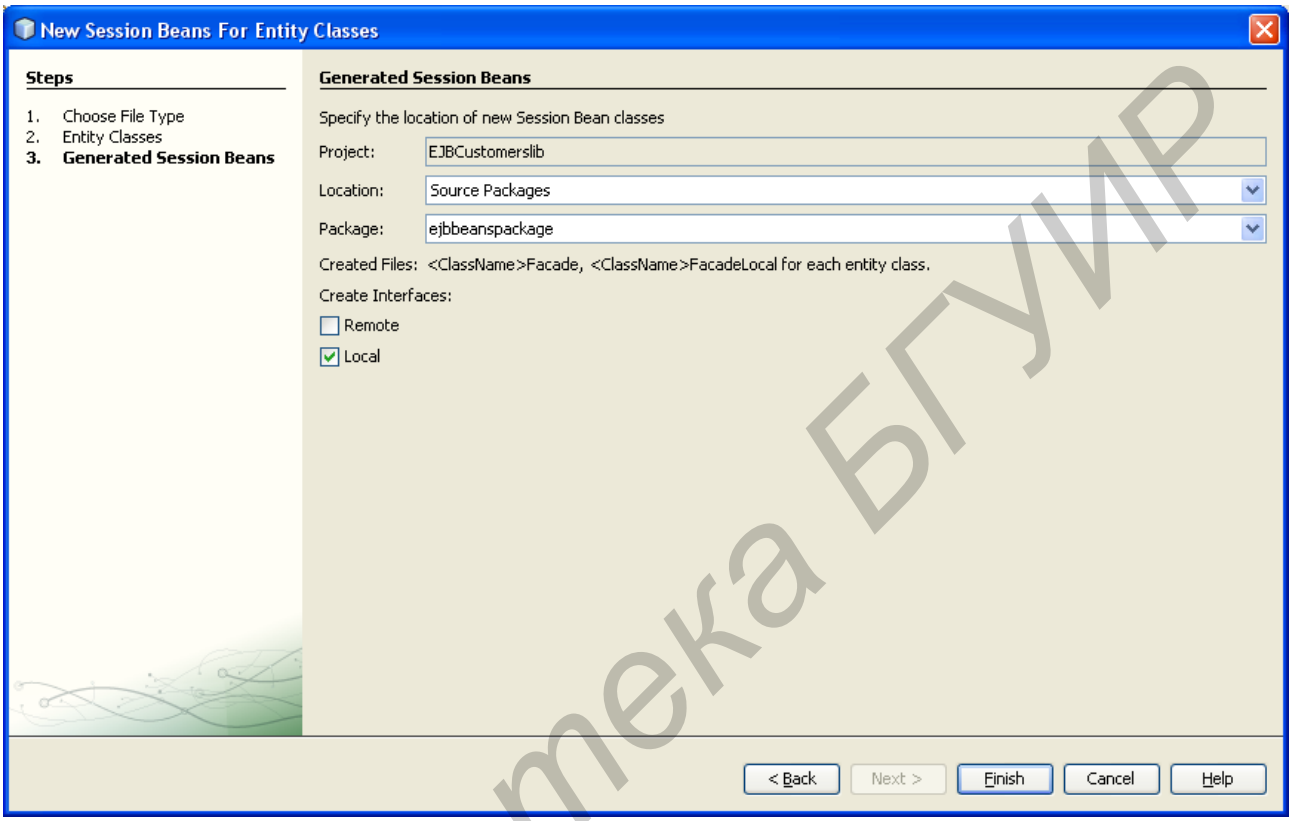


Рис. 4.8. Выбор типа сессионного компонента

В результате таких действий в проект добавится четыре класса – ***CustomerFacade, CustomerFacadeLocal***, ***DiscountCodeFacade, DiscountCodeFacadeLocal.*** Классы, имеющие в названии постфикс ***Local*** (или ***Remote*** для удалённых интерфейсов), являются интерфейсными классами сессионных компонентов. Классы, имеющие постфикс ***Facade***, содержат реализацию сессионных компонентов.

Прежде чем перейти к созданию сервиса, рассмотрим один из инструментов, предложенных разработчиками платформы Java, – **аннотации**. Аннотации были введены в язык Java начиная с пятой версии платформы. Аннотация позволяет получить дополнительную информацию о любом объекте языка – классе, методе и поле. Аннотация (и ее параметры) может быть прочитана в момент исполнения. Следовательно, необходимая информация может быть записана

прямо в файле. Аннотации широко используются в Java Persistance API (JPA, Java API для хранения). В файлах исходного кода аннотация записывается с префиксом @, например, в сгенерированных классах компонентов данных (классы ***Customer***, ***DiscountCode***) используются аннотации ***@Entity*** (говорит о том, что данный класс представляет собой сущность, которую следует сохранять в базе данных) и ***@Table*** (показывает, какая таблица используется для хранения).

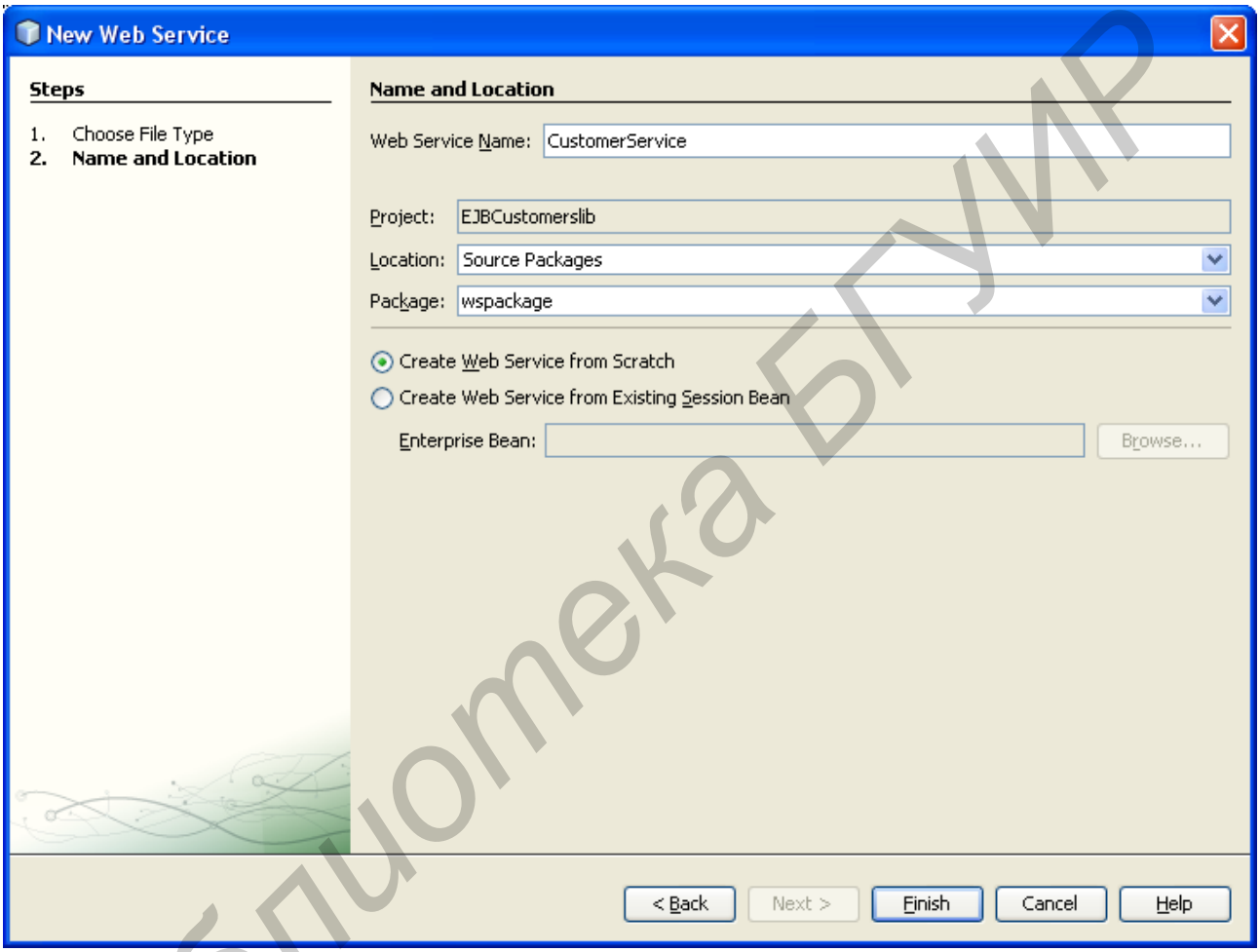


Рис. 4.9. Окно мастера создания web-сервиса

Теперь перейдем к созданию web-сервиса. Перейдем на вкладку ***Projects*** NetBeans IDE, выберем узел с именем проекта и нажмем правую кнопку мыши. Во всплывшем окне меню выберем пункт ***New->Web Service …*** .

В окне мастера создания web-сервиса (рис. 4.9) введем в поле ***Web Service Name*** имя web-сервиса ***CustomerService***. Затем в поле ***Package*** добавим имя пакета ***wspackage***, в котором разместятся файлы – исходные коды web-сервиса. Затем нажмем кнопку ***Finish***. Шаблон web-сервиса создан. Откроем файл с исходным кодом web-сервиса и добавим в него следующий код:

Файл CustomerService.java

```
package wspackage;
import ejbbeanspackage.Customer;
import java.util.List;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

// @WebService()
@Stateless()
public class CustomerService {
@PersistenceContext
private EntityManager em;
@WebMethod(operationName = "getCustomers")
public CustomersList getCustomers() {
List<Customer> lst =
em.createNamedQuery("Customer.findAll").getResultList();
CustomersList res  = new CustomersList(lst);
return res;
}
}
```

Рассмотрим добавленный код. Аннотация ***@WebService()*** указывает, что созданный класс ***CustomerService*** будет являться web-сервисом. Аннотация ***@WebMethod***, которая размещается перед методом ***getCustomers()***, позволяет указать, что этот метод будет являться операцией web-сервиса с именем ***getCustomers.*** Строка кода

```
List<Customer> lst = em.createNamedQuery("Customer.findAll").getResultList();
```

позволяет с помощью менеджера компонентов данных (он объявлен с использованием аннотации ***@PersistenceContext***) создать и выполнить именованный запрос ***Customer.findAll*** к базе данных ***sample***, который сгенерирован NetBeans IDE для работы с таблицей ***Customer***. Реализация запроса находится в файле ***Customer.java.*** Результаты выполнения запроса передаются в список, который преобразуется строкой

```
CustomersList res = new CustomersList(lst);
```

в xml-документ, который в свою очередь будет обработан сервером приложений и передан клиенту.

После изменения файла ***CustomerService.java*** создадим и добавим в пакет ***wspackage*** класс ***CustomerXml*** со следующим исходным кодом:

Файл CustomerXml.java

```
package wspackage;

import ejbbeanspackage.Customer;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class CustomerXml {
    @XmlElement
    public Integer id;
    @XmlElement
    public String zip;
    @XmlElement
    public String name;
    @XmlElement
    public String addressline1;
    @XmlElement
    public String addressline2;
    @XmlElement
    public String city;
    @XmlElement
    public String state;
    @XmlElement
    public String phone;
    @XmlElement
    public String fax;
    @XmlElement
    public String email;
    @XmlElement
    public Integer creditLimit;
    @XmlElement
    public Character discountCode;
    public CustomerXml() {  }
    public CustomerXml(Customer c) {
        this.zip = c.getZip();
        name = c.getName();
        addressline1 = c.getAddressline1();
        addressline2 = c.getAddressline2();
        city = c.getCity();
        state = c.getState();
        phone = c.getPhone();
        fax = c.getFax();
        email = c.getEmail();
        creditLimit = c.getCreditLimit();
        discountCode = c.getDiscountCode().getDiscountCode();
        id = c.getCustomerId();
    }
}
```

Ввиду того что создаваемый web-сервис позволяет организовать документно-ориентированное взаимодействие с клиентом, данные таблицы ***Customer*** будет удобно передавать как единый XML-документ. Класс ***CustomerXml*** позволяет описать структуру узла XML-документа, соответствующего одной записи таблицы ***Customer.*** Аннотация ***@XmlRootElement*** позволяет указать на то, что имя класса, следующего за аннотацией, будет являться именем корневого элемента по отношению ко всем полям класса ***CustomerXml***, объявленных с использованием аннотации ***@XmlElement.***

Затем добавим в пакет ***wspackage*** еще один класс – ***CustomersList*** – со следующим содержимым:

Файл CustomersList.java

```
package wspackage;
import ejbbeanspackage.Customer;
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class CustomersList {

    public CustomersList() { }
    public CustomersList(List<Customer> l)
    {
        for(Customer c : l)
            lst.add(new CustomerXml(c));
    }
    @XmlElement(type=CustomerXml.class)
    public List<CustomerXml> lst=new ArrayList<CustomerXml>();
}
```

Класс ***CustomersList*** является наряду с классом ***CustomerXml*** еще одним вспомогательным классом для описания структуры XML-документа, который будет передавать разработанный web-сервис своему клиенту. Этот класс позволяет с помощью аннотаций описать список XML-элементов типа ***CustomerXml***, что, по сути, эквивалентно XML-вариации таблицы ***Customer***.

Запустим проект на выполнение. После старта сервера приложений ***GlassFish*** щелкнем левой кнопкой мыши в дереве проекта ***EJBCustomerslib*** по подузлу ***Web Services***, в раскрывшемся поддереве щелкнем правой кнопкой мыши по узлу ***CustomerService*** и в появившемся всплывающем меню выберем ***Test Web Service*** (рис. 4.10).

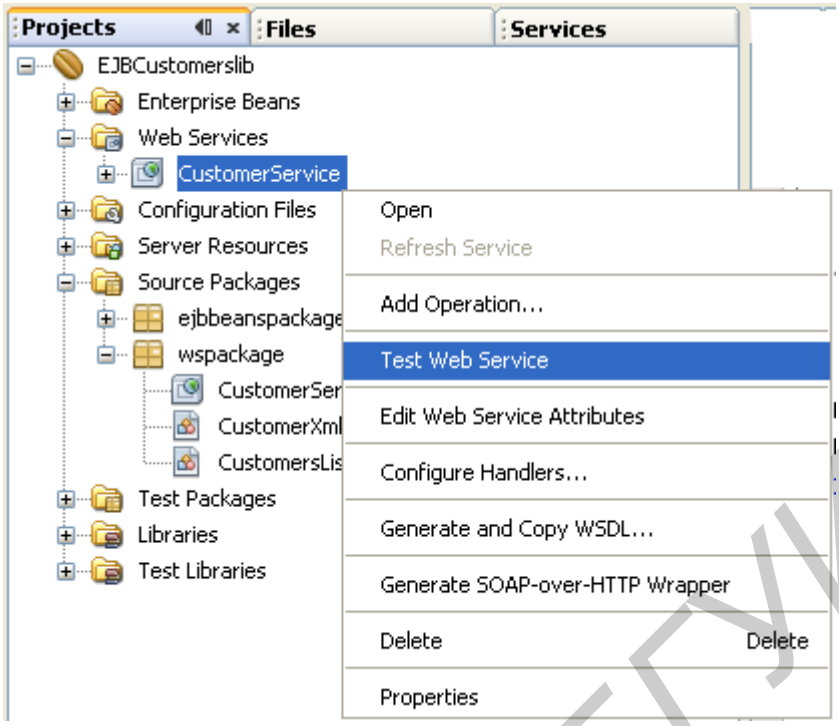


Рис. 4.10. Тестирование web-сервиса CustomerService

В появившемся окне браузера нажмем кнопку ***getCustomers,*** для того чтобы выполнить операцию web-сервиса и просмотреть результат.

4.4. Сервлеты. Технология JSP. Технология JSF

Сервлеты. Сервлетом (servlet) формально называется всякий класс, реализующий интерфейс Servlet из пакета ***javax.servlet***. Основу сервлета составляют метод ***init()***, выполняющий начальные действия сервлета, метод ***destroy()***, завершающий работу сервлета, и метод ***service()***, в котором заключена вся работа сервлета. Основная особенность сервлета заключается в том, что он работает не сам по себе, а в составе web-сервера. Когда от клиента приходит запрос к сервлету, web-сервер, а точнее один из его модулей, называемый web-контейнером, загружает сервлет и обращается сначала к его методу ***init()***, а затем к методу ***service()***. Метод ***init()*** выполняется только один раз, при загрузке сервлета, при следующих запросах к сервлету сразу выполняется метод ***service()***.

У метода ***service()*** два аргумента, его заголовок выглядит так:

```
public void service(ServletRequest req, ServletResponse resp)
throws ServletException, IOException;
```

Первый аргумент ***req*** – это ссылка на объект типа ***servletRequest***, созданный web-контейнером при получении запроса от клиента еще до обращения к

методу ***service().*** Объект содержит в себе запрос клиента, различные параметры и атрибуты которого сервлет может получить несколькими методами вида ***getParameter(String name)***, ***getAttribute(String name)***. Для получения содержимого запроса интерфейс ***ServletRequest*** предоставляет два входных потока: байтовый поток класса ***ServletInputStream*** и обычный символьный поток класса ***BufferedReader*** из пакета ***java.io***.

Второй аргумент ***resp*** – это ссылка на пустой объект типа ***servletResponse***, тоже созданный web-контейнером. Метод ***service()*** разбирает запрос, заключенный в объект ***req***, методами интерфейса ***ServietRequest***, обрабатывает его и результат обработки заносит в объект ***resp*** методами интерфейса ***ServletResponse***. Для этого интерфейс ***ServletResponse*** предоставляет байтовый выходной поток класса ***ServletOutputStream*** и символьный выходной поток класса ***PrintWriter***. После заполнения объекта ***resp*** сервлет заканчивает работу. Web-контейнер анализирует содержимое объекта ***resp***, формирует ответ и пересылает его клиенту через web-сервер, в котором работает web-контейнер. В пакете ***javax.servlet*** есть абстрактный класс ***GenericServlet***, реализующий все методы интерфейса s***ervlet,*** за исключением метода ***service()***.

Интерфейс ***servlet*** и его реализация ничего не знают о протоколе, по которому прислан запрос. Они не учитывают особенности протокола. Ввиду особой важности протокола HTTP для web-приложений, для обработки HTTP-запросов создан абстрактный класс ***HttpServlet*** – расширение класса ***GenericServlet***. В класс ***HttpServlet*** введены специализированные методы обработки различных HTTP-методов передачи данных:

```
protected void doDelete(HttpServletRequest req, HttpservletResponse resp);
protected void doGet(HttpServletRequest req, HttpservletResponse resp);
protected void doHead(HttpServletRequest req, HttpservletResponse resp);
protected void doOptions(HttpServletRequest req, HttpservletResponse resp);
protected void doPost(HttpServletRequest req, HttpservletResponse resp);
protected void doPut(HttpServletRequest req, HttpservletResponse resp);
protected void doTrace(HttpServletRequest req, HttpservletResponse resp);
```

Для того чтобы облегчить создание сервлетов, принимающих и отправляющих SOAP-послания, в пакете ***javax.xml.messaging*** приведено одно расширение класса ***HttpServlet***. Это абстрактный класс ***JAXMServlet***. Он предназначен для получения, обработки и отправки SOAP-посланий, оформленных средствами JAXM.

Технология Java Server Pages (JSP). Несмотря на то что сервлеты представляют достаточно мощные средства для разработки серверных приложений, разработка крупных программных проектов была затруднена – сервлеты позволяют строить серверные приложения на уровне запросов CGI, отдавая браузеру сформированные приложением web-страницы; алгоритмы формирования таких web-страниц должен был разрабатывать программист. Для облегчения работы программистов на смену технологии сервлетов и других технологий, подобных сервлетам, приходят технологии серверных страниц (Server Pages). Эта технология предполагает, что внутри будущей web-страницы размещаются исходные коды на каком-либо языке программирования: объявляются переменные, крутятся циклы, делаются выборки из БД и т. п. Когда же к странице обращаются, web-сервер не посылает ее сразу, а отдает на предмет достройки специальному обработчику (часто – это модуль web-сервера), который выполняет размещенный исходный код, заполняет данными, достраивает до готовой страницы и отдает назад web-серверу для отправки. Таким образом, подобная страница является на самом деле шаблоном для целого семейства страниц. Соответственно JSP – это технология, предполагающая разработку серверных страниц с использованием языка программирования Java.

Страница JSP – это сервлет наоборот: код разметки в ней можно писать как и в обычном HTML-файле, а настоящий java-код – в специальных тегах (<% ... %>). При первом обращении страница превращается в нормальный сервлет, который (как и полагается сервлету) компилируется, «ложится» в память и потом обрабатывает запросы.

Схемы разработки web-приложений. ***JSP Model1*** – это схема построения web-приложения (рис. 4.11), когда за доступ к данным отвечает разработанный под задачу набор javabeans, а все остальное берут на себя JSP-страницы. Javabeans в этом случае являются «моделью» данных, с которыми работает приложение (здесь используется паттерн Facade), а JSP-страницы, работающие с экземплярами этих javabeans, – «представлением», но при этом еще и производят всю обработку.

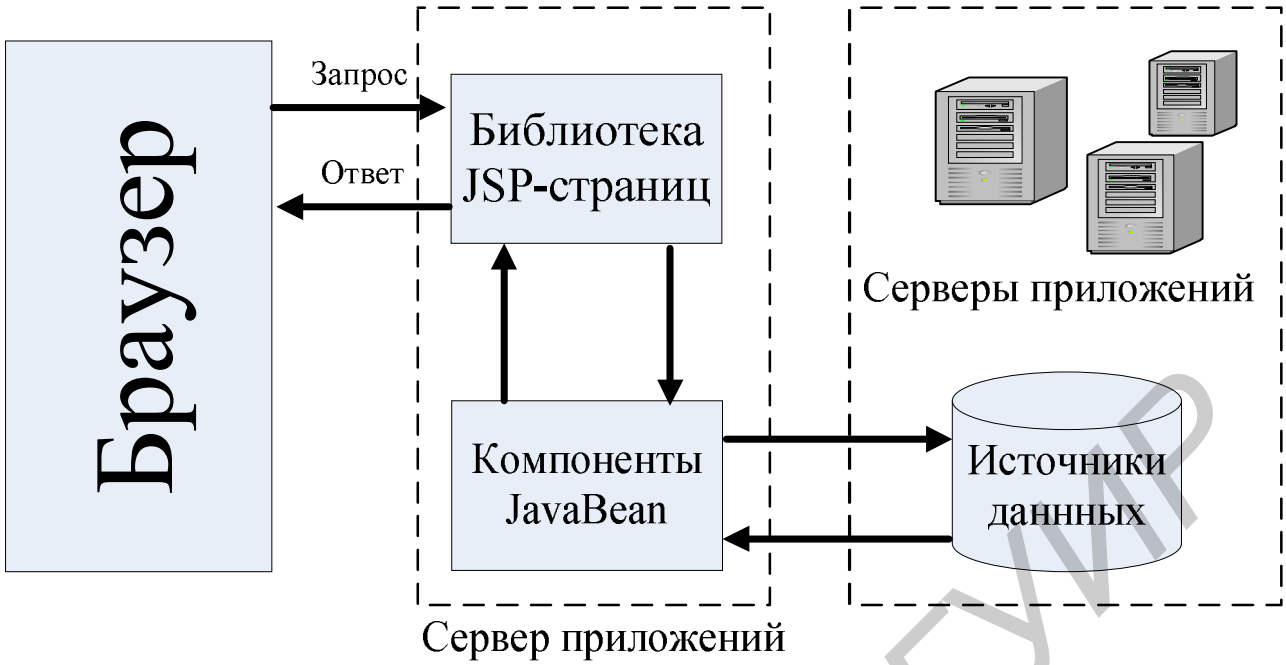


Рис. 4.11. Схема работы web-приложений JSP Model1

JSP Model2 – это улучшенная схема разработки web-приложений (рис. 4.12), использующая паттерн проектирования MVC, отличающаяся от JSP Model1 тем, что код, отвечающий за обработку данных внутри JSP-страницы при использовании модели JSP Model1, размещается в специальных классах-действиях, ориентированных на решение конкретных задач. Классы-действия подгружаются в специальный сервлет («сервлет действий»), который становится неотъемлемой частью приложения. В итоге: Javabeans отвечают за связь с данными («model»), JSP-страницы – за представление данных («view»), а сервлет действий – за обработку ("controller").

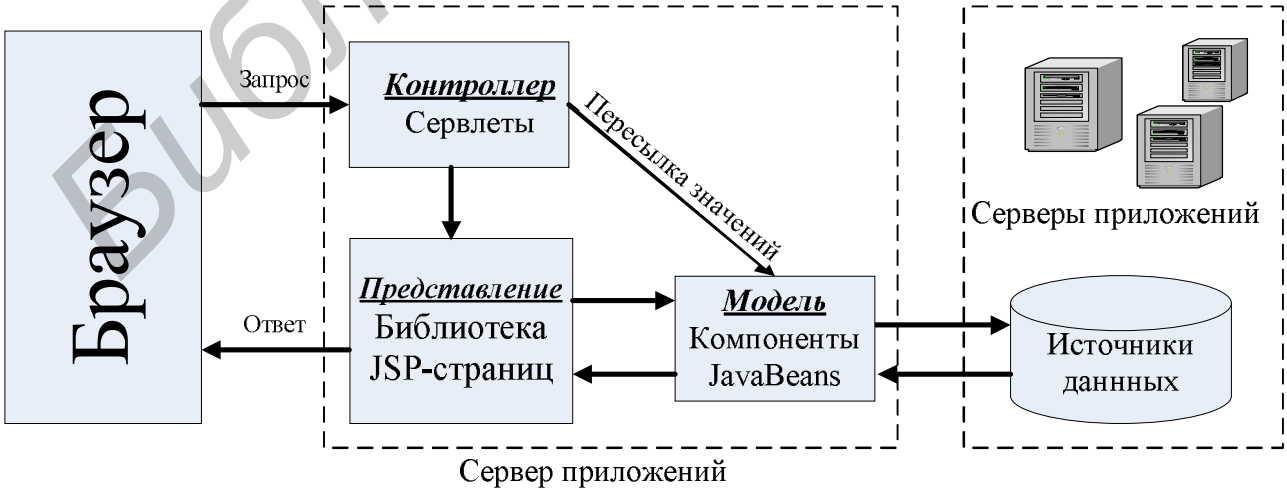


Рис. 4.12. Схема работы web-приложений JSP Model2

Model 2X – это схема разработки web-приложения, когда вывод сервлета/JSP в виде правильно оформленного XML-документа передается XML-трансформатору.

Технология Java Server Faces (JSF) использует подход на основе использования компонентов. Состояние компонентов пользовательского интерфейса сохраняется, когда пользователь запрашивает новую страницу, и затем восстанавливается, если запрос повторяется. Для отображения данных обычно используется JSP, но JSF можно приспособить и под другие технологии, например XUL.

Технология JSF включает:

- набор API для представления компонентов пользовательского интерфейса (UI) и управления их состоянием, обработкой событий и валидацией вводимой информации; определение навигации, а также поддержку интернационализации (i18n) и доступности (accessibility);
- специальную библиотеку JSP-тегов для выражения интерфейса JSF на JSP-странице.

4.5. Пример создания сервлета

Рассмотрим пример создания сервлета, который будет являться клиентом для созданного в подразд. 4.3 web-сервиса ***CustomerService***.

Выберете пункт меню ***File->New Project …***. В появившемся окне мастера (рис. 4.13) выберем категорию ***Java Web*** и тип проекта ***Web Application***. Затем нажмем кнопку ***Next.*** В следующем окне мастера необходимо в поле ***Project Name*** вписать имя проекта ***CustomerServlet*** и указать его местоположение на жестком диске. Осуществив эти действия, нажмем кнопку ***Next>***.

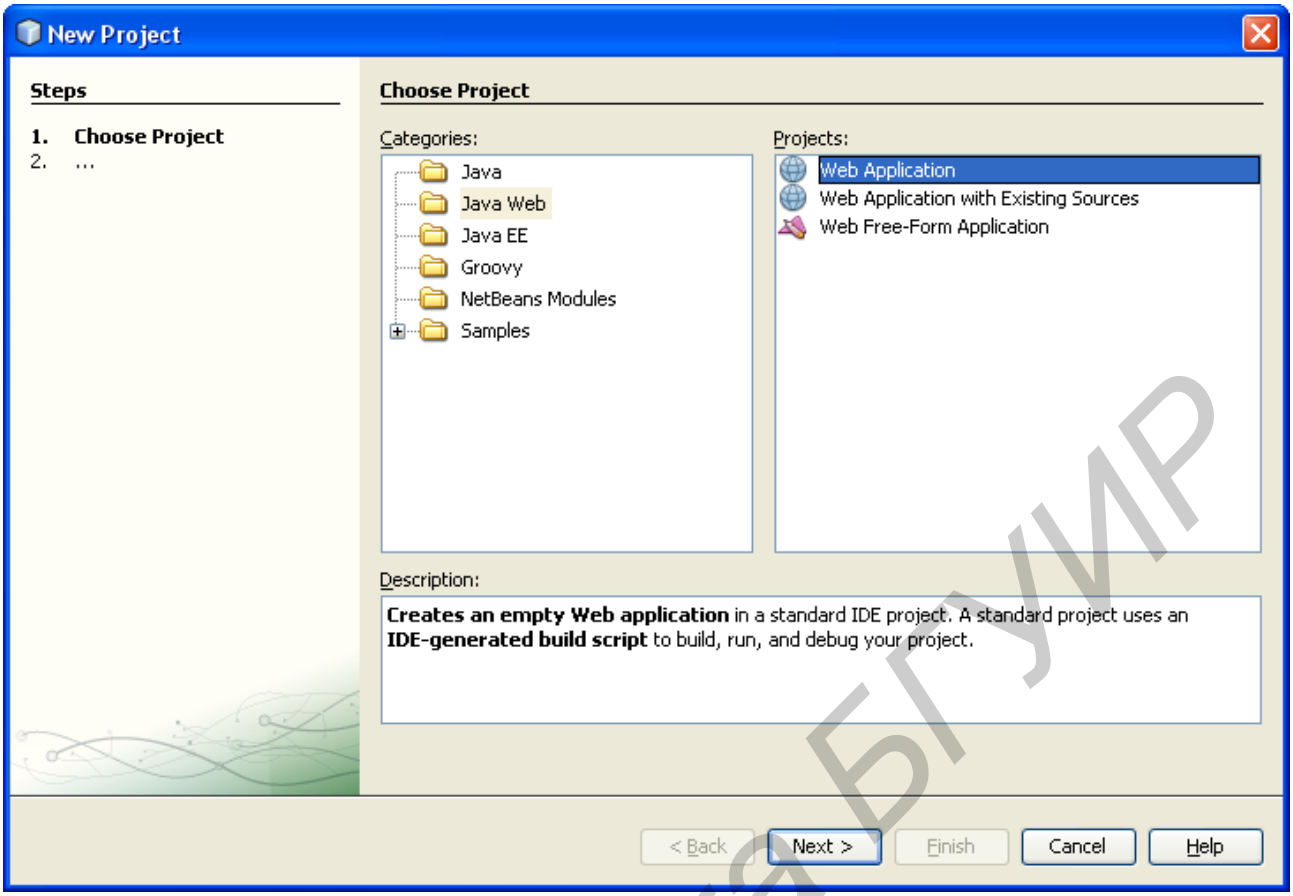


Рис. 4.13. Создание проекта web-приложения для сервлета

Следующее появившееся окно мастера ***Server and Settings*** уже знакомо по подразд. 4.3. В нем необходимо указать контейнер сервлетов, в данном случае – это Apache Tomcat, затем версию Java EE, с которой будем работать. В данном случае это пятая версия. И наконец, последним шагом необходимо будет указать контекст разрабатываемого web-приложения (пространство имен, в котором будут выполняться все сервлеты данного приложения). По умолчанию контекст будет совпадать с именем проекта, оставим его без изменений. Нажмем кнопку ***Finish***. Проект для сервлета создан.

Далее создадим класс сервлета, для этого щелкнем правой кнопкой мыши по имени нашего проекта в окне ***Projects*** NetBeans IDE. Затем в появившемся меню выберем пункт ***New->Servlet…***. В появившемся окне мастера ***New Servlet*** в поле ***Class Name*** введем имя сервлета ***CustomerServlet***, а в поле ***Package:*** введем имя пакета ***servlets***, в котором будет размещен класс сервлета, и затем нажмем кнопку ***Finish***. Класс сервлета создан.

Прежде чем писать код в классе сервлета, необходимо предварительно создать классы клиента web-сервиса ***CustomerService***, созданного в разд. 4.3.

Для этого следует предварительно убедиться, что на сервере приложений открыт и развернут проект ***EJBCustomerslib.*** Затем щелкнем правой кнопкой мыши по имени нашего проекта в окне ***Projects*** NetBeans IDE. В появившемся меню выберем пункт ***New->Web Service Client…***.

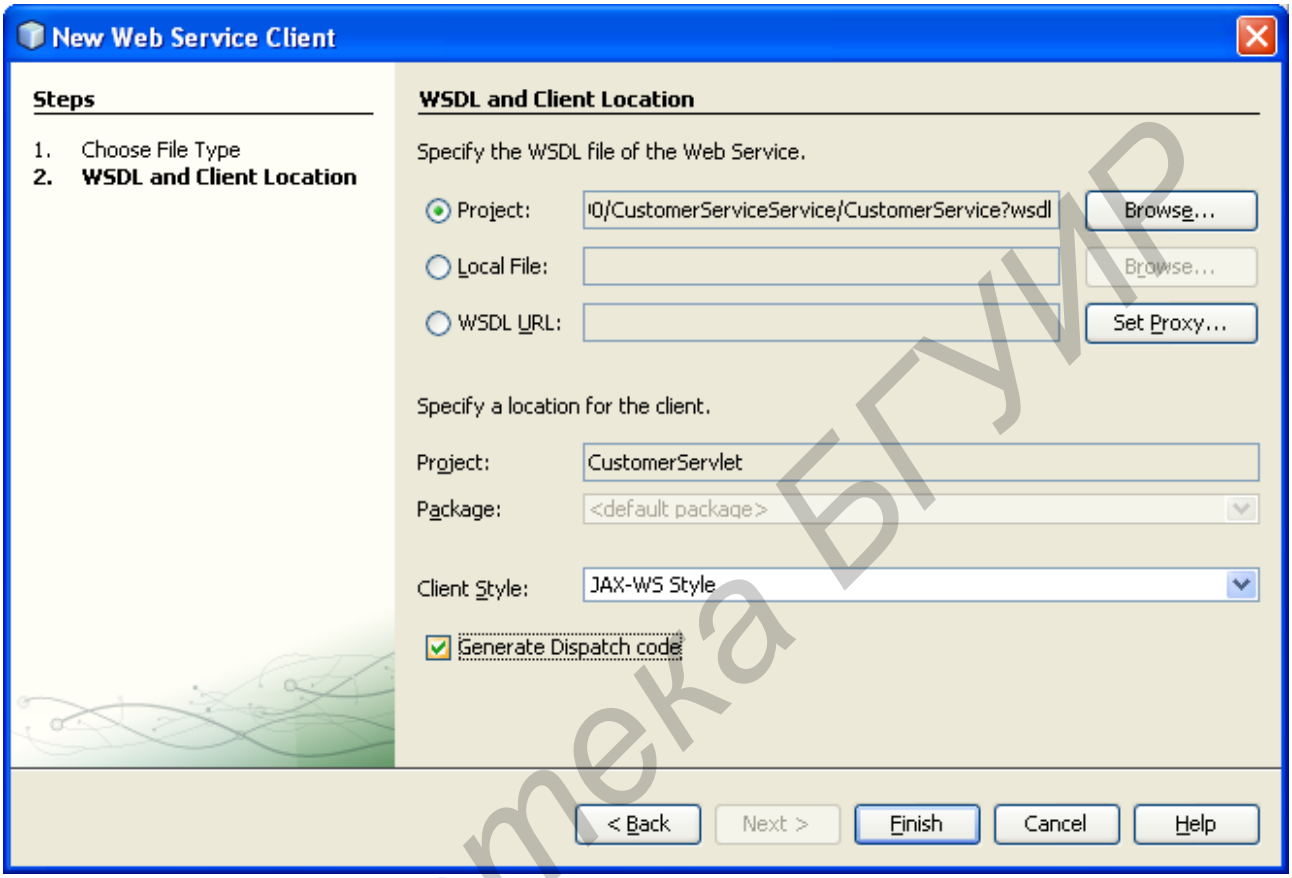


Рис. 4.14. Мастер создания клиента web-сервиса

В появившемся окне мастера (рис. 4.14) нажмем кнопку ***Browse…*** напротив поля ***Project:***, в появившемся окне из проекта ***EJBCustomerslib*** выберем web-сервис ***CustomerService*** и нажмем кнопку ***OK.*** Установим флажок ***Generate Dispatch code***. Стиль создаваемого клиента оставим ***JAX-WS Style***, как и предлагает NetBeans IDE. Затем нажмем кнопку ***Finish.*** Клиент web-сервиса ***CustomerService*** создан.

Перейдем в окно с исходным кодом класса ***CustomerServlet*** и модифицируем метод ***processRequest*** этого класса следующим образом:

```
    protected void processRequest(HttpServletRequest request, HttpServletRes-
ponse response)  throws ServletException, IOException
    {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
```

```
    try {
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet CustomerServlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Servlet CustomerServlet at " + request.getContextPath () +
"</h1>");
    wspackage.CustomerServiceService service =
new wspackage.CustomerServiceService();
    QName portQName = new QName("http://wspackage/" , "CustomerServicePort");
    String req = "<getCustomers  xmlns=\"http://wspackage/\"></getCustomers>";
    try {
    Dispatch<Source> sourceDispatch = null;
    sourceDispatch = service.createDispatch(portQName, Source.class,
    Service.Mode.PAYLOAD);
    Source result = sourceDispatch.invoke(new StreamSource(new StringRead-
er(req)));
    Result streamResult = new StreamResult(out);
    String p = "http://localhost:8084/CustomerServlet/customer.xsl";
    StreamSource styleSource = new StreamSource(p);
     Transformer t = TransformerFacto-
ry.newInstance().newTransformer(styleSource);
    t.transform(result, streamResult);
    }
    catch (Exception ex) {
    ex.printStackTrace();
    }
    out.println("</body>");
    out.println("</html>");
    } finally
    {
    out.close();
    }
    }
```

Рассмотрим код этого метода поподробнее. Метод позволяет подключиться к web-сервису ***CustomerService*** с помощью сгенерированного ранее клиента web-сервиса. Вызов метода web-сервиса создается простым перетягиванием мышью узла операции ***getCustomers()*** из поддерева ***Web Service References***, которое генерируется сразу после создания клиента web-сервиса. Для того чтобы обработать и вывести на страницу браузера результат запроса к web-сервису в виде HTML-документа, воспользуемся языком XSL(eXtensible Stylesheet Language, расширяемый язык таблиц стилей). Операции языка XSL, которые

для этого понадобятся, занесем в файл ***customer.xsl.*** Полный текст файла приведен ниже:

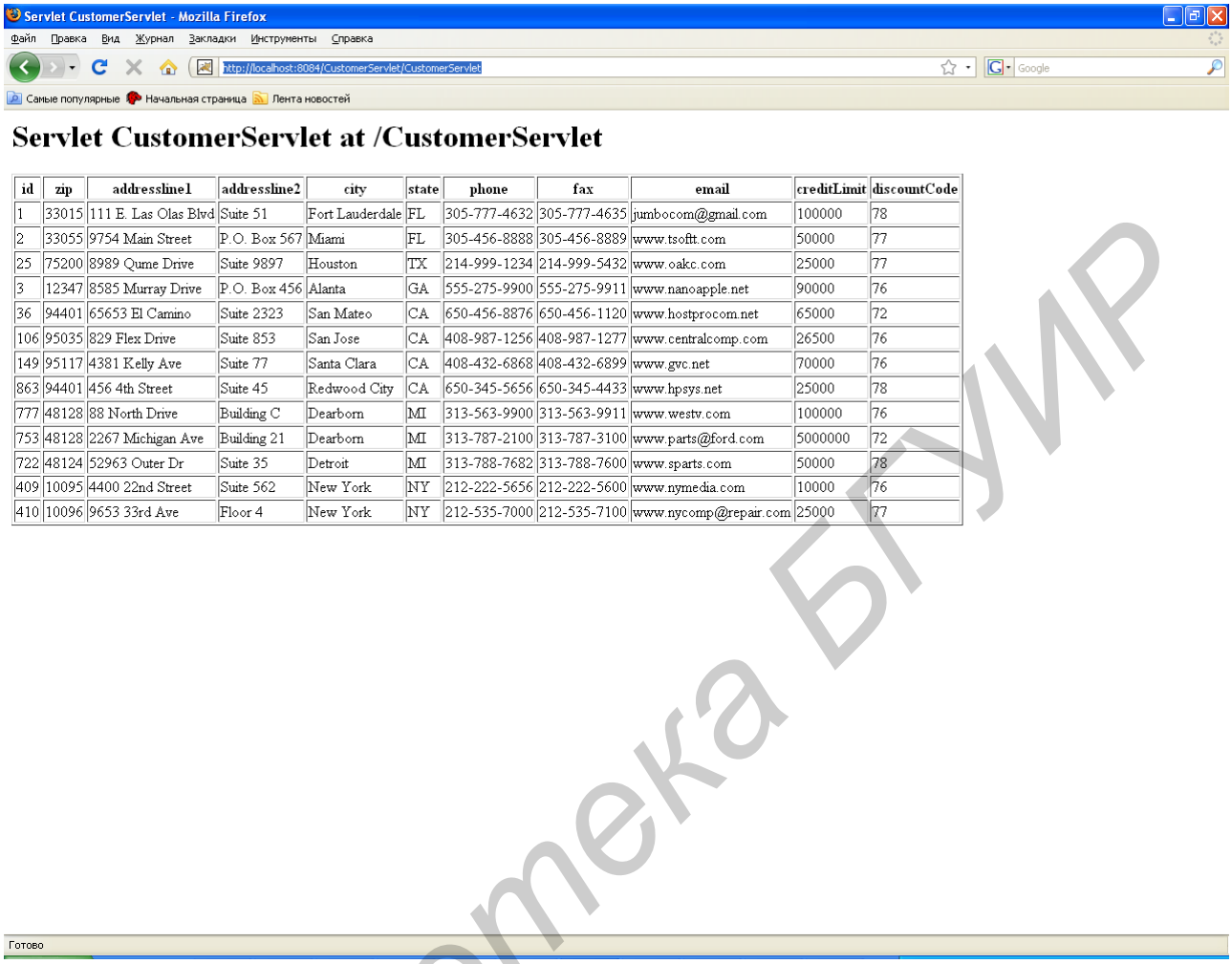
Файл customer.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html"/>
<xsl:template match="return">
<table border="1">
<tr> <th>id </th>
<th>zip</th>
<th>addressline1 </th>
<th>addressline2</th>
<th>city </th>
<th>state</th>
 <th>phone </th>
<th>fax</th>
<th>email</th>
<th>creditLimit </th>
<th>discountCode</th>
</tr>
<xsl:for-each select="lst">
<tr>
<td><xsl:value-of select="id/text()"/> </td>
<td><xsl:value-of select="zip/text()"/></td>
<td><xsl:value-of select="addressline1/text()"/> </td>
<td><xsl:value-of select="addressline2/text()"/></td>
<td><xsl:value-of select="city/text()"/> </td>
<td><xsl:value-of select="state/text()"/></td>
<td><xsl:value-of select="phone/text()"/> </td>
<td><xsl:value-of select="fax/text()"/></td>
<td><xsl:value-of select="email/text()"/></td>
<td><xsl:value-of select="creditLimit/text()"/> </td>
<td><xsl:value-of select="discountCode/text()"/></td>
</tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

Поместим этот файл в каталог ***web*** проекта ***CustomerServlet.***

Для запуска приложения предварительно установим соединение к базе данных ***sample,*** затем развернем проект ***EJBCustomerslib*** и наконец развернем приложение ***CustomerServlet.*** Запустим web-браузер и в строке адреса введем

http://localhost:8084/CustomerServlet/CustomerServlet и нажмем клавишу ***Enter***. Если всё сделано правильно, то увидим следующее (рис. 4.15).



id	zip	addressline1	addressline2	city	state	phone	fax	email	creditLimit	discountCode
1	33015	111 E. Las Olas Blvd	Suite 51	Fort Lauderdale	FL	305-777-4632	305-777-4635	jumbocom@gmail.com	100000	78
2	33055	9754 Main Street	P.O. Box 567	Miami	FL	305-456-8888	305-456-8889	www.tsoftt.com	50000	77
25	75200	8989 Qume Drive	Suite 9897	Houston	TX	214-999-1234	214-999-5432	www.oakc.com	25000	77
3	12347	8585 Murray Drive	P.O. Box 456	Alanta	GA	555-275-9900	555-275-9911	www.nanoapple.net	90000	76
36	94401	65653 El Camino	Suite 2323	San Mateo	CA	650-456-8876	650-456-1120	www.hostprocom.net	65000	72
106	95035	829 Flex Drive	Suite 853	San Jose	CA	408-987-1256	408-987-1277	www.centralcomp.com	26500	76
149	95117	4381 Kelly Ave	Suite 77	Santa Clara	CA	408-432-6868	408-432-6899	www.gvc.net	70000	76
863	94401	456 4th Street	Suite 45	Redwood City	CA	650-345-5656	650-345-4433	www.hpsys.net	25000	78
777	48128	88 North Drive	Building C	Dearborn	MI	313-563-9900	313-563-9911	www.westv.com	100000	76
753	48128	2267 Michigan Ave	Building 21	Dearborn	MI	313-787-2100	313-787-3100	www.parts@ford.com	5000000	72
722	48124	52963 Outer Dr	Suite 35	Detroit	MI	313-788-7682	313-788-7600	www.sparts.com	50000	78
409	10095	4400 22nd Street	Suite 562	New York	NY	212-222-5656	212-222-5600	www.nymedia.com	10000	76
410	10096	9653 33rd Ave	Floor 4	New York	NY	212-535-7000	212-535-7100	www.nycomp@repair.com	25000	77

Рис. 4.15. Результат запуска web-приложения CustomerServlet

4.6. Пример создания JSP-страницы

Модифицируем пример ***CustomerServlet***, созданный в подразд. 4.5. Откроем в редакторе NetBeans IDE файл ***index.jsp***. Из окна ***Projects*** IDE ссылку на этот файл можно получить, раскрыв поддерево ***Web Pages*** проекта ***Customer-Servlet.*** Модифицируем его следующим образом:

Файл index.jsp

```
<%--
Document   : index
Created on : Mar 23, 2009, 6:14:45 PM
Author     : Dmitry Kolb
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
    "http://www.w3.org/TR/html4/loose.dtd">
    <html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
    </head>
    <body>
    <center><h1>JSP Page Example!</h1>
    <%@page import="java.io.IOException" %>
    <%@page import="javax.xml.namespace.QName" %>
    <%@page import="javax.xml.transform.Source" %>
    <%@page import="javax.xml.ws.Dispatch" %>
    <%@page import="javax.xml.transform.stream.StreamSource"%>
    <%@page import="javax.xml.ws.Service"%>
    <%@page import="java.io.StringReader"%>
    <%@page import="javax.xml.transform.Result"%>
    <%@page import="javax.xml.transform.Transformer"%>
    <%@page import="javax.xml.transform.TransformerFactory"%>
    <%@page import="javax.xml.transform.dom.DOMSource"%>
    <%@page import="javax.xml.transform.stream.StreamResult"%>
    <%@page import="org.w3c.dom.Node"%>
    <%
    wspackage.CustomerServiceService service =
    new wspackage.CustomerServiceService();
    QName portQName = new QName("http://wspackage/" , "CustomerServicePort");
    String req = "<getCustomers  xmlns=\"http://wspackage/\"></getCustomers>";
    try {
    Dispatch<Source> sourceDispatch = null;
    sourceDispatch =
    service.createDispatch(portQName, Source.class, Service.Mode.PAYLOAD);
    Source result =
    sourceDispatch.invoke(new StreamSource(new StringReader(req)));
    Result streamResult = new StreamResult(out);
    String p = "http://localhost:8084/CustomerServlet/customer.xsl";
    StreamSource styleSource = new StreamSource(p) ;
    Transformer t = TransformerFacto-
ry.newInstance().newTransformer(styleSource);
    t.transform(result, streamResult);
    } catch (Exception ex) {
    ex.printStackTrace();
    }
    %>
    </center>
    </body>
    </html>
```

Заметим, что код созданной JSP-страницы очень похож на исходный код сервлета из подраздела 4.5. Развернем модифицированный проект на сервере Tomcat с учетом рекомендации по запуску, приведенной в подразд. 4.5. Для того чтобы увидеть результат проделанной работы, введем в адресной строке web-браузера ***http://localhost:8084/CustomerServlet/*** и нажмем клавишу ***Enter.***

4.7. Пример создания JSF-страницы

Попробуем реализовать более сложную версию интерфейса с использованием библиотеки JSF. Создадим проект так же, как в подразд. 4.5. В окне мастера ***Server and Settings*** после установки нужных опций нажмем кнопку ***Next>.*** На экране должно появиться окно установки библиотек проекта (рис. 4.16)/

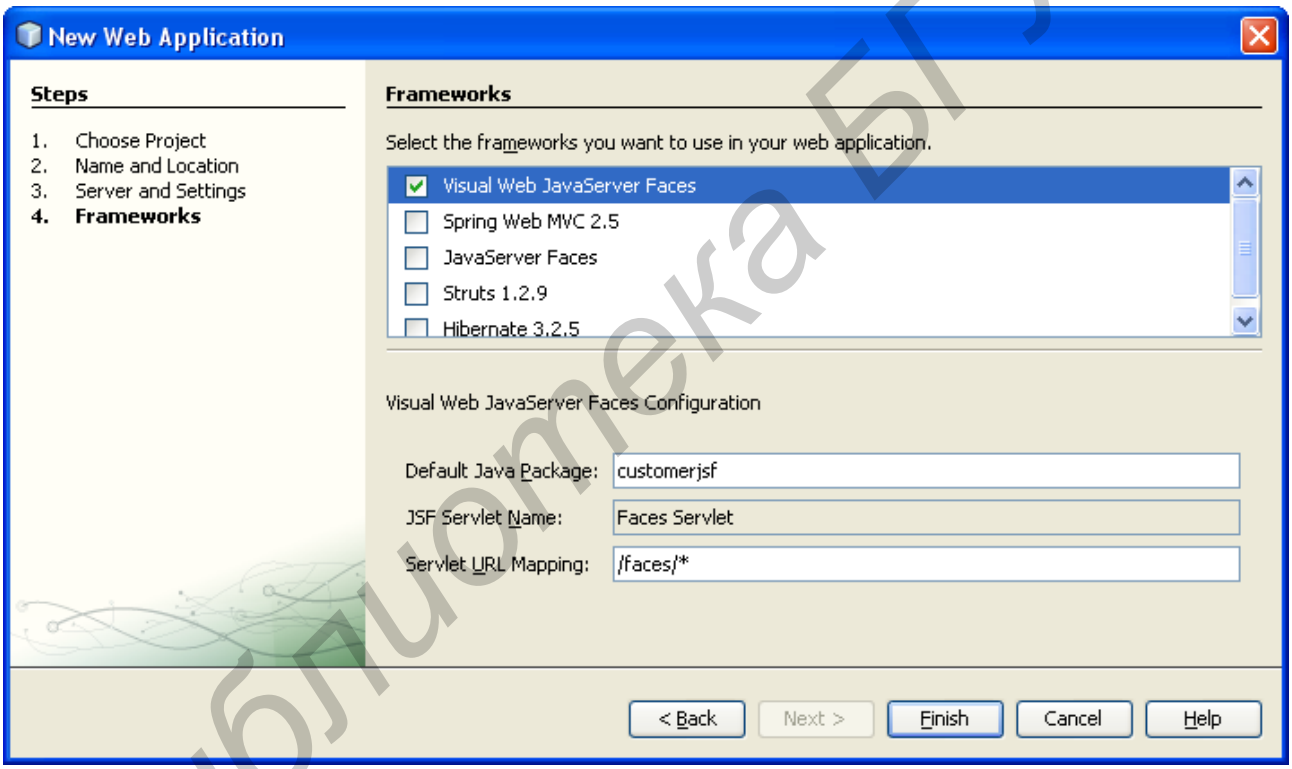


Рис. 4.16. Окно мастера установки библиотек проекта

Установим флажок напротив ***Visual Web Java Server Faces***, чтобы указать, что будет использоваться визуальная версия библиотеки JSF, позволяющая видеть в NetBeans IDE разрабатываемый интерфейс web-страницы. Затем нажмем кнопку ***Finish***.

Создадим клиента web-сервиса ***CustomerService*** для вновь созданного проекта тем же способом, что в примере из подразд. 4.5 раздела, не устанавливая флажок ***Generate Dispatch code***.

Перейдем во вкладку ***Palette*** NetBeans IDE, выберем категорию визуальных элементов ***Standart,*** найдем элемент ***Data Table*** и перетащим его в окно с активной вкладкой ***Design.*** Перейдем в окно ***Navigator,*** раскроем дерево с корневым элементом ***Page1.*** Найдем в дереве узел ***dataTable1***, раскроем поддерево этого узла, воспользовавшись комбинацией клавиш ***Ctrl+C*** и ***Ctrl+V***. Доведем количество столбцов в таблице до количества, равного количеству столбцов в таблице ***Customer*** базы данных ***sample***, как показано на рис. 4.17.

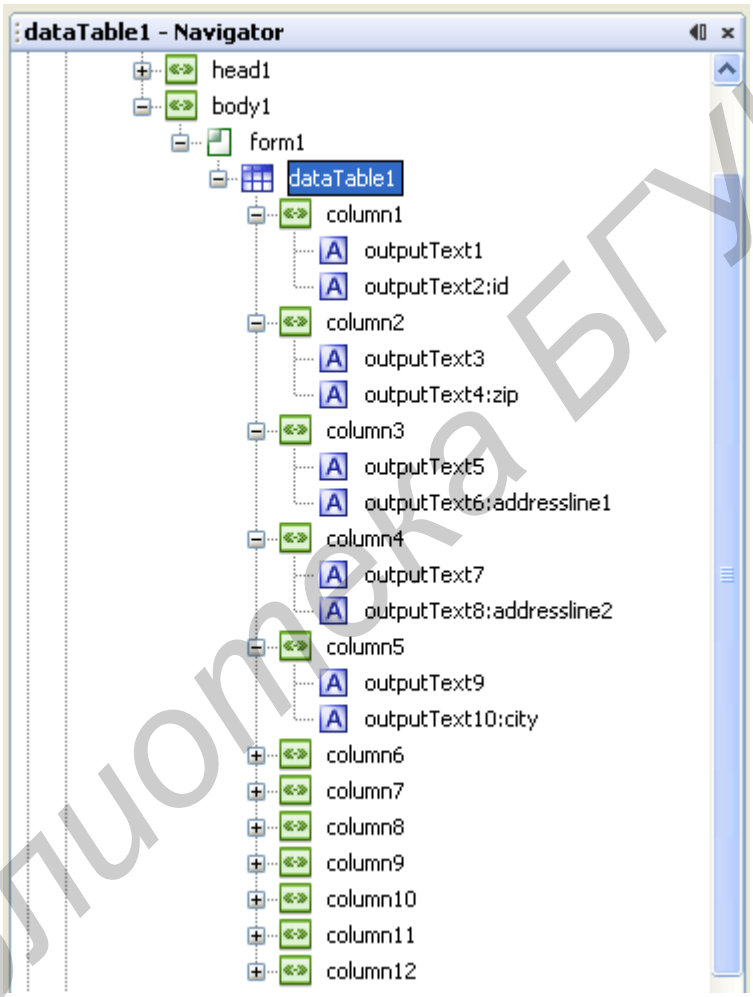


Рис. 4.17. Дерево элементов JSF-страницы

Теперь необходимо создать компонент, который будет содержать данные для строки таблицы, размещенной на JSF-странице. Добавим в пакет ***customerjsf*** класс ***CustomerBean*** со следующим содержимым:

Файл CustomerBean.java

```
package customerjsf;
import wspackage.CustomerXml;
/*
 *
 * @author Dmitry Kolb
 */
public class CustomerBean {
    protected Integer id;
    protected String zip;
    protected String name;
    protected String addressline1;
    protected String addressline2;
    protected String city;
    protected String state;
    protected String phone;
    protected String fax;
    protected String email;
    protected Integer creditLimit;
    protected Integer discountCode;

    public void XmlToBean(CustomerXml cx) {
        id = cx.getId();
        zip = cx.getZip();
        name = cx.getName();
        addressline1 = cx.getAddressline1();
        addressline2 = cx.getAddressline2();
        city = cx.getCity();
        state = cx.getState();
        phone = cx.getPhone();
        fax = cx.getFax();
        email = cx.getEmail();
        creditLimit = cx.getCreditLimit();
        discountCode = cx.getDiscountCode();
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer value) {
        this.id = value;
    }
   public String getZip() {
        return zip;
    }
    public void setZip(String value) {
        this.zip = value;
    }
    public String getName() {
```

```
        return name;
    }
    public void setName(String value) {
        this.name = value;
    }
    public String getAddressline1() {
        return addressline1;
    }
    public void setAddressline1(String value) {
        this.addressline1 = value;
    }
    public String getAddressline2() {
        return addressline2;
    }
    public void setAddressline2(String value) {
        this.addressline2 = value;
    }
    public String getCity() {
        return city;
    }
    public void setCity(String value) {
        this.city = value;
    }
    public String getState() {
        return state;
    }
    public void setState(String value) {
        this.state = value;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String value) {
        this.phone = value;
    }
    public String getFax() {
        return fax;
    }
    public void setFax(String value) {
        this.fax = value;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String value) {
        this.email = value;
    }
    public Integer getCreditLimit() {
```

```
            return creditLimit;
        }
        public void setCreditLimit(Integer value) {
            this.creditLimit = value;
        }
        public Integer getDiscountCode() {
            return discountCode;
        }

        public void setDiscountCode(Integer value) {
            this.discountCode = value;
        }
    }
```

Метод класса ***CustomerBean XmlToBean(…)*** переносит данные из класса клиента web-сервиса ***CustomerXml*** в класс, созданный для того, чтобы не зависеть от классов клиентского приложения и не использовать их при написании операций над данными JSF-страницы. После создания класса необходимо занести информацию о нём в конфигурационный файл JSF-приложения ***faces-config.xml***, который содержит информацию обо всех компонентах, используемых JSF-приложением. После модификации файл ***faces-config.xml*** будет выглядеть следующим образом:

Файл faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- =========== FULL CONFIGURATION FILE ================================
-->
<faces-config version="1.2"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">
    <managed-bean>
        <managed-bean-name>SessionBean1</managed-bean-name>
        <managed-bean-class>customerjsf.SessionBean1</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
    <managed-bean>
        <managed-bean-name>Page1</managed-bean-name>
        <managed-bean-class>customerjsf.Page1</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
    <managed-bean>
        <managed-bean-name>ApplicationBean1</managed-bean-name>
        <managed-bean-class>customerjsf.ApplicationBean1</managed-bean-
class>
```

```
        <managed-bean-scope>application</managed-bean-scope>
    </managed-bean>
    <managed-bean>
        <managed-bean-name>RequestBean1</managed-bean-name>
        <managed-bean-class>customerjsf.RequestBean1</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
        <managed-bean>
        <managed-bean-name>CustomerBean</managed-bean-name>
        <managed-bean-class>customerjsf.CustomerBean</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>
```

Далее перейдем к исходному коду класса ***Page1***, добавим туда следующие строки кода:

```
    private ListDataModel dtmCustomers = new ListDataModel();
    public ListDataModel getDtmCustomers() {
        return dtmCustomers;
    }
    public void setDtmCustomers(ListDataModel dtdm) {
        this.dtmCustomers = dtdm;
    }
```

Этот код позволит компоненту ***dataTable1*** брать данные из классов-списков базовых библиотек Java-платформы, используя модель данных списков (класс ***ListDataModel***) библиотеки JSF.

Затем модифицируйте метод ***prerender()*** следующим образом:

```
public void prerender() {
try { // Call Web Service Operation
wspackage.CustomerServiceService service =
new wspackage.CustomerServiceService();
wspackage.CustomerService port = service.getCustomerServicePort();
wspackage.CustomersList result = port.getCustomers();
List<CustomerBean> lst = new ArrayList<CustomerBean>();
for(CustomerXml c: result.getLst())
{
CustomerBean b = new CustomerBean();
b.XmlToBean(c);
lst.add(b);
}
dtmCustomers.setWrappedData(lst);
} catch (Exception ex) {
ex.printStackTrace();
}
}
```

Метод ***prerender()*** вызывается перед отрисовкой JSF-страницы, поэтому в ***prerender()*** можно разместить код, который будет изменять данные, выводимые на JSF-страницу. По сравнению с вызовами web-сервиса в предыдущих примерах код вызова web-сервиса в методе ***prerender()*** видоизменился. Теперь результат вызова метода получается не в виде XML-документа, как ранее, а в виде готовых Java-структур. После переноса данных, полученных от web-сервиса, в переменную ***lst***, содержащую список объектов класса ***CustomerBean***, необходимо передать переменную ***lst*** в метод ***setWrappedData(…)*** объекта класса ***ListDataModel***.

На последнем этапе необходимо связать написанный на языке Java код с XML-документом шаблона JSF-страницы. Для этого модифицируем файл ***Page1.jsp*** следующим образом :

Файл Page1.jsp

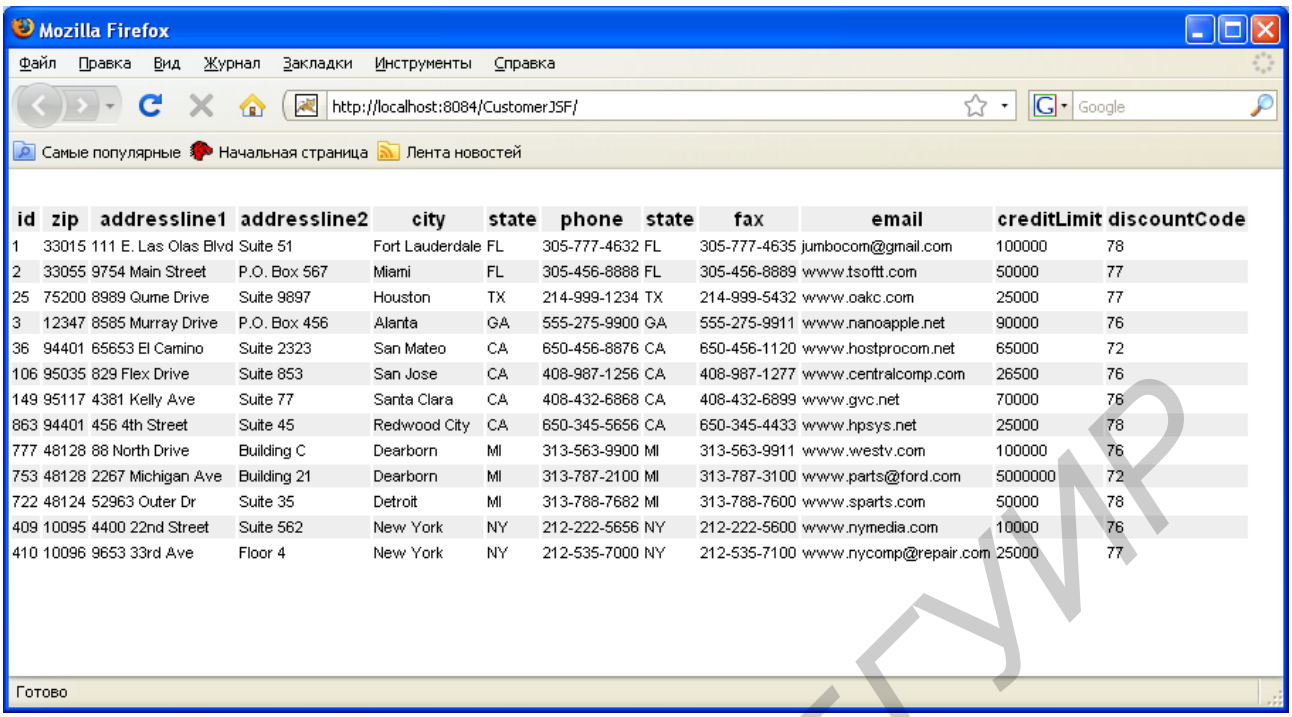
```
<?xml version="1.0" encoding="UTF-8"?>
<jsp:root version="2.1" xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html" xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:webuijsf="http://www.sun.com/webui/webuijsf">
<jsp:directive.page contentType="text/html;charset=UTF-8" pageEncod-
ing="UTF-8"/>
<f:view>
<webuijsf:page id="page1">
<webuijsf:html id="html1">
<webuijsf:head id="head1">
<webuijsf:link id="link1" url="/resources/stylesheet.css"/>
</webuijsf:head>
<webuijsf:body id="body1" style="-rave-layout: grid">
<webuijsf:form id="form1">
<h:dataTable headerClass="list-header" id="dataTable1"
rowClasses="list-row-even,list-row-odd"
style="left: 0px; top: 24px; position: absolute"
value="#{Page1.dtmCustomers}" var="CustomerBean">
<h:column id="column1">
<h:outputText id="outputText1" value="#{CustomerBean.id}"/>
<f:facet name="header">
<h:outputText id="outputText2" value="id"/>
</f:facet>
</h:column>
<h:column id="column2">
<h:outputText id="outputText3" value="#{CustomerBean.zip}"/>
<f:facet name="header">
<h:outputText id="outputText4" value="zip"/>
</f:facet>
</h:column>
```

```
<h:column id="column3">
<h:outputText id="outputText5" value="#{CustomerBean.addressline1}"/>
<f:facet name="header">
<h:outputText id="outputText6" value="addressline1"/>
</f:facet>
</h:column>
<h:column id="column4">
<h:outputText id="outputText7" value="#{CustomerBean.addressline2}"/>
<f:facet name="header">
<h:outputText id="outputText8" value="addressline2"/>
</f:facet>
</h:column>
<h:column id="column5">
<h:outputText id="outputText9" value="#{CustomerBean.city}"/>
<f:facet name="header">
<h:outputText id="outputText10" value="city"/>
</f:facet>
</h:column>
<h:column id="column6">
<h:outputText id="outputText11" value="#{CustomerBean.state}"/>
<f:facet name="header">
<h:outputText id="outputText12" value="state"/>
</f:facet>
</h:column>
<h:column id="column7">
<h:outputText id="outputText13" value="#{CustomerBean.phone}"/>
<f:facet name="header">
<h:outputText id="outputText14" value="phone"/>
</f:facet>
</h:column>
<h:column id="column8">
<h:outputText id="outputText15" value="#{CustomerBean.state}"/>
<f:facet name="header">
<h:outputText id="outputText16" value="state"/>
</f:facet>
</h:column>
<h:column id="column9">
<h:outputText id="outputText17" value="#{CustomerBean.fax}"/>
<f:facet name="header">
<h:outputText id="outputText18" value="fax"/>
</f:facet>
</h:column>
<h:column id="column10">
<h:outputText id="outputText19" value="#{CustomerBean.email}"/>
<f:facet name="header">
<h:outputText id="outputText20" value="email"/>
</f:facet>
</h:column>
```

```
<h:column id="column11">
<h:outputText id="outputText21" value="#{CustomerBean.creditLimit}"/>
<f:facet name="header">
<h:outputText id="outputText22" value="creditLimit"/>
</f:facet>
</h:column>
<h:column id="column12">
<h:outputText id="outputText23" value="#{CustomerBean.discountCode}"/>
<f:facet name="header">
<h:outputText id="outputText24" value="discountCode"/>
</f:facet>
</h:column>
</h:dataTable>
</webuijsf:form>
</webuijsf:body>
</webuijsf:html>
</webuijsf:page>
</f:view>
</jsp:root>
```

Модифицированные строки в листинге выделены жирным шрифтом. Структура страницы была определена на предыдущих шагах. На этом шаге свяжем поля таблицы с полями класса ***CustomerBean***, который моделирует строку данных таблицы, через атрибуты элементов ***<h:outputText> value,*** используя следующий шаблон: ***#{<Имя класса>.<поле класса>}.*** Класс, подставляемый в этот шаблон, должен удовлетворять соглашению Java Bean. Кроме этого, в атрибуте ***var*** элемента ***<h:dataTable>*** необходимо указать тип обрабатываемого таблицей элемента, в рассматриваемом случае это ***CustomerBean.***

Для запуска приложения предварительно установим соединение с базой данных ***sample,*** затем развернем проект ***EJBCustomerslib*** и наконец развернем приложение ***CustomerJSF.*** Запустим web-браузер, в строке адреса введем ***http://localhost:8084/CustomerJSF/*** и нажмем клавишу ***Enter***. Если все было сделано правильно, то в результате увидим выборку из таблицы ***Customer*** (рис. 4.18).



id	zip	addressline1	addressline2	city	state	phone	state	fax	email	creditLimit	discountCode
1	33015	111 E. Las Olas Blvd	Suite 51	Fort Lauderdale	FL	305-777-4632	FL	305-777-4635	jumbocom@gmail.com	100000	78
2	33055	9754 Main Street	P.O. Box 567	Miami	FL	305-456-8888	FL	305-456-8889	www.tsoftt.com	50000	77
25	75200	8989 Qume Drive	Suite 9897	Houston	TX	214-999-1234	TX	214-999-5432	www.oakc.com	25000	77
3	12347	8585 Murray Drive	P.O. Box 456	Alanta	GA	555-275-9900	GA	555-275-9911	www.nanoapple.net	90000	76
36	94401	65653 El Camino	Suite 2323	San Mateo	CA	650-456-8876	CA	650-456-1120	www.hostprocom.net	65000	72
106	95035	829 Flex Drive	Suite 853	San Jose	CA	408-987-1256	CA	408-987-1277	www.centralcomp.com	26500	76
149	95117	4381 Kelly Ave	Suite 77	Santa Clara	CA	408-432-6868	CA	408-432-6899	www.gvc.net	70000	76
863	94401	456 4th Street	Suite 45	Redwood City	CA	650-345-5656	CA	650-345-4433	www.hpsys.net	25000	78
777	48128	88 North Drive	Building C	Dearborn	MI	313-563-9900	MI	313-563-9911	www.westv.com	100000	76
753	48128	2267 Michigan Ave	Building 21	Dearborn	MI	313-787-2100	MI	313-787-3100	www.parts@ford.com	5000000	72
722	48124	52963 Outer Dr	Suite 35	Detroit	MI	313-788-7682	MI	313-788-7600	www.sparts.com	50000	78
409	10095	4400 22nd Street	Suite 562	New York	NY	212-222-5656	NY	212-222-5600	www.nymedia.com	10000	76
410	10096	9653 33rd Ave	Floor 4	New York	NY	212-535-7000	NY	212-535-7100	www.nycomp@repair.com	25000	77

Рис. 4.18. Результат работы примера JSF-страницы

4.8. Задание для лабораторной работы

Общее задание: Разработать простейший корпоративный сайт.

Серверная часть: необходимо создать базу данных в любой (по усмотрению студента) из существующих СУБД. Затем создать на сервере приложений EJB-компоненты, которые используют стратегии паттернов Session Facade и Business Delegate и дают возможность работать с базой данных. Затем необходимо создать web-сервис, который предоставлял бы интерфейс работы с базой данных для клиента.

Клиентская часть: клиент работает под управлением контейнера сервлетов Jakarta Tomcat и отображает данные, полученные от сервера, с помощью **сервлетов, JSP- или JSF-страниц.**

Для каждого справочника необходимо реализовать функции редактирования информации (добавление, изменение, удаление) и просмотра.

Рекомендуется для выполнения лабораторной работы использовать среду NetBeans IDE версии 6.5 и выше. В качестве заданий используются варианты заданий из подразд. 3.4.

4.9. Контрольные вопросы

1. Поясните понятие «интероперабельность».

2. Что такое сериализация?
3. Раскройте понятие Common Gateway Interface (CGI)
4. Поясните понятие «Model-view-controller».
5. Что такое JavaBeans?
6. Дайте определение Java Naming and Directory Interface (JNDI).
7. Раскройте понятие «Object-relational mapping».
8. Назовите основные положения технологии CORBA.
9. Перечислите основные компоненты CORBA.
10. Поясните, для чего необходима технология RMI.
11. Перечислите основные положения технологии Enterprise JavaBeans.
12. Раскройте понятия «сервер EJB» и «контейнер EJB».
13. Дайте определение компонента EJB. Поясните, что такое класс компонента.
14. Поясните понятия удаленного и внутреннего интерфейса.
15. Для чего необходим описатель развертывания?
16. Перечислите виды компонентов EJB и дайте их краткую характеристику.

ГЛОССАРИЙ

А

Авторизация *(англ. authorization) – 1. Процесс предоставления определенному лицу прав на выполнение некоторых действий. 2. Процесс подтверждения (проверки) прав пользователей на выполнение некоторых действий. Авторизацию не следует путать с аутентификацией.*

Аутентификация *(англ. authentication) – проверка принадлежности субъекту доступа предъявленного им идентификатора; подтверждение подлинности. Аутентификатор – это пакет, доказывающий, что клиент действительно является обладателем секретного ключа. Аутентификацию не следует путать с идентификацией.*

Г

Глобальная сеть, Глобальная вычислительная сеть, ГВС *(англ. Wide Area Network, WAN) – компьютерная сеть, охватывающая большие территории и включающая в себя десятки и сотни тысяч компьютеров.*

Д

Домен *– в модели OSI – административная часть распределенной системы или домен управления службой каталогов.*

Домен Windows NT *– в операционных системах семейства Microsoft Windows NT – группа компьютеров, управляемых централизованно. Также: домен (active directory).*

Доменное имя *– определенная зона в системе Интернет, выделенная владельцу домена.*

И

Идентификация *в информационной безопасности – присвоение субъектам и объектам идентификатора и (или) сравнение идентификатора с перечнем присвоенных идентификаторов. Например идентификация по штрихкоду.*

Инициализация *(от англ. initialization, инициирование) – создание, активация, подготовка к работе, определение параметров. Приведение программы или устройства в состояние готовности к использованию. Термин употребляется как для программных, так и для аппаратных средств. Действие инициализации направлено извне по отношению к инициализируемому объекту (программе, устройству) и необходимо для определения параметров и правил работы с ним.*

Интероперабельность *(от англ. interoperability) – способность системы к взаимодействию с другими системами.*

К

Кодирование *информации – процесс преобразования сигнала из формы, удобной для непосредственного использования информации, в форму, удобную для передачи, хранения или автоматической переработки.*

Консоль – *в современных компьютерах комплект устройств интерактивного ввода-вывода, присоединённых к компьютеру непосредственно (не через сеть): дисплей, клавиатура, мышь. Консольный сеанс в многопользовательских операционных системах – это сеанс, осуществляемый человеком, сидящим непосредственно перед компьютером (в противоположность сеансу удалённого доступа, например через telnet, ssh, X Window System, RDP и т. п.). Данная трактовка термина консоль безотносительна к типу пользовательского интерфейса: текстовому (CUI) или графическому (GUI).*

Компьютерная сеть *(вычислительная сеть, сеть передачи данных) – система связи компьютеров и/или компьютерного оборудования (серверы, маршрутизаторы и другое оборудование). Для передачи информации могут быть использованы различные физические явления, как правило, различные виды электрических сигналов, световых сигналов или электромагнитного излучения.*

Клиент *– компьютер или процесс, который пользуется ресурсами или услугами другого компьютера или процесса.*

Л

Локальная сеть, локальная вычислительная сеть *(ЛВС, сленг. локалка; англ. Local Area Network, LAN) – компьютерная сеть, покрывающая обычно относительно небольшую территорию или небольшую группу зданий (дом, офис, фирму, институт*

М

Маска подсети *или маска сети – в терминологии сетей TCP/IP так называется битовая маска, определяющая, какая часть IP-адреса узла сети относится к адресу сети, а какая – к адресу самого узла в этой сети.*

П

Порт **(*сетевой порт*)** *– параметр протоколов TCP и UDP, определяющий назначение пакетов данных в формате IP, передаваемых на хост по сети. Это условное число от 1 до 65535, позволяющее различным программам, выполняемым на одном компьютере, получать данные независимо друг от друга. Каждая программа обрабатывает данные, поступающие на определённый порт (иногда говорят, что программа «слушает» этот номер порта). Обычно за некоторыми распространёнными сетевыми протоколами закреплены стандартные номера портов (например, web-серверы обычно принимают данные по протоколу HTTP на TCP-порт 80), хотя в большинстве случаев программа может использовать любой порт.*

Протокол *– это формализованное правило, определяющее последовательность и формат сообщений, которыми обмениваются сетевые компоненты, лежащие на одном уровне модели OSI, но в разных узлах.*

С

Сервер *– компьютер или процесс, который предоставляет ресурсы или услуги другому компьютеру или процессу.*

Сервис **(*служба*)** *– программное обеспечение, расположенное на удаленной машине, к которому предоставлена возможность доступа клиентского программного обеспечения.*

Сериализация *– это техника обработки объектов для последующей передачи всего объекта в потоках данных, которая позволяет записывать и считывать объекты целиком.*

Сетевой интерфейс *– формализованные правила, определяющие взаимодействие сетевых компонентов соседних уровней одного узла. Сетевой интерфейс определяет набор сервисов, предоставляемых данным уровнем модели OSI соседнему уровню модели OSI.*

Стек протоколов *– это иерархически организованный набор протоколов, достаточный для организации взаимодействия узлов в сети.*

Т

Транзакция (*англ. transaction*) *– в информатике группа последовательных операций, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, с соблюдением целостности данных и независимо от других параллельно идущих транзакций, либо не выполнена вообще, и тогда она не должна произвести никакого эффекта. Транзакции обрабатываются транзакционными системами, в процессе работы которых создаётся история транзакций.*

Ш

Шлюз *– в маршрутизируемых протоколах адрес маршрутизатора, на который отправляется трафик, для которого невозможно определить маршрут исходя из таблицы маршрутизации.*

А

ARP (*англ. Address Resolution Protocol – протокол разрешения адресов*) *– протокол канального уровня (англ. Link layer), предназначенный для преобразования IP-адресов (адресов сетевого уровня) в MAC-адреса (адреса канального уровня) в сетях TCP/IP. Он определён в RFC 826.*

B

Base64 *– буквально означает – позиционная система счисления с основанием 64. Здесь 64 – это наибольшая степень двойки (*2^6*), которая может быть представлена с использованием печатных символов ASCII. Эта система широко используется в электронной почте для представления бинарных файлов в тексте письма (транспортное кодирование). Все широко известные варианты, представленные под названием Base64, используют символы A-Z, a-z и 0-9, что составляет 62 знака; для остальных двух знаков в разных системах используются различные символы.*

BSD (*англ. Berkeley Software Distribution*) *– система распространения программного обеспечения в исходных кодах, созданная для обмена опытом между учебными заведениями. Особенностью пакетов ПО BSD была специальная лицензия BSD, которую кратко можно охарактеризовать так: весь исходный код – собственность BSD, все правки – собственность их авторов. В данный момент термин BSD чаще всего употребляется как синоним BSD-UNIX общего названия вариантов UNIX, восходящих к дистрибутивам университета Беркли. К семейству BSD относятся: NetBSD, FreeBSD, OpenBSD, MirBSD, DragonFly BSD, PC-BSD, DesktopBSD, SunOS, TrueBSD, Frenzy, Ultrix и частично Darwin (ядро Mac OS X).*

C

CGI (*от англ. Common Gateway Interface*) *– это спецификация обмена данными между прикладной программой, выполняемой по запросу пользователя, и HTTP-сервером, который данную программу запускает. До появления CGI новые функции нужно было внедрять непосредственно в сервер. CGI позволила разрабатывать программы независимо от сервера, а механизм передачи им управления и данных был унаследован от программирования в среде командной строки. Последнее резко сократило трудозатраты на разработку приложений, так как не надо было программировать интерфейс пользователя: его функции выполняли формы.*

DHCP (*англ. Dynamic Host Configuration Protocol – протокол динамической конфигурации узла*) *– это сетевой протокол, позволяющий компьютерам автоматически получать IP-адрес и другие параметры, необходимые для работы в сети TCP/IP. Данный протокол работает по модели «клиент–сервер». Для автоматической конфигурации на этапе конфигурации сетевого устройства компьютер–клиент обращается к т. н. серверу DHCP и получает от него нужные параметры. Сетевой администратор может задать диапазон адресов, распределяемых сервером среди компьютеров. Это позволяет избежать ручной настройки компьютеров сети и уменьшает количество ошибок. Протокол DHCP используется в большинстве крупных (и не очень) сетей TCP/IP.*

DNS (*англ. Domain Name System – система доменных имён*) *– компьютерная распределённая система для получения информации о доменах. Чаще всего исполь-*

зуется для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почты, обслуживающих узлах для протоколов в домене.

IETF (*англ. Internet Engineering Task Force, Специальная комиссия интернет-разработок*) – *открытое международное сообщество проектировщиков, учёных, сетевых операторов и провайдеров, созданное IAB в 1986 г., которое занимается развитием протоколов и архитектуры Интернета.*

J

JavaBeans – *многократно используемые программные компоненты для Java, которыми можно манипулировать визуально в среде разработки. Это Java-классы, реализованные в соответствии с определенным соглашением. Как правило, используются для инкапсуляции множества объектов в один объект (bean). JavaBean – это «Object Plain Old Java» (POJO), сеарилизуемый, не имеющий аргументов в конструкторе и допускающий доступ к свойствам, используя методы «геттеры» и «сеттеры», объект.*

JNDI **(***от англ. - Java Naming and Directory Interface***)** – *это API-интерфейс или библиотека, которая обеспечивает приложения методами для связывания имен с объектами и поиска этих объектов в каталоге.*

ICMP (*англ. Internet Control Message Protocol – межсетевой протокол управляющих сообщений*) – *сетевой протокол, входящий в стек протоколов TCP/IP. В основном ICMP используется для передачи сообщений об ошибках и других исключительных ситуациях, возникших при передаче данных. Также на ICMP возлагаются некоторые сервисные функции.*

M

MAC-адрес (*от англ. Media Access Control – управление доступом к носителю*) – *это уникальный идентификатор, сопоставляемый с различными типами оборудования для компьютерных сетей.*

MVC (*от англ. Model-view-controller – «Модель-представление-поведение», «Модель-представление-контроллер»*) – *архитектура программного обеспечения, в которой модель данных приложения, пользовательский интерфейс и управляющая логика разделены на три отдельных компонента так, что модификация одного из компонентов оказывает минимальное воздействие на другие компоненты.*

O

ORM *(от англ. Object-relational mapping – «Объектно-реляционная проекция») – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных», т. е. позволяет записывать объекты программы в реляционную базу данных, отображая объект и его представления в виде набора таблиц.*

URI (*англ. Uniform Resource Identifier*) – *унифицированный* (*единообразный*) *идентификатор ресурса. URI – это последовательность символов, идентифицирующая абстрактный или физический ресурс. Ранее назывался Universal Resource Identifier – универсальный идентификатор ресурса.*

URL – *единый указатель ресурсов* (*англ. URL –Uniform Resource Locator*) – *единообразный локатор* (*определитель местонахождения*) *ресурса. Ранее назывался Universal Resource Locator – универсальный локатор ресурса. URL – это стандартизированный способ записи адреса ресурса в сети Интернет.*

R

RFC (*от англ. Request for Comments*) – *запрос комментариев – документ из серии пронумерованных информационных документов сети Интернет, содержащих технические спецификации и стандарты, широко применяемые во всемирной сети. В настоящее время первичной публикацией документов RFC занимается IETF под эгидой открытой организации общества Интернет (англ. Internet Society, ISOC). Правами на RFC обладает именно Общество Интернета.*

RIP2 – *разработанный в 1994 г. протокол* (*RFC 2453*), *который является расширением протокола RIP, обеспечивающим передачу дополнительной маршрутной информации в сообщениях RIP и повышающим уровень безопасности. Routing Information Protocol – протокол 3-го уровня модели OSI. RIP – так называемый дистанционно-векторный протокол, который оперирует хопами в качестве метрики маршрутизации.*

X

xDSL – *семейство технологий, позволяющих значительно расширить пропускную способность абонентской линии местной телефонной сети путём использования эффективных линейных кодов и адаптивных методов коррекции искажений линии на основе современных достижений микроэлектроники и методов цифровой обработки сигнала.*

В аббревиатуре xDSL символ ***«x»*** *используется для обозначения первого символа в названии конкретной технологии, а DSL обозначает цифровую абонентскую линию DSL* (*англ. Digital Subscriber Line цифровая абонентская линия*). *Технологии xDSL позволяют передавать данные со скоростями, значительно превышающими те скорости, которые доступны даже самым лучшим аналоговым и цифровым модемам. Эти технологии поддерживают передачу голоса, высокоскоростную передачу данных и видеосигналов, создавая при этом значительные преимущества как для абонентов, так и для провайдеров. Многие технологии xDSL позволяют совмещать высокоскоростную передачу данных и передачу голоса по одной и той же медной паре. Существующие типы технологий xDSL различаются в основном по используемой форме модуляции и скорости передачи данных.*

XUL *(от англ. XML User Interface Language) – язык разметки для создания динамических пользовательских интерфейсов на основе XML. XUL разрабатывается в рамках проекта Mozilla и является частью платформы XULRunner.*

ЛИТЕРАТУРА

1. Гери, Д. М. Java Server Pages. Библиотека профессионала / Д. М. Гери; пер. с англ. – М. : Изд. дом «Вильямс», 2002. – 448 с.

2. Документация по среде NetBeans и поддержка. [Электронный ресурс]. – 2009. Режим доступа : http://www.netbeans.org/kb/index_ru.html.

3. Закер, К. Компьютерные сети. Модернизация и поиск неисправностей / К. Закер ; пер. с англ. – СПб. : БХВ-Петербург, 2004. – 1008 с.

4. Олифер, В. Г. Компьютерные сети: принципы, технологии, протоколы : учеб. для вузов / В. Г. Олифер, Н. А. Олифер. – 3-е изд. – СПб. : Питер, 2007. – 958 с.

5. Таненбаум, Э. Компьютерные сети / Э. Танненбаум. – 4-е изд. – СПб. : Питер, 2007. – 1992 с. (Серия «Классика computer science»).

6. Хабибулин, И. Ш. Разработка Web-служб средствами Java / И. Ш. Хабибулин. – СПб. : БХВ-Петербург, 2003. – 400 с.

7. Хелд, Г. Технологии передачи данных / Г. Хелд. – 7-е изд. – СПб. : Питер; Киев : Изд. гр. ИРМб 2003. – 720 с. (Серия «Классика computer science»).

8. Элсенпитер, Р. Windows XP Professional. Администрирование сетей / Р. Элсенпитер, Т. Дж. Велт.; пер. с англ. – М. : СПЭКОМ, 2006. – 560 с. (Серия «Справочник администратора»).

9. Java: основы Web-служб / Г. Беккет [и др.]; пер. с англ. – М. : КУДИЦ-ОБРАЗ, 2004. – 464 с.

10. Core JavaServer Faces By David Geary, Cay Horstmann Publisher: Addison Wesley Pub Date: June 15, 2004 Pages: 522.

11. Deepal Jayasinghe. Quickstart Apache Axis2 Copyright © 2008 Packt Publishing, 168 p.

12. Henning, M. The Rise and Fall of CORBA, ACM Queue / M. Henning. – V. 4, № 5. – June, 2006.

13. OASIS Reference Model for Service Oriented Architecture V 1.0. [Электронный ресурс]. – 2006. – Режим доступа : http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

14. The Java EE 5 Tutorial For Sun Java System Application Server 9.1 Copyright 2007 SunMicrosystems, Inc.

15. XSLT Cookbook, 2nd Edition By Sal Mangano Publisher: O'Reilly Pub Date: December 2005 Pages: 774

16. RFC 821, Simple Mail Transfer Protocol (SMTP), by J. Postel, August 1982.

17. RFC 854, Telnet Protocol Specification by J. Postel, J. Reynolds, May 1983.

18. RFC 862, Echo Protocol by J. Postel, May 1983.

19. RFC 868, Time Protocol by J. Postel, K. Harrenstien, May 1983.

20. RFC 867, Daytime Protocol by J. Postel, May 1983.

21. RFC 959, File Transfer Protocol (FTP), by J. Postel and J. Reynolds, October 1985.

22. RFC 977, Network News Transfer Protocol, by Brian Kantor and Phil Lapsley, February 1986.

23. RFC 1282, BSD Rlogin, by Brian Kantor, December 1991.

24. RFC 1288, The Finger User Information Protocol by D. Zimmerman, December 1991.

25. RFC 1869, SMTP Service Extensions, by J. Klensin, N. Freed, M. Rose, M. Stefferud, D. Crocker, November 1995.

26. RFC 1939, Post Office Protocol Version 3, by J. Myers, May 1996.

27. RFC 2554, SMTP Service Extension for Authentication, by J. Myers, Macrh 1999.

28. RFC 1945, Hypertext Transfer Protocol version 1.0 (HTTP/1.0), by T. Berners-Lee, R. Fielding, H. Frystyk, May 1996.

29. RFC 2616, Hypertext Transfer Protocol version 1.1 (HTTP/1.1), by R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, June 1999.

30. RFC 2660, The Secure HyperText Transfer Protocol, by E. Rescorla, A. Schiffman, August 1999.

31. RFC 2980, Common NNTP Extensions, by S. Barber, October 2000.

32. RFC 4251 The Secure Shell (SSH) Protocol Architecture T. Ylonen, C. Lonvick, January 2006.

33. RFC 3501, Internet message access protocol version 4rev1 by M. Crispin, March 2003.

Учебное издание

Голенков Владимир Васильевич
Колб Дмитрий Григорьевич
Ивашенко Валерьян Петрович

АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СЕТЕЙ И ОСНОВЫ ЗАЩИТЫ ИНФОРМАЦИИ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Л. А. Шичко*
Корректор *Е. Н. Батурчик*
Компьютерная верстка *В. М. Задоля*

Подписано в печать 28.09.2010. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 6,63. Уч.-изд. л. 7,3. Тираж 100 экз. Заказ. 268.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6