СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

Серия основана в 2001 году

О.В. ИСАЧЕНКО

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

УЧЕБНОЕ ПОСОБИЕ

2-е издание, исправленное и дополненное

Рекомендовано Межрегиональным учебно-методическим советом профессионального образования в качестве учебного пособия для учебных заведений, реализующих программу среднего профессионального образования по специальностям 09.02.01 «Компьютерные системы и комплексы», 09.02.02 «Компьютерные сети», 09.02.03 «Программирование в компьютерных системах» (протокол № 16 от 28.10.2019)

Электронно-Библиотечная Система
znanium.com

Москва
ИНФРА-М
2021

УДК 004.7(075.32)
ББК 32.973.202-018.2я723
И85

Исаченко О.В.

И85 Программное обеспечение компьютерных сетей : учебное пособие / О.В. Исаченко. — 2-е изд., испр. и доп. — Москва : ИНФРА-М, 2021. — 158 с. — (Среднее профессиональное образование).

ISBN 978-5-16-015447-3 (print)
ISBN 978-5-16-108134-1 (online)

Учебное пособие включает сведения об основных видах программного обеспечения компьютерных сетей, современных Web-технологиях и популярных средствах разработки Web-приложений.

Соответствует требованиям федеральных государственных образовательных стандартов среднего профессионального образования последнего поколения.

Предназначено для студентов системы среднего профессионального образования, обучающихся по укрупненной группе специальностей 09.02.00 «Информатика и вычислительная техника» и изучающих дисциплины «Программное обеспечение компьютерных сетей» и «Программное обеспечение компьютерных сетей и Web-серверов». Может быть полезно и для учащихся высших учебных заведений.

УДК 004.7(075.32)
ББК 32.973.202-018.2я723

ISBN 978-5-16-015447-3 (print)
ISBN 978-5-16-108134-1 (online)

ПРЕДИСЛОВИЕ

Сфера информационных технологий является на сегодняшний день наиболее динамически развивающейся областью науки и техники, что приводит к расширению поля их применения в человеческой деятельности. Значительную часть современных информационных технологий составляют сетевые технологии и их более глобальное проявление — Internet-технологии.

Основу данного учебного пособия составил курс лекций, подготовленный в соответствии с современным образовательным стандартом по специальностям 09.02.01 «Компьютерные системы и комплексы», 09.02.02 «Компьютерные сети», 09.02.03 «Программирование в компьютерных системах», в рамках которого изучается дисциплина «Программное обеспечение компьютерных сетей».

В учебном пособии рассматривается главным образом организация взаимодействия между персональными компьютерами в соответствии с технологией «клиент–сервер» и особенности ее применения в Web-системах. При этом наибольшее внимание было уделено не только рассмотрению основных видов программного обеспечения, используемого в Web-системах, но и особенностям его разработки.

Кроме того, были рассмотрены вопросы, связанные с понятиями Web-дизайна и Web-программирования, включая изучение основных конструкций языка гипертекстовой разметки документов HTML, языка DHTML и JavaScript, а также особенностей разработки расширений Web-серверов на примере написания CGI-сценариев на языке программирования высокого уровня Object Pascal и с использованием специальных средств современных RAD-систем. В новое издание пособия включена еще одна глава, которая посвящена сетевым операционным системам. В этой главе рассматриваются особенности настройки и конфигурирования сетевой операционной системы на примере операционных систем из семейства Windows.

Пособие состоит из семи взаимосвязанных частей (глав), каждая из которых раскрывает одну из ключевых тем предмета:

- технология «клиент–сервер»;
- сетевые операционные системы;
- программное обеспечение Web-систем;
- Web-приложения и методы их разработки;
- средства разработки Web-приложений;
- разработка Web-приложений;
- создание приложений Web-сервера.

ВВЕДЕНИЕ

Компьютерные сети, называемые также вычислительными сетями, или сетями передачи данных, можно определить следующим образом. С одной стороны, как частный случай распределенных вычислительных систем, в которых группа компьютеров согласованно выполняет набор взаимосвязанных задач, обмениваясь данными в автоматическом режиме, а с другой — как средство передачи информации на большие расстояния с использованием методов кодирования и мультиплексирования.

Таким образом, *компьютерная сеть* — это совокупность компьютеров, соединенных линиями связи. Линии связи образованы кабелями, сетевыми адаптерами и другими коммуникационными устройствами, называемыми сетевым оборудованием. Все сетевое оборудование работает под управлением системного и прикладного программного обеспечения.

Взаимодействие между компьютерами сети происходит путем передачи сообщений через сетевые адаптеры и каналы связи. С помощью этих сообщений один компьютер обычно запрашивает доступ к локальным ресурсам другого компьютера. Таким ресурсом может являться, например, вычислительная мощность компьютера в целом. В качестве совместно используемых ресурсов часто выступают данные, хранящиеся на диске, а также разнообразные периферийные устройства: принтеры, модемы, факс-аппараты и т.д.

Разделение локальных ресурсов каждого компьютера между всеми пользователями сети — основная цель создания компьютерной сети. На тех компьютерах, ресурсы которых должны быть доступны всем пользователям сети, необходимо добавить модули, которые постоянно будут находиться в режиме ожидания запросов, поступающих по сети от других компьютеров. Обычно такие модули называются *программными серверами* (*server*), так как их главная задача — *обслуживать* (*serve*) запросы на доступ к ресурсам своего компьютера. На компьютерах, пользователи которых хотят получать доступ к удаленным ресурсам и передавать их по сети на нужный компьютер, также необходимо установить дополнительные модули. Такие модули обычно называют *программными клиентами* (*client*). Сетевые адаптеры и каналы связи решают в сети задачу передачи сообщения с запросами и ответами от одного компьютера к другому, основную же работу по организации совместного использования ресурсов выполняют клиентские и серверные части операционных систем.

Пара модулей «клиент–сервер» обеспечивает совместный доступ пользователей к определенному типу ресурсов. Термины «клиент» и

«сервер» используются для обозначения не только программных модулей, но и компьютеров, подключенных к сети. Если компьютер предоставляет свои ресурсы другим компьютерам сети, то он называется сервером, а если он их потребляет — клиентом. Иногда один и тот же компьютер может выполнять роль как сервера, так и клиента.

В самом простом случае связь компьютеров может быть реализована с помощью тех же самых средств, которые используются для связи компьютера с периферийными устройствами. Для этого в компьютере предусмотрены интерфейсы, или порты, т.е. наборы проводов, соединяющих компьютер с устройствами, а также наборы правил обмена информацией по этим проводам. Для того чтобы компьютер мог работать в сети, его операционная система (ОС) должна быть дополнена клиентским и(или) серверным модулем, а также средствами передачи данных между компьютерами. В результате такого добавления ОС компьютера становится сетевой. Схема взаимодействия программных компонентов при связи двух компьютеров представлена на рис. В.1.

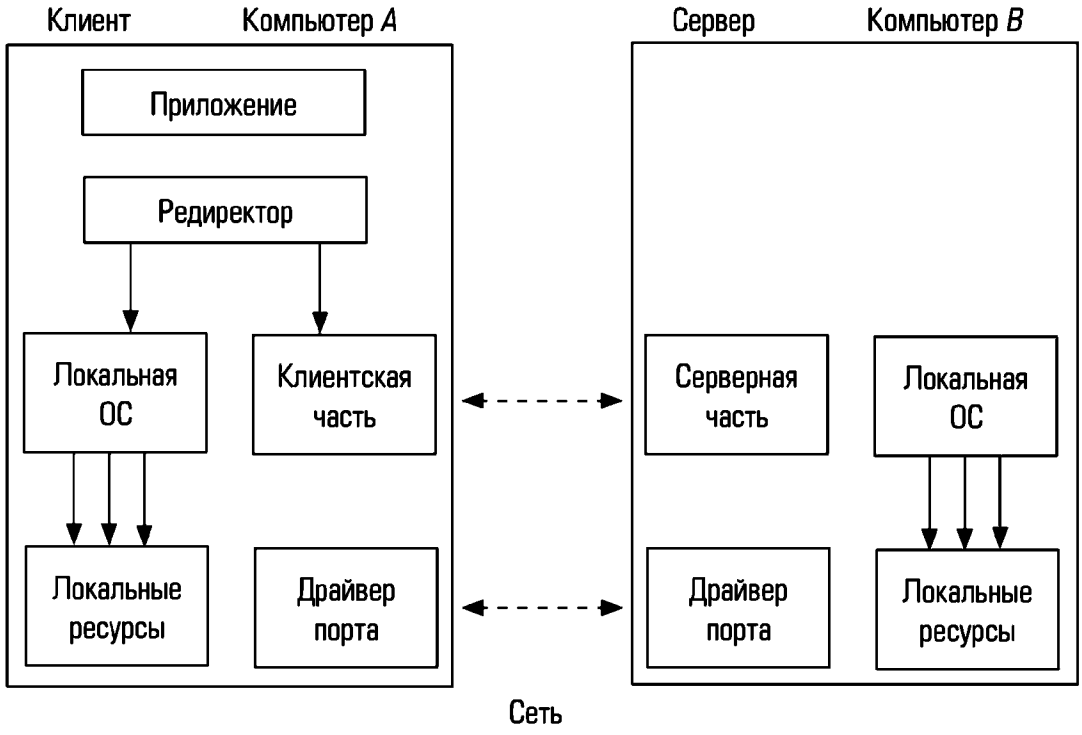

Рис. В.1. Взаимодействие программных компонентов

Развитие теории компьютерных сетей является логическим результатом эволюции двух важнейших научно-технических отраслей современной цивилизации — компьютерных и телекоммуникационных технологий.

Глава 1. ТЕХНОЛОГИЯ «КЛИЕНТ–СЕРВЕР»

1.1. Понятие о технологии «клиент–сервер»

Телекоммуникационные технологии лежат в основе современных систем коллективной человеческой деятельности, которые можно подразделить на две группы:

- *системы с разделением времени — СВР (Time Sharing System)*, каждый участник которых пользуется собственной ЭВМ и основной задачей является защита данных от несанкционированного доступа и взаимная изоляция участников;
- *системы обеспечения групповых решений — СОГР (Computer Supported Cooperative Work, groupware),* ориентированные на задачу обеспечения взаимодействия пользователей в процессе принятия решений.

С развитием представлений о распределенных вычислительных процессах и процессах обработки данных складывается концепция архитектуры «клиент–сервер» — обобщенное представление о взаимодействии двух компонентов информационной технологии (технического и(или) программного обеспечения) в вычислительных системах и сетях, среди которых логически или физически могут быть выделены активная сторона (источник запросов, клиент) и пассивная сторона (обработчик запросов, сервер).

Рассмотрим основные принципы построения «клиент-серверной» технологии. Архитектура «клиент–сервер» на сегодняшний день является доминирующей концепцией, используемой при создании распределенных сетевых приложений, она предусматривает взаимодействие и обмен данными между ними. В структуре архитектуры «клиент–сервер» можно выделить следующие основные компоненты:

- набор серверов, предоставляющих информацию или другие услуги программам, которые обращаются к ним;
- набор клиентов, использующих сервисы, которые предоставляются серверами;
- сеть, которая обеспечивает взаимодействие между клиентами и серверами.

Серверы являются независимыми друг от друга. Клиенты также функционируют параллельно и независимо друг от друга. При этом отсутствует жесткая привязка клиентов к серверам. Один сервер может одновременно обрабатывать запросы от разных клиентов, с другой стороны, клиент может обращаться то к одному серверу, то к другому. Клиенты должны знать о доступных серверах, но могут не иметь представления о существовании других клиентов.

Очень важен вопрос о том, что можно рассматривать в качестве «клиента». Можно говорить о клиентском компьютере, с которого происходит обращение к другим компьютерам; про клиентское и серверное программное обеспечение, а также о людях, которые желают с помощью соответствующего программного и аппаратного обеспечения получить доступ к той или иной информации.

Общепринятым является соглашение, что клиенты и серверы — это прежде всего программные модули. Чаще всего они находятся на разных компьютерах, но бывают ситуации, когда обе программы — и клиентская, и серверная — физически размещаются на одной машине, в такой ситуации сервер часто называют локальным.

Одним из типичных примеров клиент-серверного взаимодействия является система публикации ресурсов WWW. Существует огромное количество Web-серверов, на которых размещается та или иная информация. В простейшем случае эта информация представляет собой набор Web-страниц, которые могут сохраняться на сервере в виде файлов, размеченных с помощью языка разметки HTML. Однако значительная часть Web-ресурсов на современном этапе является динамической, т.е. они не существуют в заранее подготовленном виде, а создаются непосредственно в процессе обработки запроса от пользователя.

Для того чтобы любой пользователь Internet мог просмотреть определенную страницу, на его компьютере должно быть установлено соответствующее программное обеспечение. Программы для просмотра Web-страниц называются Web-браузерами, наиболее распространенным браузером является MSIE.

Но кроме Web-браузеров к серверам могут обращаться и другие клиенты, а именно — автономные программы. Они могут предусматривать взаимодействие с человеком, а могут работать полностью в автоматическом режиме. Типичный класс таких программ — роботы, предназначенные для автоматического просмотра Web-ресурсов. В частности, роботы являются важным элементом поисковых систем и используются ими для просмотра страниц и сбора информации.

Для запроса к Web-серверу клиентская программа должна задать местонахождение компьютера, на котором размещается серверная программа, название нужного документа и, возможно, другие данные, которые специфицируют запрос. Сеть обеспечивает обнаружение сервера и передачу ему клиентского запроса. Серверные программы обрабатывают этот запрос, ответ пересылается по сети клиенту.

Распределение обязанностей между клиентом и сервером определяет используемую модель клиент-серверного взаимодействия. Логически можно выделить три различные операции:

- уровень представления данных, который, по сути, является интерфейсом пользователя и отвечает за представление данных пользователю и введение от него управляющих команд;

- прикладной уровень, который реализует основную логику приложения и на котором осуществляется необходимая обработка информации;
- уровень управления данными, который обеспечивает сохранение данных и доступ к ним.

Двухуровневая клиент-серверная архитектура предусматривает взаимодействие двух программных модулей — клиентского и серверного. В зависимости от того, как между ними распределяются приведенные выше функции, различают:

- модель тонкого клиента, в рамках которой вся логика приложения и управление данными сосредоточена на сервере. Клиентская программа обеспечивает только функции уровня представления;
- модель толстого клиента, в которой сервер только управляет данными, а обработка информации и интерфейс пользователя сосредоточены на стороне клиента.

Тонкими клиентами часто также называют устройства с ограниченной мощностью — карманные компьютеры, мобильные телефоны и др.

Трехуровневая клиент-серверная архитектура, которая начала развиваться с середины 1990-х гг., заключается в разделении функций прикладного уровня от управления данными. Выделяется отдельный программный уровень, на котором сосредоточивается прикладная логика приложения. Программы промежуточного уровня могут функционировать под управлением специальных серверов приложений, но запуск таких программ может осуществляться и под управлением обычного Web-сервера. При этом управление данными осуществляется сервером данных.

Для работы с системой пользователь использует стандартное программное обеспечение — обычный Web-браузер. Это лишает его необходимости загружать и инсталлировать специальные программы. Но пользователю следует предоставить в распоряжение интерфейс, который разрешал бы ему взаимодействовать с системой и формировать запросы к ней; формы, которые определяют этот интерфейс, размещаются на Web-страницах и загружаются вместе с ними.

Web-браузер формирует запрос и пересылает его серверу, который осуществляет обработку. При необходимости сервер вызывает серверные программные модули, которые обеспечивают обработку запроса и по мере необходимости обращаются к серверу данных. Сервер данных осуществляет операции с данными, которые сохраняются в системе и составляют ее информационную основу. В частности, он может осуществить выборку из информационной базы, которая соответствует запросу, и передать ее модулю промежуточного уровня для дальнейшей обработки. Данные, с которыми работает сервер данных, чаще всего организованы как реляционная база данных.

Обычно Web-сервер и серверные модули промежуточного уровня размещаются на одном компьютере, хоть и представляют собой отдельные и логически независимые программные модули.

На современном этапе для реализации модулей промежуточного уровня используется язык серверных сценариев PHP, а для управления данными — СУБД MySQL. Таким образом, связку PHP–MySQL следует рассматривать как стандартный инструмент для создания сравнительно простых интерактивных Web-сайтов и систем электронной коммерции; около 90% коммерческих систем сегодня создается именно на этой основе. Вместе с тем как средства управления данными, так и middleware-средства могут быть самыми разнообразными. Так, для создания серверных приложений, кроме PHP, широко применяются Java, Perl, Python. Технологии создания Web-приложений, в частности распределенных, стремительно развиваются. Следует упомянуть о технологиях EJB (Enterprise Java Beans), CORBA, а также про .NET — сравнительно новую инициативу компании Microsoft. Для сохранения данных и их передачи часто используется так называемый расширенный язык разметки XML (Extensible Markup Language).

1.2. Разновидности функциональных структур «клиент–сервер»

Компьютер (процесс), управляющий тем или иным ресурсом, является сервером этого ресурса, а компьютер, пользующийся им, — клиентом. Каждый конкретный сервер определяется видом того ресурса, которым он владеет (обработка сервером баз данных запросов клиентов, управление файловой системой файл-сервером).

Один из основных принципов технологии «клиент–сервер» заключается в разделении функций стандартного интерактивного (диалогового) приложения на четыре группы: 1) функции ввода и отображения данных; 2) прикладные функции, характерные для данной предметной области (для банковской системы — открытие счета, перевод денег с одного счета на другой); 3) фундаментальные функции хранения и управления информационно-вычислительными ресурсами (базами данных, файловыми системами и т.п.); 4) служебные функции, осуществляющие связь между функциями первых трех групп.

В соответствии с этим в любом приложении выделяются следующие логические компоненты: компонент представления (presentation), реализующий функции первой группы; прикладной компонент (business application), поддерживающий функции второй группы; компонент доступа к информационным ресурсам (recource manager), поддерживающий функции третьей группы, а также вводятся и уточняются соглашения о способах их взаимодействия.

Различия в реализации технологии «клиент–сервер» определяются следующими факторами: виды программного обеспечения, в которые интегрирован каждый из этих компонентов; механизмы программного обеспечения, используемые для реализации функций всех трех групп; способы распределения логических компонентов между компьютерами в сети; механизмы, используемые для связи компонентов между собой.

Выделяются четыре подхода, реализованные в следующих технологиях: 1) файловый сервер (File Server — FS); 2) доступ к удаленным данным (Remote Data Access — RDA); 3) сервер баз данных (Data Base Server — DBS); 4) сервер приложений (Application Server — AS).

Технология FS является базовой при реализации локальных сетей ПК. Один из компьютеров в сети назначается файловым сервером и предоставляет другим компьютерам услуги по обработке файлов. Файловый сервер работает под управлением сетевой операционной системы и играет роль компонента доступа к информационным ресурсам (т.е. к файлам). На других ПК в сети функционирует приложение, в кодах которого совмещены компонент представления и прикладной компонент (рис. 1.1).

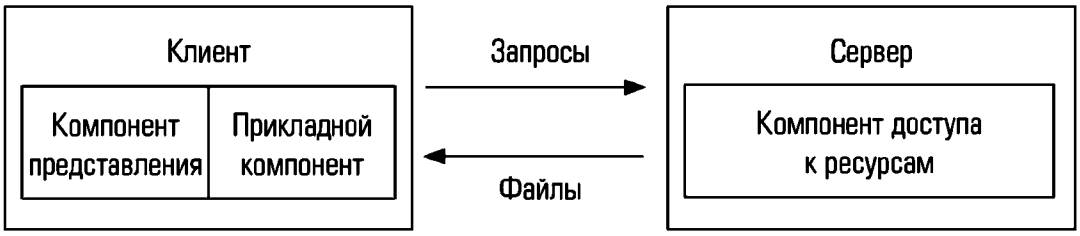

Рис. 1.1. Технология *File Server*

К недостаткам данной технологии относится низкий сетевой трафик, небольшое количество операций манипуляции с данными, отсутствие адекватных средств безопасности доступа к данным и т.д.

Технология RDA существенно отличается от FS методом доступа к информационным ресурсам, так как здесь компонент представления и прикладной компонент совмещены и выполняются на компьютере-клиенте. Доступ к информационным ресурсам обеспечивается операторами специального языка или вызовами функций специальной библиотеки. Запросы к информационным ресурсам направляются по сети удаленному компьютеру, который обрабатывает и выполняет их, возвращая клиенту блоки данных (рис. 1.2).

Достоинство RDA заключается в унификации интерфейса «клиент–сервер» в виде языка запросов и широком выборе средств разработки приложений. К недостаткам можно отнести существенную загрузку сети при взаимодействии клиента и сервера посредством

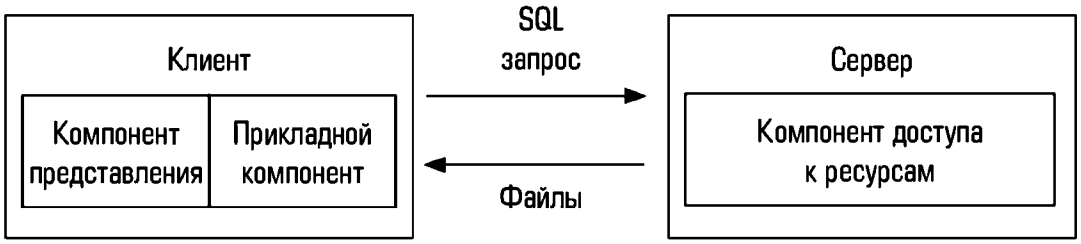

Рис. 1.2. Технология *Remote Data Access*

запросов; невозможность администрирования приложений, так как в одной программе совмещаются различные по своей природе функции (представления и прикладные).

Технология DBS реализована в некоторых реляционных (табличных) СУБД (Informix, Sybase, Oracle) (рис. 1.3).

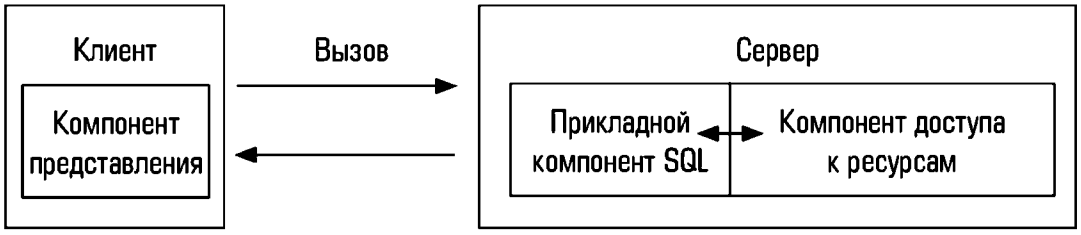

Рис. 1.3. Технология *Data Base Server*

Ее основу составляет механизм хранимых процедур (средство программирования SQL-сервера). Процедуры хранятся в базе данных, разделяются между несколькими клиентами и выполняются на том же компьютере, где функционирует SQL-сервер. В сервере баз данных компонент представления выполняется на компьютере-клиенте, в то время как прикладной компонент оформлен как набор хранимых процедур и функционирует на компьютере сервере БД. Там же выполняется компонент доступа к данным, т.е. ядро СУБД.

Достоинства технологии заключаются в следующем: возможность централизованного администрирования прикладных функций; снижение трафика; возможность разделения процедуры между несколькими приложениями; экономия ресурсов компьютера за счет использования однажды созданного плана выполнения процедуры. К недостаткам относится ограниченность средств написания хранимых процедур, представляющих собой процедурные расширения SQL, которые уступают по возможностям таким языкам, как C или Pascal.

На практике чаще используются смешанные модели, когда целостность базы данных и некоторые простейшие прикладные функ-

ции обеспечиваются хранимыми процедурами (DBS-технология), а более сложные функции реализуются непосредственно в прикладной программе, которая выполняется на компьютере клиенте (RDA-технология).

Технология AS — сервер приложений представляет собой процесс, выполняемый на компьютере-клиенте, отвечающий за интерфейс с пользователем (т.е. реализует функции первой группы) (рис. 1.4).

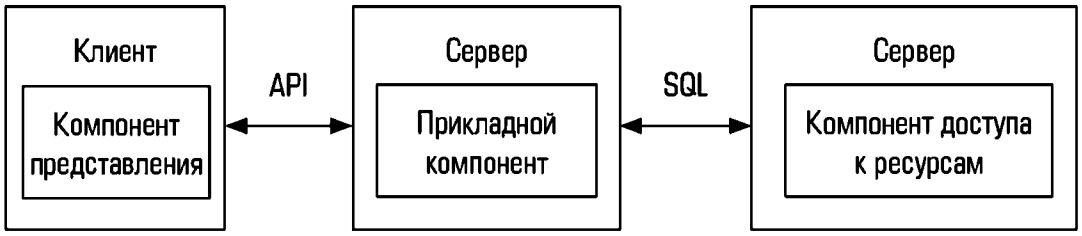

Рис. 1.4. Технология *Application Server*

Прикладной компонент реализован как группа процессов, выполняющих прикладные функции, и называется сервером приложения. Доступ к информационным ресурсам осуществляет менеджер ресурсов (например, SQL-сервер). Из прикладных компонентов доступны такие ресурсы, как базы данных, очереди, почтовые службы и др. AS, размещенная на компьютере, где функционирует менеджер ресурсов, избавляет от необходимости направления SQL-запросов по сети, что повышает производительность системы.

Технологии RDA и DBS опираются на двухзвенную схему разделения функций: в RDA прикладные функции отданы программе-клиенту (прикладной компонент комбинируется с компонентом представления); в DBS ответственность за их выполнение берет на себя ядро СУБД (прикладной компонент интегрируется в компонент доступа к информационным ресурсам).

В AS реализована трехзвенная схема разделения функций. Здесь прикладной компонент выделен как важнейший изолированный элемент приложения. Сравнивая модели, можно заключить, что AS обладает наибольшей гибкостью и имеет универсальный характер.

1.3. «Клиент-серверные» системы на основе Web-технологий

Распространенным примером «клиент-серверных» систем может служить система, построенная на основе Web-технологий. При этом в качестве клиента выступают Web-браузеры (NetScape Navigator, MS Internet Explorer и др.), в качестве сервера — Web-серверы (NCSA, Raily, Apachie и т.д.).

Web-технологии являются базовыми при объединении внутренних и глобальных сетей в единую информационную систему и, таким образом, представляют собой платформу взаимодействия различных информационных источников в рамках единой информационной системы.

Основными компонентами Web-технологий, состоящих в применении гипертекстовой модели к информационным ресурсам, распределенным в Internet, являются (рис. 1.5): HTML (HyperText Markup Language) — язык гипертекстовой разметки документов; URL (Universal Resource Locator, uniform resource identificator) — универсальный способ адресации ресурсов в сети; HTTP (HyperText Transfer Protocol) — протокол обмена гипертекстовой информацией; дополнительные средства (CGI, Java, JavaScript). Также можно говорить о наличии еще одного компонента данной технологии — интерпретатора языка. В Web функции интерпретатора разделены между сервером гипертекстовой базы данных и интерфейсом пользователя. Сервер, кроме обеспечения доступа к документам и реализации гипертекстовых ссылок, осуществляет также предпроцессорную обработку документов, в то время как интерфейс пользователя проводит интерпретацию конструкций языка, связанных с представлением информации.

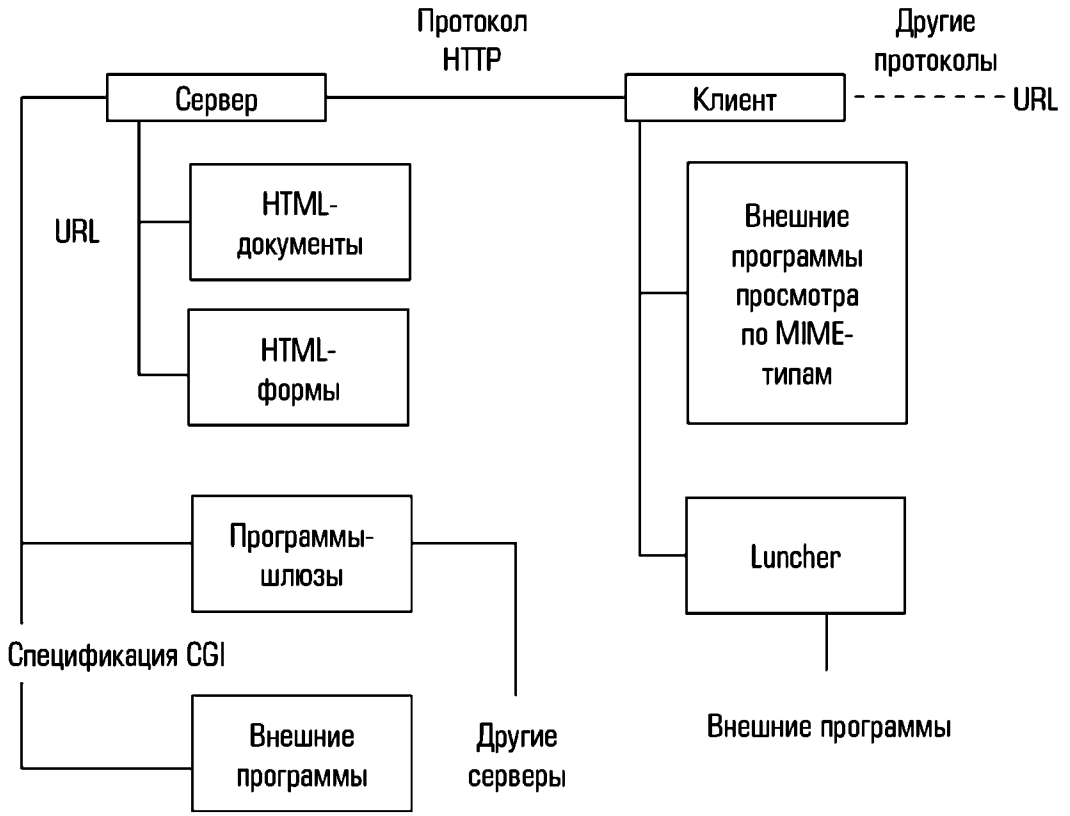

Рис. 1.5. Архитектура Web-технологии

CGI (Common Gateway Interface) — средство расширения возможностей Web-технологии путем написания прикладных пользовательских программ (CGI-скриптов).

CGI-скрипт — программа, написанная в соответствии с CGI на одном из языков программирования (C, Perl, Pascal и пр.) или командном языке ОС.

CGI-шлюз — CGI-скрипт, используемый для обмена данными с другими информационными ресурсами Internet.

MIME — расширенный стандарт формата почтового сообщения (RFC-1341), обеспечивающий возможности передачи разнородной информации по электронной почте (текст, мультимедиа, графика и пр.).

1.4. Разработка «клиент-серверной» системы на основе технологии File Server

Рассмотрим создание простейшего варианта «клиент-серверного» приложения на языке Delphi. Для этого нам потребуются следующие компоненты: TLabel, TButton, TEdit, TListBox, TComboBox, TRadioGroup, TPanel, TPopupMenu, TListView. Все перечисленные компоненты расположены на вкладке Standart. А оставшийся компонент TListView – на вкладке Win32.

Компонент TLabel (Надпись) предназначен для вывода данных, которые записываются в свойство Caption. TButton (Кнопка) используется в программе для интерактивного общения с пользователем. Этот компонент предназначен для выполнения команд, реализуемых после нажатия кнопки. Компонент TEdit (Текстовое поле) применяется для ввода информации в программу. Его основным свойством является свойство Text, в котором и помещается информация, введенная пользователем. Данная информация представляется в текстовом виде.

Компонент TListBox (Список) предназначен для вывода в виде списка той или иной текстовой информации. В списке может быть несколько столбцов, их количество определяется свойством Columns. Если оно больше 0, то на каждую колонку отводится часть общей длины списка. Основным свойством TListBox является свойство Items (список строк, класс TStrings). Строки можно задать на этапе проектирования в специальном редакторе, а можно и во время работы программы с помощью метода Add класса TStrings. Для очистки всего содержимого списка используется метод Clear. Чтобы удалить конкретный элемент, можно воспользоваться методом DeleteString, в параметрах которого указывается номер удаляемого элемента.

Компонент TComboBox (Поле со списком) представляет собой вариант списка с присоединенным дополнительным полем, в кото-

ром отображается выбранный элемент списка. Режим работы компонента TComboBox определяется свойством Style, которое может принимать следующие значения: csDropDown, csDropDownList и csSimple. Подробнее об этом компоненте будет сказано в главе 6 «Разработка Web-приложений» в разделе «Разработка Web-браузера».

Компонент TRadioGroup (Группа переключателей) необходим в случае, если требуется использовать несколько переключателей. Создание переключателей в этом компоненте осуществляется с помощью свойства Items, имеющего тип TString. Свойство Columns задает число столбцов, образованных переключателем. Свойство ItemIndex содержит номер выделенного переключателя (исходное значение -1, которое означает, что не выбран ни один переключатель).

Компонент TPanel (Панель) используется для создания декоративных элементов при оформлении внешнего вида формы. Свойства BevelInner и BevelOuter задают стили внутренней и внешней рамок панели, BevelWidth определяет расстояние между внутренней и внешней рамками, а свойства BorderStyle и BorderWidth задают соответственно стиль и ширину границы.

Компонент TPopupMenu (Контекстное меню) является невизуальным компонентом. Создание меню осуществляется при помощи специального редактора, который вызывается двойным щелчком мыши по компоненту. Программный вызов контекстного меню осуществляется при помощи метода Popup, в качестве параметров которого указываются координаты верхнего левого угла меню. Для указания корректных значений координат необходимо воспользоваться методом формы GetClientOrigin, который обращается к стандартным функциям Windows и возвращает корректное значение смещения, учитывая высоту строки заголовка формы и панели командных кнопок. Данная функция возвращает значение типа TPoint (Координатная точка), представляющее запись из двух элементов: X и Y.

Компонент TListView (Список элементов) представляет собой стандартный список строк, но при этом содержит значительно больше возможностей для представления информации. Режим работы данного компонента определяется свойством ViewStyle, которое может принимать одно из следующих значений:

- vsIcon – каждый элемент представлен полноразмерным значком с подписью, который можно перетаскивать. Так работают папки Windows в режиме «Крупные значки»;
- vsSmollIcon – каждый элемент представлен маленьким значком с подписью справа от него. Эти значки можно перетаскивать. Так работают папки Windows в режиме «Мелкие значки».

- vsList – каждый элемент представлен маленьким значком с подписью справа от него. Эти значки, расположенные по столбцам, перетаскивать нельзя. Так работают папки Windows в режиме «Список».
- vsReport – объект работает как обычный список с несколькими столбцами.

Свойство определяет число и свойства столбцов и представляет собой набор объектов типа TListColoumn. Их основным свойством является свойство Caption, которое содержит заголовок столбца. Это свойство можно редактировать во время работы программы, если значение свойства ReadOnly не равно true и если заголовки не работают в режиме кнопок.

Данные списка формируются в свойстве Items (тип TListItems), который состоит из списка объектов TListItem. Эти объекты могут создаваться на этапе проектирования с помощью специального редактора (рис. 1.6).

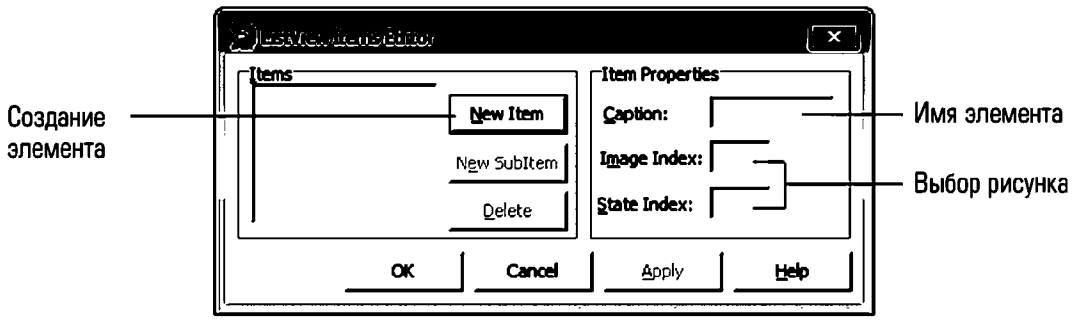

Рис.1.6. Редактор списка объектов

Новый элемент списка создается щелчком на кнопке New Item. Затем задаются его параметры. В поле Caption (Заголовок) задается его имя, в поле Image Index (Номер рисунка) – номер рисунка (значка) из списка рисунков LargeImages или SmallImages, а в поле State Index (Номер состояния) – номер рисунка из списка рисунков StateImages. Каждый элемент может состоять из нескольких вложенных элементов, представляющих собой свойства этого элемента. Обращение к этим объектам возможно через подсвойство SubItems, вложенное в свойство Items.

Компонент TListView имеет следующие свойства:

- property AllocBy – число элементов списка, хранимое в памяти. Управляя этим значением, можно существенно повысить быстродействие некоторых операций по работе со списком, например, сортировки;

- property Checkboxes – имеет значение true, если в начале каждой строки списка отображается флажок;
- property ColumnClick – имеет значение true, если заголовкам столбцов разрешено работать в режиме кнопок: допустимы щелчки по заголовкам и обработка этих щелчков. Такая возможность полезна для сортировки содержимого щелчком по его заголовку;
- property FlatScrollBars – имеет значение true, если полосы прокрутки должны выглядеть плоскими;
- property FullDrag – имеет значение true, если разрешается полностью перерисовывать заголовки столбцов во время перетаскивания, а не только отображать границы;
- property GridLines – имеет значение true, если между элементами списка рисуются разделительные линии;
- property HideSelection – имеет значение true, если выделение текущего элемента списка автоматически сбрасывается при переключении фокуса на другой элемент формы. Такая возможность используется, когда ведется одновременная работа с несколькими списками, что позволяет быстро определить, какой список имеет фокус;
- property HotTrack – имеет значение true, если выбор элемента осуществляется наведением на него указателя (без щелчка);
- property HotTrackStyles – если значение свойства HotTrack равно true, то способ выделения элемента определяется комбинацией значений данного свойства (множества). Возможные значения: htHandPoint (указатель мыши принимает вид руки), htUnderlineCold (невыделенные элементы подчеркиваются), htUnderlineHot (выделенный элемент подчеркивается);
- property HoverTime – если значение свойства HotTrack равно true, то данное свойство определяет время (в миллисекундах), по истечении которого после наведения на него указателя элемент будет считаться выделенным;
- property IconOptions – способ упорядочения значков в списке. Имеет три подсвойства. Подсвойство Arrangement определяет порядок выравнивания (слева направо или сверху вниз). Подсвойство AutoArrange определяет, будут ли значки переупорядочиваться автоматически. Подсвойство WrapText определяет, будет ли заголовок выравниваться по ширине значка или располагаться слева от него;
- property LargeImages – Список картинок-значков. Стиль отображения определен значением vsIcon;
- property MultySelect – имеет значение true, если разрешается выбирать несколько элементов списка;

- property OwnerData – имеет значение true, если обработка содержимого списка выполняется в тексте программы. Подобный список называется виртуальным, и разработчик берет на себя программирование его основных функций, связанных с динамическим формированием значений элементов. Это требуется при обработке больших наборов данных;
- property OwnerDraw – имеет значение true, если рисование списка и его элементов явно выполняется в тексте программы по алгоритму разработчика;
- property RowSelect – имеет значение true, если разрешается выделять целую строку списка;
- property ShowColumnHeaders – имеет значение true, отображаются заголовки столбцов;
- property SmallImages – список картинок-значков. Стиль отображения определен значением, отличающимся от vsIcon;
- property SortType – способ автоматической сортировки списка. Возможные значения: stNone (сортировка не выполняется), stData (сортировка выполняется на основе значений свойства Data каждого элемента списка), stText (сортировка выполняется на основе значений свойства Caption каждого элемента списка), stBoth (сортировка выполняется на основе значений как свойства Data, так и свойства Caption);
- property StateImages – список картинок, отражающих промежуточное состояние объекта.

Перечислим методы компонента TListView:

- procedure Arrange(Code: TListArrangement) – задает способ выравнивания значков, когда значение свойства ViewStyle равно vsIcon или vsSmallIcon;
- function FindeCaption(StartIndex: Integer; Value: string;
  Partial, Inclusive, Wrap: Boolean): TListItem – поиск элемента списка, заголовок которого совпадает со значением параметра Value. Если параметр Inclusive имеет значение true, то поиск выполняется с элемента с номером, хранящимся в параметре StartIndex. При этом если параметр Wrap также имеет значение true, то поиск до достижения конца списка продолжается с его начала. Параметр Partial разрешает не искать полное совпадение заголовков, а использовать значение Value как подстроку;
- function FindData(StartIndex: Integer; Value: TPoint;
  Inclusive, Wrap: Boolean): TListItem – аналогично предыдущему, только производится сравнение не заголовков, а связанных с объектом данных (свойство Data);
- function GetHitTestInfoAt(X, Y: Integer): THiteTests – возвращает подробную информацию об указанной точке клиентской обла-

сти. Тип THitsTests описывает различные местоположения точки;

- function GetItemAt(X, Y: Integer): TListItem – возвращает элемент списка, области которого принадлежит указанная точка;
- function GetNearestItem(Point: TPoint;
  Direction: TSerchDirection): TListItem – возвращает элемент списка, ближайший к указанной точке по заданному направлению (параметр Direction);
- function GetNextItem(StartItem: TListItem;
  Direction: TSearchDirection;
  States: TItemStates): TListItem – возвращает элемент списка, следующий за указанным в параметре StartItem в направлении Direction. Если используется список StateImages, то можно учитывать и наличие свойства State;
- function GetSearchString: string – возвращает текущую строку, которую пользователь ввел для поиска нужного элемента;
- procedure Scroll(DX, DY: Integer) – прокрутка содержимого списка на DX пикселов по горизонтали и DY пикселов по вертикали;
- function StringWidth(S: string): Integer – возвращает ширину строки S в пикселах с учетом текущего шрифта списка;
- procedure UpdateItems(FirstIndex, LastIndex: Integer) – перерисовка диапазона элементов списка в диапазоне от FirstIndex до LastIndex.

Теперь рассмотрим события класса TListView:

- OnCustomDraw, OnAdvencedCustomDraw – программная отрисовка внешнего вида списка;
- OnCustomDrawItem, OnAdwencedCustomDrawItem – программная отрисовка элемента списка;
- OnCustomDrawSubItem, OnAdwencedCustomDrawSubItem – программная отрисовка элемента (свойства) списка;
- OnChange – элемент списка был изменен;
- OnChanging – происходит изменение элемента списка;
- OnColumnClick – щелчок мышкой на заголовке столбца;
- OnColumnDragged – заголовок столбца был перемещен (перетащен мышью) в новое место;
- OnColoumnRightClick – щелчок на заголовке столбца правой кнопкой мыши;
- OnData – генерируется перед тем, как элемент списка должен быть нарисован. Данное сообщение обрабатывается, когда содержимое каждого элемента формируется программно (режим «виртуального списка»);
- OnDataFind – запрос на поиск данных от метода FindData;

- OnDataHint – изменен диапазон видимых на экране элементов (например, при прокрутке);
- OnDataStateChange – изменение состояния элемента (событие возникает, только если значение свойства OwnerData равно true);
- OnDeletion – пользователь отдал команду на удаление элемента;
- OnDrawItem – программная отрисовка содержимого элемента (событие возникает, только если значение свойства OwnerData равно true);
- OnEdited – завершено редактирование элемента;
- OnEditing – происходит редактирование элемента;
- OnGetImageIndex – генерируется перед отображением элемента на экране. Его можно обрабатывать, чтобы динамически задавать номер картинки-значка из списка картинок;
- OnGetSubItemImage – то же самое только для вложенного элемента SubItem;
- OnInfoTip – пользователь навел указатель мыши на элемент и задержал его;
- OnInsert – в список добавлен новый элемент;
- OnSelectItem – в списке выбран элемент.

Рассмотрим теперь методы класса TListItems, необходимые для работы со свойством Items:

- function Add: TListItem – создание нового элемента и его добавление в конец списка. Функция возвращает ссылку на этот элемент;
- procedure BeginUpdate, procedure EndUpdate – первая процедура блокирует перерисовку списка, а вторая снимает блокировку. Эти методы обычно используют во время выполнения большого числа изменений, чтобы не замедлять работу перерисовкой ненужных временных деталей;
- procedure Clear – удаление всех элементов списка и освобождение ими памяти;
- procedure Delete(Index: Integer) – удаление указанного элемента;
- function IndexOf(Value: TListItem): Integer – возвращает номер элемента, указанного в качестве параметра;
- function Insert(Index: Integer): TListItem – создание нового элемента и его добавление в указанную позицию списка. Функция возвращает ссылку на этот элемент;
- procedure SetCount(Value: Integer) – задание числа элементов в списке.

Рассмотрим теперь свойства и методы класса TListItem, который характеризует конкретный элемент списка:

- property Caption – заголовок элемента;
- property Checked – имеет значение true, если флажок элемента включен (свойство Checkboxes должно иметь значение true);
- property Cut – элемент рисуется в виде, показывающем, что он вырезан командой Cut (Вырезать). Все действия по реализации процедуры такого рисования разработчик должен программировать самостоятельно;
- property Data – свойство, имеющее тип Pointer и указывающее на связанный с элементом объект;
- property Focused – имеет значение true, если элемент имеет фокус;
- property ImageIndex – номер значка в списке картинок;
- property Index – положение элемента в колекции TListItems;
- property Left – горизонтальный сдвиг от левой границы списка;
- property Position – свойство типа TPoint, определяющее координаты (в пикселах) элемента внутри списка;
- property Selected – имеет значение true, если элемент выделен;
- property StateIndex – номер значка из дополнительного списка картиок;
- property SubItemImages – список картинок для свойств данного элемента;
- property SubItems – список названий свойств элемента (тип TStrings);
- procedure Delete – удаление элемента из списка. Для освобождения занимаемой им памяти надо использовать метод Free;
- function DisplayRect(Code: TDisplayCode): TRect – определяет прямоугольные координаты элемента с учетом параметра Code (границы всего элемента, только значка, только текста, значка и текста);
- function GetPosition: TPoint – определение положения элемента в списке: смещение верхнего левого угла относительно начала списка;
- procedure SetPosition(const Value: TPoint) – установка нового положения элемента;
- procedure MakeVisible(PartialOK: Boolean) – прокрутка списка так, чтобы элемент стал видимым. Если значение параметра PartialOK равно true и элемент уже частично виден, то прокрутка не выполняется;
- procedure Update – перерисовка элемента.

Теперь перейдем непосредственно к разработке приложения. Вначале необходимо создать файловую структуру базы данных. Мы будем разрабатывать базу данных «Книги». Для этого в корневом каталоге, в котором у нас будет располагаться программа, создаем пап-

ку под названием Base, в нее включаем четыре файла. Файлы просто можно создать с помощью программы «Блокнот» и затем заменить расширения файлов с txt на db. Нам потребуется четыре файла: Autors.db, Cites.db, Publ.db и Books.db. Первые три будут хранить информацию соответственно об авторах книг, городах их издания и издательствах. Файл Books.db будет содержать собственно каталог книг. Кроме того, приложение будет включать две программы: одна реализует функции сервера, другая – клиента.

Рассмотрим программу-сервер. Интерфейс программы представлен на рис. 1.7.

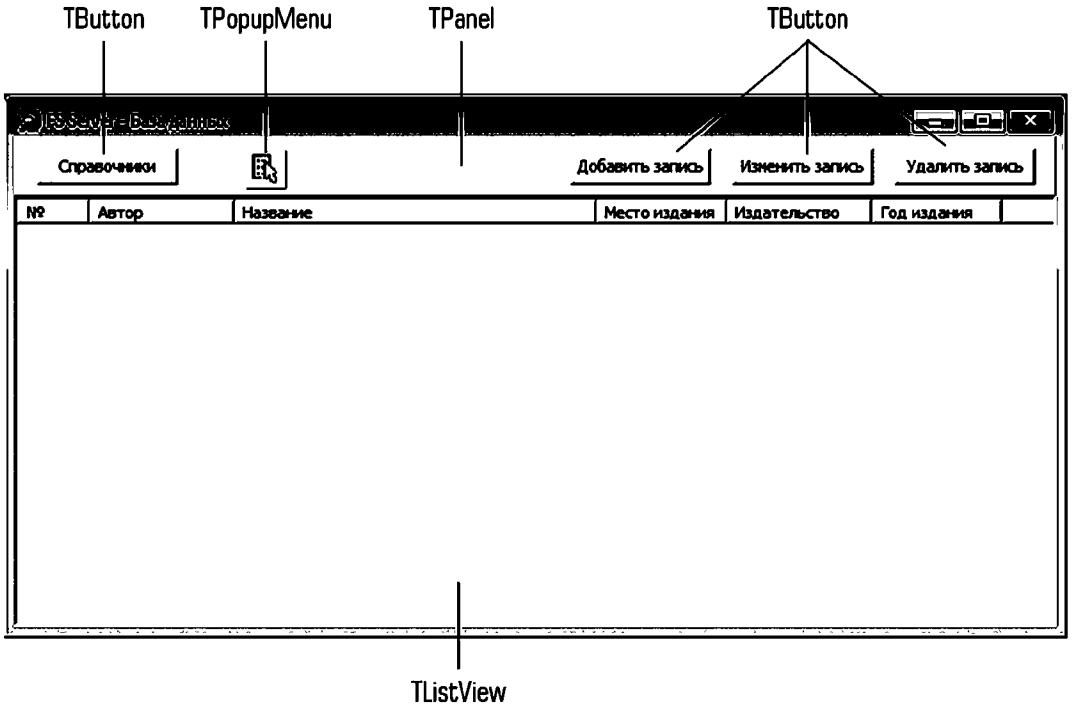

Рис. 1.7. Интерфейс программы-сервера

Вначале настроим компонент TListView. Для этого обратимся к свойству Columns. В редакторе создадим шесть колонок: №, Автор, Название, Место издания, Издательство и Год издания. Также изменим некоторые из свойств компонента: ReadOnly – true; Align – alClient; RowSelect – true; ViewStyle – vsReport. Настроим компонент TPopupMenu. В редакторе настройки создадим три пункта меню: Авторы, Место издания и Издательство. Они будут использоваться для вызова соответствующего справочника.

Еще нам понадобятся две формы: одна будет работать как окно справочника (рис. 1.8), другая – как редактор содержимого базы данных (рис. 1.9).

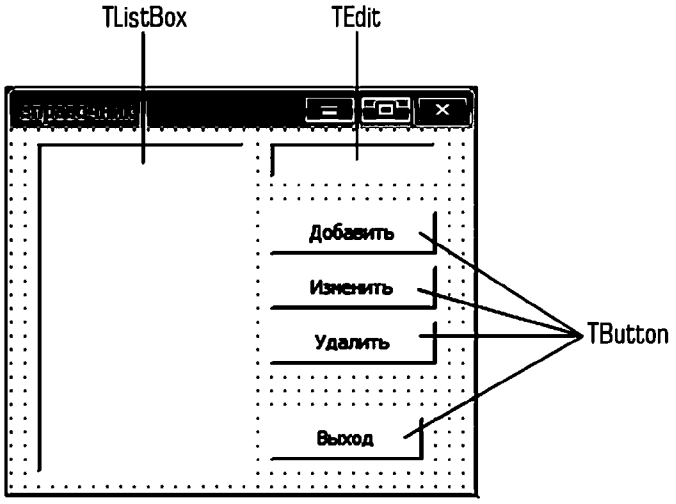

Рис. 1.8. Интерфейс окна «Справочник»

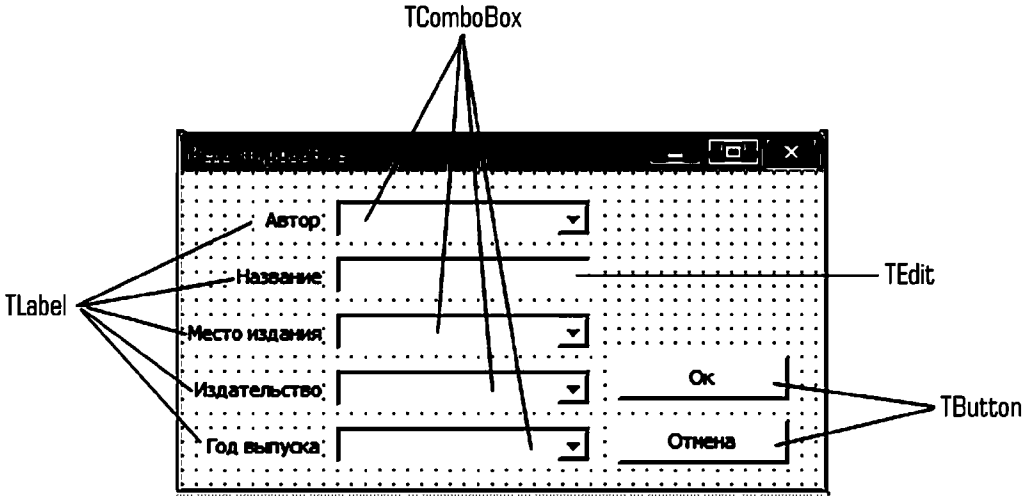

Рис. 1.9. Интерфейс окна «Редактирование»

Теперь приступим непосредственно к написанию кода программы. Опишем вначале в модуле главного окна программы Unit 1 сложный тип данных record (запись):

```
TBook = record
        Num: integer;
        Autor: string[20];
        Book: string[100];
        City: string[15];
        Publ: string[15];
        Year: integer;
end;
```

И две глобальные переменные:

```
F: File of TBook;
Fs: TextFile;
```

Нам будет необходимо работать с такой структурой языка Delphi, как файл, поэтому опишем технологию работы с файлами в этой системе. Эта технология предполагает соблюдение определенного порядка действий:

- открытие файла;
- определение режима работы с файлом;
- работа с файлом;
- закрытие файла.

Для определения переменной файловой структуры используется ключевое слово File, которое определяет нетипизированный файл. Для определения файла определенной структуры используется конструкция:

```
var имя-переменной: File of тип;
```

Для определения текстового файла указывается ключевое слово TextFile:

```
var имя-переменной: TextFile;
```

Теперь рассмотрим непосредственно порядок работы с файлами.

Для открытия файла используется процедура AssignFile. Первый параметр этой процедуры – имя переменной, второй – строка, содержащая название файла. Если полный путь поиска файла не указан, то файл будет разыскиваться в текущем каталоге.

Для определения режима работы с файлом используется одна из двух процедур: Rewrite – для открытия в режиме записи (при этом происходит полное уничтожение его содержимого, а размер файла становится равным нулю) и Reset – для открытия файла в режиме чтения (при этом вносить изменения в содержимое файла не разрешается). Процедура Rewrite может также применяться для создания нового файла. Каждая из этих процедур может иметь второй необязательный параметр, который определяет длину записи нетипизированного файла в байтах.

Для записи данных в типизированный файл применяется процедура Write, в качестве первого параметра которой указывается файловая переменная, далее следует список переменных типа, соответствующего типу файла. Считывание содержимого файла производится с помощью процедуры Read, ее параметры аналогичны процедуре записи в файл. Для перезаписывания определенной записи без перезаписи всего содержимого файла используется процедура Seek, первый ее параметр – файловая переменная (определенного типа или нетипизированная), второй параметр – номер записи в файле, начиная с которой будет выполняться следующая

операция ввода/вывода. Нумерация позиций в файле начинается с нуля. Есть еще одна специальная процедура Truncate, которая позволяет удалить все содержимое файла, начиная с текущей позиции до конца.

Закрытие файла выполняется вызовом процедуры CloseFile, в качестве ее параметра указывается файловая переменная.

Уже упоминавшиеся текстовые файлы предназначены только для хранения строк, которые описываются базовым типом String. Считывание и запись в таких файлах производится построчно. Для таких файлов реализованы две процедуры, которые осуществляют ввод и вывод с новой строки: ReadLn и WriteLn. Если применять процедуры Read и Write, то считывание и запись текста производится сплошным потоком, без разделения на строки.

Для работы с текстовыми файлами имеется набор стандартных подпрограмм:

- procedure Append(var F: Text) – открытие текстового файла в режиме записи. Отличается от процедуры Rewrite тем, что не стирает все содержимое, а устанавливает текущую позицию в самый конец файла, что позволяет добавлять информацию;
- procedure AssignPrn(var Text) – вся информация, записываемая в файл, перенаправляется на принтер. Файл должен быть открыт в режиме Rewrite;
- function Eoln(var F: Text): Boolean – возвращает значение true, если текущая позиция в файле расположена либо в конце файла, либо в конце строки. Такая проверка может понадобиться, если ввод выполняется с помощью процедуры Read, не переходящей к началу следующей строки автоматически;
- procedure Erase(var F: Text) – удаление файла. Он должен быть определен с помощью процедуры AssignFile, но не должен быть открыт;
- procedure Flush(var F: Text) – информация, которая была записана в файл из программы, но находится во временном буфере, физически перемещается на диск;
- function SeekEof(var F: Text): Boolean – возвращает значение true, если текущая позиция расположена в конце файла;
- function SeekEoln(var F: Text): Boolean – возвращает значение true, если текущая позиция расположена в конце строки;
- procedure SetTextBuf(var F: Text; var Buf; Size: Integer) – устанавливает размер буфера для операций ввода/вывода с текстовыми файлами (параметр Size). Этот буфер располагается внутри программы. Он указывается в качестве второго параметра.

Теперь продолжим разбирать код нашей программы. Следующий программный код необходимо прописать в процедуре обработчика события создания формы OnCreate:

```
procedure TForm1.FormCreate(Sender: TObject);
var
    Book: TBook;
        Item: TListItem;
begin
    AssignFile(F, 'Base\Books.db');
    Reset(F);
    while not Eof(F) do
    begin
        Read(F, Book);
        Item := ListView1.Items.Add;
        Item.Caption:= IntToStr(Book.Num);
        Item.SubItems.Add(Book.Autor);
        Item.SubItems.Add(Book.Book);
        Item.SubItems.Add(Book.City);
        Item.SubItems.Add(Book.Publ);
        Item.SubItems.Add(IntToStr(Book.Year));
    end;
    CloseFile(F);
end;
```

Здесь осуществляются чтение файла базы данных и вывод его содержимого в компонент TListView.

Далее запрограммируем процедуру нажатия на кнопку «Справочники» на главной форме:

```
procedure TForm1.btn_ListClick(Sender: TObject);
var
    P: TPoint;
begin
    Form2.ListBox1.Clear;
    Form2.e_Name.Clear;
    P := GetClientOrigin;
    PopupMenu1.Popup(P.X+btn_List.Left, P.Y+btn_List.Top);
end;
```

Теперь запишем программный код в обработчики событий нажатия на пункты контекстного меню, которые будут записывать содержимое файлов Autors.db, Cites.db, Publ.db в компонент TList второй формы, в зависимости от того, какой пункт меню выбран:

```
procedure TForm1.mAutorsClick(Sender: TObject);
var
    Str: string[20];
```

```
begin
    Mode := 1;
    Form2.Caption := 'Справочник Авторы';
    AssignFile(Fs, 'Base\Autors.db');
    Reset(Fs);
    while not Eof(Fs) do
    begin
        ReadLn(Fs, Str);
        Form2.ListBox1.Items.Add(Str);
    end;
    CloseFile(Fs);
    Form2.Show;
end;

procedure TForm1.mCityClick(Sender: TObject);
var
    Str: string;
begin
    Mode := 2;
    Form2.Caption := 'Справочник Место издания';
    AssignFile(Fs, 'Base\Cites.db');
    Reset(Fs);
    while not Eof(Fs) do
    begin
        ReadLn(Fs, Str);
        Form2.ListBox1.Items.Add(Str);
    end;
    CloseFile(Fs);
    Form2.Show;
end;

procedure TForm1.mPublClick(Sender: TObject);
var
    Str: string[15];
begin
    Mode := 3;
    Form2.Caption := 'Справочник Издательства';
    AssignFile(Fs, 'Base\Publ.db');
    Reset(Fs);
```

```
        while not Eof(Fs) do
        begin
            ReadLn(Fs, Str);
            Form2.ListBox1.Items.Add(Str);
        end;
        CloseFile(Fs);
        Form2.Show;
    end;
```

И наконец, запишем код обработки событий нажатия на кнопки «Добавить запись», «Изменить запись» и «Удалить запись» соответственно:

```
    procedure TForm1.btn_AddClick(Sender: TObject);
    begin
        Mode := 0;
        Form3.Show;
    end;

    procedure TForm1.bnt_EditClick(Sender: TObject);
    begin
        if ListView1.ItemIndex = -1 then
            ShowMessage('Выделите нужную запись.')
        else
        begin
            Mode := 1;
            Form3.Show;
        end;
    end;

    procedure TForm1.btn_DeleteClick(Sender: TObject);
    var
        I: integer;
        Book: TBook;
    begin
        if ListView1.ItemIndex = -1 then
            ShowMessage('Выделите нужную запись.')
        else
        begin
            AssignFile(F, 'Base\Books.db');
            Reset(F);
```

```
            Seek(F, ListView1.ItemIndex);
            Truncate(F);
            for I := ListView1.ItemIndex + 1 to ListView1.Items.Count-1 do
            begin
                Book.Num := I;
                Book.Autor := ListView1.Items[I].SubItems[0];
                Book.Book := ListView1.Items[I].SubItems[1];
                Book.City := ListView1.Items[I].SubItems[2];
                Book.Publ := ListView1.Items[I].SubItems[3];
                Book.Year := StrToInt(ListView1.Items[I].SubItems[4]);
                Write(F, Book);
                ListView1.Items[I].Caption := IntToStr(I);
            end;
            CloseFile(F);
            ListView1.Items[ListView1.ItemIndex].Delete;
        end;
    end;
```

Теперь рассмотрим программный код формы «Справочники». Определим глобальные переменные:

```
    F: TextFile;
    NameF: string;
```

Запишем подпрограммы обработчика нажатий на кнопки «Добавить», «Изменить», «Удалить» и «Выход»:

```
    procedure TForm2.btn_AddClick(Sender: TObject);
    begin
        if e_Name.Text = '' then
            ShowMessage('Не указано значение поля!')
        else
        begin
            case Form1.Mode of
                1: NameF := 'Base\Autors.db';
                2: NameF := 'Base\Cites.db';
                3: NameF := 'Base\Publ.db';
            end;
            AssignFile(F, NameF);
            Append(F);
            Form2.ListBox1.Items.Add(e_Name.Text);
            WriteLn(F, e_Name.Text);
            CloseFile(F);
```

```
            e_Name.Clear;
        end;
end;

procedure TForm2.btn_EditClick(Sender: TObject);
var
    I: integer;
begin
    if (e_Name.Text = '')or(ListBox1.ItemIndex = -1) then
        ShowMessage('Не выбран элемент списка!')
    else
    begin
        case Form1.Mode of
            1: NameF := 'Base\Autors.db';
            2: NameF := 'Base\Cites.db';
            3: NameF := 'Base\Publ.db';
        end;
        Form2.ListBox1.Items.Strings[ListBox1.ItemIndex] := e_Name.Text;
        AssignFile(F, NameF);
        Rewrite(F);
        for I := 0 to ListBox1.Items.Count-1 do
            WriteLn(F, ListBox1.Items[I]);
        CloseFile(F);
        e_Name.Clear;
    end;
end;

procedure TForm2.btn_DeleteClick(Sender: TObject);
var
    I: integer;
begin
    if e_Name.Text = '' then
        ShowMessage('Не выбран элемент списка!')
    else
    begin
        case Form1.Mode of
            1: NameF := 'Base\Autors.db';
            2: NameF := 'Base\Cites.db';
            3: NameF := 'Base\Publ.db';
```

```
                end;
                Form2.ListBox1.Items.Delete(ListBox1.ItemIndex);
                AssignFile(F, NameF);
                Rewrite(F);
                for I := 0 to ListBox1.Items.Count-1 do
                    WriteLn(F, ListBox1.Items[I]);
                CloseFile(F);
                e_Name.Clear;
            end;
        end;

        procedure TForm2.bnt_CloseClick(Sender: TObject);
        begin
            Form2.Close;
        end;
```

Также определим события нажатия на компонент TList и показа формы:

```
        procedure TForm2.bnt_CloseClick(Sender: TObject);
        begin
            Form2.Close;
        end;

        procedure TForm2.FormShow(Sender: TObject);
        begin
            ListBox1.ItemIndex := -1;
        end;
```

Теперь перейдем к описанию третьей формы «Редактирование». Опишем тип данных TBook:

```
        TBook = record
                Num: integer;
                Autor: string[20];
                Book: string[100];
                City: string[15];
                Publ: string[15];
                Year: integer;
            end;
```

Глобальные переменные:

```
        Fs: TextFile;
        Fbook: File of TBook;
```

Определим событие нажатия на кнопку «OK». Здесь мы производим редактирование записей в файле Books.db:

```
procedure TForm3.btn_OkClick(Sender: TObject);
var
    Book: TBook;
    Item: TListItem;
begin
    if (box_Autor.Text = '')or(e_Name.Text = '')or(box_City.Text = '')or(
box_Publ.Text = '')or(box_Year.Text = '')then
        ShowMessage('Введены не все данные')
    else
    begin
        if Form1.Mode = 0 then
        begin
            AssignFile(Fbook, 'Base\Books.db');
            Reset(Fbook);
            Seek(Fbook, FileSize(FBook));
            Book.Num := FileSize(FBook) + 1;
            Book.Autor := box_Autor.Text;
            Book.Book := e_Name.Text;
            Book.City := box_City.Text;
            Book.Publ := box_Publ.Text;
            Book.Year := StrToInt(box_Year.Text);
            Write(Fbook, Book);
            CloseFile(Fbook);
            Item := Form1.ListView1.Items.Add;
            Item.Caption := IntToStr(Form1.ListView1.Items.Count);
            Item.SubItems.Add(box_Autor.Text);
            Item.SubItems.Add(e_Name.Text);
            Item.SubItems.Add(box_City.Text);
            Item.SubItems.Add(box_Publ.Text);
            Item.SubItems.Add(box_Year.Text);
            Form3.Close;
        end
    else
        begin
            AssignFile(Fbook, 'Base\Books.db');
            Reset(Fbook);
            Seek(Fbook, Form1.ListView1.ItemIndex);
```

```
                    Book.Num := Form1.ListView1.ItemIndex+1;
                    Book.Autor := box_Autor.Text;
                    Book.Book := e_Name.Text;
                    Book.City := box_City.Text;
                    Book.Publ := box_Publ.Text;
                    Book.Year := StrToInt(box_Year.Text);
                    Write(Fbook, Book);
                    CloseFile(Fbook);
  Form1.ListView1.Items[Form1.ListView1.ItemIndex].SubItems[0] :=
                    Form3.box_Autor.Text;
  Form1.ListView1.Items[Form1.ListView1.ItemIndex].SubItems[1] :=
                    Form3.e_Name.Text;
  Form1.ListView1.Items[Form1.ListView1.ItemIndex].SubItems[2] :=
                    Form3.box_City.Text;
  Form1.ListView1.Items[Form1.ListView1.ItemIndex].SubItems[3] :=
                    Form3.box_Publ.Text;
  Form1.ListView1.Items[Form1.ListView1.ItemIndex].SubItems[4] :=
                    Form3.box_Year.Text;
                        Form3.Close;
                    end;
                end;
            end;
```

Запишем процедуру показа формы, в которой будем заполнять данными компоненты TComboBox, считанными из файлов:

```
procedure TForm3.FormShow(Sender: TObject);
var
    I: integer;
    Str: string;
begin
    box_Autor.Clear;
    AssignFile(Fs, 'Base\Autors.db');
    Reset(Fs);
    while not Eof(Fs) do
    begin
        ReadLn(Fs, Str);
        box_Autor.Items.Add(Str);
    end;
    CloseFile(Fs);
```

```
box_City.Clear;
AssignFile(Fs, 'Base\Cites.db');
Reset(Fs);
while not Eof(Fs) do
begin
      ReadLn(Fs, Str);
box_City.Items.Add(Str);
end;
CloseFile(Fs);

box_Publ.Clear;
AssignFile(Fs, 'Base\Publ.db');
Reset(Fs);
while not Eof(Fs) do
begin
      ReadLn(Fs, Str);
      box_Publ.Items.Add(Str);
end;
CloseFile(Fs);

box_Year.Clear;
for I := 0 to 20 do
      box_Year.Items.Add(IntToStr(2000+I));

if Form1.Mode = 0 then
begin
      e_Name.Clear;
      box_Autor.ItemIndex := -1;
      box_City.ItemIndex := -1;
      box_Publ.ItemIndex := -1;
      box_Year.ItemIndex := -1;
end
else
begin
      box_Autor.ItemIndex := box_Autor.Items.IndexOf(
Form1.ListView1.Items[Form1.ListView1.ItemIndex].SubItems[0]);
      e_Name.Text := Form1.ListView1.Items[
Form1.ListView1.ItemIndex].SubItems[1];
```

```
            box_City.ItemIndex := box_City.Items.IndexOf(
        Form1.ListView1.Items[Form1.ListView1.ItemIndex].SubItems[2]);
            box_Publ.ItemIndex := box_Publ.Items.IndexOf(
        Form1.ListView1.Items[Form1.ListView1.ItemIndex].SubItems[3]);
            box_Year.ItemIndex := box_Year.Items.IndexOf(
        Form1.ListView1.Items[Form1.ListView1.ItemIndex].SubItems[4]);
        end;
end;
```

И напоследок опишем обработчик события нажатия на кнопку «Отмена»:

```
procedure TForm3.btn_CanselClick(Sender: TObject);
begin
    Form3.Close;
end;
```

После этого нам необходимо подключить модули (раздел uses) в главной форме Unit 2 и Unit 3, во второй – Unit 1 и в третьей – Unit 1. Кроме того, надо в модуле Unit 3 подключить модуль ComCtrls для корректной работы программы.

На этом написание программы-сервера можно считать законченным.

Рассмотрим программу-клиент. Интерфейс главного окна изображен на рис. 1.10.

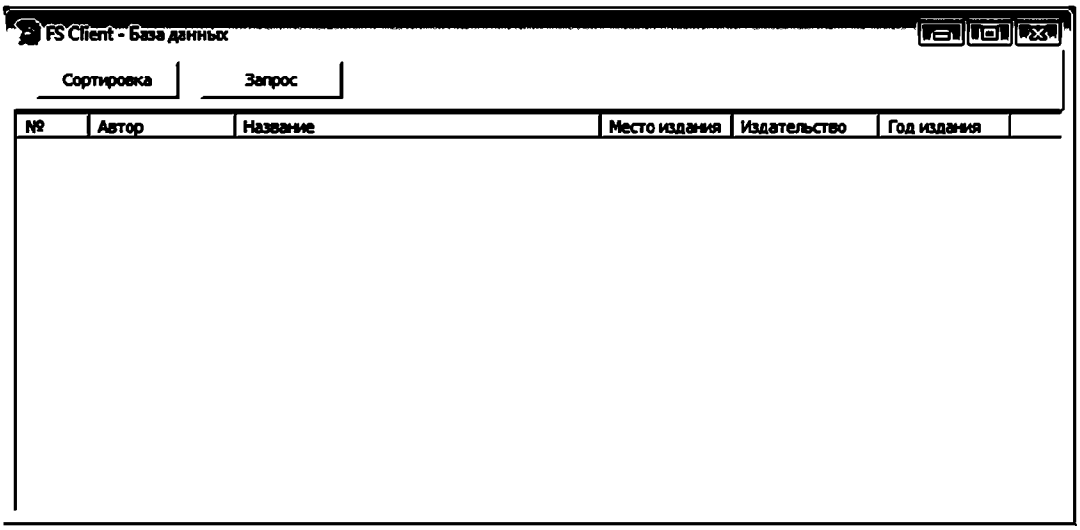

Рис. 1.10. Главное окно программы-клиента

Также программа будет включать форму «Сортировка» (рис. 1.11) и форму «Запрос» (рис. 1.12).

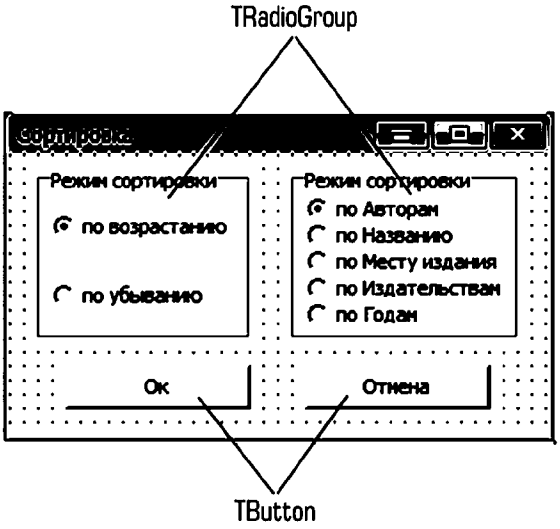

Рис. 1.11. Окно формы «Сортировка»

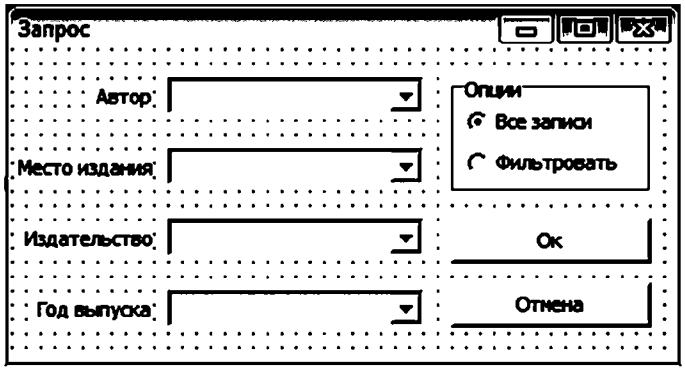

Рис. 1.12. Окно формы «Запрос»

Рассмотрим модуль главного окна программы. Вначале в описании класса формы в разделе public определим переменную Mode типа byte.

Далее опишем сложный тип данных записью:

```
TBook = record
        Num: integer;
        Autor: string[20];
        Book: string[100];
        City: string[15];
        Publ: string[15];
        Year: integer;
    end;
```

И глобальную переменную: F: File of TBook;

Процедуру создания формы:

```
procedure TForm1.FormCreate(Sender: TObject);
var
    Book: TBook;
    Item: TListItem;
begin
    AssignFile(F, 'Base\Books.db');
    Reset(F);
    while not Eof(F) do
    begin
        Read(F, Book);
        Item := ListView1.Items.Add;
        Item.Caption:= IntToStr(Book.Num);
        Item.SubItems.Add(Book.Autor);
        Item.SubItems.Add(Book.Book);
        Item.SubItems.Add(Book.City);
        Item.SubItems.Add(Book.Publ);
        Item.SubItems.Add(IntToStr(Book.Year));
    end;
    CloseFile(F);
end;
```

Также определим обработчики нажатия на кнопки «Сортировка» и «Запрос»:

```
procedure TForm1.btn_SortClick(Sender: TObject);
begin
    Form2.Show;
end;
procedure TForm1.btn_QueryClick(Sender: TObject);
begin
    Form3.Show;
end;
```

В модуле второй формы опишем процедуру, которая будет осуществлять перезапись элементов списка в главной форме:

```
procedure ChangePos (I: integer);
var
    S: string;
    K: integer;
begin
    S := Form1.ListView1.Items[I].Caption;
```

```
            Form1.ListView1.Items[I].Caption := Form1.ListView1.Items[I+1].Caption;
            Form1.ListView1.Items[I+1].Caption := S;
            for k := 0 to 4 do
            begin
                S := Form1.ListView1.Items[I].SubItems[K];
                Form1.ListView1.Items[I].SubItems[K] :=
            Form1.ListView1.Items[I+1].SubItems[K];
                Form1.ListView1.Items[I+1].SubItems[K] := S;
            end;
        end;
```

Также опишем обработчик нажатия на кнопку «OK», в котором описывается порядок сортировки записей на главной форме программы:

```
        procedure TForm2.btn_OkClick(Sender: TObject);
        var
            I, J: integer;
        begin
            Form2.Close;
            for J := 0 to Form1.ListView1.Items.Count-1 do
            for I := 0 to Form1.ListView1.Items.Count-2 do
            begin
                if RadioGroup1.ItemIndex = 0 then
                begin
                    if (Form1.ListView1.Items[I].SubItems[
                RadioGroup2.ItemIndex]>Form1.ListView1.Items[I+1].SubItems[
                RadioGroup2.ItemIndex])then
                    ChangePos(I);
                end
                else
                begin
                    if (Form1.ListView1.Items[I].SubItems[
                RadioGroup2.ItemIndex]<Form1.ListView1.Items[I+1].SubItems[
                RadioGroup2.ItemIndex])then
                    ChangePos(I);
                end;
            end;
        end;
```

Зададим обработчик нажатия на кнопку «Отмена»:

```
        procedure TForm2.btn_CanselClick(Sender: TObject);
```

```
begin
    Form2.Close;
end;
```

В обработчике события показа формы OnShow запишем:

```
RadioGroup1.ItemIndex := 0;
RadioGroup2.ItemIndex := 0;
```

Теперь рассмотрим третий модуль программы Unit 3. Опишем в нем новый тип:

```
TBook = record
        Num: integer;
        Autor: string[20];
        Book: string[100];
        City: string[15];
        Publ: string[15];
        Year: integer;
    end;
```

И глобальные переменные:

```
F: File of TBook;
Fs: TextFile;
```

Также определим новую процедуру, выводящую на экран главной формы данные в компонент TListView:

```
procedure PrintItem(Book: TBook);
var
    Item: TListItem;
begin
    Item := Form1.ListView1.Items.Add;
    Item.Caption:= IntToStr(Book.Num);
    Item.SubItems.Add(Book.Autor);
    Item.SubItems.Add(Book.Book);
    Item.SubItems.Add(Book.City);
    Item.SubItems.Add(Book.Publ);
    Item.SubItems.Add(IntToStr(Book.Year));
end;
```

Процедуру обработки события показа окна:

```
procedure TForm3.FormShow(Sender: TObject);
var
    I: integer;
    Str: string;
begin
    RadioGroup1.ItemIndex := 0;
```

```
        box_Autor.Clear;
        AssignFile(Fs, 'Base\Autors.db');
        Reset(Fs);
        while not Eof(Fs) do
        begin
            ReadLn(Fs, Str);
            box_Autor.Items.Add(Str);
        end;
        CloseFile(Fs);

        box_City.Clear;
        AssignFile(Fs, 'Base\Cites.db');
        Reset(Fs);
        while not Eof(Fs) do
        begin
            ReadLn(Fs, Str);
            box_City.Items.Add(Str);
        end;
        CloseFile(Fs);

        box_Publ.Clear;
        AssignFile(Fs, 'Base\Publ.db');
        Reset(Fs);
        while not Eof(Fs) do
        begin
            ReadLn(Fs, Str);
            box_Publ.Items.Add(Str);
        end;
        CloseFile(Fs);

        box_Year.Clear;
        for I := 0 to 20 do
            box_Year.Items.Add(IntToStr(2000+I));
    end;
```

В обработчик события нажатия на кнопку «OK» запишем:

```
    procedure TForm3.btn_OkClick(Sender: TObject);
    var
        Book: TBook;
    begin
```

```
        Form1.ListView1.Clear;
        AssignFile(F, 'Base\Books.db');
        Reset(F);
        while not Eof(F) do
        begin
            Read(F, Book);
            if (RadioGroup1.ItemIndex = 1)and(
        (box_Autor.Text=Book.Autor)or(box_Autor.ItemIndex=-1))and(
        (box_City.Text=Book.City)or(box_City.ItemIndex=-1))and(
        (box_Publ.Text=Book.Publ)or(box_Publ.ItemIndex=-1))and(
        (box_Year.Text=IntToStr(Book.Year))or(box_Year.ItemIndex=-1))then
                PrintItem(Book)
            else if RadioGroup1.ItemIndex = 0 then
                PrintItem(Book);
        end;
        CloseFile(F);
        Form3.Close;
    end;
```

В обработчике события OnClick кнопки «Отмена» укажем:

```
    Form3.Close;
```

После настройки всех свойств обоих приложений и написания программного кода должны быть созданы серверное и клиентское приложения по обработке файловой базы данных по технологии File Server.

Вопросы для самопроверки

1. Перечислите современные системы коллективной человеческой деятельности.
2. Сформулируйте понятие «клиент-серверная технология», опишите ее принципы и перечислите основные компоненты.
3. Какие существуют подходы для реализации «клиент-серверной» технологии?
4. Что такое Web-технологии?
5. Опишите архитектуру Web-технологий.

Тесты

1. Какими особенностями характеризуются системы с разделением времени (TSS)?
   А. Защита данных от несанкционированного доступа.
   Б. Взаимная изоляция участников.
   В. Использование общих ресурсов всеми участниками.
   Г. Использование каждым участником собственной ЭВМ.

2. Какими особенностями характеризуются системы обеспечения групповых решений (CSCW)?
   А. Использование общих ресурсов всеми участниками.
   Б. Ваимная изоляция участников.
   В. Защита данных от несанкционированного доступа.
   Г. Взаимодействие пользователей в процессе принятия решений.

3. Какие функции выполняет компонент представления (presentation) интерактивного приложения?
   А. Функции хранения и управления информационно-вычислительными ресурсами.
   Б. Функции ввода и отображения данных.
   В. Прикладные функции.
   Г. Служебные функции.

4. В каких подходах реализована двухзвенная схема разделения функций между компонентами приложений?
   А. Сервер баз данных (DBS).
   Б. Файловый сервер (FS).
   В. Сервер приложений (AS).
   Г. Доступ к удаленным данным (RDA).

5. В каких подходах реализована трехзвенная схема разделения функций между компонентами приложений?
   А. Сервер баз данных (DBS).
   Б. Файловый сервер (FS).
   В. Сервер приложений (AS).
   Г. Доступ к удаленным данным (RDA).

ГЛАВА 2. СЕТЕВЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ

2.1. Назначение и функциональные компоненты

Сетевая операционная система (*сетевая ОС*) – это комплекс взаимосвязанных программ, предоставляющих пользователям, программистам и администраторам эффективный способ разделения ресурсов между выполняемыми в сети процессами путем предоставления им виртуальной вычислительной системы.

Сетевая операционная система может выполнять функции автономного компьютера, а также решать задачи по организации взаимодействия между процессами, выполняющимися на разных машинах.

Перечислим основные функции сетевой ОС.

Управление процессами связано с организацией хранения данных о ресурсах вычислительной системы, в том числе и о фактически выделенных ресурсах для процессов, развивающихся в системе. В качестве таких ресурсов выступают оперативная память, процессор, файлы, устройства ввода вывода и т.п. Принято выделять пользовательские и системные процессы, которые основаны на следующих принципах:

- поддержка очереди заявок к ресурсам с учетом их характеристик;
- защита ресурсов, выделенных процессу, от других ресурсов;
- организация работы процесса в своем адресном пространстве оперативной памяти;
- обеспечение возможности многократного прерывания и возобновления процесса.

Управление памятью заключается в следующем:

- распределение физической памяти между процессами и ее защита;
- настройка адресно-зависимых частей кодов процесса;
- загрузка кодов в отведенную область памяти.

Управление файлами и внешними устройствами состоит в виртуализации набора данных, хранящихся на внешнем накопителе, в виде файла, осуществляя, таким образом, представление набора данных, разбросанных на физическом носителе информации, в виде иерархической структуры файлов и каталогов.

Защита данных и администрирование заключается в следующем:

- защита данных от несанкционированного доступа;
- фиксация событий, влияющих на безопасность системы (так называемый аудит ОС);

- поддержка отказоустойчивости и резервирование;
- резервное копирование информации;
- наличие утилит для администратора.

Сетевое программное обеспечение представляет собой совокупность операционных систем отдельных компьютеров, работающих в сети. По причине того, что в одной сети могут работать компьютеры с различными ОС, для организации работы между ними используется согласованный набор коммуникационных протоколов.

Рассмотрим функциональные компоненты сетевой ОС (рис. 2.1). Среди них можно выделить следующие:

- средства управления локальными ресурсами включают в себя все ресурсы автономного компьютера;
- клиентское ПО представляет собой средства запроса на доступ к удаленным ресурсам и услугам;
- серверное ПО состоит из средств, предоставляющих ресурсы и услуги в общее пользование;
- транспортные средства являются компонентами системы, которые обеспечивают передачу сообщений между компьютерами сети через коммуникационную систему.

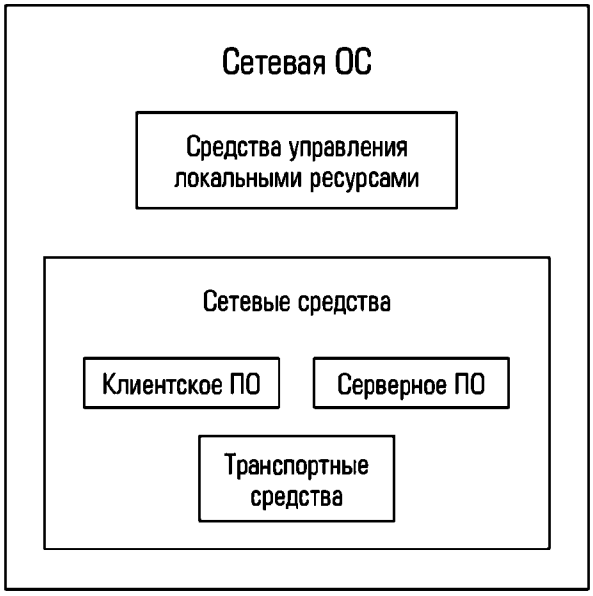

Рис. 2.1. Компоненты сетевой ОС

2.2. Администрирование и конфигурирование сетевых операционных систем

Рассмотрим особенности установки сетевой операционной системы на примере Windows-системы. Процесс установки включает в себя следующие этапы:

- запрос необходимой информации;
- копирование файлов;
- перезапуск системы.

Завершающим этапом установки ОС является настройка сервера с помощью Configure Your Server Wizard (Мастер настройки сервера).

Для запуска установки ОС с использованием загрузочного диска следует выполнить ряд действий.

1. Вставьте диск в дисковод D: и перезагрузите компьютер или откройте командную строку DOS и выполните команду D:\winnt. Выполнение этой команды позволит загрузить в память компьютера минимальный объем файлов, необходимый для установки ОС.
2. После появления на экране информации о лицензировании необходимо принять условия лицензионного соглашения и продолжить установку, приступив к созданию разделов жесткого диска.
3. Далее программа установки запросит указать раздел, в который необходимо установить операционную систему. Для этого выберите существующий раздел либо создайте новый.
4. Укажите желаемый тип файловой системы (FAT16, FAT32 или NTFS). После этого произойдет форматирование указанного раздела и программа установки приступит к копированию в него файлов. Завершив копирование файлов, программа установки сохранит начальную конфигурацию новой системы и выполнит перезагрузку компьютера.

После перезагрузки системы на экране появится окно мастера установки Windows. Файлы операционной системы будут установлены в папку C:\Winnt. Мастер установки запросит ввести сведения о себе и о компьютере, организации или компании, владеющей лицензией на программное обеспечение. После получения этих сведений мастер начнет автоматический процесс установки, в результате чего на жесткий диск компьютера будет скопированы файлы, необходимые для поддержки конфигурации аппаратного обеспечения, установленных устройств и т.п.

Затем система произведет установку необходимых сетевых компонентов. В частности, программа установки Windows попытается распознать сетевую плату, настройка которой состоит из следующих этапов.

Распознавание сетевой платы. После распознания сетевой платы и установки драйверов программа установки определит местоположение в сети DHCP-сервера (то есть попытается получить динамический IP-адрес), в случае неудачи система самостоятельно присвоит себе IP-адрес.

Выбор сетевых компонентов. На этом этапе необходимо выбрать сетевые компоненты. В число основных компонентов входят Client

for Microsoft (Клиент для сетей Microsoft), File and Print Sharing for Microsoft Networks (Служба доступа к файлам и принтерам сетей Microsoft) и протокол TCP/IP.

Рабочая группа или домен. В этом разделе осуществляется настройка доменов и групп. Если сервер необходимо присоединить к домену, то потребуются имя и пароль учетной записи, обладающей привилегией создания новых учетных записей домена. В случае невозможности присоединения к домену добавьте сервер в рабочую группу. Если существующих рабочих групп нет, то создайте новую группу с любым именем. В любом случае после завершения установки всегда можно изменить имя рабочей группы либо присоединить сервер к домену.

Завершающим действиями установки операционной системы являются окончательное копирование файлов, настройка системы и удаление временных файлов. После этого программа установки перезагрузит компьютер.

В операционных системах любой пользователь, процесс, компьютер или программа, которым необходимо взаимодействовать с другим объектом в сети, трактуется как пользователь (user). Объединение пользователей, контактов или компьютеров называют группой (group). Пользователи и Группы поддерживаются службой каталога Active Directory, в которую входят средства управления учетными записями. К ним относятся средства Stored User Names and Passwords (Сохранение имен пользователей и паролей) и Local Users and Groups (Локальные пользователи и группы), установленные на локальных компьютерах и рядовых серверах, а также Active Directory Users and Computers (Active Directory – пользователи и компьютеры) на контроллерах доменов. Хранящиеся в Active Directory учетные записи пользователей называют учетными записями домена (или сетевыми учетными записями). Люди, компьютеры и процессы используют учетные записи домена, входя в сеть и получая доступ к ее ресурсам. При каждой попытке входа в сеть осуществляется идентификация, в процессе которой система сравнивает друг с другом пароль, введенный пользователем, и пароль, сохраненный в Active Directory. В случае успешной проверки пароля пользователь получает доступ к ресурсам, компьютерам и коммуникациям. Учетные записи пользователей объединены в группы, что позволяет более безопасно предоставлять и контролировать доступ к ресурсам.

Инструментом контроля изменений в Windows является средство Group Policy Object Editor (Редактор объектов групповой политики). Групповая политика используется для управления безопасностью и конфигурирования аппаратного обеспечения. Применение групповой политики заключается в создании объекта, свойства которого

определяют права доступа компьютера и пользователя к ресурсам сети и ресурсам локального компьютера. Этот объект называется объектом групповой политики (Group Policy Object). Конкретная политика создается на основе шаблонов, хранящихся на рабочей станции или сервере.

Существуют следующие типы политик.

Развертывание приложений используется для управления доступом пользователя к приложениям.

Развертывание файлов позволяет размещать файлы в определенных папках на компьютерах пользователей.

Создание сценариев позволяет назначать сценарии для запуска в строго определенное время. Они применяются в случае необходимости выполнения сценариев при включении или выключении компьютера.

Программное обеспечение позволяет настраивать программное обеспечение на компьютерах пользователей как на глобальном, так и на целевом уровнях.

Безопасность используется для защиты системы от взлома и несанкционированного доступа к ресурсам.

Взаимодействие между компьютерами в сети осуществляется с использованием IP-адресов, которые назначаются каждому из узлов сети. Каждый узел должен обладать уникальным IP-адресом. IP-адреса могут быть определены одним из двух способов – динамически или статически. При статическом назначении адреса присваиваются вручную каждому узлу. Динамическое назначение адресов производится с использованием протокола DHCP (Dynamic Host Configuration Protocol – протокол динамической конфигурации узла). Данный протокол позволяет автоматически назначать IP-адреса и соответствующие свойства при загрузке узла.

Одним из компонентов сетевой ОС является служба печати, предназначенная для организации работы принтеров в системе. Принято выделять логическую и физическую среду печати. Логическая среда представляет собой абстракцию физического устройства печати, видимого пользователю, и включает в себя необходимое для него программное обеспечение. Физическая среда – это собственно те устройства, которые предназначены для печати. В данном случае операционная система предоставляет пользователям логический интерфейс, позволяющий управлять процессом печати, а также среду очереди печати и включения принтера в пул. Логический принтер представляет собой пиктограмму и свойства. Они устанавливаются на пользовательских компьютерах (локальные принтеры) либо на серверах, предназначенных для предоставления служб печати (сетевые принтеры).

2.3. Типы серверов и способы удаленного управления сервером

Сетевые операционные системы семейства Windows объединяют несколько служб, которые могут быть использованы клиентами интрасети и Internet, называемые информационные службы Internet (Internet Information Services, IIS). Перечислим их:

- *серверные расширения фоновой интеллектуальной службы передачи* (Background Intelligent Transfer Service, BITS) позволяют передавать данные маленькими частями, используя свободную пропускную способность сети;
- *диспетчер информационных служб Internet* представляет собой консоль управления, которая позволяет настраивать службы IIS;
- *сервер WWW* (World Wide Web) позволяет настроить Windows для выполнения функций HTTP-сервера в WWW;
- *сервер FTP* (File Transfer Protocol) обеспечивает обмен фалами между удаленными компьютерами по сети;
- *служба SMTP* (Simple Mail Transfer Protocol) позволяет настроить компьютер для выполнения функций SMTP-сервера (для передачи электронной почты);
- *служба NNTP* (Network News Transfer Protocol) позволяет настроить компьютер для выполнения функций NNTP-сервера (для передачи сетевых новостей);
- *серверные расширения FrontPage* позволяют поддерживать Web-узлы, созданные с помощью приложения Microsoft FrontPage;
- *печать через Internet* позволяет создать виртуальный каталог Printers и скопировать файлы, необходимые для обнаружения принтера, его установку и отправку заданий печати посредством протокола HTTP.

Одной из разновидностей технологии «клиент-сервер» является служба терминалов, вернее, модели вычислений «тонкий клиент-сервер». Первоначально согласно этой технологии «тонкий клиент» функционировал только в качестве простого терминала ввода-вывода. В его обязанности входили вывод изображения на экран клиента и передача серверу нажатий клавиш и щелчков кнопками мыши. Вся обработка данных, выполнение программ и доступ к хранилищам производись на сервере. Сейчас согласно технологии «тонкий клиент-сервер» пользователь имеет в своем распоряжении только монитор, клавиатуру и мышь, а сервер полностью обеспечивает функционирование пользовательской среды и всех приложений, то есть рабочее пространство каждого приложения пользователя (процессор, жесткий диск и память) находится на сервере.

Службы терминалов Windows состоят из следующих компонентов:

- многопользовательское ядро;
- протокол RDP;
- клиент Remote Desktop Connection (подключение к удаленному рабочему столу);
- клиент Remote Desktop Web Connection (интернет-подключение к удаленному рабочему столу);
- средство Remote Assistance (удаленный помощник);
- служба лицензирования сервера терминалов;
- средства администрирования служб терминалов.

Для управления службами терминалов используются два стандартных средства администрирования: Terminal Services Manager (Диспетчер служб терминалов) и Terminal Services Configuration (Настройка служб терминалов).

Технология «тонкий клиент-сервер» в Windows реализуется на основе компонента Remote Desktop (Удаленный рабочий стол), который позволяет получить доступ к рабочему столу, запущенному на удаленном сервере. При этом графический интерфейс не всегда возможен при низкоскоростных соединениях, а также при подключении к серверам Windows клиентов под управлением других операционных систем. Для обхода этих ограничений используют средства протокола Telnet. Он активно используется администраторами Unix. Протокол Telnet является частью стека протоколов TCP/IP и состоит из клиентского и серверного компонентов. Клиент Telnet входит в состав каждой стандартной реализации TCP/IP независимо от типа операционной системы.

Для открытия сеанса Telnet с компьютера клиента наберите в командной строке слово telnet и нажмите клавишу <Enter> или выполните команду Start → Run и в появившемся окне Run введите слово telnet. Для получения списка поддерживаемых команд выполните команду help либо ?. Приведем некоторые популярные команды:

o имя_узла порт – открывает подключение Telnet к конкретному порту узла с определенным именем;

c – закрывает текущее подключение;

q – завершает работу Telnet;

display – отображает текущие параметры работы.

Вопросы для самопроверки

1. Дайте определение сетевой операционной системы и перечислите ее основные функции.
2. В чем заключается процесс установки сетевой операционной системы?
3. Каким образом настраивается сетевая плата?
4. Что такое пользователи и группы пользователей и в чем состоит их настройка?
5. Что такое служба терминалов и каковы ее основные компоненты?

Тесты

3.1. Какая из функций сетевой операционной системы связана с организацией хранения данных о ресурсах вычислительной системы?
- А. Управление процессами.
- Б. Управление памятью.
- В. Управление файлами и внешними устройствами.
- Г. Защита данных и администрирование.

4.1. Какая из функций сетевой операционной системы состоит в виртуализации набора данных, хранящихся на внешнем накопителе?
- А. Управление процессами.
- Б. Управление памятью.
- В. Управление файлами и внешними устройствами.
- Г. Защита данных и администрирование.

5.1. Какие функциональные компоненты сетевой операционной системы представляют собой средства запроса на доступ к удаленным ресурсам и услугам?
- А. Средства управления локальными ресурсами.
- Б. Клиентское ПО.
- В. Серверное ПО.
- Г. Транспортные средства.

6.1. Какие функциональные компоненты сетевой операционной системы обеспечивают передачу сообщений между компьютерами сети через коммуникационную систему?
- А. Средства управления локальными ресурсами.
- Б. Клиентское ПО.
- В. Серверное ПО.
- Г. Транспортные средства.

7.1. Какой из типов групповой политики используется для защиты системы от взлома и несанкционированного доступа к ресурсам?
- А. Развертывание приложений.
- Б. Развертывание файлов.
- В. Создание сценариев.
- Г. Безопасность.

Глава 3. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ WEB-СИСТЕМ

3.1. Классификация программного обеспечения

Программное обеспечение для Web-систем можно разделить на группы по направлениям использования. Каждое из этих направлений определяется либо схемой взаимодействия компонентов Web-технологии, либо особенностями применения его субъектами обмена информацией в рамках World Wide Web (WWW). Принята следующая классификация:

- программы клиенты (в том числе многопротокольные программы-браузеры);
- программы просмотра документов в форматах, отличных от стандартных форматов Web;
- программы-серверы протокола обмена гипертекстовой информацией (Web-серверы);
- программы подготовки публикаций; поисковые машины;
- программы анализа статистики посещений.

Программы-клиенты стандартны для Web (например, MS Internet Explorer, Opera). Основная задача этих программ — интерпретация разметки на языке HTML, интерпретация встроенных в HTML программ на одном из командных языков Web — JavaScript или VBScript, интерпретация Java-апплетов, разбор спецификации ресурсов сети (обработка URL), взаимодействие с серверами по протоколам прикладного уровня стека протоколов TCP/IP.

Программы просмотра файлов специальных форматов. В том случае, если информация, передаваемая в рамках Web, не может быть просмотрена программой-браузером, необходим запуск дополнительного программного обеспечения. К подобного рода информации относятся файлы в форматах Postscript и PDF, документы Word и WordPerfect и ряд других информационных компонентов, которые доступны по сети. Для их просмотра браузер запускает дополнительную программу, которой передает файл для обработки. Наиболее часто используются: Acrobat (PDF-файлы); Ghost в комплекте с GhostView (файлы формата PostScript); программы просмотра аудио-, видеоинформации (формат MPEG); программы просмотра графических файлов и ряд других.

Программы-серверы — это программы, принимающие запросы от Web-клиентов и отвечающие на них. В качестве ответа может быть

возвращен HTML-документ, хранящийся в базе данных сервера, графический образ, аудиозапись, фильм или ответ внешней программы. Наиболее популярным сервером является Apachie по следующим причинам: свободно распространяемый программный код, обилие дополнительных модулей, расширяющих стандартный набор программного обеспечения поставки сервера, наименьшее количество ошибок, высокая мобильность и эффективность кода для различных платформ, простота установки.

Программы подготовки публикаций предназначены для создания и редактирования HTML-документов (Page Compouser (Netscape), MS FrontPage (Microsoft)). Эти программы позволяют упростить процесс разработки HTML-документа за счет автоматизации расстановки внутри него тегов языка HTML (т.е. разметки документа). Практически все редакторы позволяют запускать внешнюю программу просмотра для тестирования набранного документа, а некоторые из них синхронно отображают набираемый текст и графику, демонстрируя Web-страницу в том виде, как ее увидят пользователи.

Информационно-поисковые системы могут быть разделены по функционально-структурному принципу на следующие классы:

- полностью распределенные системы, где реализуются принципы распределенных вычислений и распределенного хранения данных. Процесс поиска реализуется на совокупности распределенных по сети серверов, которые опрашивают друг друга при обработке запроса, причем исходные и промежуточные данные поиска также имеют распределенный характер;
- частично распределенные — распределенные данные и локализованная обработка. Здесь промежуточные данные расположены на поисковом сервере;
- локальные системы — локализованные данные и их обработка. В данном случае и первичные и промежуточные данные располагаются на поисковом сервере.

Программы анализа статистики посещений подразделяются на свободно распространяемые и коммерческие. Задача этих программ сводится к подготовке сводного отчета, в который входит: общее количество посещений сервера за анализируемый период; число перенаправлений; среднее число запросов в день; число обслуженных хостов; количество страниц, с которых осуществлялся доступ; число некорректных записей в файле посещений; общее число байтов, переданных клиентам; среднее число байтов, переданных за сутки. Коммерческие программы обычно являются частью большого коммерческого пакета и обладают не только возможностями по группированию данных из файлов посещений и их отображению в виде графиков и гистограмм, но и некоторыми дополнительными возможностями.

3.2. Назначение и функции Web-браузера. Популярные Web-браузеры

Web-браузеры имеют наибольшее распространение и использование из всего массива программного обеспечения Web-систем, так как они предназначены для загрузки и просмотра Web-ресурсов Internet.

До недавнего времени широко использовались Web-браузеры Netscape Navigator и Internet Explorer. В последнее же время появилось множество более современных программ, одна из которых — Opera. Данные программные продукты достаточно удобны в использовании и обеспечивают просмотр практически всех видов информации, доступных в WWW.

Программа Netscape Navigator стала первым высококачественным Web-браузером, который распространялся практически бесплатно, поэтому она заняла подавляющее лидерство в данной области ПО (более 75% рынка). Это подтолкнуло корпорацию Microsoft под угрозой потери ведущего положения в области производства программного обеспечения начать усовершенствование собственного Web-браузера Internet Explorer, который в последующем был включен в новые версии Windows. Это привело к выравниванию положения этих двух разработок.

В последнее время большое количество пользователей предпочитают выбирать разработки других производителей, примером таких программных продуктов могут служить Mozilla Firefox, Opera и др., распространяемые условно бесплатно. Причинами этого процесса являются значительные проблемы с безопасностью, функциональностью и расширяемостью. Первая проблема возникла в результате большой популярности MSIE, благодаря чему многие злоумышленники пытались взломать защиту данного браузера, что им во многих случаях удавалось. Таким образом, было обнаружено значительное количество «брешей» в MSIE. Вторая проблема развилась, так как компания Microsoft мало внимания уделяла развитию своего продукта. С течением времени ее конкуренты расширили функциональные возможности своих разработок, связанные в первую очередь с возможностью работать с вкладками, блокировать рекламу и поддерживать «мышиные жесты». И наконец, третья проблема была связана с ограниченными возможностями Internet Explorer по расширению функциональных возможностей, которые ограничиваются только установкой надстроек, в отличие, например, от Mozilla Firefox, для которого существуют тысячи дополнений.

Большинство Web-браузеров имеют сходный интерфейс и аналогичные функции, заложенные еще в первых подобных разработках.

Основными элементами интерфейса пользователя являются:

- главное меню, содержащее команды, доступные пользователю;

- панель навигации, позволяющая осуществлять навигацию в Internet;
- панель адреса, предназначенная для ввода адреса Web-ресурса;
- панель состояния, предназначенная для отображения информации об элементе, на который указывает в данный момент курсор мыши.

Главным предназначением Web-браузеров является загрузка, интерпретация и отображение информации, содержащейся в публикуемых в системе WWW документах, для реализации которого и используются функциональные возможности Web-браузеров.

Рассмотрим наиболее широко используемый Web-браузер MSIE. Окно обозревателя Internet Explorer во многом похоже на обычное окно Windows, в котором можно выделить основные элементы: строка меню, панели инструментов (обычные кнопки и ссылки), адресная строка, строка состояния. В основной области окна открывается загружаемая страница. На панели Обычные кнопки расположены кнопки, с помощью которых осуществляется управление работой обозревателя и переход между Web-ресурсами. Адресная строка служит для задания адреса Web-ресурса, который должен загрузить Web-браузер. Последние 20 адресов Web-страниц, на которые заходил пользователь, сохраняются в списке, прикрепленном к адресной строке. Последние версии MSIE поддерживают многие дополнительные функции, в частности распознавание жестов, блокировку рекламных баннеров, сохранение страниц в различных графических форматах и т.д. Кроме того, основным нововведением стала поддержка вкладок.

Наиболее популярными на сегодняшний день альтернативами MSIE являются Mozilla Firefox и Opera. Первым реальным конкурентом Internet Explorer стал Mozilla Firefox, что было связано в первую очередь с более высоким уровнем безопасности, расширенными возможностями по блокировке рекламы, высокой стабильностью работы и т.д. Популярность этого Web-браузера постоянно растет еще благодаря его мультиплатформенности. Еще одной его особенностью является защита от фишинга.

Web-браузер Opera менее популярен по сравнению с двумя ранее рассмотренными программами, и связано это с тем, что первоначально он был платным. Основным его преимуществом является стабильная работа и ограниченное потребление ресурсов компьютера. В рамках Opera реализован механизм так называемых виджетов (widgets), представляющих собой небольшую программу, которая может работать даже при свернутом окне Web-браузера. Сфера применения виджетов очень широка — от конверторов валют и переводчиков до часов, игр и HTML-редакторов.

Рассмотрим Web-браузер Opera более подробно. Сторонников у этого обозревателя гораздо меньше чем у Mozilla Fierfox и Internet Explorer, но постепенно их становиться все больше. Помимо указанных положительных сторон Opera поддерживает вкладки, виджеты и протокол BigTorrent. Все эти функции сделали Opera достойным конкурентом других обозревателей Internet. Однако поддержка вкладок, блокировка всплывающих окон и другие функции не являются особенностями этого обозревателя. Основным преимуществом Opera является поддержка так называемых виджетов, которые представляют собой небольшую программу. Они могут работать и при свернутом окне Opera. Виджет можно перетаскивать в любую область экрана. Сфера применения виджетов очень широка, это могут быть конверторы валют, переводчики, часы, игры, HTML-редакторы и множество других. Виджеты могут создаваться сторонними разработчиками. На сайте Opera есть подробная инструкция по их созданию.

3.3. Редакторы для Web-дизайна. MS FrontPage

На сегодняшний день существует множество интегрированных оболочек для построения отдельных Web-страниц и целых Web-узлов, с помощью которых даже неопытный пользователь, незнакомый с языками программирования, сможет самостоятельно создать свой собственный Web-узел и опубликовать его в Internet. С точки зрения предложенной классификации данные виды программ получили название программ подготовки публикаций, также их иногда называют Web-редакторами. Примером таких программ может служить разработка Microsoft FrontPage и программы компании Macromedia HomeSite+, Dreamwear и многие другие.

Программы подготовки публикаций могут быть использованы в качестве средств и Web-дизайна, и Web-программирования. Это интегрированные среды, содержащие в общем случае редактор Web-страниц, модули управления структурой узла и инструменты публикации узла на сервере.

Работа в этих системах ориентирована на построение так называемого Web-узла, представляющего собой набор файлов в формате HTML, расположенных в определенной папке и связанных друг с другом гиперссылками. Один из файлов Web-узла назначается главным, он представляет собой домашнюю страницу и открывается в браузере пользователя при подключении к Web-узлу. Остальные Web-страницы выводятся в окно браузера по мере перехода к ним по гиперссылкам. Кроме файлов HTML в состав узла входит набор графических объектов формата GIF или JPG, предназначенных для

оформления страниц. С развитием браузеров, предлагающих самые разнообразные расширения стандарта HTML, в Web-узлах все чаще стали появляться файлы других форматов.

Рассмотрим один из распространенных на сегодняшний день редактор для Web-дизайна MS FrontPage. FrontPage — это интегрированная среда, содержащая редактор Web-страниц, модули управления структурой узла и инструменты публикации узла на сервере. Во FrontPage три отдельных модуля — редактор страниц, компоновщик узла и средств поддержки Web-сервера — объединены в одну интегрированную оболочку, обеспечивающую удобный доступ ко всем инструментам. Чтобы создать законченный Web-узел, недостаточно просто разместить в одной папке несколько HTML-файлов. Грамотно построенный узел имеет хорошо продуманную структуру. Это облегчает пользователю поиск необходимой информации. Если у вас нет большого опыта работы с Web-страницами, мастер Web-узла поможет правильно скомпоновать узел, а вам останется только наполнить страницы содержанием.

Чтобы прибегнуть к услугам мастера, необходимо выполнить следующие шаги.

1. Запустите FrontPage.

2. Выберите команду Файл → Создать → Страница или Web-узел (File → New → Web). В области задач приложения откроется окно Создание Web-страниц (New Page or Web) со списком шаблонов и мастеров, которыми можно воспользоваться для построения Web-узла.

3. В разделе Создание с помощью шаблона (New from Template) щелкните на значке Шаблоны Web-узлов (Web Site Template) и в появившемся окне диалога Шаблоны Web-узлов (Web Site Templates) щелкните на значке Мастер корпоративного Web-узла (Corporate Presence Wizard).

4. В поле раздела Options (Параметры) введите название папки, в которой будут храниться файлы узла. Щелкните на кнопке OK. В первом окне мастера щелкните на кнопке Далее (Next).

Второе окно предлагает список основных Web-страниц, которые можно включить в новый Web-узел:

- Домашняя страница (Home Page);
- Что нового (What's New);
- Товары и услуги (Products/Services);
- Оглавление (Table of Contents);
- Обратная связь (FeedBack form);
- Форма поиска (Search form).

5. Оставьте установленными все флажки и щелкните на кнопке Далее. Следующее окно мастера предлагает определить вид домаш-

ней страницы. Устанавливая и сбрасывая флажки этого окна, вы добавляете или убираете соответствующие разделы домашней страницы.

6. Установите все четыре флажка. Щелкните на кнопке Далее.

Шесть следующих окон диалога мастера настраивают вид страницы определенного типа (из тех, которые были выбраны во втором окне мастера). Последовательно изучите каждое окно и установите флажки для тех компонентов, которые необходимо включить в Web-узел.

7. Десятое окно мастера задает общее оформление всех страниц. Щелкните два раза на кнопке Далее.

8. Введите полное название компании, то же самое название, сокращенное до одного слова, и адрес компании. Щелкните на кнопке Далее.

9. В следующем окне введите телефон компании, номер факса, электронный адрес Web-мастера и адрес информационной поддержки. Щелкните два раза на кнопке Далее, а затем на кнопке Готово (Finish). Мастер сгенерирует новый Web-узел и откроет его в режиме просмотра задач со списком действий, которые необходимо выполнить для получения законченного узла. Пункты этого списка содержат операции, с помощью которых вы должны наполнить смысловым содержанием сформированные Web-страницы. В процессе разработки узла можно вручную добавлять новые задачи, связанные с той или иной Web-страницей.

FrontPage предлагает шесть режимов просмотра Web-узла, кнопки которых имеются на панели режимов:

- Страница (Page) — редактор отдельной Web-страницы. В этом режиме можно изменять содержание любой страницы узла, настраивать ее оформление и просматривать HTML-код, на основе которого генерируется страница;
- Папки (Folders) — список папок и файлов Web-узла с их подробными характеристиками, подобный списку окна Проводника Windows;
- Отчеты (Reports) — статистическая информация об узле и отдельных его компонентах;
- Переходы (Navigation) — редактор структуры Web-узла, позволяющий в графическом режиме изменять связи и перестраивать гиперссылки;
- Гиперссылки (Hyperlinks) — список Web-страниц узла и схема гиперссылок выделенной страницы;
- Задачи (Tasks) — список задач, связанных с определенными файлами, которые нужно не забыть выполнить для завершения разработки узла.

Переключать режимы просмотра можно как с помощью команд меню Вид (View), так и посредством кнопок панели режимов, расположенной в левой части окна FrontPage.

Мастер Web-узла конструирует ссылки между страницами по определенному алгоритму. Вам может не понравиться созданная им структура связей. С помощью режима просмотра Переходы можно увидеть общую схему узла, добавить или разорвать определенные ссылки, скорректировать правила размещения ссылок в панелях навигации Web-страниц.

Режим просмотра гиперссылок позволяет просматривать дерево гиперссылок и проверять работоспособность внутренних и внешних ссылок узла.

Объединение группы Web-страниц в единую структуру узла позволяет программе FrontPage не только отслеживать схему построения ссылок между страницами, но и обеспечивать однотипное оформление всех страниц узла.

Каждый Web-узел строится на основе определенной схемы, задающей однотипное оформление всем стандартным элементам страниц. При разработке узла в любой момент можно сменить схему. Для этого следует выполнить следующие шаги.

1. Выберите команду Формат → Тема (Format → Theme). Откроется окно диалога. В нем вы увидите выбранную схему, с помощью которой оформлена страница.

2. В списке схем подыщите ту расцветку, которая больше всех подходит для разрабатываемого узла. В левой нижней части окна диалога есть четыре флажка, установка которых позволяет выполнять следующие действия:

- Яркие цвета (Vivid Color) — раскрашивает выбранную схему более яркими цветами;
- Активные рисунки (Active Graphics) — усложняет рисунок ссылок, кнопок и маркеров, добавляя тени и мелкие детали;
- Фоновый рисунок (Background Picture) — размещает на заднем плане страницы фоновый рисунок;

Применить с помощью CSS (Apply Using CSS) — назначает новое оформление с помощью каскадных таблиц стилей, добавляемых в HTML-код. Каскадные таблицы со стилями можно подключать к любому HTML-файлу, формируя на основе этих стилей оформление элементов страницы. При таком подходе для изменения вида целого Web-узла достаточно лишь модифицировать таблицы стилей, которые, как правило, хранятся в отдельных файлах. Это помогает администраторам больших узлов сохранять единообразие всех документов, размещенных на Web-сервере.

В основе практически любой Web-страницы лежит текст. Его можно делить на абзацы, оформлять в виде пунктов маркированных списков или помещать в ячейки таблиц. Правильное форматирова-

ние текста во многом определяет удобство восприятия информации. Для реализации функции форматирования в FrontPage включен редактор Web-страниц, который во многом похож на MS Word.

3.4. Понятие и структура Web-приложения

Структура обычного Web-приложения включает ряд элементов, определенных типов файлов, каждый из которых несет свою функциональную нагрузку и предназначен для определенных целей. Сюда входят: HTML-страницы, отображающие данные в пользовательском браузере; файлы сервера, выполняющие различные работы на сервере и дополнительные файлы, улучшающие внешний вид данных и выполняющие ряд служебных функций. Каждый из этих типов файлов представляет собой определенный ресурс, необходимый для исполнения Web-приложения. При проектировании будущего Web-приложения каждый ресурс (файл) описывается и создается отдельно, для того чтобы позже стать доступным для компонента Web-приложения.

При создании Web-приложений особое внимание уделяется ключевым компонентам: архитектура Web-приложения; Web-клиент; Web-сервер; протоколы связи; хранилища данных; визуальные элементы.

Web-клиент — это удаленный компьютер Web-системы, использующий ресурсы другого компьютера, называемого сервером. Web-клиенты могут делиться на несколько типов по различным характеристикам: по используемому протоколу передачи данных (HTTP, WAP — Wireless Application Protocol); по типу используемого приложения (Web-браузер или клиентская программа); по толщине (тонкий, не требующий специального программного обеспечения для выполнения Web-приложения, и толстый клиент обладает дополнительными возможностями, например, возможностью сохранения данных приложений на локальном компьютере); по способу реализации (Java-клиент — приложение реализованное на языке Java, и XML-документ, разработанный на основе нового стандарта разметки текста XML (Extensible Markup Language)). На сегодняшний момент самым распространенным типом Web-клиентов являются Web-браузеры.

Web-серверы обрабатывают клиентские запросы, пересылают их дальше, формируют на основе запросов обращения к внешним источникам данных. Web-сервер работает с разными протоколами передачи данных, обеспечивая клиент-серверное взаимодействие и не предназначен для обработки пользовательских событий без формирования клиентского запроса. Всякое действие пользователя формируется на стороне клиента в виде запроса.

Протоколы связи обеспечивают взаимодействие между клиентом и сервером. Клиент посылает запрос, используя протокол, понятный серверу. На настоящий момент самыми распространенными протоколами, используемыми в Web-приложениях, являются:

- HTTP (HyperText Transfer Protocol) — применяется при взаимодействии Web-браузера с Web-сервером;
- RMI (Remote Method Invocation) — создан для организации взаимодействия между Java-объектами, находящимися на различных платформах;
- IIOP (Internet Inter-ORB Protocol) — стандартный протокол, основанный на технологии CORBA. В задачу протокола входит организация взаимодействия между приложениями, написанными на различных языках.

Хранилища данных решают задачу хранения информации во внешних, относительно приложения, источниках. Такими источниками могут быть различные базы данных, файловые системы и внешние носители информации, не содержащие программные элементы.

Визуальные элементы предназначены для отражения информации в форме, облегчающей понимание переданной информации. Чаще всего в качестве них выступают различные HTML-элементы, видимые в Web-браузере: кнопки, таблицы, изображения и т.д. При этом под визуальными элементами подразумевается не только графический интерфейс клиентской части, но и административная консоль управления приложением на сервере.

Рассмотренные компоненты включаются в структуру любого Web-приложения. Практически все Web-приложения являются распределенными системами. Распределенные системы — это системы, расположенные на различных физических узлах, будь то персональные компьютеры, большие ЭВМ или какие либо другие устройства. Обычно под распределенными приложениями понимаются программные модули, обладающие возможностями выполнения на различных операционных системах. Кроме того, что выполнение распределенного приложения осуществляется на нескольких различных компьютерах, некоторые части распределенного приложения могут выполняться в других операционных программных средах.

Основными элементами Web-приложения являются клиентская и серверная части. Клиент выполняет запрос и получает ответ, сформулированный серверной частью. Серверная часть включает в себя обработку запроса, формирование ответа и посылку ответа клиенту. Для клиента приложения формируется соответствующая графическая форма, содержащая элементы пользовательского интерфейса. Для клиента, использующего Web-браузер, существует несколько форм ответов. Самым распространенным из них является HTML-страница. Также в последнее время очень активно стали внедряться

динамический HTML (Dynamic HTML) и программы, написанные на языке Java. Последним же решением в данной области является использование XML, улучшенной версии стандартного HTML.

Выбор *архитектуры приложения* зависит от цели его создания. Одноуровневая архитектура характеризуется отсутствием сетевых сервисов, а внешние источники данных находятся практически всегда на одном компьютере или в одном сегменте сети. Двухуровневая архитектура отличается наличием Web и application сервера на одном компьютере, а базы данных на другом. Большинство Web-приложений базируется на трехуровневой архитектуре, в которой физические данные распределены на три основных звена (рис. 2.2).

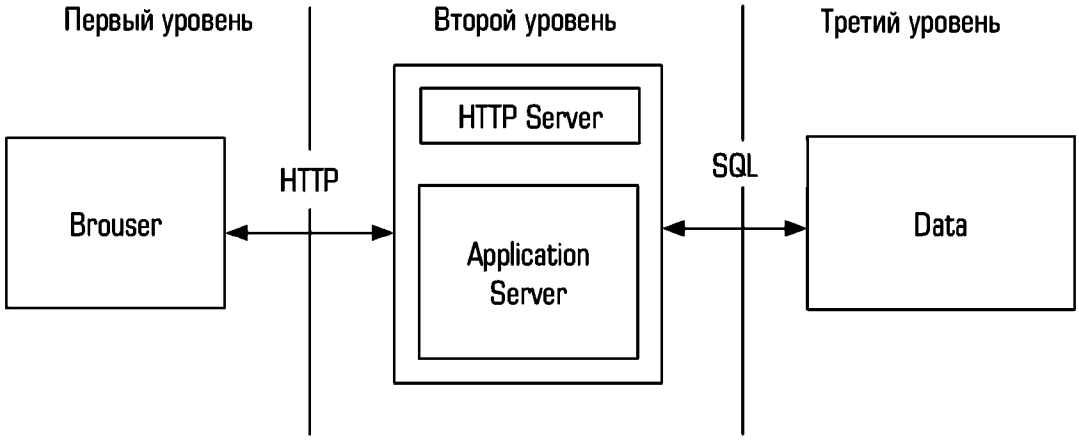

Рис. 3.1. Архитектура Web-приложения

Расширением данной архитектуры является многоуровневая топология, примером которой может служить глобальная сеть, где распределенные системы и данные находятся не только на разных машинах, но и в разных сетях.

3.5. Разработка программы HTML-редактор

Рассмотрим процесс разработки простейшей программы, позволяющей создавать и редактировать HTML-документы. Для ее создания нам потребуются следующие компоненты: TMainMenu, TOpenDialog, TSaveDialog, TPanel, TMemo, TWebBrowser. Подробно на описании свойств, методов и событий этих классов мы останавливаться не будем, так как ряд из них были изучены ранее либо будут рассмотрены в последующих главах книги.

Интерфейс приложения будет выглядеть так, как показано на рис. 3.2.

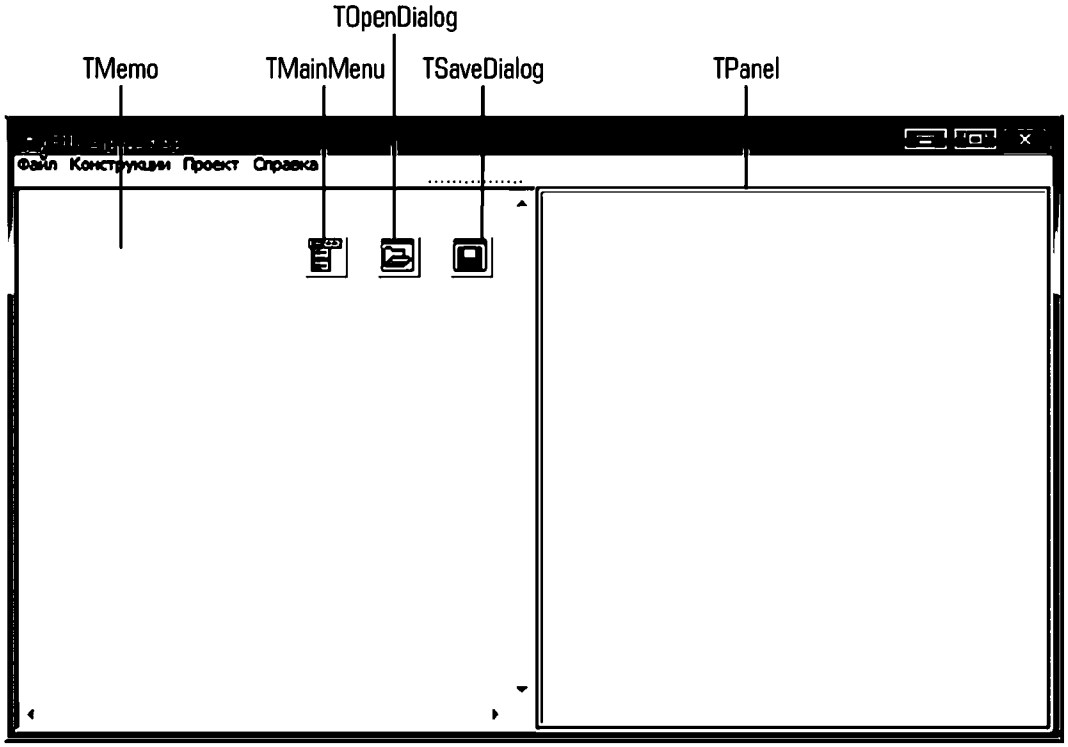

Рис. 3.2. Интерфейс программы HTML-редактор

Последовательно выполняем следующие настройки свойств компонентов. С помощью TMainMenu создаем Главное меню программы. Основные заголовки: «Файл», «Конструции», «Проект» и «Справка». Затем вложенные: в меню «Файл» — «Создать», «Открыть», «Сохранить», «Сохранить как...», «Выход»; «Конструкции» — «<HTML></ HTML>», «<HEAD></ HEAD>», «<BODY></ BODY>», «<P></ P>», «<B></ B>»; «Проект» — «Запустить»; «Справка» — «О программе». Настроим свойство Filter компонентов TOpenDialog и TSaveDialog, указав в специальном редакторе значения параметров: Filter Name — HTML-документы; Filter — *.html.

Теперь перейдем к написанию программного кода. Определим глобальные переменные:

```
F: TextFile;
FileName: string;
P: TPoint;
new: Boolean;
```

Добавим процедуру InsTag для добавления записей в TMemo:

```
procedure InsTag(Tag: string);
begin
```

```
        P := Form1.Memo1.CaretPos;
        Form1.Memo1.Lines.Insert(P.Y,Tag);
    end;
```

В обработчике события создания формы запишем:

```
    new := true;
    FileName := 'Document.html';
```

В процедуре обработки нажатия на кнопку меню «Создать» запишем:

```
    FileName := 'Document.html';
    AssignFile(F, FileName);
    Rewrite(F);
    CloseFile(F);
    new := true;
    Memo1.Lines.Clear;
    WebBrowser1.Navigate(FileName);
    Form1.Caption := 'HTML-редактор – '+ ExtractShortPathName(FileName);
```

При нажатии на пункт меню «Открыть» будет выполнять следующий код:

```
    if OpenDialog1.Execute then
    begin
        new := false;
        FileName := OpenDialog1.FileName;
        Memo1.Lines.LoadFromFile(FileName);
        Form1.Caption := 'HTML-редактор – '+
        ExtractShortPathName(FileName);
    end;
```

При нажатии на пункт меню «Сохранить»:

```
    Memo1.Lines.SaveToFile(FileName);
```

В обработчике события нажатия на пункт меню «Сохранить как...»:

```
    if SaveDialog1.Execute then
    begin
        new := false;
        FileName := SaveDialog1.FileName + '.html';
        Memo1.Lines.SaveToFile(FileName);
        Form1.Caption := 'HTML-редактор – '+
        ExtractShortPathName(FileName);
    end;
```

При нажатии на «Выход»:

```
    Form1.Close;
```

Нажатие на «О программе» будет вызывать метод показа диалогового окна:

```
ShowMessage('HTML-редактор. Версия 1.0.');
```

В обработчиках нажатия на подпункты меню «Конструкции» будем записывать вызов описанной нами ранее процедуры InsTag, в качестве параметра которой укажем соответсвующую конмтрукцию языка HTML, например, для пункта меню «HTML></HTML» укажем конструкцию:

```
InsTag('<HTML></HTML>');
```

Осталась последняя процедура, которая является обработчиком события нажатия на пункт меню «Запустить», в результате чего отображается HTML-страница сконструированная в компоненте TMemo:

```
procedure TForm1.msRunClick(Sender: TObject);
var
    Path:string;
begin
    AssignFile(F, FileName);
    Rewrite(F);
    CloseFile(F);
    Memo1.Lines.SaveToFile(FileName);
    if new then
        Path := GetCurrentDir + '\' + FileName
        else
        Path := FileName;
    WebBrowser1.Navigate(Path);
end;
```

На этом написание кода заканчивается. В результате должна получиться программа, которая позволяет конструировать HTML-страницы и отображать их в окне программы.

В написанной программе были использованы два метода, предназначенные для работы с именами файлов и каталогами, в частности ExtractShortPathName и GetCurrentDir. Приведем описание этих методов и ряда других, которыми также можно пользоваться.

Стандартные подпрограммы для работы с файлами:

- procedure ChDir(S: string),
  function SetCurrentDir(const Dir: string): Boolean — устанавливает текущий каталог в соответсвии с путем поиска, заданным параметром S (Dir);
- function GetCurrentDir: string — возвращает имя текущего каталога;

- function CreateDir(const Dir: string): Boolean,
  procedure MkDir(S: string) — создает новый каталог, путь поиска которого указан в качестве параметра. Все промежуточные каталоги (кроме самого последнего) должны существовать;
- function DeleteFile(const FileName: string): Boolean — удаляет файл;
- function DirectoryExists(Name: string): Boolean — проверяет, существует ли каталог;
- function DiskFree(Drive: Byte): Int64 — определяет число свободных байтов на диске, который задан параметром Drive. Значение 0 соответствует текущему диску, 1 — диску A, 2 — диску B, 3 — диску C и т.д.;
- function DiskSize(Drive: Byte): Int64 — возвращает объем указанного диска в байтах;
- function FileExists(const FileName: string): Boolean — проверяет, существует ли указанный файл;
- function FileGetAttr(const FileName: string): Integer,
  function FileSetAttr(const FileName: string; Attr: Integer): Integer — получает или устанавливает атрибуты указанного файла. Набор атрибутов представляет число типа Integer, разные биты которого определяют разные характеристики файла. faReadOnly — только для чтения; faHidden — скрытый; faSysFile — системный; faVolumeID — метка диска; faDirectory — каталог; faArchive — архивный; faAnyFile — произвольный;
- function FilePos (var F): Longint — возвращает номер текущей записи в файле. Нумерация начинается с нуля;
- function FileSize(var F): Integer — возвращает число записей в файле или его размер в байтах;
- function ForceDirectories(Dir: string): Boolean — создает каталог. Если указаны несуществующие промежуточные каталоги, то они также будут созданы;
- procedure GetDir(D: Byte; var S: string) — для диска, заданного параметром D (0 — A, 1 — C, 2 — D и т.д.), записывает в переменную S полный путь поиска для текущего каталога этого диска;
- procedure RmDif(S: string),
  function RemoveDir(const Dir: string): Boolean — удаляет пустой каталог, полный путь поиска для которого задан параметром S (Dir);
- procedure Rename(var F; Newname: string) — переименовывает файл, ассоциированный с переменной F при помощи процедуры AssignFile;

- function RenameFile(const OldName, NewName: string): Boolean — файл получает имя, определяемое строковым параметром NewName.

Теперь перечислим стандартные подпрограммы для работы с именами файлов:

- function ChangeFileExt(
  const FileName, Extension: string): string — изменяет расширение имени файла FileName на новое, заданное параметром Extension;
- function ExcudeTrailingBackslash(
  const S: string): string — удаляет завершающий символ \, если он присутствует в сторе;
- function ExpandFileName(
  const FileName: string): string — формирует полный путь поиска для файла FileName;
- function ExpandUNCFileName(
  const FileName: string): string — формирует полный путь поиска файла в формате UNC (\\имя-сервера\имя-ресурса);
- function ExtractFileDir(
  const FileName: string): string — выделяет имя каталога и диска в формате, пригодном для непосредственного использования в функциях CreateDir, GetCurrentDir, RemoveDir и SetCurrentDir;
- function ExtractFileDrive(
  const FileName: string): string — выделяет имя локального диска или сетевого устройства;
- function ExtractFileExt(
  const FileName: string): string — выделяет расширение имени файла;
- function ExtractFileName(
  const FileName: string): string — выделяет имя файла, отделяя его пути поиска;
- function ExtractFilePath(
  const FileName: string): string — выделяет путь поиска (с завершающем символом \) из полного имени файла;
- function ExtractRelativePath(
  const BaseName, DestName: string): string — преобразует полный путь поиска файла (параметр DestName) в строку, которая указывает относительный путь поиска каталога, заданного параметром BaseName. Элементы пути поиска могут включать промежуточные переходы на уровень вверх;
- function ExtractShortPathName(
  const FileName: string): string — преобразует путь поиска в формат коротких имен (не более 8 символов на имя файла или каталога);

- function IncludeTrailingBlackslash(
  const FileName: string): string — добавляет в конец строки символ \, если он там отсутсвует;
- function IsPathDelimiter(const S: string;
- Index: Integer): Boolean — определяет, находится ли в позиции строки, определяемой параметром Index, символ \;
- function MatchesMask(
  const FileName, Mask: string): Boolean — возвращает значение true, если файл FileName соответствует строке Mask, в которой задана маска имен файлов, содержащая подстановочные символы;
- procedure ProcessPath(const EdirText: string;
  var Drive: Char; var DirPart: string; var FilePart: string) — выделяет имя диска (один символ), путь поиска и имя файла из параметра EditText.

Вопросы для самопроверки

1. Перечислите и опишите назначение основных видов программ Web-систем.
2. Каково назначение и основные функции Web-браузеров?.
3. Каковы основные элементы пользовательского интерфейса Web-браузера Opera?
4. Дайте характеристику редакторов для Web-дизайна на примере MS FrontPage.
5. Какова архитектура Web-приложения?

Тесты

1. Какие программы предназначены для интерпретации разметки на языке HTML?
   А. Программы-клиенты.
   Б. Программы-серверы.
   В. Программы подготовки публикаций.
   Г. Информационно-поисковые системы.

2. Какие программы предназначены для приема и ответа на сообщения?
   А. Программы-клиенты.
   Б. Программы-серверы.
   В. Программы подготовки публикаций.
   Г. Информационно-поисковые системы.

3. Какие программы предназначены для создания и редактирования HTML-документов?
   А. Программы-клиенты.
   Б. Программы-серверы.
   В. Программы подготовки публикаций.
   Г. Информационно-поисковые системы.

4. Какой компонент Web-приложения обеспечивает взаимодействие между клиентом и сервером?
   А. Web-клиент.
   Б. Протоколы связи.
   В. Хранилища данных.
   Г. Визуальные элементы.

5. Какой компонент Web-приложения предназначен для отражения информации в форме, облегчающей понимание переданной информации?
   А. Web-клиент.
   Б. Протоколы связи.
   В. Хранилища данных.
   Г. визуальные элементы.

Глава 4. WEB-ПРИЛОЖЕНИЯ И МЕТОДЫ ИХ РАЗРАБОТКИ

4.1. Особенности разработки Web-приложений

В настоящее время наиболее развитой частью Internet является Всемирная паутина (World Wide Web, WWW, или Web) — система публикации ресурсов, представленных в виде гипертекстовых документов. Под публикацией обычно понимают размещение на сервере некоторого гипертекстового документа, содержащего как статические, так и динамические данные.

По структуре Web-страницы можно разделить на статические и динамические. Статические страницы содержат некоторую жестко заданную информацию, для изменения которой необходимо вносить изменения в гипертекстовый документ. Динамические страницы позволяют отображать данные (например, информацию из базы данных), которые могут изменяться без изменения самого HTML-документа.

Для создания динамических HTML-страниц обычно используют специальные серверные расширения, называемые сценариями. Типичная задача, выполняемая сценарием, — получение информации из некоторого внешнего источника, которая затем представляется в виде HTML-документа и передается серверу, а он в свою очередь отсылает ее клиенту. Кроме того, сценарии позволяют обеспечить интерактивное взаимодействие с клиентом, обрабатывая данные, передаваемые от клиента к серверу.

Реализация и поддержка Web-серверов предполагает решение трех основных задач:

1) подготовка материалов к Web-публикации, редактирование, дизайн, соблюдение единого стиля и единообразного оформления Web-страниц, поддержание связности Web-документов и т.п.;

2) обеспечение динамического представления информации на Web-странице: создание интерактивных Web-страниц, организация разных видов поиска информации, ведение статистики посещений страницы, регистрация пользователей, регламентация доступа к страницам, организация доступа к базам данных;

3) администрирование Web-сервера как компонента Web-системы.

Первая задача состоит в разработке и создании статической Web-страницы (статической части Web-документа). Для ее решения ис-

пользуются специальные средства разработки HTML-документов. Вторая задача подразумевает разработку средств, расширяющих возможности Web-сервера, и состоит в реализации возможности динамического изменения содержимого страниц в сочетании с возможностью интерактивного режима работы. Третья задача связана с созданием серверных расширений с помощью самых разных средств разработки.

В процессе создания Web-страницы принято выделять две составляющие — Web-дизайн и Web-программирование. Между ними нет четкой границы. Чаще всего под Web-дизайном понимают разработку статической части Web-страницы на языке HTML. Включение же в HTML-документ фрагментов на языке Java обычно относят к области Web-программирования. Исключительно к области Web-программирования относят разработку расширений для Web-сервера. В дальнейшем под Web-программированием будет пониматься разработка сценариев, необходимых для организации динамических документов.

Программы, обеспечивающие работу Web, используют для обмена данными протокол HTTP. Для передачи двоичных файлов по протоколу HTTP используется спецификация MIME (Multipurpose Internet Mail Extension — многоцелевые расширения почты Internet), согласно которой формат данных описывается следующим образом: <тип>/<подтип>.

Тип определяет, какого рода информация содержится в двоичном файле (текст, приложение, изображение, видеозапись и т.п.), а подтип — формат файла. Сеанс взаимодействия с HTTP-сервером в наиболее общем виде состоит из нескольких шагов: установление TCP-соединения, запрос клиента, ответ сервера, разрыв TCP-соединения.

4.2. Основные типы серверов приложений

Задачу выполнения каких-либо нестандартных действий по запросу Web-клиента Web-серверу решают специальные программы, называемые серверами приложений и используемые в качестве расширений Web-серверов. Основными из них, которые могут создаваться средствами разработки приложений, функционирующих под управлением операционной системы Windows, являются CGI-сценарии, ISAPI-расширения, ASP-страницы.

CGI-сценарии — это первый и общепринятый интерфейс для создания расширений Web-серверов. CGI-сценарий представляет собой обычное консольное приложение, обменивающееся данными с сервером через переменные окружения. Их недостатком является в первую очередь то, что они выполняются в своем адресном про-

странстве (а не в адресном пространстве Web-сервера), поэтому скорость взаимодействия с сервером довольно низка. Достоинство состоит в универсальности CGI-сценариев, так как они поддерживаются практически всеми Web-серверами, работающими на любых платформах. CGI-сценарий является основным видом серверных расширений для Web-серверов, работающих под управлением различных разновидностей операционной системы UNIX, в отличие от Windows NT, для которой эта технология является устаревшей и вытесняется другими технологиями.

Спецификация ISAPI (Internet Server Application Programming Interface — прикладной программный интерфейс для интернет-серверов) представляет собой динамическую библиотеку (DLL, Data Link Library), загружаемую как программный поток, принадлежащий Web-серверу (а не как отдельный процесс в случае с CGI-сценарием). При этом основными преимуществами ISAPI-расширений являются: наименьшая загрузка сервера за счет меньшего потребления ресурсов программным потоком; увеличение скорости обработки в связи с выполнением в адресном пространстве сервера; возможность сохранения режима загрузки в памяти и повышение тем самым скорости обработки запросов. Главный недостаток ISAPI-расширений — опасность возникновения ошибок при выполнении, приводящих к нарушению работы сервера, что связано с загрузкой в одно адресное пространство с Web-сервером.

ASP (Active Server Pages — активные серверные страницы) является наиболее современной из рассматриваемых технологий. Данная технология не предлагает новых концепций — сценарий выполняется на сервере, а клиенту отправляется формируемый им HTML-файл. Входные данные от клиентских приложений поступают с помощью методов GET и POST. Существенным отличием ASP от выше рассмотренных технологий является то, что сценарий ASP-страницы формирует не полный HTML-документ, а лишь его часть, добавляемую к исходному документу, из которого производится вызов сценария. В одном HTML-документе может содержаться несколько обращений к разным ASP-серверам. Посылаемый клиенту результирующий HTML-документ формируется на основе откликов всех ASP-сценариев.

4.3. Публикация данных на Web-сервере

При публикации информации в Internet широко используются базы данных, что значительно расширяет возможности Web-сервера и позволяет решить ряд проблем, связанных с ограничением доступа к информации и поиском данных.

Использование Web-браузера в качестве клиентской программы для базы данных имеет ряд преимуществ, главным из которых является возможность работы с базой данных, размещенной на Web-сервере любой компьютерной платформы. Это связано с тем, что стандарт языка HTML одинаково интерпретируется браузерами независимо от модели операционной системы, а также в случае внесения каких-либо изменений в базу данных нет необходимости проводить обновление программного обеспечения пользователей этой базы данных.

Модель взаимодействия с базой данных в рамках технологий доступа к базе данных, размещенной на стороне сервера, можно изобразить в виде схемы (рис. 4.1).

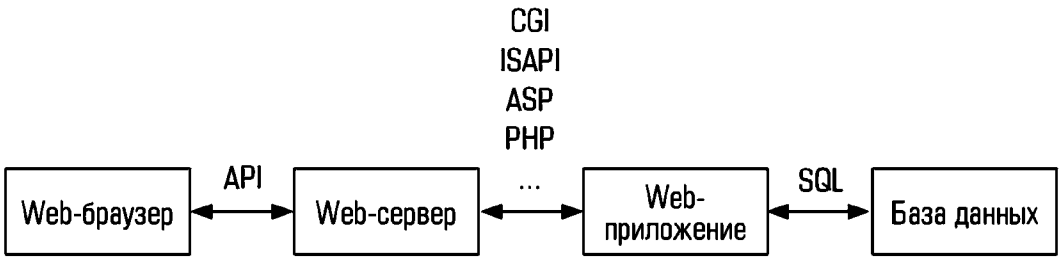

Рис. 4.1. Модель доступа к данным

При обеспечении доступа к базам данных через Web возможен ряд технологических и организационных решений, выбор которых зависит от специфики СУБД и ряда других факторов.

Для доступа к базам данных через Web наиболее часто используется один из двух подходов: однократное или регулярное преобразование содержимого базы данных в статические гипертекстовые документы; динамическое создание гипертекстовых документов на основе информации, содержащейся в базе данных, и информации, переданной клиентом Web-серверу.

В случае однократного или регулярного преобразования содержимого базы данных в статические гипертекстовые документы база данных периодически просматривается специальной программой, создающей множество связанных HTML-документов, содержащих информацию из базы данных. Полученные HTML-файлы размещаются на одном или нескольких компьютерах, выполняющих функции Web-серверов. Доступ к ним осуществляется как к статическим гипертекстовым документам. Этот способ достаточно эффективен при работе с небольшими редко обновляемыми базами данных, имеющими простую структуру, а также при невысоких требованиях к актуальности данных, предоставляемых через Web.

В случае динамического создания гипертекстовых документов на основе информации, содержащейся в базе данных, и информации, переданной клиентом Web-серверу, доступ к базе данных обеспечивается специальным приложением (CGI, ISAPI, ASP, PHP и т.п.), вызываемым Web-сервером в ответ на запрос, полученный от клиента. Приложение обрабатывает запрос, производит необходимую выборку из базы данных и на ее основе формирует выходной HTML-документ, возвращаемый клиенту. Такое решение эффективно для больших баз данных со сложной структурой. Данный вариант позволяет также обеспечить возможность изменения информации, хранящейся в базе данных, с помощью Web-интерфейсов.

Вопросы для самопроверки

1. Охарактеризуйте понятие «публикация Web-ресурсов».
2. Перечислите, в чем заключаются основные задачи реализации и поддержки Web-серверов.
3. В чем состоит процесс создания Web-страниц?
4. Опишите основные виды серверов приложений.
5. Опишите модель публикации данных на Web-сервере.

Тесты

1. Что в себя включает создание статической части Web-документа?
   А. Подготовка материалов к Web-публикации.
   Б. Редактирование и дизайн Web-документа.
   В. Организация разных видов поиска информации.
   Г. Регламентация доступа к страницам.

2. Что в себя включает создание динамической части Web-документа?
   А. Поддержание связности Web-документов.
   Б. Редактирование и дизайн Web-документа.
   В. Организация доступа к базам данных.
   Г. Регламентацию доступа к страницам.

3. Что подразумевает администрирование Web-сервера?
   А. Создание интерактивных Web-страниц.
   Б. Создание серверных расширений.
   В. Ведение статистики посещений страницы.
   Г. Организация доступа к базам данных.

4. Что представляет собой обычное консольное приложение, обменивающееся данными с сервером через переменные окружения?
   А. CGI-сценарии.
   Б. Спецификация ISAPI.
   В. ASP.
   Г. PHP.

5. Что представляет собой динамическую библиотеку, загружаемую как программный поток, принадлежащий Web-серверу?
   А. CGI-сценарии.
   Б. Спецификация ISAPI.
   В. ASP.
   Г. PHP.

Глава 5. СРЕДСТВА РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ

5.1. Средства разработки статической части Web-документа

Популярность World Wide Web и неотъемлемой ее части, HTML, стала причиной повышенного внимания к системам гипертекстовой разметки документов. Хотя понятие гипертекста было введено В. Бушем еще в 1945 г. и начиная с 1960-х гг. стали появляться первые приложения, использующие гипертекстовые данные, всплеск активности вокруг этой технологии начался лишь тогда, когда возникла реальная необходимость в механизме объединения множества информационных ресурсов, обеспечения возможности создания, просмотра нелинейного текста. И примером реализации этого механизма послужила паутина WWW.

Язык HTML является одним из основных средств разработки статической части Web-документа. По своей структуре HTML-документ представляет собой обычный текстовый файл, содержащий текст документа и специальные языковые конструкции — теги, используемые для разметки документа и управления его отображением. При создании HTML-документа можно использовать любой простейший текстовый редактор. Если в состав HTML-документа входят графические изображения, то они хранятся в отдельных файлах. При этом в тексте HTML-документа указывается ссылка на соответствующий файл. Для хранения изображений в основном используются файлы форматов JPEG, GIF, PNG.

Таким образом, язык разметки документов — это набор специальных инструкций, называемых тегами, предназначенных для формирования в документах какой-либо структуры и определения отношений между различными элементами этой структуры. Теги языка, или, как их иногда называют, управляющие дескрипторы, в таких документах каким-то образом кодируются, выделяются относительно основного содержимого документа и служат в качестве инструкций для программы, производящей показ содержимого документа на стороне клиента.

Использование гипертекстовой разбивки текстового документа в современных информационных системах во многом связано с тем, что гипертекст позволяет создавать механизм нелинейного просмотра информации. В таких системах данные представляются не в виде непрерывного потока текстовой информации, а набором взаимос-

вязанных компонентов, переход по которым осуществляется при помощи гиперссылок.

Самый популярный на сегодняшний день язык гипертекстовой разметки HTML был создан специально для организации информации, распределенной в сети Internet, и является одной из ключевых составляющих технологии WWW. С использованием гипертекстовой модели документа способ представления разнообразных информационных ресурсов в сети стал более упорядочен, а пользователи получили удобный механизм поиска и просмотра нужной информации.

HTML — это упрощенная версия стандартного общего языка разметки SGML (Standard Generalized Markup Language), который был утвержден ISO в качестве стандарта еще в 1980-х гг. Этот язык предназначен для создания других языков разметки, он определяет допустимый набор тегов, их атрибуты и внутреннюю структуру документа. Контроль за правильностью использования дескрипторов осуществляется при помощи специального набора правил, называемых DTD-описаниями, которые используются программой клиента при разборе документа. Для каждого класса документов определяется свой набор правил, описывающих грамматику соответствующего языка разметки. С помощью SGML можно описывать структурированные данные, организовывать информацию, содержащуюся в документах, представлять эту информацию в некотором стандартизованном формате. Но ввиду некоторой своей сложности SGML использовался в основном для описания синтаксиса других языков (наиболее известным из которых является HTML), и немногие приложения работали с SGML-документами напрямую.

Гораздо более простой и удобный, чем SGML, язык HTML позволяет определять оформление элементов документа и имеет некий ограниченный набор инструкций — тегов, при помощи которых осуществляется процесс разметки. Инструкции HTML в первую очередь предназначены для управления процессом вывода содержимого документа на экране программы-клиента и определяют этим самым способ представления документа, но не его структуру. В качестве элемента гипертекстовой базы данных, описываемой HTML, используется текстовый файл, который может легко передаваться по сети с использованием протокола HTTP. Эта особенность, а также то, что HTML является открытым стандартом и огромное количество пользователей имеет возможность применять возможности этого языка для оформления своих документов, безусловно, повлияли на рост популярности HTML и сделали его сегодня главным механизмом представления информации в Web.

Однако современные приложения нуждаются не только в языке представления данных на экране клиента, но и в механизме, позволяющем определять структуру документа, описывать содержащиеся

в нем элементы. HTML обладает несложным набором команд и вполне успешно справляется с задачей описания текстовой информации и отображением ее на экране программы просмотра — браузера. Однако сами отображаемые данные никак не связаны с теми тегами, которые используются для форматирования, поэтому у программ-анализаторов нет возможности использовать теги HTML для поиска нужных нам фрагментов документа.

Другим существенным недостатком HTML можно назвать ограниченность набора его тегов. DTD-правила для HTML определяют фиксированный набор дескрипторов, и поэтому у разработчика нет возможности вводить собственные специальные теги. Хотя время от времени появляются новые расширения языка, но долгий путь их стандартизации, сопровождаемый постоянными разногласиями между основными производителями браузеров, делают практически невозможной быструю адаптацию языка, его использование для отображения специализированной информации (например, мультимедийной, математических, химических формул и т.д.).

Подводя итог всему сказанному, можно утверждать, что HTML уже сегодня не удовлетворяет в полной мере требованиям, предъявляемым современными разработчиками к языкам подобного рода. И ему на смену был предложен новый язык гипертекстовой разметки — мощный, гибкий, и одновременно удобный язык XML.

XML (Extensible Markup Language) — это язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Этот язык используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. т.е. сам по себе XML не содержит никаких тегов, предназначенных для разметки, он просто определяет порядок их создания.

Процесс создания XML-документа очень прост и требует лишь базовых знаний HTML и понимания тех задач, которые необходимо выполнить, используя XML в качестве языка разметки. Таким образом, у разработчиков появляется уникальная возможность определять собственные команды, позволяющие им наиболее эффективно определять данные, содержащиеся в документе. Автор документа создает его структуру, строит необходимые связи между элементами, используя те команды, которые удовлетворяют его требованиям, и добивается такого типа разметки, который необходим ему для выполнения операций просмотра, поиска, анализа документа.

Еще одним из очевидных достоинств XML является возможность использования его в качестве универсального языка запросов к хранилищам информации. Кроме того, XML-документы могут выступать в качестве уникального способа хранения данных, который включает в себя одновременно средства для разбора информации и

представления ее на стороне клиента. В этой области одним из перспективных направлений является интеграция Java и XML-технологий, позволяющая использовать мощь обеих технологий при построении машинно-независимых приложений, использующих, кроме того, универсальный формат данных при обмене информацией.

XML позволяет также осуществлять контроль за корректностью данных, хранящихся в документах, производить проверки иерархических соотношений внутри документа и устанавливать единый стандарт на структуру документов, содержимым которых могут быть самые различные данные. Это означает, что его можно использовать при построении сложных информационных систем, в которых очень важным является вопрос обмена информацией между различными приложениями, работающими в одной системе.

Также одним из достоинств XML является то, что программы — обработчики XML-документов не сложны, и уже сегодня появились и свободно распространяются всевозможные программные продукты, предназначенные для работы с XML-документами. XML поддерживается сегодня в Microsoft Internet Explorer, Netscape Communicator, СУБД Oracle, DB-2, в приложениях MS Office. Все это дает основания предполагать, что скорее всего в ближайшем будущем XML станет основным языком обмена информации для информационных систем, заменив собой HTML. На основе XML уже сегодня созданы такие известные специализированные языки разметки, как SMIL, CDF, MathML, XSL, и список рабочих проектов новых языков, находящихся на рассмотрении W3C, постоянно пополняется.

5.2. Язык гипертекстовой разметки документов HTML

Теги HTML-документов в большинстве своем просты для понимания и использования, ибо они образованы с помощью общеупотребительных слов английского языка, понятных сокращений и обозначений.

Теги обычно используются парами, состоящими из открывающегося и закрывающего тегов. Все теги начинаются с символа "<" и оканчиваются символом ">". Открывающий тег имеет следующий формат: <имя тега [атрибуты]>, закрывающий — </имя тега>.

Атрибуты тега следуют за именем и отделяются друг от друга одним или несколькими знаками табуляции, пробелами или символами возврата к началу строки. Порядок записи атрибутов в теге значения не имеет. Значение атрибута, если таковое имеется, следует за знаком равенства, стоящим после имени атрибута. Если значение атрибута — одно слово или число, то его можно просто указать после знака равенства, не выделяя дополнительно. Все остальные значе-

ния необходимо заключать в одинарные или двойные кавычки, особенно если они содержат несколько разделенных пробелами слов. Длина значения атрибута ограничена 1024 символами. Регистр символов в именах тегов и атрибутов не учитывается, чего нельзя сказать о значениях атрибутов. Например, особенно важно использовать нужный регистр при вводе URL других документов в качестве значения атрибута HREF. Конечные теги никогда не содержат атрибутов.

При использовании вложенных тегов в документе следует соблюдать особую аккуратность. Вложенные теги нужно закрывать, начиная с самого последнего и двигаясь к первому. Некоторые HTML-теги не имеют конечного компонента, поскольку они являются автономными элементами.

В некоторых случаях конечные теги в документе можно опускать. Большинство браузеров реализованы так, что при обработке текста документа начальный тег воспринимается как конечный тег предыдущего.

Любой документ в формате HTML начинается с открывающего тега <HTML> и заканчивается тегом </HTML>. Он состоит из двух частей: раздел заголовка (HEAD); тело (BODY). В общем виде HTML-документ имеет следующую структуру:

```
<HTML>
    <HEAD>
        [заголовок документа]
    </HEAD>
    <BODY>
        [тело документа]
    </BODY>
</HTML>
```

Раздел заголовка содержит различного рода служебную информацию. Наиболее часто в заголовке применяются следующие теги:

- <TITLE> (</TITLE>) — заголовок HTML-документа, который отображается в строке заголовка окна браузера;
- <BASE> — базовый адрес, используемый при обработке относительных URL-адресов;
- <LINK> — тег, используемый для связи с другими HTML-документами;
- <META> — дополнительная информация о HTML-документе.

Тело документа включает всю информацию, которая отображается в окне браузера. В теле документа специальные теги обеспечивают форматирование текста документа и включают в него изображения, таблицы и формы.

HTML-документ может содержать шесть уровней заголовков. Для задания заголовка используются пары тегов: <H1> (</H1>), <H2> (</H2>), ..., <H6> (</H6>). В открывающем теге можно указать дополнительный атрибут ALIGN, определяющий способ выравнивания текста заголовка. Данному атрибуту можно задать одно из трех значений: LEFT, RIGHT, CENTER.

```
<H1 ALIGN=CENTER> текст заголовка </H1>
```

Текст, относящийся к одному абзацу, заключается между тегами <P> и </P>. Каждый абзац отделяется от предыдущего увеличенным межстрочным интервалом. Для абзацев также можно задать способ выравнивания текста с помощью атрибута ALIGN.

Если требуется начать текст с новой строки в пределах одного абзаца, то используется тег
.

HTML-документ может содержать как маркированные, так и нумерованные списки. Для создания маркированных списков используются теги <UL> и </UL>, нумерованных — <OL> и </OL>. В обоих случаях каждый элемент списка помещается между тегами <LI> и </LI>. Допускается создание вложенных списков.

Язык HTML позволяет выделять отдельные слова и даже символы документа. Для этого используют теги:

- <B> (</B>) — выделение полужирным шрифтом;
- <I> (</I>) — выделение курсивом;
- <U> (</U>) — выделение подчеркиванием.

Перечисленные теги задают способ явного выделения фрагмента текста. Наряду с ними имеются теги, которые лишь указывают, что текст должен быть выделен, не указывая способа выделения. В этом случае выбор способа выделения определяется браузером:

- <EM> (</EM>) — выделенному тексту должно уделяться внимание;
- <STRONG> (</STRONG>) — выделенному тексту должно уделяться особое внимание;
- <KBD> (</KBD>) — обозначение клавиши клавиатуры;
- <VAR> (</VAR>) — обозначение переменной;
- <CITE> (</CITE>) — цитата.

Связь между различными HTML-документами обеспечивается посредством гиперссылок. Гиперссылка представляет собой фрагмент HTML-документа (текст или изображение), щелчок на котором приводит к загрузке другого документа.

Для создания гиперссылки используется пара тегов <A> (</A>). Заключенный между ними фрагмент HTML-документа при просмотре будет отображаться как гиперссылка. Тег <A> обязательно должен указываться с атрибутом HREF. С его помощью задается ссылка

на документ, к которому должен быть произведен переход при щелчке на гиперссылке.

<A HREF=URL_ресурса> фрагмент документа </A>

Здесь URL-адрес может быть абсолютным и относительным. Абсолютный URL-адрес содержит полную информацию о местоположении ресурса и протоколе обращения к ресурсу. Относительный URL-адрес задает расположение ресурса относительно местоположения текущего HTML-документа.

Тег <A> имеет еще одно назначение: помимо создания гиперссылок он позволяет устанавливать маркеры для организации переходов по гиперссылкам в пределах одного HTML-документа. Для создания маркера тег <A> используется совместно с атрибутом NAME (имя маркера должно задаваться латинскими буквами и может содержать цифры, кроме первого символа):

<A NAME="имя_маркера" > текст </A>

В этом случае текст, заключенный между тегами, при отображении никак не выделяется, но к помеченному таким образом фрагменту HTML-документа можно перейти с помощью гиперссылки следующего вида:

<A HREF="имя_маркера" > текст </A>

Гиперссылки такого вида часто используются в документах большого объема.

По мере роста системы WWW стало ясно, что средств, которые заложены в HTML, не достаточно для качественного отображения различного типа документов. Недостатком HTML было отсутствие в его составе средств отображения таблиц. Для этой цели обычно использовался предформатированный текст (тег <PRE>), в котором таблица обрисовывалась символами ASCII. Но такая форма представления таблиц была недостаточно высокого качества и выделялась из общего стиля документа.

Для описания таблиц был введен тег <TABLE>. Формирование строк и столбцов таблицы осуществляется соответственно с помощью тегов <TR> и <TD>.

При задании заголовков для столбцов и строк таблицы используются теги заголовка <TH></TH> (Table Header, заголовок таблицы). Эти теги аналогичны <TD></TD>. Отличие состоит в том, что текст, заключенный между тегами <TH></TH>, автоматически записывается жирным шрифтом и по умолчанию располагается посередине ячейки. Центрирование можно отменить и выровнять текст по левому или правому краю. Если воспользоваться <TD></TD> с тегом <B> и атрибутом <ALIGN=CENTER>, текст тоже будет выглядеть как заголовок. Однако следует иметь в виду, что не все браузеры под-

держивают жирный шрифт в таблицах, поэтому лучше задавать заголовки таблиц с помощью <TH>.

<CAPTION> позволяет создавать заголовки таблицы. По умолчанию заголовки центрируются и размещаются либо над (<CAPTION ALIGN=TOP>), либо под таблицей (<CAPTION ALIGN=BOTTOM>). Заголовок может состоять из любого текста и изображений. Текст будет разбит на строки, соответствующие ширине таблицы. Иногда тег <CAPTION> используется для подписи под рисунком. Для этого достаточно описать таблицу без границ.

Обычно любой текст в таблице, не помещающийся в одну строку ячейки, переходит на следующую строку. Однако при использовании атрибута NOWARP с тегами <TH> или <TD> длина ячейки расширяется настолько, чтобы заключенный в ней текст поместился в одну строку.

Теги <TD> и <TH> модифицируются с помощью атрибута COLSPEN (Column Span, соединение столбцов). Для того чтобы сделать какую-нибудь ячейку шире, чем верхняя или нижняя, можно воспользоваться атрибутом COLSPEN, таким образом она будет растянута над любым количеством обычных ячеек. Атрибут ROWSPAN, используемый в тегах <TD> и <TH>, аналогичен атрибуту COLSPEN, только он задает число строк, на которые растягивается ячейка. Если вы указали в атрибуте ROWSPAN число, большее единицы, то соответствующее количество строк должно находиться под растягиваемой ячейкой.

Атрибут WIDTH применяется в двух случаях. Можно поместить его в тег <TABLE> для задания ширины всей таблицы, а можно использовать в тегах <TR> или <TH> для задания ширины ячейки или группы ячеек. Ширину можно указывать в пикселях или процентах. Например, если вы задали в теге <TABLE> WIDTH=250, вы получите таблицу шириной 250 пикселей независимо от размера страницы на мониторе. При задании WIDTH=50% в теге <TABLE> таблица будет занимать половину ширины страницы при любом размере изображения на экране.

Атрибут UNIT тега <TABLE> определяет единицы измерения, используемые при указании размеров как всей таблицы, так и ее отдельных столбцов. Атрибут UNIT может принимать следующие три значения:

- UNIT=EN — это значение присваивается по умолчанию и задает единицу измерения, равную en-пробелу. En-пробел — это типографская единица измерения, равная ширине буквы <n>. Реальный размер пробела зависит от выбранного шрифта: для крупного шрифта en-пробел больше, чем для мелкого. Обычно en-пробел равен половине размера шрифта. Например, при использовании 12-пунктового шрифта ширина en-пробела будет

6 пунктов, для 8-пунктового шрифта en-пробел занимает 4 пункта;

- UNIT=RELATIVE используется для задания относительной ширины столбцов в процентах от общей ширины таблицы. Этот способ следует по возможности применять вместо указания ширины в процентах. UNIT=RELATIVE выполняет ту же функцию, но поддерживается бóльшим количеством браузеров. При задании относительных (RELATIVE) единиц вводимые числа воспринимаются как ширина столбцов в процентах;
- UNIT=PIXELS — это значение применяется, когда вам нужно точно знать ширину столбца на экране. В этом случае лучше всего задать ее в пикселях. Например, тег <TABLE UNIT=PIXELS WIDTH=340> сформирует таблицу шириной 340 пикселей.

Атрибут COLSPEC, используемый с атрибутом UNIT, определяет, сколько места занимает каждый столбец таблицы и как в нем выравниваются данные. Применяется только в теге <TABLE>. COLSPEC перечисляет все столбцы и для каждого из них задает выравнивание и размер. Для столбца (или ячейки) существует пять способов выравнивания: L — по левому краю, C — по центру, R — по правому краю, J — по правому и левому краю и D — по десятичной запятой. Если у вас пять столбцов, вы можете определить ширину и выравнивание каждого из них следующим образом:

```
<TABLE UNIT=PIXELS COLSPEC="L10 C15 R20 J25 D30">
```

В данном случае была описана таблица, в которой первый столбец имеет ширину 10 пикселей и его содержимое выравнивается по левому краю, второй столбец шириной 15 пикселей с выравниванием по центру, третий — шириной 20 пикселей выровнен по правому краю, четвертый — шириной 25 пикселей выровнен с двух сторон, а пятый — шириной 30 пикселей выравнивается по десятичным запятым.

В теге <TABLE> часто определяют, как будут выглядеть рамки, т.е. линии, окружающие ячейки таблицы и саму таблицу. Если вы не зададите рамку, то получите таблицу без линий, но пустое пространство для них будет отведено. Того же результата можно добиться, задав <TABLE BORDER=0>. При создании вложенных таблиц приходится делать границы различной толщины для разных таблиц, чтобы их легче было различать.

Атрибут CELLSPASING определяет в пикселях ширину промежутков между ячейками. Если этот атрибут не задан, по умолчанию задается величина, равная двум пикселям. Атрибут CELLSPASING можно использовать, чтобы поместить текст и графику непосредственно там, где вам нужно. Если вы хотите оставить пустое место, можно вписать в ячейку пробел.

Для интерактивного взаимодействия с пользователем, работающим на клиентской машине, и Web-приложениями, выполняющимися на стороне сервера, предназначены формы. Для создания формы используется пара тегов <FORM> (</FORM>). Между ними располагаются строки, описывающие различные элементы управления: кнопки, поля ввода, флажки и т.п. Совместно с тегом <FORM> практически всегда используются атрибуты:

- ACTION — предназначен для указания URL-адреса программы (сценария), которая будет выполнять обработку данных, введенных пользователем;
- METOD — определяет метод, с помощью которого данные, введенные пользователем, будут передаваться на сервер. Данный атрибут может принимать одно из двух значений: GET или POST.

Основные элементы управления создаются с помощью тега <INPUT>, который используется без закрывающего тега. Тип элемента управления задается с помощью атрибута TYPE, тег <INPUT> содержит ряд других атрибутов, определяющих параметры элемента управления.

Для создания полей ввода атрибуту TYPE следует присвоить значение "TEXT" (с кавычками). Параметры поля ввода задаются следующими атрибутами тега <INPUT>:

- NAME — идентификатор элемента управления;
- VALUE — начальное значение, отображаемое в поле сразу после загрузки документа;
- SIZE — максимальное количество отображаемых символов;
- MAXLENGTH — максимальное количество символов, которые могут быть введены с помощью поля.

Существует разновидность полей, предназначенных для ввода пароля. Для создания такого поля ввода атрибуту TYPE следует задать значение "PASSWORD". Все символы, вводимые в этом поле, отображаются на экране в виде звездочек (*).

Для создания флажка атрибуту TYPE тега <INPUT> задается значение "CHECKBOX". Параметры флажка определяются следующими атрибутами:

- NAME — идентификатор элемента управления;
- VALUE — атрибут, определяющий значение, которое передается на сервер в случае, если флажок установлен;
- CHICKED — атрибут, указывающий, что после загрузки документа флажок должен быть установлен (значение данному атрибуту не задается).

Переключатель представляет собой элемент управления, подобный флажку. Однако в отличие от флажков в установленном состоянии может находиться только один из переключателей, входящих

в группу. Для создания переключателей атрибуту TYPE задается значение "RADIO". Параметры переключателей определяются следующими атрибутами:

- NAME — идентификатор переключателя (должен быть одинаковым для всех переключателей, входящих в одну группу);
- VALUE — значение, передаваемое серверу при установленном переключателе;
- CHICKED — атрибут, указывающий, какой из переключателей должен быть установлен в группе при загрузке документа.

Различают два вида кнопок:

- SUBMIT — производит передачу введенных пользователем данных на сервер;
- RESET — возвращает все элементы управления в исходные состояния.

Для создания кнопки атрибуту TYPE присваивается одно из двух указанных значений, надпись на кнопке задается с помощью атрибута VALUE.

5.3. Общие сведения о языке гипертекстовой разметки XML

Дальнейшим развитием языков разметки Web-документов является расширяемый язык разметки XML. С помощью этого языка разработчики получили возможность создавать собственные языки разметки. Для этого достаточно в терминах XML описать предполагаемые нововведения, и если описание произведено корректно, браузер должен понять такой документ. Так с помощью XML был расширен язык HTML до новой версии языка — XHTML.

Знакомство с HTML облегчает изучение XML. Хотя XML сильно отличается по своим возможностям и предназначению от языка гипертекстовой разметки, оба эти языка являются подмножествами SGML и, зачит, наследуют его базовые принципы.

Простейший XML- документ может выглядеть так:

```
<?xml version=»1.0»?>
<list_of_items>
<item id=»1»><first/>Первый</item>
<item id=»2»>Второй <sub_item>подпункт 1
</sub_item></item>
<item id=»3»>Третий</item>
<item id=»4»><last/>Последний</item>
</list_of_items>
```

Этот документ очень похож на обычную HTML-страницу. Также, как и в HTML, инструкции, заключенные в угловые скобки, назы-

ваются тегами и служат для разметки основного текста документа. В XML существуют открывающие, закрывающие и пустые теги.

Тело документа XML состоит из элементов разметки (markup) и непосредственно содержимого документа — данных (content). XML-теги предназначены для определения элементов документа, их атрибутов и других конструкций языка.

Любой XML-документ должен всегда начинаться с инструкции <?xml?>, внутри которой также можно задавать номер версии языка, номер кодовой страницы и другие параметры, необходимые программе-анализатору в процессе разбора документа.

В общем случае XML-документы должны удовлетворять следующим требованиям:

- в заголовке документа помещается объявление XML, в котором указывается язык разметки документа, номер его версии и дополнительная информация;
- каждый открывающий тег, определяющий некоторую область данных в документе, обязательно должен иметь закрывающий тег, т.е. в отличие от HTML, нельзя опускать закрывающие теги;
- в XML учитывается регистр символов;
- все значения атрибутов, используемых в определении тегов, должны быть заключены в кавычки;
- вложенность тегов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тегов;
- вся информация, располагающаяся между начальным и конечным тегами, рассматривается в XML как данные, и поэтому учитываются все символы форматирования (т.е. пробелы, переводы строк, табуляции не игнорируются, как в HTML).

Если XML-документ не нарушает приведенные правила, то он называется формально-правильным, и все анализаторы, предназначенные для разбора XML-документов, смогут работать с ним корректно. Однако кроме проверки на формальное соответствие грамматике языка в документе могут присутствовать средства контроля над содержанием документа, за соблюдением правил, определяющих необходимые соотношения между элементами и формирующих структуру документа.

Для того чтобы обеспечить проверку корректности XML-документов, необходимо использовать анализаторы, производящие такую проверку и называемые верифицирующими. На сегодняшний день существуют два способа контроля правильности XML-документа: DTD-определения (Document Type Definition) и схемы данных (Semantic Schema).

Для того чтобы использовать данные, определяемые элементами XML, например отображать их на экране пользователя, необходимо

написать программу-анализатор, которая бы выполняла эти действия. Уже сегодня таких программ появилось достаточное количество, и у разработчиков существует возможность выбора наиболее подходящей из них для решения конкретных проблем.

Как уже отмечалось ранее, в общем случае программы-анализаторы можно разделить на две группы: верифицирующие (т.е. использующие DTD-описания для определения корректности документа) и неверифицирующие. При создании своего языка и описании его грамматики на основе DTD для анализа документов, написанных на этом языке, безусловно, потребуется программа, проверяющая корректность составления документа. Но так как использование DTD в XML не является обязательным, то любой правильно оформленный документ может быть распознан и разобран программой, предназначенной для анализа XML-документов. В любом случае, используя универсальные XML-анализаторы, можно быть уверенным в том, что если заданные в документе конструкции языка являются синтаксически правильными, то программа-анализатор сможет правильно извлечь определяемые ими элементы документа и передать их прикладной программе, выполняющей необходимые действия по отображению. То есть после разбора документа в большинстве случаев предоставляется объектная модель, отображающая содержимое документа, и средства, необходимые для работы с ней (прохода по дереву элементов). При этом в некоторых анализаторах способ представления структуры документа основывается на спецификации DOM. Поэтому появляется также возможность использовать строгую иерархическую модель DOM для построения собственных документов.

Если на компьютере установлен браузер Internet Explorer, то можно использовать встроенный в этот браузер XML-анализатор msxml в своих сценариях, написанных на Java Script или VBScript. В настоящий момент существуют две его реализации, одна предназначена для использования в виде написанного на C++ ActiveX-объекта (реализация на базе COM-технологии), другая, написанная на Java, не зависит от платформы. Обе они имеют поддержку иностранных языков, т.е. в составе Internet Explorer C++-анализатор работает со всеми языками, «понимаемыми» браузерами, а анализатор на Java — с теми языками, с которыми может работать виртуальная Java-машина.

Объектная модель XML-анализатора Microsoft может быть представлена в виде следующего набора внутренних объектов: XML Document, XML Element и Element Collection. Объект XML Document содержит свойства и методы, необходимые для работы с XML-документом в целом. XML Element отвечает за работу с каждым из элементов XML-документа. Element Collection представляет собой набор элементов, доступ к которым возможен при помощи имени или порядкового номера.

5.4. Язык DHTML (Dynamic HTML)

DHTML представляет собой программный интерфейс, предназначенный для управления элементами HTML-документа. Используя его, можно интерактивно изменять практически любые свойства как самой страницы, так и ее элементов в ответ на действия пользователя. Для изучения того, каким образом можно обратиться к тому или иному элементу страницы, изменить его свойства или выполнить какие-либо действия, необходимо познакомиться с объектной моделью документа.

На рис. 5.1 схематически показан набор объектов с соблюдением их иерархии в объектной модели. Рассмотрим эти объекты:

- window (окно) — предназначен для работы с окнами браузера, дает возможность открывать новые окна с установленными параметрами;
- screen (экран) — позволяет определить такие параметры монитора клиента, как разрешение и глубина цвета;
- frames (фреймы) — позволяет сценариям работать с фреймовыми страницами;
- location (расположение) — предназначен для задания адресов страниц (с помощью сценария посетителя можно перенаправлять на другие страницы);
- navigator (навигатор) — позволяет определять различные параметры компьютера клиента, такие как тип браузера, его версия, тип операционной системы;
- history (история) — позволяет выполнять навигацию в браузере (отправлять посетителя вперед или назад на определенное число просмотренных страниц);
- document (документ) — служит для обращения к элементам страницы.

Чтобы обратиться к элементу страницы, перед его именем необходимо указать объект document. Объекты при записи должны разделяться точкой:

```
document.имя_элемента
```

При обращении к свойству элемента указывается следующее:

```
document.имя_элемента.свойство
```

Для задания свойству элемента значения применяется простое присваивание:

```
document.имя_элемента.свойство=значение
```

Свойство позволяет узнать и задать значение соответствующего параметра того или иного элемента. Кроме свойств существуют еще методы объектов. Метод дает возможность выполнить с элементом определенное действие. Форма записи для определения значения

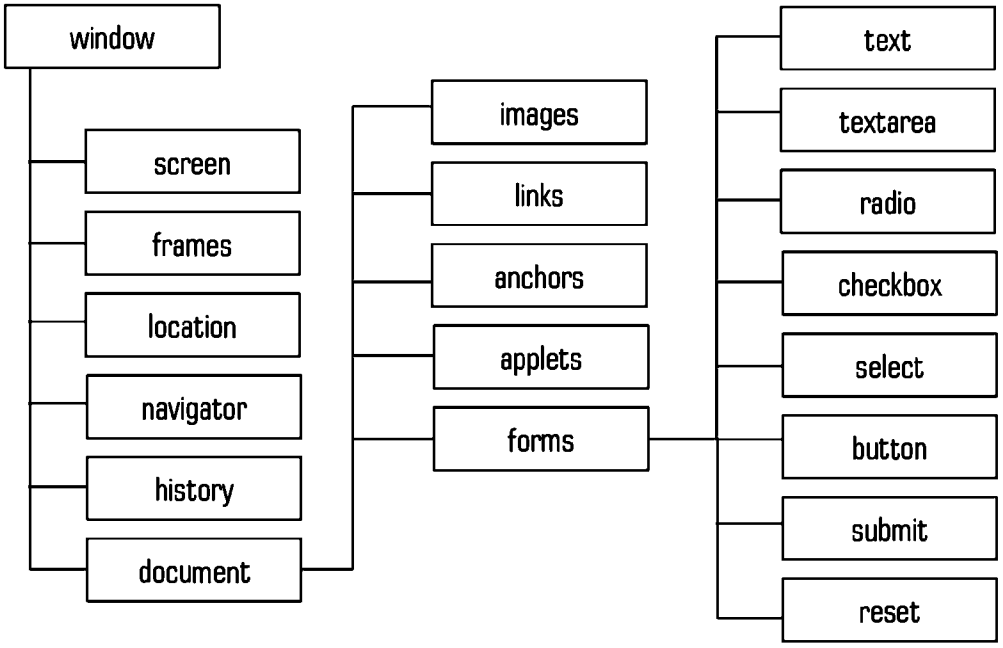

Рис. 5.1. Объектная модель документа

свойства или выполнения метода точно такая же, как и при обращении к самому объекту, — вначале объект, потом точка, затем свойство или метод:

```
объект.свойство
объект.метод
```

Рассмотрим свойства объекта window. Они позволяют определять размеры окон и текст в строке состояния, а также обращаться к другим объектам. Перечислим свойства объекта и возвращаемые ими значения:

- closed — возвращает значение true, если текущее окно закрыто (может быть использовано при работе с несколькими окнами);
- defaultStatus — сообщение, которое отображается в строке состояния окна по умолчанию;
- document — ссылка на документ, загруженный в текущее окно;
- frames — ссылка на фрейм;
- history — ссылка на объект папки History (Журнал) браузера;
- innerHeight — высота клиентской области окна (без рамок, меню и панелей инструментов) в пикселях, поддерживается только браузерами Netscape начиная с версии 4.0;
- innerWidth — ширина клиентской области окна в пикселях, поддерживается только браузерами Netscape начиная с версии 4.0;
- location — ссылка на объект location документа, загруженного в текущее окно;

- name — имя окна или фрейма;
- navigator — ссылка на объект браузера;
- outerHeight — полная высота окна (с рамками, меню и панелями инструментов) в пикселях, поддерживается только браузерами Netscape начиная с версии 4.0;
- outerWidth — полная ширина окна в пикселях, поддерживается только браузерами Netscape начиная с версии 4.0;
- pageXOffset — расстояние по горизонтали между текущей позицией окна и левой границей документа (при прокручивании содержимого окна вправо значение этого свойства увеличивается, а при прокручивании влево — уменьшается), поддерживается только браузерами Netscape начиная с версии 4.0;
- pageYOffset — расстояние по вертикали между текущей позицией окна и левой границей документа (при прокручивании содержимого окна вниз значение этого свойства увеличивается, а при прокручивании вверх — уменьшается), поддерживается только браузерами Netscape начиная с версии 4.0;
- parent — ссылка на родительское окно, если текущий объект window представляет собой фрейм (в противном случае возвращается ссылка на само это окно);
- screen — ссылка на объект screen;
- screenLeft — горизонтальная координата левого верхнего угла окна, поддерживается только браузерами IE начиная с версии 5.0;
- screenTop — вертикальная координата левого верхнего угла окна, поддерживается только браузерами IE начиная с версии 5.0;
- screenX — горизонтальная координата левого верхнего угла окна, поддерживается только браузерами Netscape начиная с версии 4.0;
- screenY — вертикальная координата левого верхнего угла окна, поддерживается только браузерами Netscape начиная с версии 4.0;
- scrollbars — ссылка на полосу прокрутки окна;
- self — ссылка на объект window текущего окна;
- status — текст, отображаемый в строке состояния окна браузера;
- statusbar — ссылка на строку состояния окна;
- toolbar — ссылка на панель инструментов окна;
- top — ссылка на родительское окно самого верхнего уровня, если текущий объект window представляет собой фрейм (в противном случае возвращается ссылка на само это окно).

Рассмотрим методы объекта window. Они позволяют открывать новые окна, причем с указанием их размера и определенным набором элементов окна, а также закрывать их. Кроме того, можно выводить на экран окна с предупреждениями, подтверждениями и окна для ввода информации. Перечислим эти методы:

- alert([{текст}]) — выводит на экран окно с предупреждением, текст которого передается как параметр;
- back — возвращает к предыдущему документу, как если бы на панели инструментов браузера щелкнули на кнопке Back (Назад);
- blur — убирает фокус с текущего окна;
- close — закрывает текущее окно. Если оно было открыто методом open, то закрывается сразу, если же было открыто пользователем, сначала появляется окно, где пользователь должен выбрать, закрывать окно или нет;
- confirm([{текст}]) — выводит на экран окно, в котором пользователю предлагается определить, будет ли текст передаваться как параметр. После щелчка на кнопке OK возвращается значение true, на кнопке Cancel (Отмена) — значение false;
- focus — передает фокус текущему окну;
- forward — переходит к следующему документу в списке журнала, как если бы на панели инструментов нажали кнопку Forward (Вперед);
- home — переходит на «домашнюю» страницу, заданную в настройках браузера, как если бы на панели инструментов нажали кнопку Home (Домой);
- open({адрес}, {имя_окна} [,{список свойств окна, разделенных запятыми}]) — открывает окно браузера, загружает в него документ, адрес которого предан в первом параметре, и присваивает окну имя, переданное во втором параметре (в третьем параметре может быть передан список свойств окна);
- print — выводит содержимое окна на печать;
- prompt({текст_приглашения} [,{значение_по_умолчанию}]) — выводит на экран диалоговое окно с полем ввода, приглашающее пользователя ввести какой-либо текст (текст самого приглашения передается в первом параметре, во втором параметре передается текст, отображаемый в поле ввода по умолчанию);
- resizeBy({X}, {Y}) — увеличивает окно на X пикселей по горизонтали и на Y пикселей по вертикали (для уменьшения размеров окна необходимо задать отрицательные значения параметров X и Y);
- resizeTo({X}, {Y}) — увеличивает или уменьшает окно до размера заданного значениями параметров X и Y;
- scroll({X}, {Y}) — прокручивает содержимое окна до точки, имеющей координаты X и Y;
- scrollBy({X}, {Y}) — прокручивает содержимое окна на X пикселей вправо и на Y пикселей вниз (для прокрутки влево и вверх необходимо задать отрицательные значения параметров X и Y);
- scrollTo({X}, {Y}) — прокручивает содержимое окна в точку, заданную значениями параметров X и Y;

- stop — останавливает загрузку текущей страницы.

Рассмотрим свойства объекта history. Они позволяют определять адреса текущего, предыдущего и следующего документов в списке журнала, а также возвращают размер списка. Перечислим их:

- current — адрес документа, загруженного в настоящее время;
- next — адрес следующего документа;
- previous — адрес предыдущего документа.

Методы объекта history позволяют перемещаться назад и вперед по списку журнала:

- back — загружает в окно браузера предыдущий документ;
- forward — загружает в окно браузера следующий документ;
- go({адрес}) — загружает в окно браузера документ, адрес которого наиболее близок к переданному в качестве параметра;
- go({позиция}) — перемещает в списке журнала на количество позиций, переданных в качестве параметра (можно задавать как положительные, так и отрицательные значения приращения).

Рассмотрим свойства объекта screen. С их помощью можно получить данные о размерах и глубине цвета экрана пользователя. Перечислим их:

- availHeight — высота рабочей области экрана (без панели задач и подобных ей элементов графического интерфейса системы) в пикселях;
- availWidth — ширина рабочей области экрана в пикселях;
- colorDepth — глубина цвета (для 16 цветов возвращает значение 2, для 256 — 8, для 16,7 млн. цветов (режим High Color) — 32);
- height — полная высота экрана в пикселях;
- width — полная ширина экрана в пикселях.

Рассмотрим свойства объекта navigator. Они позволяют определить некоторые данные о компьютере клиента, в частности тип и версию браузера, операционную систему и даже класс процессора:

- appMinorVersion — младшая цифра номера версии программы браузера (например, для MSIE 5.0 это свойство возвратит “0”, а для 5.5 — “5”), поддреживается только браузерами MSIE начиная с версии 4.0;
- appName — название браузера, например «Netscape» или «Microsoft Internet Explorer»;
- appVersion — версия браузера;
- cpuClass — класс процессора;
- language — код языка браузера;
- onLine — возвращает значение true, если клиент в настоящее время подключен к сети Internet (находится в режиме on-line), и значение false, если отключен (off-line);
- platform — название клиентской платформы, например «Win32»;

- systemLanguage — код языка операционной системы.

Рассмотрим методы объекта location. Они позволяют выполнять загрузку новых документов. Перечислим их:

- assign({адрес}) — загружает документ, адрес которого передан в качестве параметра, поддерживается только браузерами MSIE начиная с версии 4.0;
- reload([true|false]) — перезагружает документ с Web-сервера;
- replace({адрес}) — загружает документ, адрес которого передан в качестве параметра, и заменяет в списке журнала браузера адрес предыдущего документа адресом нового.

Для доступа к нужному элементу страницы используется универсальный метод, поддерживаемый большинством современных браузеров:

```
document.getElementById("имя_элемента")
```

В скобках указывается имя элемента, которое присваивается ему с помощью атрибута Name или ID.

Однако браузеры MSIE версии 4 и ниже не поддерживают данный метод, и к элементам следует обращаться следующим образом:

```
document.all["имя_элемента"]
```

Кроме того, существуют универсальные способы доступа к отдельным группам элементов. Все браузеры понимают такие способы обращения к изображениям:

```
document.images[номер]
document.images["имя_изображения"]
```

К элементам форм можно обратиться тремя способами:

```
document.forms[номер].имя_элемента
document.forms["имя_формы"].имя_элемента
document.имя_формы.имя_элемента
```

Перейдем к рассмотрению событий, так как именно их обработка позволяет сделать страницы динамическими. Событием в DHTML может быть любое действие пользователя: манипуляции с помощью мыши (щелчок правой или левой кнопкой, перетаскивание), использование клавиатуры, а также загрузка документа в окно браузера, его выгрузка и т.д. Полный список событий в алфавитном порядке приведен в табл. 5.1.

Для обработки события его необходимо записать как атрибут того тега, который это событие будет инициализировать. В общем виде запись будет выглядеть следующим образом:

```
<ТЕГ атрибут1=значение ... атрибутn=значение
событие1=значение ... событиеn=значение>
```

В одном теге можно обрабатывать любое количество событий (если они не противоречат друг другу и допустимы для этого тега).

Таблица 5.1

Список событий DHTML

СОБЫТИЕ	ОБРАБОТЧИК	УСЛОВИЕ ВОЗНИКНОВЕНИЯ
abort	onAbort	Прекращение загрузки объекта
blur	onBlur	Потеря фокуса
click	onClick	Щелчок на элементе
dblClick	onDblClick	Двойной щелчок на элементе
dragDrop	onDragDrop	Перетаскивание объекта в окно браузера
error	onError	Ошибка при загрузке документа или изображения
focus	onFocus	Получение элементом фокуса
keyDown	onKeyDown	Нажатие клавиши на клавиатуре (в противоположность опусканию клавиши) — обычно используется в паре с обработчиком onKeyUp
keyPress	onKeyPress	Нажатие клавиши на клавиатуре (одиночное событие)
keyUp	onKeyUp	Опускание клавиши на клавиатуре (в противоположность нажатию клавиши) — обычно используется в паре с обработчиком onKeyDown
load	onLoad	Завершение загрузки документа
mouseDown	onMouseDown	Нажатие кнопки мыши
mouseMove	onMouseMove	Перемещение указателя мыши
mouseOut	onMouseOut	Перемещение указателя мыши за пределы области объекта
mouseOver	onMouseOver	Размещение указателя мыши в пределах области объекта
mouseUp	onMouseUp	Опускание кнопки мыши
move	onMove	Перемещение окна браузера
reset	onReset	Очистка формы (щелчок по кнопке Reset)
resize	onResize	Изменение размеров окна браузера
scroll	onScroll	Прокрутка документа в окне браузера
select	onSelect	Выбор элемента формы
submit	onSubmit	Отправка формы
unload	onUnload	Выгрузка документа

Рассмотрим порядок возникновения событий связанных с использованием мыши. Здесь представлена последовательность выполнения событий.

Следующие события мыши, для которых не требуется щелчок любых кнопок мыши, происходят в таком порядке:

onMouseOver

onMouseMove

onMouseOut

Для событий, связанных с нажатием кнопки, порядок следующий:

onMouseOver

onMouseMove
onMouseDown
onMouseUp
onClick
onDblClick
onMouseOut

При обработке событий в том же элементе, свойства которого надо изменить, нет необходимости обращаться непосредственно к объекту. Достаточно использовать ключевое слово this, т.е. вместо

```
document.имя_элемента.style.свойство=значение
```

указать

```
this.style.свойство=значение
```

Например, чтобы при щелчке на блоке менялся его фоновый цвет, необходимо первоначально создать блок, затем задать ему фоновый цвет, а потом добавить в тег событие. Если изменение должно происходить после щелчка мыши, нужно добавить событие onClick:

```
<DIV STYLE="width:200px; height:100px; background-color: #0099FF"
onClick="this.style.backgroundColor='yellow'">текст в блоке</DIV>
```

Фоновый цвет можно изменить и при наведении указателя мыши на элемент, для этого необходимо использовать событие onMouseOver. А для того чтобы элементу возвращался первоначальный цвет при убирании указателя мыши, надо добавить также событие onMouseOut:

```
<DIV STYLE="width:200px; height:100px; background-color: #0099FF"
onMouseOver="this.style.backgroundColor='yellow'"
onMouseOut="this.style.backgroundColor='#0099FF'">текст в блоке</DIV>
```

Конечно, можно изменять не только фоновый цвет блока, но и значение любого свойства, которое задается при помощи стилей. По такому же принципу можно изменять все свойства элементов страницы, которые применимы к конкретному элементу. По такому же принципу можно изменять свойства одних элементов с помощью событий других (обычно используются события ссылок и кнопок). Для этого событие необходимо объявить в элементе, который будет производить действие, и в этом событии обратиться к изменяемому элементу и указать новые свойства. В тег изменяемого элемента никаких дополнений вносить не нужно.

Например, чтобы щелчком на кнопке (или ссылке) изменить ширину расположенной на странице таблицы, достаточно в теге таблицы указать ее имя с помощью атрибута NAME или ID:

```
<TABLE ID="table1" WIDTH="300px">
...
</TABLE>
```

После этого в любом месте страницы нужно создать кнопку, добавить к ней событие onClick и в событии обратиться к объекту, указав для него новую ширину:

```
<INPUT TYPE="button" VALUE="Изменить"
onClick='document.getElementById("table1").style.width="500px"' >
```

Любой элемент может менять свойства другого элемента: щелчок на изображении может изменить размеры текста, а наведение мыши на текст привести к появлению или исчезновению изображения. Для осуществления таких действий нет практически никаких препятствий — все зависит от того, можно ли применять тот или иной стиль к конкретному элементу.

5.5. Язык JavaScript

Для применения сценариев JavaScript в HTML-документах используется контейнер <SCRIPT>. Обычно его размещают в служебной части документа, чтобы он был загружен первым. Однако при этом никаких ограничений на местоположение сценариев не существует. Также количество сценариев, а значит контейнеров <SCRIPT> на странице не ограничено.

Тег <SCRIPT> имеет несколько атрибутов. Для указания версии языка используется атрибут LANGUAGE. Чаще всего в качестве его значения используется строка javascript, что указывает на применение именно языка JavaScript (без уточнения его версии), а не другого языка программирования (например, для языка Visual Basic Script используется значение vbscript).

Также можно использовать сценарий, сохраненный в отдельном файле с расширением .js, в котором никакие теги языка HTML не применяются, только JavaScript. Возможность включения файла сценария в HTML-документ может быть удобной в тех случаях, когда один и тот же сценарий используется на нескольких страницах либо когда нужно скрыть его текст от пользователей. В случае помещения сценария прямо в документ это сделает его доступным клиентам, если же сценарий внедрен с помощью файла, можно видеть только ссылку на файл. Внедрение сценария таким способом осуществляется с помощью атрибута SCR:

```
<SCRIPT SCR="адрес/имя_файла">
```

В этом случае закрывающий тег </SCRIPT> указывать не нужно.

Также можно воспользоваться контейнером <NOSCRIPT> (его можно разместить где угодно). Он предназначен для применения в браузерах, не поддерживающих работу со сценариями. Внутри контейнера располагается текст, который будет отображаться в окне таких браузеров, например:

```
<NOSCRIPT>
    Текст, отображаемый браузерами, не поддерживающими сценарии
</NOSCRIPT>
```

Содержание сценария по этим же причинам заключается в контейнер комментария: если браузер не поддерживает сценарии, они будут отображаться на странице.

```
<SCRIPT>
    <!--
        содержание сценария
    //-->
<SCRIPT>
```

Рассмотрим основные элементы языка JavaScript.

- Переменные — предназначены для хранения значений, объявляются с помощью ключевого слова var (например, var a=4) либо инициализируются с помощью операции присваивания (например, a=4). Область действия переменных, объявленных вторым способом, ограничивается текущим сценарием. Переменные, объявленные с помощью ключевого слова var, действуют в пределах всего документа. Если переменная объявлена внутри функции, то ее область ограничена телом функции.
- Операция присваивания (=) — используется, например, в случае, когда переменной необходимо присвоить значение.
- Бинарные арифметические операции — обычные математические операции над двумя операндами. К ним относятся операции сложения (+), вычитания (–), умножения (×), деления (/) и операция получения остатка от деления (%).
- Унарные арифметические операции, инкремента (++) и декремента (--), — позволяют соответственно увеличивать и уменьшать значение операнда на единицу.
- Операции сравнения — осуществляются с помощью знаков больше (>), меньше (<), меньше или равно (<=), больше или равно (>=), не равно (!=) и равно (==).
- Логические операции — служат для вычисления логического результата. В JavaScript используются логические операции И (&&), ИЛИ (||) и НЕ (!).
- Условные операторы — позволяют выполнить определенные действия в зависимости от результата проверки одного или нескольких условий:

```
if(условие){
        действие, если условие истинно
```

```
}else{
        действие, если условие ложно
}
```

Функции — представляют собой подпрограммы, позволяющие выполнять определенные задачи. Сначала функцию необходимо объявить на текущей странице:

```
function имя_функции(параметры){
        тело_функции
}
```

Чтобы функцию можно было использовать, ее нужно вызвать. Зачастую функция привязывается к событию, например:

```
<A HREF=# onClick=имя_функции(параметры)>
```

Комментарии — указываются с помощью символов // и позволяют ввести поясняющий текст.

Рассмотрим заново пример, связанный с изменением ширины таблицы, теперь уже с использованием конструкций языка JavaScript. Как и раньше, событие onClick свяжем с кнопкой, но теперь оно вызывает функцию:

```
<INPUT TYPE="button" VALUE="Изменить" onClick=change_widht()>
```

Сценарий с объявлением функции поместим в контейнер <HEAD>:

```
<SCRIPT>
  <!--
        function change_width(){
        document.getElementById(«table1»).style.width=»500px»;
        }
  //-->
<SCRIPT>
```

Таким способом можно изменять свойства и других элементов.

Рассмотрим теперь пример сценария, который позволяет при наличии двух кнопок при щелчке на каждой из них отображать определенный элемент, а отображаемый до этого элемент скрывать. Для этого создаем кнопки, связываем с ними события, каждое событие вызывает функцию:

```
<INPUT TYPE="button" VALUE="Кнопка 1" onClick=func1()>
<INPUT TYPE="button" VALUE="Кнопка 2" onClick=func2()>
```

Теперь необходимо создать сценарий с объявлениями функций, каждая из которых должна делать видимым свой объект, а остальные скрывать:

```
<SCRIPT>
<!--
    function func1(){
```

```
            document.getElementById("имя_элемента_1"
            ).style.visiblity="visible";
            document.getElementById("имя_элемента_2"
            ).style.visiblity="hidden";
            }
        function func2(){
            document.getElementById("имя_элемента_1"
            ).style.visiblity="hidden";
            document.getElementById("имя_элемента_2"
            ).style.visiblity="visible";
            }
    //-->
    <SCRIPT>
```

Модифицируя представленный пример, сделаем так, чтобы повторный щелчок на той же кнопке приводил к сокрытию связанного с ней элемента. Для этого в функции необходимо выполнять проверку условия — если элемент уже отображается, то его необходимо скрыть, а если не отображается, наоборот, отобразить:

```
function func1(){
    if(document.getElementById("имя_элемента_1"
    ).style.visiblity=="visible"){
        document.getElementById("имя_элемента_1").style.visiblity=
        "hidden";
    }else{
        document.getElementById("имя_элемента_1").style.visiblity=
        "visible";
}
document.getElementById("имя_элемента_2").style.visiblity=
"hidden";
}
```

Аналогично выполняется проверка и в функции func2().

Теперь рассмотрим другой пример. Необходимо создать библиотеку из 10 изображений. При загрузке страницы должны отобразиться первое изображение, а также кнопки навигации по библиотеке. Щелчок на кнопке Следующее приводит к отображению следующего изображения, щелчок на кнопке Предыдущее — наоборот, к отображению предыдущего.

Вначале необходимо загрузить все изображения, но только первое из них сделать видимым:

```
<IMG SCR="изображение_1"> STYLE="position:absolute; top:...;
left:...; visibility:visible">
<IMG SCR="изображение_2"> STYLE="position:absolute; top:...;
```

```
left:...; visibility:hidden">
...
<IMG SCR="изображение_10"> STYLE="position:absolute; top:...;
left:...; visibility:hidden">
```

Для того чтобы рисунки отображались на одном и том же месте, применяется абсолютное позиционирование.

Кнопки создаются обычным образом и с ними связываются функции обработки события щелчка:

```
<INPUT TYPE="button" VALUE="Предыдущее" onClick=back()>
<INPUT TYPE="button" VALUE="Следующее" onClick=forward()>
```

В начале сценария объявим переменную i. В функции forward() будем увеличивать значение I, чтобы каждый щелчок на кнопке приводил к сокрытию текущего изображения и отображению следующего. Увеличение значения i будет производиться до тех пор, пока оно не достигнет предела. В этом случае i обнулится и изображения снова будут отображаться, начиная с первого. Аналогичным образом устроена функция перемещения назад back():

```
<SCRIPT>
  <!--
    var i=0;
    function forward(){
        document.images[i].style.visiblity="hidden";
        if(i<9){
            i=i+1;
        }else{
            i=0;
        }
        document.images[i].style.visiblity="visible";
        }
    function back(){
        document.images[i].style.visiblity="hidden";
        if(i>0){
            i=i-1;
          }else{
            i=9;
          }
            document.images[i].style.visiblity="visible";
        }
        //-->
        <SCRIPT>
```

Рассмотрим, каким образом с помощью сценариев JavaScript можно открывать и закрывать окна, а также выводить на экран окна предупреждений, подтверждений и ввода информации.

Для открытия окна используется метод open объекта window. Оформим это как отдельную функцию в сценарии:

```
<SCRIPT>
    <!--
        function имя_функции(){
            window.open("адрес/имя_файла", "имя_окна", "width=300,
        height=300, screenX=300, screenY=20, status=no,
    toolbar=no,
        menubar=no, resizeable=no, scrollbar=yes, titlebar=no");
        }
    //-->
<SCRIPT>
```

Новое окно можно открыть, не прибегая к созданию сценария, прямо связав вызов метода с событием, например, щелчок на ссылке:

```
<A HREF=# onClick=' window.open("адрес/имя_файла", "имя_окна",
"width=300, height=300, screenX=300, screenY=20, status=no,
toolbar=no, menubar=no, resizeable=no, scrollbar=yes,
titlebar=no")'>текст</A>
```

Для закрытия окна применяется метод close, например, в теге кнопки:

```
<INPUT TYPE="button" VALUE="Закрыть" onClick='window.close()'>
```

Окно предупреждения отображается с помощью метода alert, объекта window. Следующий код выводит окно предупреждения, которое клиент должен закрыть щелчком на единственной кнопке OK, прежде чем сможет загрузить следующую страницу:

```
<A HREF="адрес_страницы" onClick='window.alert(
"текст сообщения")'>текст</A>
```

Окно подтверждения отличается от окна предупреждения наличием уже двух кнопок. При щелчке на кнопке OK выполнение операции продолжается, а при щелчке на кнопке Cancel (Отмена) действие будет прервано. Окно подтверждения реализуется с помощью метода confirm:

```
<INPUT TYPE="button" VALUE="Действие"onClick=
    'window.confirm("текст сообщения")'>
```

Окно третьего вида, предназначенное для ввода информации, отображается методом prompt объекта window:

```
<BODY onLoad=your_func()>
<SCRIPT>
    <!--
```

```
function your_func(){
        var name= window.open("Укажите ваше имя", "");
        document.write("<H2 ALIGN=center>Здравствуйте, "+name+
        «!</H>")
    }
//-->
<SCRIPT>
```

Некоторые свойства объектов содержат информацию о компьютере клиента. Эти сведения можно использовать как для сбора статистики, так и для изменения страниц — они могут подстраиваться под конкретные браузеры, разрешение, количество цветов и т.д.

Для реализации такой настройки в сценарии лучше всего применять оператор if, позволяющий проводить сравнение и выполнять в зависимости от его результата одно из двух различных действий. Например, следующий сценарий заменяет текущую страницу на страница_1, если используется браузер MSIE, или на страница_2 — если используется браузер Netscape, или на страница_3 в противном случае:

```
<SCRIPT>
    <!--
        if(navigator.AppName="Microsoft Internet Explorer"){
        location.replace("страница_1");
        }else{
        if(navigator.AppName="Netscape"){
        location.replace("страница_2");
        }else{
        location.replace("страница_3");
        }
        }
//-->
</SCRIPT>
```

С помощью JavaScript легко создавать панели навигации. Используя ссылки или кнопки, можно дать клиентам возможность переходить к предыдущей или к следующей просмотренной странице:

```
<A HREF="javascript:history.back()">Назад</A>
<INPUT TYPE="button" VALUE="Назад" onClick='history.forward()'>
```

Сценарии позволяют проверять правильность заполнения полей формы, а именно запрещать и разрешать ввод тех или иных значений. Для запрета ввода в текстовое поле определенных символов необходимо обработать событие onKeyPress. Приведем пример сценария, разрешающего ввод в текствое поле только цифр:

```
<INPUT TYPE="text" onKeyPress='if((event.keyCode<48)||(
event.keyCode>57))event.returnValue=false;'>
```

Существует еще одна возможность — использование раскрывающегося списка как набора ссылок на другие страницы. В этом случае клиент выбирает в списке нужный ему элемент и автоматически переходит на другую страницу:

```
<SCRIPT>
    <!--
        function имя_функции(){
            window.location.href=
        document.имя_формы.имя_списка.options[
        document.имя_формы.имя_списка.selectedIndex].value;
        }
    //-->
<SCRIPT>
. . .
<FORM NAME=имя_формы>
<SELECT NAME=имя_списка SIZE=1 onChange=имя_функции()>
    <OPTION VALUE=# SELECTED>Выберите вариант</OPTION>
    <OPTION VALUE="страница_1">Вариант 1</OPTION>
    <OPTION VALUE=»страница_2»>Вариант 2</OPTION>
    . . .
    <OPTION VALUE=»страница_n»>Вариант n</OPTION>
</SELECT>
</FORM>
```

Реализованная функция достаточно проста. С помощью свойства href объекта location производится переход на страницу, адрес которой является значением выбранного элемента. В самой форме адреса ссылок указываются как значения элементов списка, а для запуска функции сценария используется метод onChange.

5.6. Разработка теста с использованием HTML и JavaScript

Рассмотрим особенности применения языков HTML и JavaScript на примере разработки теста к этой книге. В оболочку войдут все тестовые задания, указанные в конце каждой главы пособия. Для этого вначале следует определить документ HTML, то есть создать его либо воспользовавшись одним из редакторов HTML, либо с использованием простого «Блокнота». Затем запишем основные конструкции, определяющие HTML-документ:

```
<HTML>
    <HEAD>
        <TITLE> Тест </TITLE>
```

```
</HEAD>
<BODY background=»background.JPG»>
    <H2 align=center> Тестирование по дисциплине "Программное
    обеспечение компьютерных сетей"</H2>
</BODY>
</HTML>
```

Затем определим в теле документа форму и расположенные на ней элементы управления. Это могут быть или флажки (checkbox) или переключатели (radio). Для того чтобы, например, разместить на форме группу флажков, следует в теле документа указать следующее:

```
<FORM name=f>
    <P><center><B>Тема №1. Технология "клиент-сервер"</B>
    </center></P>
    <P>
        1. Какими особенностями характеризуются системы
        с разделением времени (TSS)?
    </P>
        <INPUT TYPE=checkbox NAME=test11 VALUE="1"> а) защита
        данных от несанкционированного доступа <Br>
        <INPUT TYPE=checkbox NAME=test11 VALUE="2">
        б) взаимная изоляция участников <Br>
        <INPUT TYPE=checkbox NAME=test11 VALUE="3">
        в) использование общих ресурсов всеми участника <Br>
        <INPUT TYPE=checkbox NAME=test11 VALUE="4">
        г) использование каждым участником собственной ЭВМ <Br>
</FORM>
```

Таким способом добавляется столько вопросов и тестовых заданий, сколько может потребоваться. После их добавления в теле формы указываются теги, позволяющие создать кнопку, для вывода результатов теста:

```
<P><CENTER>
    <INPUT TYPE=»button» VALUE=»Завершить тестирование»
    onClick= «ResTest()»>
</P></CENTER>
```

В качестве ответа на нажатие указанной кнопки, то есть в результате события на ее нажатие, вызывается функция ResTest(), которую следует описать в скрипте:

```
<SCRIPT language=»javascript»>
    <!--
        function ResTest()
        {
            var n=0
```

```
            if(document.f.test11[0].checked && document.f.test11[1].
            checked && document.f.test11[3].checked){n++}
            if(document.f.test12[0].checked && document.f.test12[3].
            checked){n++}
            if(document.f.test13[1].checked){n++}
            if(document.f.test14[0].checked && document.f.test14[1].
            checked && document.f.test14[3].checked){n++}
            if(document.f.test15[2].checked){n++}
            n=n/35*100
            var res
            if (n<50){res=»Отметка 2!»}
            if (n>60 && n<80){res=»Отметка 3!»}
            if (n>80 && n<90){res=»Отметка 4!»}
            if (n>90){res=»Отметк а 5!»}
        window.alert(«Результат: «+ Math.round(n) +»%. " + res)
        }
    //-->
</SCRIPT>
```

Программа подсчитывает количество правильных ответов и выводит на экран диалоговое окно с результатом теста. Переменная n содержит процент правильных ответов. В данном случае число вопросов указано число 35, поэтому для вычисления полученного процента записывается выражение n=n/35*100.

В результате документ должен выводить на экран HTML-документ с тестами и кнопку «Завершить тестирование», при нажатии на которую выводится окно с полученным процентом правильных ответов и отметкой.

Вопросы для самопроверки

1. Дайте общую характеристику средствам разработки статической части Web-страницы.
2. Опишите основные конструкции языка HTML.
3. Охарактеризуйте язык гипертекстовой разметки XML.
4. Дайте определение языка DHTML и опишите объектную модель документа.
5. Опишите основные элементы языка JavaScript и способы их применения.

Тесты

1. В каких форматах в основном хранятся изображения в HTML-документах?
   А. JPEG.
   Б. GIF.

В. BMP.
Г. PNG.

2. Какая конструкция обозначает начало HTML-документа?
   А. <HTML>.
   Б. <?xml version=»1.0»?>.
   В. <BODY>.
   Г. <SCRIPT>.

3. Какой элемент является корневым в объектной модели документа?
   А. Window.
   Б. Screen.
   В. Frames.
   Г. Document.

4. Сколько событий можно обрабатывать в одном теге?
   А. Одно.
   Б. Два.
   В. Три.
   Г. Любое количество.

5. Что из перечисленного относится к бинарным операциям в JavaScript?
   А. ++.
   Б. !=.
   В. %.
   Г. ||.

Глава 6. РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ

6.1. Возможности системы Delphi по созданию Web-приложений

Система визуального проектирования Delphi обладает большим набором классов, реализованных в виде компонентов, предназначенных для работы с Internet, WWW и сетями TCP/IP. Они располагаются на панели Internet (и др.) и позволяют выполнять следующие базовые функции: поддерживать работу Web-сервера (сервера, ответственного за функционирование конкретных Web-узлов в Internet), организовывать связь с помощью сокетов и по протоколу HTTP, а также доступ к данным и их публикацию на Web-узле.

В данном случае под Web-узлом будем понимать виртуальный узел сети, имеющий свой IP-адрес и обеспечивающий набор функций для рассылки обращающимся к нему пользователям HTML-страниц и прочей вспомогательной информации; а под Web-сервером — программу или набор программ, которые поддерживают работу Web-узлов на физическом уровне, предоставляя возможность динамического формирования HTML-страниц (например, на основе информации, хранимой в базе данных), позволяя запускать специализированные программы, а также выполняя вспомогательные функции по сопровождению программного обеспечения, установленного на компьютере, где работает Web-сервер.

Еще одна панель, FastNet (система ООП Delphi 5.0), содержит большое число компонентов, поддерживающих популярные службы и протоколы Internet: работу с электронной почтой и группами новостей (телеконференциями), обмен файлами и многое другое.

Реализация приложений на основе перечисленных компонентов в некоторых случаях может потребовать доступа к Internet и возможность запуска собственного Web-сервера. Для этого необходимо оплатить выделенный канал и установить свой компьютер, на котором будет работать Web-сервер и реализуемые программы.

Однако существует более простой способ, связанный с возможностями протокола TCP/IP допускать работу с локальным компьютером, как если бы он был подключен к сети, поддерживающей этот протокол (например, к Internet). Каждый компьютер имеет стандартный IP-адрес и соответствующее ему имя localhost. Соответствие IP-адресов и названий в сети TCP/IP, даже если она состоит из одного компьютера, задается в файле HOSTS каталога Windows. Так как в сети, насчитывающей несколько компьютеров, не может быть оди-

наковых имен (и одинаковых IP-адресов), то операционная система позволяет в настройках протокола указать уникальное сетевое имя компьютера.

Чтобы компьютер мог выполнять функции Web-сервера, на нем должна быть запущена соответствующая программа, например Microsoft Personal Web Server. Если такую программу установить на компьютере и выполнить все необходимые настройки, то этот компьютер во время подключения к Internet будет работать наравне со всеми остальными Web-узлами.

6.2. Разработка Web-браузера

Наиболее простым примером реализации возможностей таких систем, как Delphi, в разработке приложений для Web-систем является создание Web-браузера, позволяющего осуществлять просмотр Web-страниц.

Для его простейшей реализации необходимо использовать следующие компоненты: TWebBrowser (Internet), TToolBar (Win32), TMainMenu (Standard), TImageList (Win32), TStatusBar (Win32), TProgressBar (Win32), TOpenDialog (Dialogs), TComboBox (Standard).

Панели инструментов создаются с помощью компонента TToolBar, строка состояния — TStatusBar, главное меню приложения разрабатывается на основе компонента TMainMenu. Функции собственно Web-браузера реализуются с помощью компонента TWebBrowser, для организации работы которого используется вызов процедур, обеспечивающих загрузку, навигацию, обновление и другие возможности.

Рассмотрим основные методы и свойства каждого из перечисленных компонентов, использование которых может потребоваться в данном случае.

TWebBrowser включает следующие свойства, методы и события:

- property Busy — имеет значение true, если компонент находится в процессе перехода к новой странице или в процессе загрузки документа;
- property LocationName — сокращенный текст адреса URL;
- property LocationURL — полный адрес URL;
- property Offline — имеет значение true, если компонент работает в отключенном от Internet (автономном) режиме;
- property RegisterAsBrouser — имеет значение true, если компонент используется в системе в качестве основного браузера;
- property Visible — имеет значение true, если компонент виден на экране;
- procedure GoBack — вернуться к предыдущему просмотренному документу;

- procedure GoForward — перейти к следующему просмотренному документу;
- procedure GoHome — перейти к начальной основной странице;
- procedure GoSearch — перейти к текущей поисковой странице;
- procedure Navigate(const URL: WideString);
- procedure Navigate(const URL: WideString;
  var Flags: OleVariant;
  var TargetFrameName: OleVariant;
  var PostData: OleVariant; var Headers: OleVariant) — переместиться к ресурсу, указанному в параметре URL;
- procedure Navigate2(var URL: OleVariant);
- procedure Navigate2(var URL: OleVariant;
  var Flags: OleVariant;
  var TargetFrameName: OleVariant;
  var PostData: OleVariant; var Headers: OleVariant) — переместиться к ресурсу, указанному в параметре URL (он не обязательно представляет собой адрес URL в текстовом виде);
- procedure Refresh — перезагрузить текущий документ;
- procedure Refresh2;
- procedure Refresh2(var Level: OleVariant) — перезагрузить текущий документ с учетом вида перезагрузки (параметр Level);
- procedure Stop — прервать текущую операцию;
- OnBeforeNavigate2 — перед переходом к новому документу;
- OnDocumentComplete — документ загружен полностью;
- OnDownloadBegin — начата загрузка документа;
- OnDownloadComplete — загрузка документа завершена;
- OnNavigateComplete2 — успешно завершен переход к новому ресурсу;
- OnNewWindow2 — браузер собирается создать новое окно;
- OnProgressChange — изменился процент загрузки документа;
- OnTitleChange — изменена информация о названии текущего документа;
- OnVisible — браузер становится видимым или невидимым.

На основе класса TToolBar могут быть созданы многострочные наборы инструментов, содержащие элементы управления быстрого доступа. Сразу после создания панель инструментов автоматически размещается у края формы. Для использования вместе с панелью разработаны кнопки класса TToolButton, для их добавления используют команду контекстного меню панели NewButton. Массив таких кнопок хранится в свойстве Buttons. Их текущее число определяется значение свойства ButtonCount.

TToolBar включает следующие свойства:

- property EdgeBorders — определяет наличие видимых краев с каждой из четырех сторон панели;

- property EdgeInner, property EdgeOuter — определяют вид окаймляющих полос с внутренней и внешней стороны;
- property ButtonWidth, property ButtonHeight — ширина и высота в (пикселях) элементов управления, располагаемых на панели;
- property DisabledImages — указывает на объект TImageList, который содержит изображения кнопок, отображающих их «отключенное» состояние. Если значение для данного свойства не задано, используются обычные картинки кнопок, преобразованные к палитре, содержащей только тона серого;
- property Flat — имеет значение true, если используется стиль «прозрачных» кнопок, когда фон панели и кнопок не рисуется;
- property HotImages — указывает на объект TImageList, который содержит изображения кнопок, появляющихся при наведении на кнопку указателя мыши;
- property Images — указывает на объект типа TImageList, который содержит рабочие изображения кнопок;
- property Indent — сдвиг левой границы панели в пикселях;
- property List — имеет значение true, если необходимо показывать заголовки кнопок справа от изображений, а не под ними;
- property ShowCaptions — имеет значение true, если текстовые заголовки кнопок должны отображаться на панели;
- property Transparent — имеет значение true, если панель прозрачная. Это свойство не влияет на прозрачность объектов на панели;
- property Wrapable — имеет значение true, если элементы управления на панели должны автоматически образовывать новые строки, если им не хватает пространства формы.

Перечислим основные свойства класса TToolButton:

- property AllowAllUp — имеет значение true, если все кнопки группы, к которой относится данная кнопка, могут одновременно находиться в отжатом состоянии;
- property Down — состояние кнопки (значение true, если кнопка нажата);
- property DropdownMenu — ссылка на меню кнопки (компонент TPoupMenu). Дополнительно для свойства кнопки Style надо задать значение tbsDropDown. При этом к правой части кнопки добавляется небольшая панель со стрелкой, при щелчке на которой открывается меню;
- property Grouped — имеет значение true, если данная кнопка входит в состав группы. Режим работы в группе задает значение tbsCheck для свойства кнопки Style. В группе может быть нажата только одна кнопка, при щелчке на неопущенной кнопке все остальные отжимаются, а она остается в нажатом состоянии. Такой режим действует только в диапазоне смежных кнопок,

которые не отделены друг от друга разделителями и все имеют значение true для свойства Grouped и значение tbsCheck для свойства Style;

- property ImageIndex — номер картинки из списка картинок, на который ссылается родительский объект (TToolBar);
- property Indeterminate — имеет значение true, если кнопка находится в «промежуточном» состоянии. Это позволяет показывать «частичность» выполняемых функций. Свойство Down при этом автоматически принимает значение false;
- property Index — порядковый номер кнопки;
- property Marked — имеет значение true, если кнопка изображается как отключенная;
- property MenuItem — если в данном свойстве указан один из доступных на форме пунктов любого из существующих меню, то щелчок на данной кнопке приведет к выполнению действия, связанного с этим пунктом;
- property Style — стиль кнопки. Возможные значения — tbsButton (командная кнопка), tbsCheck (кнопка-переключатель), tbsDropDown (кнопка-меню), tbsSeparator (кнопка-разделитель), tbsDivider (широкий разделитель);
- property Wrap — имеет значение true, если требуется, чтобы следующая кнопка размещалась на новой строке панели.

Ни одно приложение Windows не обходится без главного меню, для создания которого в Delphi предусмотрен компонент TMainMenu. Настройка меню производится с помощью редактора меню, вызываемого двойным щелчком на компоненте. Для создания пункта меню в Инспекторе объектов нужно в свойство Caption вписать название первого пункта, а затем нажать клавишу ENTER. Редактор меню переключится обратно в проектируемое меню, где уже появится первый пункт. Для задания горячей клавиши необходимо перед соответствующей буквой названия пункта меню вставить символ «&». Чтобы вставить линию-разделитель, надо в свойстве Caption в первой позиции указать символ «-» (дефис). Для добавления новых пунктов служит клавиша INSERT, для удаления клавиша — DELETE. Каждый создаваемый пункт меню формируется классом TMenuItem.

Для хранения изображений кнопок панели инструментов используется невизуальный компонент TImageList. Он позволяет хранить наборы картинок фиксированного размера, обращаться к ним по номерам и осуществлять вывод изображений на экран различными способами.

Перед загрузкой изображений в данный компонент надо настроить ряд основных его свойств:

- property AllocBy — число новых элементов, добавляемых, если при добавлении очередного изображения требуется расширение

списка. Например, если значение свойства AllocBy равно 4 и список содержит четыре картинки, то при добавлении пятой длина списка увеличится до 8 элементов (на 4). С помощью данного свойства можно регулировать распределение памяти, если планируется активная работа программы с большим числом изображений;

- property BkColor — фоновый цвет рисунка. При выводе этим цветом заполняется маскируемая область;
- property BlendColor — цвет переднего плана, используется, когда изображение рисуется как выделенное (например, имеющее фокус);
- property Count — число картинок в списке;
- property DrawingStyle — способ вывода изображения на экран. Возможные значения — dsFocused (цвет картинки смешивается с цветом BlendColor на 25%), dsSelected (цвет картинки смешивается с цветом BlendColor на 50%), dsNormal (в качестве фонового цвета используется цвет BkColor). Если значение свойства равно clNone (цвет не задан), то изображение рисуется с использованием прозрачного цвета и маски. Значение dsTransparent указывает, что изображение рисуется прозрачным независимо от значения свойства BkColor;
- property Height — высота одного изображения в пикселях;
- property ImageType — тип изображения. Возможные значения — itImage (картинка), itMask (маска);
- property Masked — имеет значение true, если при выводе изображения используется маска. Маскируемая часть изображения либо выводится как прозрачная, либо заполняется цветом, указанным в свойстве BkColor;
- property ShareImages — имеет значение false, если область памяти, выделенная для списка картинок, освобождается после завершения работы программы;
- property Width — ширина одного изображения в пикселях.

Изображения загружаются в список с помощью встроенного редактора. Он вызывается двойным щелчком на объекте ImageList (рис. 6.1).

Новые картинки добавляются с помощью кнопки Add (Добавить). Пока редактор не закрыт, добавленные изображения можно редактировать. Раскрывающийся список TransparentColor (Прозрачный цвет) позволяет выбрать цвет, который будет считаться прозрачным. Пиксели этого цвета не выводятся на холст. Если прозрачность не нужна, следует выбрать значение clNone.

На панели Options (Параметры) задается способ позиционирования картинки:

- Crop (Обрезка) — означает, что изображение вкладывается в элемент списка, начиная от его верхнего угла;

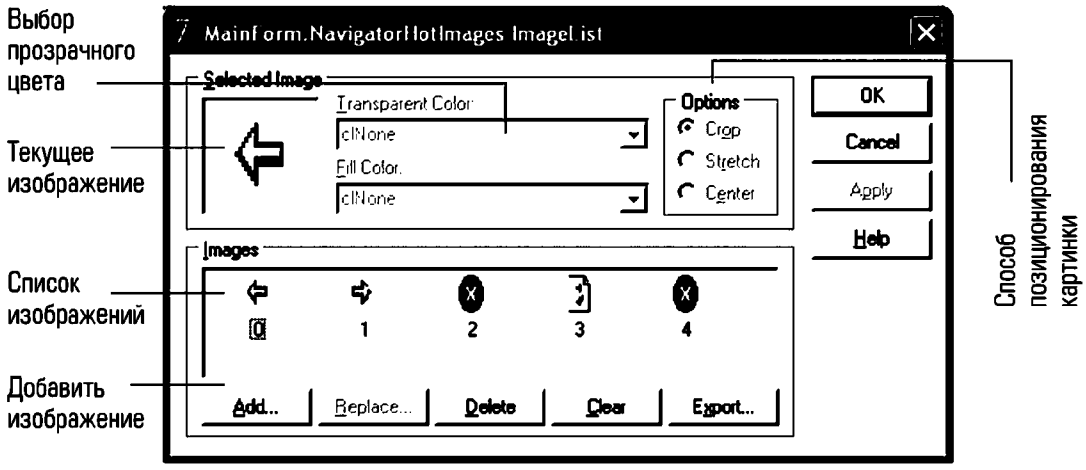

Рис. 6.1. Редактор списка изображений

- Stretch (Растяжение) — указывает, что изображение сжимается или растягивается, чтобы соответствовать параметрам ширины и высоты элемента списка;
- Center (По центру) — позиционирует изображение по центру допустимого прямоугольника. Если оно в нем не помещается, то выступающие части обрезаются.

Перечислим основные методы класса TImageList:

- function Add(Image, Mask: TBitmap): Integer — добавление в список изображения и его маски (четно-белого точечного изображения, где белый цвет обозначает прозрачную область). Возвращаемое значение — номер изображения в списке. Нумерация начинается с нуля;
- function AddIcon(Image: TIcon): Integer — добавление значка. Значки всегда рисуются прозрачными;
- procedure AddImages(Value: TCustomImageList) — копирование всех изображений из другого объекта ImageList;
- function AddMasked(Image: TBitmap; MaskColor: TColor): Integer — аналогично методу Add, но вместо дополнительной картинки-маски задается цвет изображения, который считается прозрачным, подобно работе с редактором списка;
- procedure Clear — удаление всех изображений из списка;
- procedure Draw(Canvas: TCanvas; X, Y, Index: Integer; Enabled: Boolean=True) — основная процедура вывода изображения с номером Index на холст Canvas в позицию (X, Y). Способ отображения определяется значением свойства DrawingStyle. Если значение свойства Enabled равно false, то изображение будет рисоваться в сером «выключенном» стиле;
- function FileLoad(ResType: TResType; Name: string; MaskColor: TColor): Boolean — загрузка изображения из файла ресурсов с

именем Name. В параметре ResType указывается тип ресурса, в параметре MaskColor задается прозрачный цвет;

- procedure GetBitmap(Index: Integer; Image: TBitmap) — получение элемента изображения с номером Index как объекта класса TBitmap. Результат возвращается во втором параметре, который должен быть переменной;
- procedure GetIcon(Index: Integer; Image: TIcon) — получение элемента изображения (или его части) с номером Index как объекта класса TIcon;
- procedure GetImages(Index: Integer; Image, Mask: TBitmap) — разделение изображения с номером Index на две части: цветное точечное изображение и монохромную маску. Эти изображения возвращаются во втором и третьем параметрах;
- procedure Insert(Index: Integer; Image, Mask: TBitmap) — добавление изображения и маски в конкретную позицию списка;
- procedure InsertIcon(Index: Integer; Image: TIcon) — добавление значка в конкретную позицию;
- procedure InsertMasked(Index: Integer; Image: TBitmap; MaskColor: TColor) — добавление изображения с маскируемым цветом в конкретную позицию списка;
- procedure Move(CurIndex, NewIndex: Integer) — перестановка изображения в списке из позиции CurIndex в позицию NewIndex;
- procedure Replace(Index: Integer; Image, Mask: TBitmap) — замена элемента списка на новое изображение (с маской);
- procedure ReplaceIcon(Index: Integer; Image: TIcon) — замена элемента списка (значка) на новое изображение;
- procedure ReplaceMasked(Index: Integer; NewImage: TBitmap, MaskColor: TColor) — замена элемента списка на новое изображение (с маскируемым цветом);
- function ResourceLoad(ResType: TResType; Name: string; MaskColor: TColor) — загрузка изображения из ресурса программы под названием Name. В параметре ResType указывается тип ресурса, в параметре MaskColor — прозрачный цвет.

Строка состояния, реализуемая компонентом TStatusBar, предназначена для отображения какой-либо дополнительной информации (например, о текущем режиме работы приложения). После размещения на форме она автоматически прикрепляется к нижней ее части. Высоту строки можно менять с помощью свойства height.

Текст в строке состояния можно привязать к текущей подсказке любого элемента управления на форме, тогда в строке состояния будет автоматически отображаться подсказка для текущего элемента управления. Для этого свойство AutoHint должно иметь значение true.

В самом простом варианте строка состояния работает, как одна большая панель. При этом свойство SimplePanel получает значение

true, выводимый текст записывается в свойство SimpleText. Часто используется свойство Panels, представляющее собой коллекцию объектов типа TStatusPanel, из панелей которых составляется строка состояния. Для их создания и обработки применяется стандартный редактор коллекций.

Приведем главные свойства класса TStatusPanel:

- property Alignment — способ выравнивания текста на данной панели;
- property Bevel — вид панели (выпуклая, вдавленная или обычная);
- property Style — определяет вывод текста или программное формирование содержимого;
- property Text — текст панели;
- property Width — ширина панели в пикселях.

Компонент TProgressBar предназначен для отображения сведений о ходе какого-либо длительного процесса (в данном случае загрузки документа).

Рассмотрим основные свойства данного класса:

- property Min, property Max — минимальная и максимальная допустимые границы;
- property Orientation — ориентация, горизонтальная (значение trHorizontal) или вертикальная (значение trVertical);
- property Smooth — имеет значение true, если в полосе заполнения отображается сплошная линия, в противоположном случае — сегментированная.

Если требуются более гибкие возможности оформления индикатора, то можно воспользоваться компонентом TGauge с панели Samples (Образцы), который позволяет дополнительно определять цвет индикатора и выводить процент заполнения.

Компонент TOpenDialog предназначен для выбора файла с целью последующего открытия. Приведем основные свойства и события этого класса:

- property DefaultExt — расширение имени, используемое по умолчанию. Добавляется в конец выбранного пользователем имени файла, если расширение не указано явно;
- property FileName — выбранное пользователем имя файла вместе с полным путем поиска;
- property Files — список выбранных имен файлов. В свойстве Options должен быть включен флажок ofAllowMultiSelect;
- property Filter — набор масок, в соответсвии с которыми отбираются имена файлов для отображения в диалоговом окне. Каждая маска состоит из двух частей: названия и шаблона, разделенных символом |. Одному названию могут соответствовать несколько шаблонов. Маски разделяются друг от друга символом |;

- property FilterIndex — номер текущей маски. Нумерация начинается с 1;
- property HistoryList — список ранее выбранных фалов (тип TStrings);
- property InitialDir — текущий каталог, содержимое которого отображается при первом открытии диалогового окна;
- property Options — набор флажков, определяющих работу выбора файлов;
- property Title — заголовок диалогового окна;
- On CanClose — пользователь пытается закрыть диалоговое окно. Обработчик этого события позволяет проконтролировать правильность выбранного или введенного в соответствующее поле окна имени файла и разрешить или запретить закрытие;
- OnFolderChange — пользователь переключился в другой каталог;
- OnIncludeItem — к текущему списку файлов в диалоговом окне будет добавлено новое имя. Обработчик данного события дает возможность отбирать доступные имена по алгоритму, определенному программистом;
- OnSelectionChange — пользователь выбрал новое имя файла в диалоговом окне;
- OnTypeChange — пользователь выбрал новую маску файлов (свойство Filter).

Для реализации адресной строки в данном случае применяется компонент TComboBox. Данный компонент может работать в тех разных режимах, определяемых значением свойства Style:

- csDropDown — в присоединенном поле можно указывать значения, отсутствующие в списке. Свойство MaxLength определяет максимально допустимое число символов, которое можно ввести в это поле (значение 0 указывает на отсутствие ограничений). Текст, введенный пользователем, доступен через свойство Text. Список раскрывающийся;
- csDropDownList — допустим только выбор значений, уже имеющихся в списке. Список раскрывающийся;
- csSimple — отличается от стиля csDropDown только тем, что список не является раскрывающимся.

Свойство CharCase управляет регистром вводимого пользователем текста:

- ecUpperCase — автоматическое преобразование к верхнему регистру;
- ecLowerCase — преобразование к нижнему регистру;
- ecNormal — текст никак не преобразовывается (по умолчанию).

Свойство DropDownCount задает максимальное число элементов, одновременно отображаемых в видимой части списка.

Свойство ItemIndex хранит номер в списке Текущий выбранной строки.

Наиболее важными событиями являются:

- OnChange — пользователь изменил текст в присоединенном поле;
- OnDropDown — список раскрывается.

Разработка любого приложения не ограничивается настройкой свойств компонентов, а также требует написания программного кода, обеспечивающего полноценное взаимодействие этих компонентов. Рассмотрим порядок использования перечисленных компонентов для разработки Web-браузера.

Вначале необходимо разместить на форме все указанные компоненты, как показано на рис. 6.2.

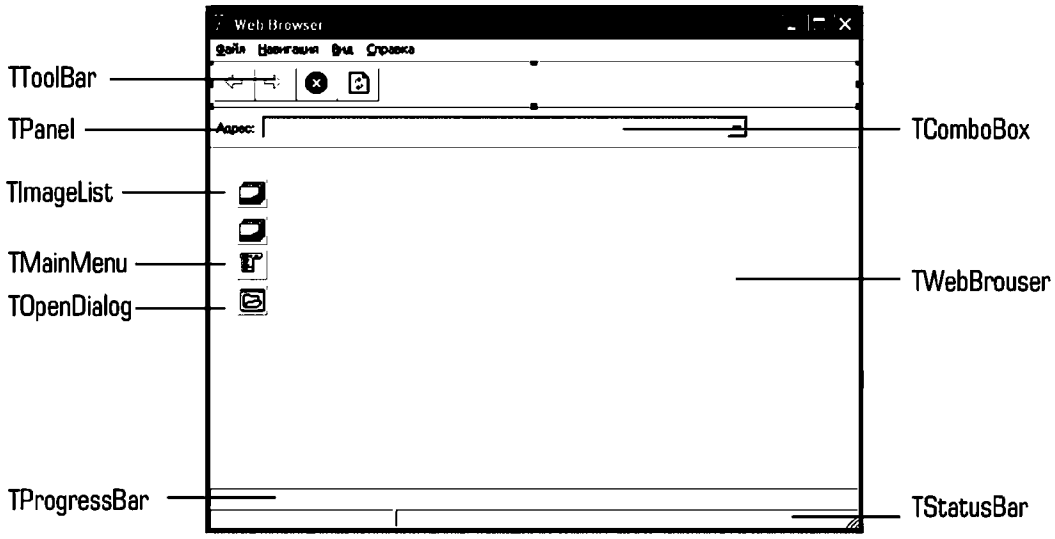

Рис. 6.2. Интерфейс Web-браузера

Целесообразнее вначале настроить главное меню приложения. Для этого с помощью редактора меню создается набор пунктов меню: Файл, Навигация, Вид и Справка. В меню Файл создаются подпункты: Открыть с названием Open и Закрыть — Exit1. В меню Навигация: Назад — Back, Вперед — Forward1, Стоп — Stop, Обновить — Refresh. При этом в свойство Enabled пунктов Назад и Вперед надо сделать равным false. В меню Вид создаются подпункты: Панель инструментов — ToolBar и Строка состояния — StatuaBar. Свойство Checked этих подпунктов делают равным true. В пункте Справка создают один элемент «О программе» с названием About.

Затем производят настройку компонента TToolBar. В данном случае нам понадобятся два экземпляра данного компонента: первый — для создания панели навигации, а второй — для размещения адресной строки. Разместим на первой панели четыре кнопки: Назад — BackBtn, Вперед — ForwardBtn, Остановить — StopBtn и Обновить — Re-

freshBtn. Для настройки изображений кнопок навигации разместим два экземпляра компонента TImageList. В первом с названием NavigatorImages будут находиться изображения кнопок в рабочем состоянии, а во втором — NavigatorHotImages — изображения, появляющиеся при наведении на кнопки курсора мыши. Изображения кнопок можно загрузить из каталога c:\Program Files\Common Files\Borland Shared\Images\Buttons\. После загрузки изображений в Инспекторе объектов в свойствах HotImages и Images необходимо указать соответственно компоненты NavigatorHotImages и NavigatorImages.

На втором компоненте TToolBar размещают компонент TPanel и записывают в его свойстве Caption строку «Адрес:», в свойстве Alignment выбирают значение taLeftJustify (выравнивание по левому краю), а в свойстве BevelOuter — bvNone (без рамки). Затем здесь же размещают компонент TComboBox, которому присваивают имя URLs.

После этого надо разместить на форме компонент TWebBrouser и указать в его свойстве Align значение alClient (во всю клиентскую область формы).

Затем размещают компонент TProgressBar, в свойстве Align которого указывают значение alBottom (в нижней части формы), а в свойстве Max — значение 10 000.

После на форму вытаскивают компонент TStatusBar. В его свойстве AutoHint устанавливают значение true для того, чтобы в нем отображались значения свойств Hint (Подсказка) расположенных на форме компонентов. Кроме того, нужно создать в данном компоненте две панели, что выполняется двойным щелчком по компоненту.

Последний компонент, который нужно разместить, —TOpenDialog. В его свойстве Filter записываем строку HTML-файлы|*.html;*.htm|Все файлы.

На этом настройка свойств компонентов заканчивается, и необходимо программным образом обработать логику работы Web-браузера.

Вначале опишем глобальные переменные:

```
HistoryIndex: Integer;//номер ресурса
HistoryList: TStringList;//список просмотренных ресурсов
```

После этого надо в процедуре создания формы FormCreate прописать следующие строки:

```
HistoryIndex:= -1;
HistoryList:= TStringList.Create;
OpenDialog1.InitialDir:= GetCurrentDir;
URLs.Text:= 'http://www.borland.com';
FindAddress;
```

а в процедуре уничтожения формы FormDestroy записать:

```
HistoryList.Free;
```

Затем пропишем процедуру загрузки документа:

```
procedure TMainForm.FindAddress;
begin
WebBrowser1.Navigate(WideString(Urls.Text));
end;
```

Предварительно в разделе private класса TMainForm (главная форма нашего Web-браузера) необходимо прописать процедуру FindAddress:

```
type
TMainForm = class(TForm)
. . .
private
procedure FindAddress;
end;
```

Теперь нужно прописать обработчики выбора (нажатий) пунктов меню.

Пункт меню Открыть:

```
procedure TMainForm.OpenClick(Sender: TObject);
begin
        if(OpenDialog1.Execute)then
        begin
            URLs.Text:= OpenDialog1.FileName;
            FindAddress;
        end;
end;
```

Пункт меню Закрыть:

```
procedure TMainForm.Exit1Click(Sender: TObject);
begin
 Close;
end;
```

Пункт меню Назад:

```
procedure TMainForm.BackClick(Sender: TObject);
begin
    URLs.Text:=HistoryList[HistoryIndex-1];
    FindAddress;
end;
```

Пункт меню Вперед:

```
procedure TMainForm.Forward1Click(Sender: TObject);
```

```
begin
    URLs.Text := HistoryList[HistoryIndex+1];
    FindAddress;
end;
```

В обработчике щелчка по пункту меню Стоп нужно записать:

```
WebBrowser1.Stop;
```

Пункт меню Обновить:

```
procedure TMainForm.RefreshClick(Sender: TObject);
begin
    FindAddress;
    if HistoryList.Count > 0 then
        Forward1.Enabled:=HistoryIndex<HistoryList.Count-1
    else
        Forward1.Enabled := False;
end;
```

В пункте меню Вид имеет два подпункта — Панель инструментов и Строка состояния, которые управляют отображением соответственно Панели инструментов и Строки состояния:

```
procedure TMainForm.ToolbarClick(Sender: TObject);
begin
        MainForm.Toolbar.Checked:=not(MainForm.Toolbar.Checked);
        Toolbar1.Visible:=MainForm.Toolbar.Checked;
        Toolbar2.Visible:=MainForm.Toolbar.Checked;
end;
procedure TMainForm.StatusbarClick(Sender: TObject);
begin
        MainForm.Statusbar.Checked:=not(MainForm.Statusbar.Checked);
        StatusBar1.Visible:=MainForm.StatusBar.Checked;
end;
```

Пункт меню О программе вызывает диалоговое окно, содержащее сведения о программе:

```
AboutBox.ShowModal;
```

После прописки всех обработчиков выбора пунктов меню необходимо их связать с соответствующими кнопками навигации на Панели инструментов.

Далее нужно решить более сложную задачу — реализовать логику работы компонента TWebBrouser, которая связана с обработкой необходимых событий этого компонента.

Сначала надо прописать обработчик события перехода к новому ресурсу:

```
procedure TMainForm.WebBrowser1BeforeNavigate2(Sender: TObject;
    const pDisp: IDispatch; var URL, Flags, TargetFrameName,
    PostData, Headers: OleVariant; var Cancel: WordBool);
var
    NewIndex: Integer;
begin
    NewIndex := HistoryList.IndexOf(URLs.Text);
    if NewIndex = -1 then
        HistoryIndex:=HistoryList.Add(URLs.Text)
    else
        HistoryIndex:= NewIndex;
    NewIndex:=URLs.Items.IndexOf(URLs.Text);
    if NewIndex = -1 then
            URLs.Items.Insert(0,URLs.Text)
        else
            URLs.Items.Move(NewIndex, 0);
    URLs.Text:= URLs.Items.Strings[0];
    Statusbar1.Panels[1].Text:= URLs.Text;
    if HistoryList.Count>0 then
    begin
        BackBtn.Enabled:=HistoryIndex>0;
        Back.Enabled:=HistoryIndex>0;
    end
    else
    begin
        BackBtn.Enabled:=false;
        Back.Enabled:=false;
    end;
    if HistoryIndex<HistoryList.Count-1 then
    begin
        Forward1.Enabled:=true;
        ForwardBtn.Enabled:=true;
    end
    else
    begin
        Forward1.Enabled:=false;
        ForwardBtn.Enabled:=false;
    end;
end;
```

Далее обрабатывается событие начал загрузки документа:

```
procedure TMainForm.WebBrowser1DownloadBegin(Sender: TObject);
begin
    StatusBar1.Panels[0].Text:='Загрузка ресурса;
    ProgressBar1.Position:=0;
    ProgressBar1.Visible:=true;
    StopBtn.ImageIndex:= 4;
end;
```

В обработчике события ProgressChange компонента TWebBrouser записывается:

```
ProgressBar1.Position:=Progress;
```

В обработчике события успешного завершения перехода к новому ресурсу прописывается следующее:

```
StatusBar1.Panels[0].Text:='Готово';
ProgressBar1.Visible:=false;
```

В обработчике события завершения загрузки документа записывается:

```
StopBtn.ImageIndex:= 2;
```

После выполнения всех указанных инструкций по настройке свойств компонентов и правильной записи обработчиков процедур должен получиться полноценный Web-браузер, позволяющий загружать и просматривать Web-ресурсы.

6.3. Создание распределенных многопользовательских приложений

Примером использования архитектуры клиент–сервер может являться приложение, работающее по протоколу TCP/IP. Серверная часть запускается на компьютере, подключенном к сети, и выполняет обработку информации, поступающей от клиентских программ. Типичный пример такого приложения — форумы (интернет-чат), позволяющие множеству людей общаться друг с другом в реальном времени. Каждый пользователь форума имеет специальную клиентскую программу, в которую заложены координаты определенного сервера и которая после запуска напрямую с ним соединяется. Когда мы вводим в своей программе строку и щелкаем на кнопке «Отправить», эта строка посылается серверу, а он ее рассылает другим клиентским программам, подключенным к нему в данный момент. Получив от сервера строку, клиентская программа выводит ее в своем окне. Таким образом, вся работа сервера заключается в рассылке поступающей информации всем подключенным к нему пользователям.

Связь клиентской программы с серверной осуществляется с помощью технологии сокетов (sockets), представляющих собой виртуальные аналоги разъемов. Сервер имеет набор таких разъемов, к которым подключаются клиентские программы, что позволяет организовывать прямую связь между двумя компьютерами в сети.

В системе Windows поддержка технологии связи программ TCP/IP с помощью сокетов реализована в виде программного интерфейса Windows socket API, для работы с которым требуется установка на компьютере библиотеки WinSock.DLL, распространяемой бесплатно.

Сокеты бывают трех видов: клиентские, слушающие и серверные. *Клиентские сокеты* устанавливают связь с удаленным компьютером (сервером) на основании его IP-адреса, после чего могут обмениваться с ним информацией. *Слушающие сокеты* выполняют пассивную роль, непрерывно опрашивая специальный порт компьютера, предназначенный для приема/передачи данных. Как только на него приходит запрос от нового подключающегося к серверу клиентского сокета, этот запрос становится в очередь. Когда серверный сокет освободится от выполнения текущей работы, он на основании запросов из очереди формирует связь с новым клиентским сокетом, а слушающий сокет освобождается и снова ждет поступления запросов в порт. *Серверные сокеты* обмениваются данными с клиентскими сокетами по уже установленным с помощью слушающих сокетов соединениям.

Для того чтобы клиентские программы могли установить соединение с сервером, им надо указать два параметра такого соединения: адрес компьютера в сети и незанятый порт компьютера, на который будет посылаться информация. Сервер будет постоянно прослушивать этот порт. Как только на него поступят данные, он сразу же их обработает. Серверному объекту надо выделить свой собственный порт. Обычно на компьютере почти все номера портов свободны за исключением нескольких сотен, отведенных на системные нужды. Просмотреть список портов можно в файле services (без расширения), расположенном в каталоге Windows.

Этот файл организован следующим образом. Сначала указывается условное название службы (service), которая привязана к данному порту, затем номер порта, за которым через символ «/» следует название поддерживаемого протокола. Далее следует необязательный псевдоним службы и после символа # — комментарий. Все службы располагаются в порядке возрастания номеров портов. При добавлении нового порта надо придерживаться описанного правила записи и соответствующих отступов. Например: testport 777/tcp. После внесения изменений в файл services систему Windows надо перезагрузить, чтобы новый порт был зафиксирован в системе.

Для организации связи по протоколу TCP/IP одного номера порта недостаточно. Клиентской программе надо еще знать, на каком компьютере в сети запущена нужная программа-сервер. Координаты сервера определяются его IP-адресом.

Для разработки распределенных многопользовательских приложений в последних версиях Delphi предусмотрены компоненты, размещенные на панелях Indy (Internet Direct): IndyClients, IndyServers, IndyIntercepts, Indy I/O Handlers, IndyMisc. Данные компоненты обладают упрощенным интерфейсом, позволяющим разрабатывать клиент-серверные программы для Internet. При этом они поддерживают две операционные системы — Windows и Linux и применяются в Delphi, C++ Builder и Kylix.

Определим назначение компонентов, расположенных на каждой из перечисленных вкладок:

- IndyClients — содержит компоненты, ответственные за поддержку Internet-протоколов на клиентских местах;
- IndyServers — компоненты, предназначенные для организации работы Internet-серверов;
- Indy I/O Handlers — компоненты, позволяющие работать с идентификаторами ввода/вывода при использовании сокетов;
- IndyIntercepts — компоненты, реализующие функции сжатия и протоколирования;
- IndyMisc — вспомогательные компоненты.

Основными из рассматриваемых компонентов являются: TIdTCPServer (компонент TCP-сервер), TIdTCPClient (компонент TCP-клиент), TIdThread (компонент Indy-поток).

TIdTCPServer является базовым для всех серверных систем, поддерживающих протокол TCP. Он наследует свойства и методы класса TIdComponent — базового класса для всех компонентов Indy, на основе которых создаются клиентские и серверные приложения. Каждый TCP-сервер способен поддерживать несколько соединений с удаленными машинами. Сервер контролирует и прослушивает состояние различных портов удаленных приложений через клиентские соединения, которые вместе с портами указываются в коллекции Bindings (тип TIdSocketHandles) в визуальном редакторе. Если сервер контролирует состояние портов компьютера, на котором он запущен, то IP-адрес этого компьютера можно не указывать. Достаточно лишь ввести номер порта.

TIdTCPClient — наследует свойства и методы класса TIdComponent, которые определяют конкретное TCP-соединение. Данный компонент может быть использован при создании различных клиентских приложений.

TIdThread — наследник стандартного класса TThread, используемый в многопотоковых компонентах Indy. Дополнен функциональными возможностями по контролю за состоянием потоков.

Для создания простейшей распределенной серверной системы, работающей по протоколу TCP, достаточно воспользоваться компонентами TIdTCPClient и TIdTCPServer. Функционирование сервера должно сводиться к приему текстовых сообщений от множества программ-клиентов и пересылке им ответных подтверждений о приеме сообщения.

Первоначально производят настройку TIdTCPServer, указывая в свойстве Bindings подходящий порт для прослушивания. Работа сервера реализуется в обработчике события OnExecute, которое возникает, когда один из клиентов пытается установить связь с сервером. В качестве параметра обработчику передается объект класса TIdPeerThread. Такой объект создается автоматически для каждой установленной клиент-серверной TCP-связи, порождающей в свою очередь отдельный выполняющийся на сервере поток. В простейшем случае этот обработчик будет содержать в себе три оператора:

- function CurrentreadBuffer: string — функция считывания из буфера информации, полученной от клиентского приложения;
- procedure WriteLn (const AOut: string) — процедура отсылки информации клиентской программе;
- procedure Disconnect — закрытие текущего соединения.

Закрытие сформированного соединения очень важно, так как для каждого такого соединения на сервере запускается отдельный поток. При большом количестве незакрытых соединений существенно возрастает нагрузка на процессор.

Организация работы клиентского приложения сводится к настройке компонента TIdTCPClient. Для этого указывается IP-адрес сервера в свойстве Host, а также значение свойства Port. Взаимодействие с сервером организуется путем вызова следующих методов данного компонента:

- procedure Connect (const ATimeout: Integer) — открытие соединения;
- procedure Write (const AOut: string) — пересылка информации в виде текстовой строки серверу;
- function ReadLn (ATerminator: string; const ATimeout: Integer; AMaxLineLength: Integer): string — получение информации от сервера;
- procedure Disconnect — закрытие текущего соединения.

Рассмотрим пример разработки приложения, реализующего технологию клиент-серверного взаимодействия на основе указанных компонентов системы Borland Delphi. Касаться настройки компонентов мы не будем, а сразу перейдем к технологии написания программного кода.

Для разработки серверной части приложения потребуются компонент TIdTCPServer (будет выполнять функции сервера), два эк-

зкземпляра компонента TButton (кнопки для активизации сервера и закрытия приложения) и компонент TListBox (для отображения передаваемых сообщений) (рис. 6.3).

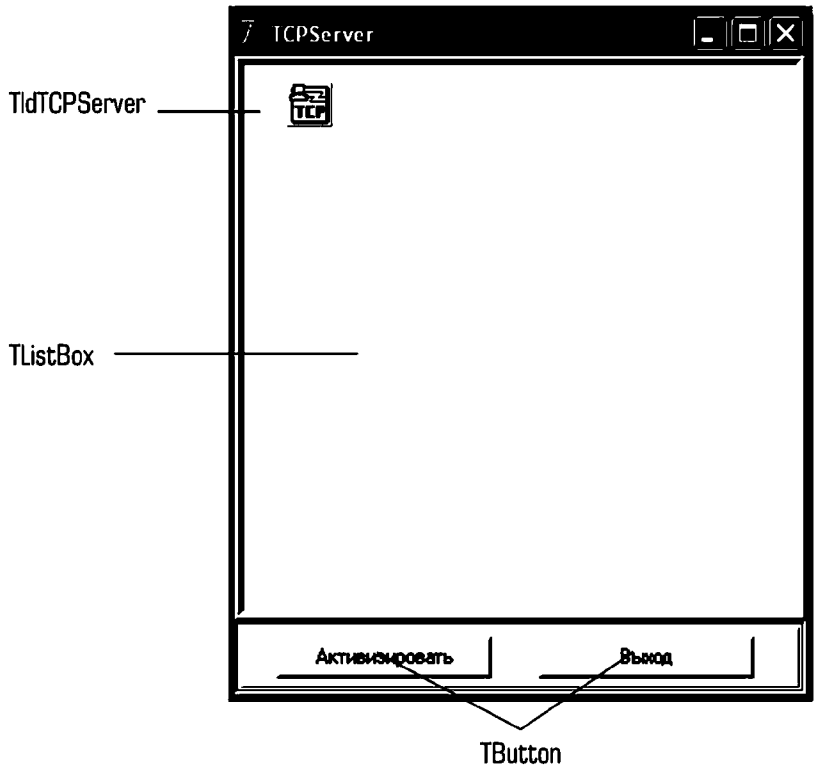

Рис. 6.3. Интерфейс серверной части TCP-приложения

Вначале нужно определить глобальную переменную, контролирующую, активен сервер в данный момент или нет:

```
ActiveCon:boolean;
```

И в процедуре создания формы прописать строку:

```
ActiveCon:=false;
```

Теперь надо прописать процедуру нажатия кнопки активизации сервера:

```
procedure TServer.ActivateBtnClick(Sender: TObject);
begin
    try
        ActiveCon:=not(ActiveCon);
        ListBox1.Items.Clear;
        IdTCPServer1.Threads.Clear;
        IdTCPServer1.Active:=ActiveCon;
        if(not(ActiveCon))then
```

```
                ActivateBtn.Caption:='Активизировать'
            else
                ActivateBtn.Caption:='Отключить';
        except
            ShowMessage('Нарушение работы сервера!');
        end;
    end;
```

Затем в обработчике события закрытия формы нужно записать:

```
    if(ActiveCon)and(MessageDlg(
    'Сервер активизирован. Произвести отключение?',mtConfirmation,
    mbYesNoCancel,0)=mrYes)then
        ActivateBtnClick(Sender);
    if(not(ActiveCon))then
    begin
        IdTCPServer1.Active:=false;
        Action:=caFree;
    end
    else
        Action:=caNone;
```

В обработчике нажатия кнопки закрытия приложения прописывается процедура закрытия формы Close.

Вся логика работы серверного компонента должна быть реализована в обработчике события OnExecute:

```
    procedure TServer.IdTCPServer1Execute(AThread: TIdPeerThread);
    var
        s:string;
    begin
        try
            with AThread.Connection do
            begin
                s:=CurrentReadBuffer;
                if(s='Connect')then
                        s:='Произведено подключение';
                if(s='Disconnect')then
                        s:='Произведено отключение';
                if(s<>'')then
                begin
                        s:=IntToStr(AThread.Handle)+'> '+s;
                        ListBox1.Items.Add(s);
```

```
                WriteLn(s);
            end;
        end;
    except
        ShowMessage('Ошибка взаимодействия!');
    end;
end;
```

Рассмотрим теперь порядок реализации клиентской части TCP-приложения. Для этого необходимы компонент TIdTCPClient (будет реализовывать работу клиентской части), три экземпляра компонента TButton (кнопки подключения к серверу, отправки сообщений и закрытия формы приложения), компонент TListBox (для вывода на экран переданных сообщений) и компонент TEdit (для записи отправляемой строки) (рис. 6.4).

Вначале нужно прописать глобальную переменную:

```
ActiveCon:boolean;
```

После этого в обработчике события создания формы нужно ей присвоить значение false.

Затем необходимо прописать процедуру, связанную с событием нажатия кнопки подключения к серверу:

```
procedure TClient.ConnectBtnClick(Sender: TObject);
begin
    try
        ActiveCon:=not(ActiveCon);
        ListBox1.Items.Clear;
        if(ActiveCon)then
        begin
            IdTCPClient1.Connect;
            IdTCPClient1.Write('Connect');
            SendEdit.Text:=IdTCPClient1.ReadLn();
            ConnectBtn.Caption:='Отключиться';
        end
        else
        begin
            IdTCPClient1.Write('Disconnect');
            SendEdit.Text:=IdTCPClient1.ReadLn();
            IdTCPClient1.Disconnect;
            ConnectBtn.Caption:='Подключиться';
        end;
    except
        on EIdSocketError do
        begin
```

```
                ShowMessage('Нарушена работа сервера!');
                ActiveCon:=false;
            end;
        else
            ShowMessage('Неизвестная ошибка!');
        end;
    end;
```

После этого в обработчике события закрытия формы надо записать:

```
if(ActiveCon)and(MessageDlg(
'Активизировано соединение. Произвести отключение?',
mtConfirmation,mbYesNoCancel,0)=mrYes)then
    ConnectBtnClick(Sender);
if(not(ActiveCon))then
begin
    IdTCPClient1.Disconnect;
    Action:=caFree;
end
else
    Action:=caNone;
```

В обработчике события нажатия кнопки закрытия приложения прописывается процедура закрытия формы Close.

И последнее, что нужно прописать, — это процедура отправки сообщений на сервер, связанная с событием нажатия соответствующей кнопки:

```
procedure TClient.SendBtnClick(Sender: TObject);
begin
    try
        IdTCPClient1.Write(SendEdit.Text);
        ListBox1.Items.Add(IdTCPClient1.ReadLn());
    except
        ShowMessage('Нарушена работа сервера!');
        ActiveCon:=false;
        ConnectBtn.Caption:='Подключиться';
    end;
end;
```

Таким образом реализуется простейшее приложение, основанное на технологии клиент–сервер с использованием компонентов TIdTCPServer и TIdTCPClient.

Реализация работы рассмотренной системы возможна с использованием компонентов TServerSocket и TClientSocket (отсутствующих

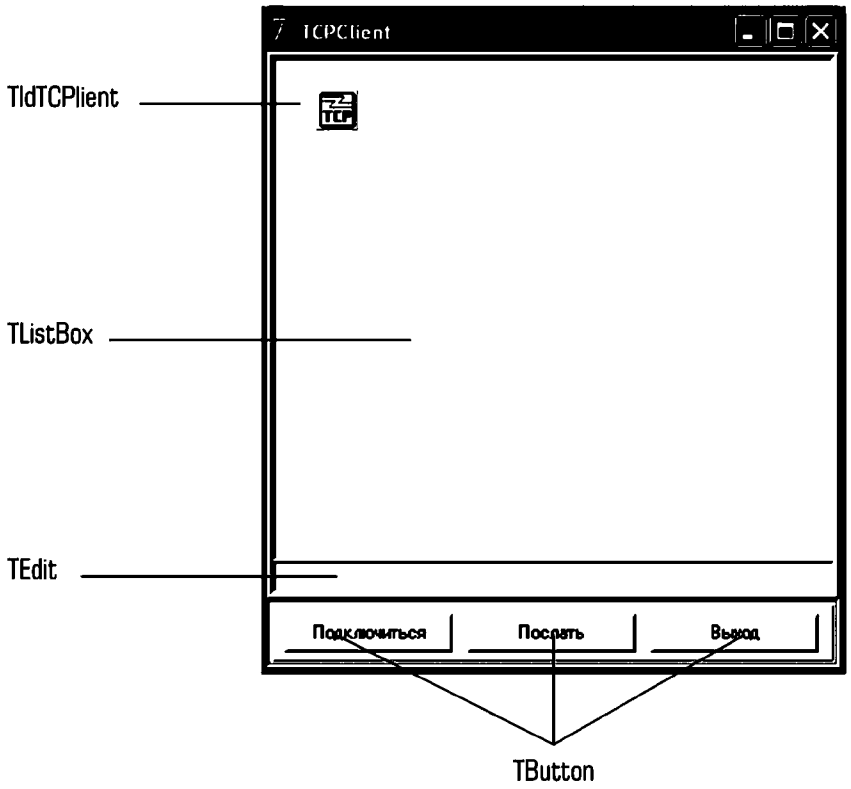

Рис. 6.4. Интерфейс клиентской части TCP-приложения

на стандартной палитре компонентов), настройка которых более сложна по сравнению с указанными классами. Чтобы получить к ним доступ, их необходимо поместить на палитре компонентов, для этого надо подключить новую библиотеку компонентов. Это делается выбором пункта меню системы Delphi Component □ Install Packages. В открывшемся диалоговом окне (рис. 6.5) нужно нажать кнопку Add.

Затем следует выбрать в каталоге размещения программы в папке Bin файл библиотеки dclsockets70.bpl. После чего на вкладке Internet появятся два компонента TServerSocket и TClientSocket.

Процесс реализации рассмотренной системы с использованием этих компонентов аналогичен тому, как это было сделано с помощью TCP- компонентов, поэтому на их применении мы останавливаться не будем.

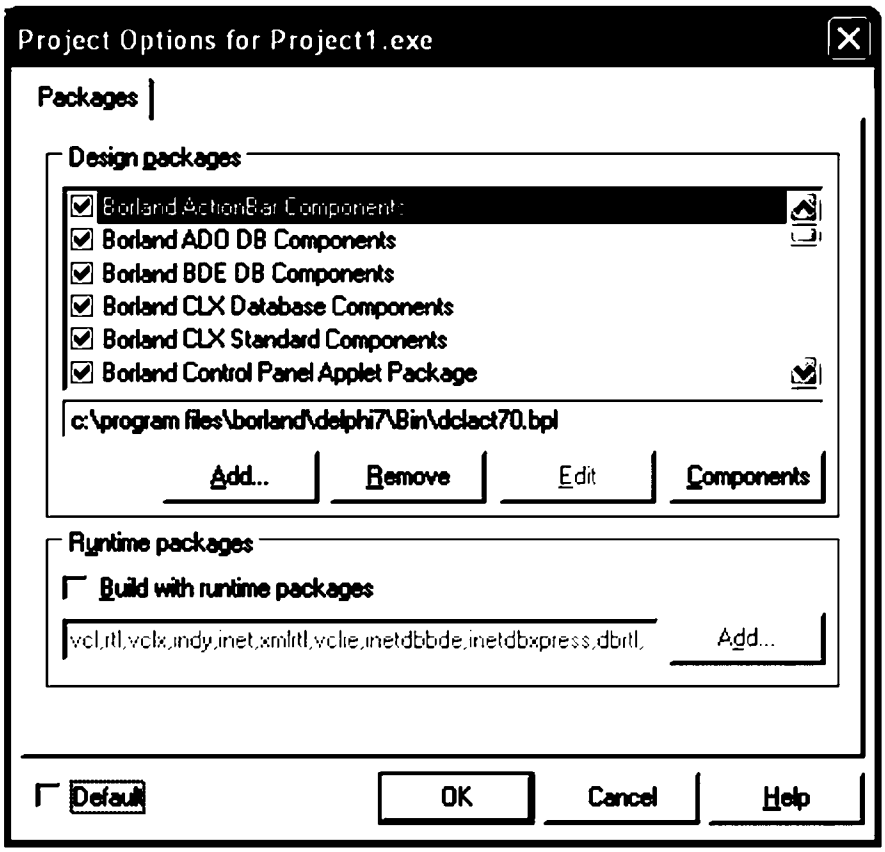

Рис. 6.5. Окно инсталляции библиотек компонентов

Вопросы для самопроверки

1. Опишите возможности системы Delphi по разработке Web-приложений.
2. Какие компоненты можно использовать для простейшей реализации Web-браузера?
3. Перечислите основные свойства, методы и процедуры компонента TWebBrouser.
4. Что такое распределенные многопользовательские приложения?
5. Опишите назначение и порядок использования компонентов TIdTCPServer, TIdTCPClient, TIdThread.

Тесты

1. С помощью какого компонента реализуется панель инструментов?
   А. TToolBar.
   Б. TProgressBar.
   В. TComboBox.
   Г. TStatusBar.

2. На какой вкладке Палитры компонентов системы Delphi располагается компонент TWebBrouser?
   А. WebServices.
   Б. InternetExpress.
   В. Internet.
   Г. WebSnap.

3. Какое событие компонента TWebBrouser генерируется после полной загрузки документа?
   А. OnDocumentComplete.
   Б. OnDownloadBegin.
   В. OnDownloadComplete.
   Г. OnNavigateComplete2.

4. На какой вкладке Indy располагаются компоненты, реализующие функции сжатия и протоколирования?
   А. IndyClients.
   Б. Indy I/O Handlers.
   В. IndyIntercepts.
   Г. IndyMisc.

5. На какой вкладке Indy располагаются компоненты, реализующие вспомогательные функции?
   А. IndyMisc.
   Б. IndyClients.
   В. IndyIntercepts.
   Г. Indy I/O Handlers.

Глава 7. СОЗДАНИЕ ПРИЛОЖЕНИЙ WEB-СЕРВЕРА

7.1. Разработка расширений Web-серверов

В задачу Web-сервера входит обеспечение работы Web-узлов Internet (или локальной сети на базе протокола TCP/IP), чтобы множество пользователей одновременно могли обращаться к одним и тем же страницам HTML. Web-сервер рассылает копии этих страниц клиентским программам-браузерам. Современные подходы к созданию приложений Internet требуют активного взаимодействия с пользователем и обеспечения тесной обратной связи с ним по аналогии с обычными офисными приложениями. Для этого необходимо, чтобы содержание HTML-страницы, отправляемой пользователю, не было простой копией файла на сервере, а создавалось динамически, программным путем, в зависимости от действий пользователя.

Web-сервер не занимается генерацией страниц. При поступлении запроса на выполнение нестандартной операции Web-сервер загружает в виде отдельного процесса один из специальных модулей, который выполняет создание нужной HTML-страницы и затем передает ее обратно Web-серверу, который отсылает ее соответствующему клиенту. Такие модули пишутся на самых разных языках программирования и могут работать под управлением самых разных операционных систем.

Рассмотрим разработку расширений Web-серверов на примере CGI-сценариев. При создании CGI-приложения решаются две основные задачи: разработка Web-интерфейса (с использованием языка HTML) и разработка собственно приложения (с использованием языка программирования).

Для запуска CGI-приложения пользуются двумя способами:

- щелчком на кнопке SUBMIT формы (<FORM action="/cgi-bin/test.exe" metod="GET"><INPUT type="SUBMIT"></FORM>);
- щелчком на ссылке (<A href="/cgi-bin/test.exe">Run CGI</A>).

Наиболее часто используется первый способ, так как он позволяет организовать интерактивное взаимодействие с пользователем.

CGI-сценарии представляют собой традиционные консольные приложения, поэтому для их разработки не требуется никаких специальных средств. Для вывода результатов выполнения CGI-сценария используются обычные процедуры вывода на консоль. На языке Object Pascal это процедуры write и writeln. Однако выводимая таким образом информация должна соответствовать протоколу

HTTP. Первая строка заголовка (HTTP/1.0 200 OK) формируется Web-сервером. Информационные же поля заголовка и тела ответа должны формироваться CGI-приложением. В большинстве случаев достаточно одного поля — Content-type. При этом заголовок и тело ответа должны разделяться пустой строкой.

Пример формирования HTML-документа в результате выполнения CGI-сценария, написанного в виде консольного приложения на языке Pascal:

```
cal
writeln('Content-Type: text/html');
writeln;
writeln('<HTML>');
writeln('<HEAD>');
writeln('<TITLE> Пример CGI-приложения'</TITLE>);
writeln('</HEAD>');
writeln('<BODY>');
writeln('<H2 align=center>Hello!'</H2>);
writeln('</BODY>');
writeln('</HTML>');
```

Для вызова рассмотренного CGI-сценария необходимо разработать HTML-документ, из которого будет производиться его вызов:

```
<HTML>
  <HEAD>
   <TITLE>Пример CGI-приложения</TITLE>
  </HEAD>
   <BODY>
     <A href="/cgi-bin/consol.exe">Run CGI</A><Br><Br>
     <FORM method="get" action="/cgi-bin/consol.exe">
   <INPUT type="submit">
 </FORM>
</BODY>
</HTML>
```

Для выполнения Web-приложений необходимо наличие на компьютере специальной программы, реализующей функции Web-сервера, заключающиеся в организации взаимодействия с другими приложениями по протоколу HTTP. Такое приложение может быть установлено на любой локальный компьютер, даже не подключенный к сети.

Обращение к соответствующему расширению сервера возможно после его размещения в специальном каталоге Web-сервера, предназначенном для этого.

7.2. Основные инструменты организации взаимодействия с Web-клиентом и их использование

Ввод данных пользователем производится средствами интерфейса, реализованного с помощью Web-формы. Щелчок на кнопке submit, расположенной на форме, вызывает CGI-сценарий, указанный в теге <FORM> с помощью атрибута action. Перед запуском сценария сервер формирует строку параметров. Содержимое этой строки будет определяться интерактивными элементами, расположенными на форме. Каждый из этих элементов имеет идентификатор, задаваемый атрибутом name, и значение, определяемое атрибутом value или последовательностью символов, введенных пользователем. Из идентификаторов элементов управления и их значений формируется строка параметров следующего формата:

```
идентификатор1=значение1&идентификатор2=значение2...
```

Каждый параметр этой строки соответствует одному элементу управления и представляет собой имя элемента и его значение, разделенные знаком равенства. Параметры, относящиеся к разным элементам управления, разделяются в строке символами &.

Если символ "=" или "&" входит в состав имени или значения элемента управления, то он кодируется последовательностью из трех знаков: первый знак — %, за ним следуют две шестнадцатеричные цифры, являющиеся кодом символа (например, символ "=" кодируется как %3D, а символ "&" — как %26). Помимо этих двух знаков трехсимвольными последовательностями обычно кодируются все знаки, за исключением латинских букв, цифр и символа пробела. Символ пробела заменяется символом "+".

Полученную строку параметров следует декодировать, для чего используют следующую последовательность действий:

1) разделение строки на пары: идентификатор_N=значение_N;

2) выделение в каждой паре идентификатора и значения;

3) замена символа "+" в каждом идентификаторе и каждом значении пробелом;

4) преобразование каждой трехсимвольной последовательности, начинающейся со знака %, в символ ASCII.

Решение этой задачи можно реализовать в виде следующего программного кода, иллюстрирующего процесс преобразования определенного параметра и его значения:

```
var InParams: string;
// Читаем переданные параметры из переменной окружения
procedure InitParams;
 var SS: string;
begin
```

```
 SetLength(SS,10000);
 GetEnvironmentVariable('QUERY_STRING',@SS[1],2000);
 InParams:=PChar(@SS[1]);
end;
// Функция переводит шестнадцатеричный символ в число
function HexToInt(CH: char): integer;
begin
        Result:=0;
        case CH of
             '0'..'9': Result:=Ord(CH)-Ord('0');
             'A'..'F': Result:=Ord(CH)-Ord('A')+10;
             'a'..'f': Result:=Ord(CH)-Ord('a')+10;
        end;
end;
// Преобразует символы, записанные в виде %2B к правильному виду
function Decode(Value: string): string;
var
        I, L: integer;
begin
        Result:='';
        L:=0;
        for I:=1 to Length(Value) do
        begin
             if(Value[I]<>'%')and(Value[I]<>'+')and(L<1)then
                  Result:=Result+Value[I];
             else
             begin
                  if(Value[I]='+')then
                       Result:=Result+' '
                  else if(Value[I]='%')then
                  begin
                       L:=2;
                       if(I<Length(Value)-1)then
        Result:=Result+Chr(HexToInt(Value[I+1])*16+HexToInt(
                  Value[I+2]));
             end
             else
                  Dec(L);
             end;
```

```
        end;
    end;
    // Возвращает значение параметра, заданного в Name
    function ParamByName(Name: string): string;
    var
        SS, ST: string;
    K: integer;
    begin
            Result:='';
            SS:=InParams;
            while Length(SS)<>0 do
            begin
                K:=Pos('&',SS);
                if(K<>0)then
                begin
                    ST:=Copy(SS,1,K-1);
                    SS:=Copy(SS,K+1,10000);
                end
                else
                begin
                    ST:=SS;
                    SS:='';
                end;
                K:=Pos('=',ST);
                if(K<>0)then
                    if(Name=Copy(ST,1,K-1))then
      Result:=Decode(Copy(ST,K+1,6000));
        end;
    end;
```

Строка параметров может передаваться Web-серверу либо методом GET, либо методом POST. Метод передачи данных определяется значением атрибута method в теге <FORM>:

- при использовании метода GET строка параметров передается вместе с URL-адресом вызываемого CGI-приложения, а для разделения URL-адреса и строки параметров используется символ "?"; при этом строка параметров передается CGI-приложению через переменную окружения QUERY_STRING;
- в случае использования метода POST строка параметров передается в теле HTTP-запроса; в данном случае данные передаются CGI-приложению через стандартный поток ввода консольной

программы, и длина строки в этом случае может быть определена через переменную окружения CONTENT_LENGTH.

Для получения строки параметров можно использовать функцию Win32 API, возвращающую значение переменной окружения с заданным именем:

```
function GetEnviromentVariable(lpName: PChar; lpBuffer: PChar; nSize:
DWORD): DWORD,
```

где

- lpName — имя переменной окружения;
- lpBuffer — строка PChar, в которую будет занесено значение указанной переменной окружения;
- nSize — длина строки lpBuffer.

Значение, возвращаемое функцией GetEnviromentVariable, равно нулю в том случае, если переменная окружения не определена.

Фрагмент программы, иллюстрирующий считывание строки параметров при передаче ее методом GET, может выглядеть таким образом:

```
var
    buff: PChar;
    st: String;
...
begin
    GetMem(buff, 200);
    GetEnviromentVariable('QUERY_STRING', buff, 200);
    St:=StrPas(buff);
    FreeMem(buff);
...
end;
```

В случае с методом POST программный код изменяется, так как следует выполнять считывание именно такого количества символов, какое содержится в передаваемой строке — попытка считать больше символов приведет к «зависанию» программы. Если же считать не все символы, то часть информации будет потеряна.

Для реализации процедуры считывания данных из стандартного потока ввода проще использовать стандартную процедуру read языка Pascal. Количество символов, которые требуется считать, передается через переменную окружения CONTENT_LENGTH. При этом процедура будет выглядеть таким образом:

```
var
    buff: PChar;
    ContentLength,i: Integer;
    St: String;
```

```
    C: Char;
...
begin
    GetMem(buff, 50);
    GetEnviromentVariable('CONTENT_LENGTH', buff, 50);
    ContentLength:=StrToInt(StrPas(buff));
    FreeMem(buff);
    St:=' ';
    for i:=1 to ContentLength do
    begin
        Read(C);
        St:=St+C;
    end;
...
end;
```

При разработке CGI-сценариев можно использовать и ряд других переменных окружения, через которые Web-сервер передает дополнительную информацию о пользователе, запустившем сценарий. Ряд переменных окружения, содержащих наиболее важную информацию, приведен в табл. 7.1.

Таблица 7.1

Переменные окружения

НАЗВАНИЕ	ОПИСАНИЕ
1	2
PATH_INFO	Строка параметров, расположенная в запросе после имени сценария, но до данных запроса
REMOTE_HOST	Имя компьютера, пославшего запрос
REMOTE_ADDR	IP-адрес компьютера, пославшего запрос
GATEWAY_INTERFACE	Наименование интерфейса, по которому был вызван скрипт (CGI/1.1)
SCRIPT_NAME	Выполняемый файл скрипта (/cgi-bin/cgi_t.exe)
REQUEST_METHOD	Метод передачи параметров скрипту (GET или POST)
HTTP_ACCEPT	Описывает, результат какого типа может быть принят клиентом: text/html — html-документ image/gif — картинка gif */* — любой и т.д. Может передавать несколько типов через запятую
HTTP_ACCEPT_CHARSET	Может содержать кодовую страницу, в которой клиент хотел бы получить результат
HTTP_ACCEPT_ENCODING	Может содержать тип кодировки, поддерживаемый клиентом (например, gzip-результат может быть сжат gzip'ом)
HTTP_ACCEPT_LANGUAGE	Может содержать язык, на котором клиент желает получить результат (ru — русский)

Окончание табл. 7.1

1	2
HTTP_HOST	Название хоста, к которому обратился клиент
HTTP_REFERER	Содержит URL страницы, ссылающейся на скрипт
HTTP_USER_AGENT	Идентификатор клиента, пославшего запрос
QUERY_STRING	Если параметры передаются скрипту методом GET, то здесь содержится строка параметров запроса
SERVER_SOFTWARE	Идентификатор Web-сервера
SERVER_NAME	Название Web-сервера, вызвавшего скрипт
SERVER_PROTOCOL	Протокол, по которому был получен запрос (HTTP/1.0, HTTP/1.1, ...)
SERVER_PORT	Порт сервера, по которому был принят запрос
CONTENT_LENGTH	Количество символов, содержащихся в строке параметров
USER_NAME	Имя пользователя
USER_PASSWORD	Пароль пользователя
AUTH_TYPE	Тип авторизации
HTTP_X_FORWARDED	Может содержать название прокси-сервера, через который был произведен запрос
HTTP_X_FORWARDED_FOR	Может содержать IP-адрес компьютера, пославшего запрос через прокси-сервер (эту переменную устанавливают не анонимные прокси-серверы)

Перед получением строки параметров обычно следует проверить, какой метод передачи информации использован. Это обеспечит правильность считывания передаваемой информации.

7.3. Разработка Web-приложений специальными средствами Delphi

Разработка Web-приложений возможна с использованием конструкций различных языков программирования, но использование специализированных средств позволяет в значительной степени упростить разработку Web-приложений.

Система Delphi позволяет создавать пустые модули Web-приложений. Для вызова мастера создания модуля пользуются командой File → New и выбирают значок Web Server Application. При этом будет предложено выбрать тип будущего Web-приложения (рис. 7.1):

- ISAPI/NSAPI Dynamic Link Library — создание библиотеки (.DLL), автоматически загружаемой Web-сервером по мере необходимости, данные передаются с помощью специального программного интерфейса;
- CGI Stand-along executable — создание серверного консольного приложения, передача данных осуществляется напрямую;

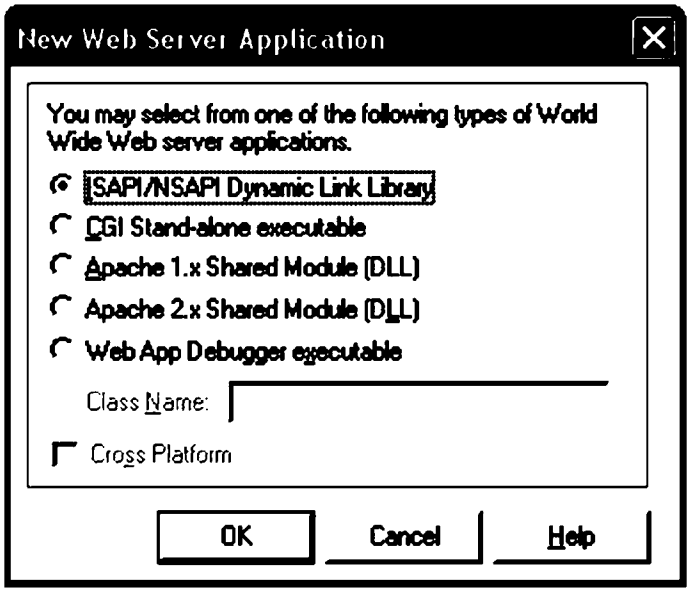

Рис. 7.1. Диалог Мастера подготовки Web-модуля

- Win-CGI Stand-along executable — создание серверного приложения в формате программы Windows, вызываемой как консольное приложение, данные передаются через специальный промежуточный файл (.INI). (Исключен из Delphi начиная с седьмой версии системы);
- Apache Shared Module — создание библиотеки (.DLL) для Web-сервера Apache, применяемой, как правило, на платформе Linux.

После этого формируется форма будущего Web-приложения, представляющая собой экземпляр класса TWebModule. Процесс разработки Web-приложения на основе этого класса можно подразделить на три этапа: выбор типа создаваемого приложения; задание действий, выполняющих обработку запросов клиента; компиляция и тестирование созданного приложения.

Компонент TWebModule является основой при разработке любых Web-приложений в Delphi. С помощью этого компонента приложение выполняет интерпретацию HTTP-запросов.

Основное свойство компонента TWebModule — свойство Action, которое содержит список действий, являющихся обработчиками запросов, поступающих от клиента. Каждый элемент этого списка имеет тип TWebActionItem и характеризуется собственными свойствами:

- property PathInfo: String — указывает, при обработке какой строки параметров (расположенной в запросе после имени сценария, но до данных запроса) будет вызваться данное действие;

- property MethodType: TMethodType — указывает метод, используемый клиентом для передачи запроса, на который данное действие может ответить. Может принимать значения: mtGet, mtPost, mtHead, mtPut, mtAny. В зависимости от значения данного свойства действие будет обрабатывать запросы, переданных соответствующим методом, либо отвечать на запрос любого вида;
- property Default: Boolean — используется для задания обработчика по умолчанию. Если данное свойство установлено равным true, то действие будет обрабатывать запросы со строками параметров, для которых не заданы обработчики;
- property Enabled: Boolean — указывает, может (true) или нет (false) данное действие обработать HTTP-запрос с параметрами PathInfo и MethodType соответствующими свойствами данного действия;
- property Produser: TCustomContentProduser — указатель на специальный компонент, используемый для формирования ответа Web-приложения.

Каждый элемент списка Actions может обрабатывать всего одно событие — OnActions. Обработчик этого события и выполняет формирование ответа серверу на принятый запрос клиента:

```
property OnAction: THTTPMethodEvent;
type THTTPMethodEvent = procedure (Sender: TObject;
 Request: TWebRequest; Response: TWebResponse;
 var Handled: Boolean) of object;
```

С помощью параметра Request передается запрос, полученный от клиента. Этот параметр является экземпляром класса TWebRequest, основными свойствами которого являются:

- property Content: String — строка параметров, переданная с помощью метода POST. Фактически это строка, содержащая тело HTTP-запроса, полученного от клиента;
- property ContentFields: TStrings — расшифрованная строка параметров, переданная с помощью метода POST. Каждый элемент коллекции ContentField представляет собой строку, соответствующую одному элементу управления, расположенному на форме, и представляет собой имя управляющего элемента и его значение, разделенные знаком равенства (идентификатор=значение);
- property Query: String — строка параметров, переданная с помощью метода GET;
- property QueryFields: TStrings — расшифрованная строка параметров, переданная с помощью метода GET. Список коллекции QueryFields полностью аналогичен формату коллекции ContentField;

- property RemoteAddr: String — IP-адрес клиента, пославшего запрос;
- property RemoteHost: String — доменное имя клиента, пославшего запрос;
- property Method: String — метод, используемый для передачи данных серверу.

Параметр Response используется для формирования ответа клиенту и представляет собой экземпляр класса TWebResponse — базового класса, предназначенного для формирования ответа на HTTP-запрос. Приведем его основные свойства:

- property Content: String — содержимое тела ответа;
- property ContentType: String — тип данных, содержащихся в теле ответа;
- property ContentLenght: Integer — число символов, содержащихся в теле ответа;
- property ContentStream: TStream — определяет объект Stream, который будет передан клиенту. Данное свойство обычно используется для передачи клиенту бинарных файлов. Если свойство ContentStream установлено, оно заменяет свойство Content.

Параметр Handled применяется в том случае, когда требуется указать, что запрос не обработан. Для этого данному параметру следует присвоить значение false.

Для обработки запросов и изменения содержимого ответа можно использовать действия, задаваемые в свойстве Action, а также события самого компонента TWebModule, задаваемые в свойстве Action, и события самого компонента TWebModule. В этом компоненте предусмотрена возможность обработки четырех событий:

- property AfterDispatch: THTTPMethodEvent — вызывается после того, как HTTP-ответ был успешно сформирован, но еще не передан клиенту. Обработчик этого события можно использовать для проверки сформированного HTTP-ответа;
- property BeforeDispatch: THTTPMethodEvent — вызывается перед тем, как будет установлено соответствие HTTP-запроса с каким-либо действием. Может использоваться для предварительной обработки HTTP-запроса;
- property OnCreate: TNotifyEvent — вызывается при создании экземпляра TWebModule. Данное событие применяется для инициализации переменных и объектов, содержащихся в приложении. Например, если модуль содержит базу данных, в обработчике этого события можно выполнить подключение базы данных;
- property OnDestroy: TNotifyEvent — вызывается перед уничтожением экземпляра TWebModule. Обычно используется для осво-

бождения объектов, созданных динамически. При работе с базами данных в обработчике этого события можно разрывать соединение с базой данных.

7.4. Средства формирования Web-ответа в формате HTML

Компонент TWebModule значительно упрощает интерпретацию данных, полученных от пользователя, однако формирование ответа осложняется написанием его вручную в формате HTML. Для тех случаев, когда необходимо выдавать сложные ответы, предусмотрен набор специальных компонентов, упрощающих формирование сложных HTML-документов.

Компоненты для формирования HTML-документов располагаются на вкладке Internet палитры компонентов. Условно их можно разделить на две группы: компоненты для генерирования HTML-документов без использования баз данных и компоненты для генерирования HTML-документов на основе данных, хранящихся в базе данных.

Первая из них включает компонент TPageProduser, представляющий собой простейший компонент для генерации HTML-документа на основе некоторого заданного шаблона. Шаблон представляет собой обычный HTML-документ, в который помимо обычных HTML-тегов могут включаться специальные теги, имеющие следующий формат:

```
<#TagName Param1=Value1 Param2=Value2 ...>.
```

Задание шаблона производится с помощью свойств:

- property HTMLDoc: TStrings — содержит код шаблона HTML-документа;
- property HTMLFile: TFileName; — задает имя файла, в котором содержится шаблон HTML-документа.

Данные свойства являются взаимоисключающими, т.е. задание значения производится только в одном из них.

Обработка документа производится с помощью метода function Content: string. Этот метод производит обработку шаблона и возвращает результат обработки в виде HTML-документа.

При обработке шаблона со специальными тегами TPageProduser генерирует событие OnHTMLTag (являющееся единственным событием данного класса):

```
type TTag = (tgCustom, tgLink, tgImage, tgTable, tgImageMap,
 tgObject, tgEmbed);
type THTMLTagEvent = procedure (Sender: TObject; Tag: TTag;
 const TagString: string; TagParams: TStrings; var ReplaceText:
 string) of object;
```

Параметр Tag может принимать одно из семи значений в зависимости от имени тега:

- tgLink — тег в формате <#LINK Dest=Value>, где параметр Dest определяет гипертекстовую ссылку;
- tgImage — тег в формате <#IMAGE Ref=Value>, где параметр Ref ссылается на графический объект;
- tgTable — тег в формате <#TABLE Param1=Value1 Param2=Value2 ... Paramn=Valuen>, где параметры определяют характер отображения таблицы;
- tgImageMap — тег в формате <#IMAGEMAP Param1=Value1 Param2=Value2 ... Paramn=Valuen>, где параметры определяют графический объект с выделенными областями;
- tgObject — тег в формате <#OBJECT Control=Value>, где параметр Control ссылается на объект ActiveX;
- tgEmbed — тег в формате <#EMBED AddIn=Value>, где параметр AddIn ссылается на встраиваемую библиотеку (.DLL);
- tgCustom — <#TagName Param1=Value1 Param2=Value2 ... Paramn=Valuen>, где TagName и все паремтры являются строками. Используется в случае определения пользовательского тега, не совпадающего ни с одним из предопределенных.

Имя тега содержится в аргументе TagString. Параметры тега передаются через аргумент TagParams в виде строк. Через аргумент ReplaceText возвращается строка, на которую должен быть заменен обрабатываемый тег.

В обработчике события OnHTMLTag определяется порядок обработки каждого специального тега и его замены на необходимое значение.

7.5. Публикация данных из приложений Web-сервера

Чаще всего информация, публикуемая на Web-сервере, хранится в базе данных. Для организации вывода информации, хранимой в базе данных, в HTML-документ в системе Delphi предусмотрено несколько специальных компонентов. Таких компонентов на вкладке Internet палитры компонентов три: TDataSetPageProducer, TDataSetTableProducer, TQueryTableProducer.

Компонент TDataSetPageProducer во многом схож с TPageProducer, но при этом обладает возможностью работы с набором данных, указываемым в свойстве DataSet. При обработке шаблона теги, имена которых совпадают с именами полей набора данных, заменяются значениями этих полей из текущей записи.

Компонент TDataSetTableProducer предназначен для вывода в окне Web-браузера информации, содержащейся в базе данных, в табличной форме. Его основными свойствами являются:

- property DataSet: TDataSet — набор данных, на основе которого будет формироваться выводимая таблица;
- property Dispatcher: IWebDispatcherAccess — компонент-диспетчер, выполняющий обработку HTTP-запросов. Обычно значение этого свойства устанавливается автоматически, в нем указывается имя компонента TWebModule, на форме которого располагается TDataSetTableProducer;
- property Caption: string — заголовок генерируемого HTML-документа;
- property CaptionAlignment: THTMLCaptionAlignment — местоположение заголовка. Может принимать следующие значения: caDefault — местоположение заголовка определяется Web-браузером; caTop — заголовок располагается над HTML-таблицей; caBottom — заголовок размещается под HTML-таблицей;
- property Header: TStrings — текст, располагаемый перед таблицей;
- property Footer: TStrings — текст, выводимый после таблицы;
- property Columns: THTMLTableColumns — поля, включаемые в формируемую таблицу, а также атрибуты отображения полей в таблице. Для установки этого свойства используется специальный редактор, открывающийся при щелчке на кнопке с многоточием в поле ввода свойства Coloumns окна инспектора объектов или при двойном щелчке на значке компонента TDataSetTableProducer, размещенного на форме;
- property MaxRows: Integer — максимальное количество строк (записей), выводимых в таблице;
- property TableAttributes: THTMLTableAttributes — атрибуты отображения таблицы. Класс THTMLTableAttributes имеет следующие свойства: Align — способ выравнивания таблицы относительно окна браузера; BgColor — цвет фона таблицы; Border — толщина линий в пикселях, разделяющих ячейки таблицы (если задано значение –1, то линии не отображаются); CellPadding — расстояние между ячейками таблицы в пикселях (если задано значение –1, то расстояние определяется Web-браузером); CellSpacing — расстояние от текста до границы ячейки в пикселях (если задано значение –1, то расстояние определяется Web-браузером); Width — ширина таблицы в процентах от ширины окна браузера;
- property RowAttributes: THTMLTableRowAttributes — атрибуты отображения строк в таблице. Класс THTMLTableRowAttributes имеет четыре свойства, два из которых, Align и BgColor, полностью аналогичны соответствующим свойствам класса THTMLTableAttributes. Оставшиеся свойства: VAlign — вертикальное вы-

равнивание текста в ячейке таблицы; Custom — дополнительные атрибуты вывода строки.

Компонент TQueryTableProducer может связываться только с набором данных TQuery и позволяет настраивать параметры заданного SQL-запроса в соответствии с полученной с его помощью строкой параметров. Данный компонент предоставляет возможность пользователю задавать критерии выборки данных, на основе которых будет формироваться HTML-таблица.

Рассмотрим пример публикации информации из базы данных на Web-сервере. Создадим проект Web-модуля (класс TWebModule) и разместим на форме компоненты TDataSetTableProducer (вкладка Internet) и TTable (вкладка BDE). В свойстве компонента TTable DatabaseName укажем DBDEMOS, в свойстве TableName — animals.dbf. Настроим теперь свойства компонента TDataSetTableProducer. В свойство Caption впишем строку «<H2>Информация о животных<H2>» (то, что будет отображаться в заголовке таблицы), в свойстве CaptionAlignment укажем caTop (в верху страницы). Свяжем эти два компонента, выбрав в свойстве DataSet компонента TDataSetTableProducer значение Table1.

После настройки свойств компонентов необходимо программно прописать логику их работы. Вначале в обработчике события создания Web-модуля нужно активизировать набор данных:

```
Table1.Open;
```

Затем нужно в обработчике события завершения Web-модуля прописать закрытие набора данных, если он активен:

```
if(Table1.Active)then
 Table1.Close;
```

Теперь нужно создать в свойстве Action новый элемент (действие, которое будет обрабатывать запрос на публикацию данных). После нужно прописать обработчик события OnAction элемента WebActionItem1. Процедура проверяет введенные пользователем логин и пароль и в случае их совпадения выводит данные:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender:
    TObject; Request: TWebRequest; Response: TWebResponse;
    var Handled: Boolean);
const
    DBlogin:string='User';
    DBpassword:string='password';
var
    login:string;
    password:string;
    begin
```

```
try
    if(Request.Method='GET')then
    begin
        login:=Request.QueryFields.Values['login'];
        password:=Request.QueryFields.Values['password'];
    end;
    if(Request.Method='POST')then
    begin
        login:=Request.ContentFields.Values['login'];
        password:=Request.ContentFields.Values['password'];
    end;
    if(DBlogin=login)and(DBpassword=password)then
        Response.Content:=
    '<HEAD><TITLE>База данных</TITLE></HEAD>'+
    DataSetTableProducer1.Content
    else
        Response.Content:=
    '<HEAD><TITLE>База данных</TITLE></HEAD>'+
    '<H2 align=center>Incorrect login/password<H2>';
except
    Response.Content:=
'<HEAD><TITLE>База данных</TITLE></HEAD>'+
'<H2 align=center>Error connect!</H2>';
end;
end;
```

После компиляции и размещения сценария на Web-сервере нужно создать страницу, которая будет вызывать информацию из базы данных, которую также нужно разместить в специальном каталоге Web-сервера:

```
<HTML>
    <HEAD>
        <TITLE>Загрузка данных о животных</TITLE>
    </HEAD>
    <BODY>
    <CENTER><B>Для загрузки данных введите пароль
    и нажмите на кнопку "Connect"</B><CENTER>
    <FORM metod="get" action="/cgi-bin/example.exe">
        <TABLE>
        <TR>
```

```
            <TD>Login:</TD>
            <TD><INPUT type="text" name="login"></TD>
        </TR>
        <TR>
            <TD>Password:</TD>
            <TD><INPUT type="password"
            name="password"></TD>
    </TR>
    </TABLE><BR><BR>
    <INPUT type="submit" value="Connect">
    </FORM>
    </BODY>
</HTML>
```

В результате выполнения этих действий после обращения к Web-серверу на экран будет выводиться окно запроса, показанное на рис. 7.2.

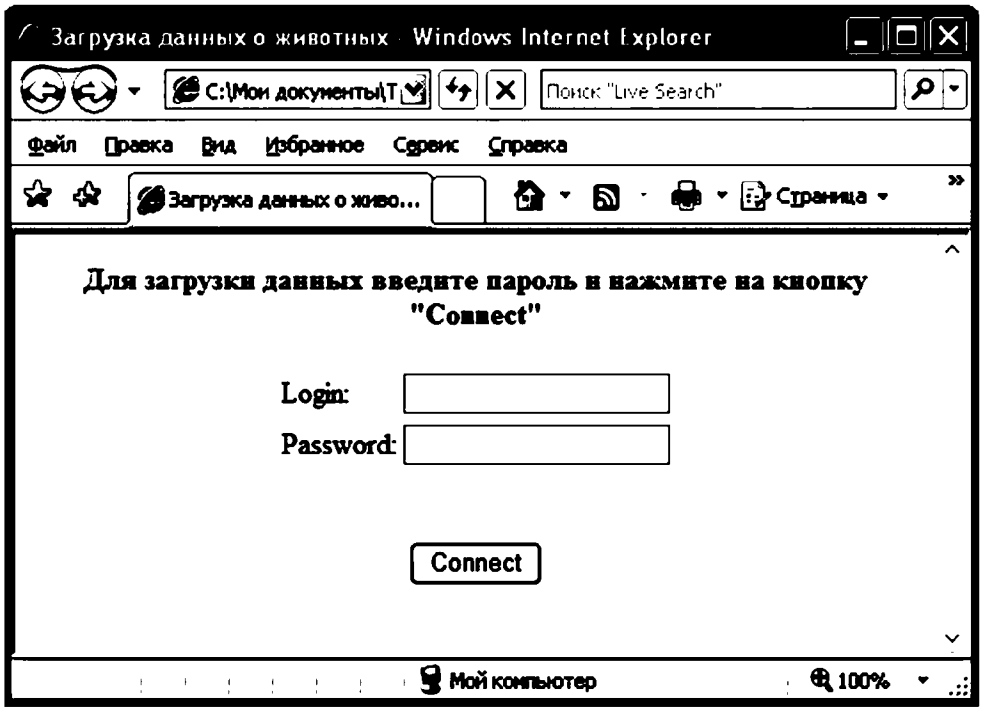

Рис. 7.2. Окно запроса на публикацию данных

В случае правильного ввода логина и пароля на вновь загруженной странице отобразится содержимое таблицы animals.dbf базы данных DBDEMOS.

Вопросы для самопроверки

1. В чем состоит особенность разработки расширений Web-серверов?
2. Перечислите основные переменные окружения, используемые для передачи информации Web-сервером при выполнении CGI-сценария?
3. Каковы основные средства разработки Web-приложений системы Delphi?
4. Перечислите компоненты Delphi формирования HTML-документов.
5. Опишите назначение и порядок использования компонентов TDataSetPageProducer, TDataSetTableProducer, TQueryTableProducer.

Тесты

1. Какой символ разделяет разные параметры в строке параметров, формируемой сервером?
   А. &.
   Б. =.
   В. %.
   Г. +.

2. Какая переменная окружения содержит строку параметров при ее передаче методом GET?
   А. QUERY_STRING.
   Б. SCRIPT_NAME.
   В. CONTENT_TYPE.
   Г. CONTENT_LENGTH.

3. Какой тип Web-приложения при его проектировании средствами Delphi нужно выбрать для создания серверного консольного приложения?
   А. ISAPI/NSAPI Dynamic Link Library.
   Б. CGI Stand-along executable.
   В. Win-CGI Stand-along executable;
   Г. Apache Shared Module.

4. Какой компонент предназначен для вывода в окне Web-браузера информации, содержащейся в базе данных, в табличной форме?
   А. TDataSetPageProducer.
   Б. TPageProducer.
   В. TDataSetTableProducer.
   Г. TQueryTableProducer.

5. Какие из перечисленных компонентов позволяют генерировать HTML-документ на основе некоторого заданного шаблона?
   А. TDataSetPageProducer.
   Б. TPageProducer.
   В. TDataSetTableProducer.
   Г. TQueryTableProducer.

ЗАКЛЮЧЕНИЕ

Учебная дисциплина «Программное обеспечение компьютерных сетей» является в соответствии с образовательным стандартом специальной дисциплиной и логическим продолжением общепрофессиональной дисциплины «Компьютерные сети» и связана с изучением основных видов и методов разработки программных продуктов, используемых в компьютерных сетях.

В результате изучения дисциплины студент должен:

1) иметь представление о значении и областях применения данной дисциплины;

2) знать:

- основные понятия и определения;
- структуру технологии «клиент–сервер»;
- конструкции и особенности применения языков гипертекстовой разметки (SGML, HTML, XML);
- типы серверов приложений и прикладные протоколы;
- инструментальные средства создания серверной части сетевых приложений (CGI, ISAPI, ASP);
- принципы построения и основные задачи, выполняемые серверными программами;
- инструментальные средства создания клиентской части сетевых приложений (DHTML, JavaScript);

3) уметь:

- применять языки гипертекстовой разметки документов;
- осуществлять проектирование сетевых приложений;
- создавать серверные и клиентские части сетевых приложений с использованием современных систем разработки.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Анкудинов Г.И.* Сети ЭВМ и телекоммуникации. Архитектура и сетевые технологии [Текст]: учебное пособие / Г.И. Анкудинов, И.Г. Анкудинов, А.И. Стрижаченко. — СПб., 2006.
2. *Бобровский С.И.* Технологии Delphi 2006. Новые возможности [Текст] / С.И. Бобровский. — СПб., 2006.
3. *Гук М.* Аппаратные средства локальных сетей [Текст]: энциклопедия / М. Гук. — СПб., 2015.
4. *Избачков Ю.С.* Информационные системы [Текст]: учебник для вузов / Ю.С. Избачков, В.И. Петров. — СПб., 2006.
5. *Исаченко О.В.* Введение в информационные технологии [Текст] / О.В. Исаченко. — Ростов н/Д., 2009.
6. *Кириленко А.* Самоучитель HTML [Текст] / А. Кириленко. — СПб., 2005.
7. *Максимов Н.В.* Компьютерные сети [Текст]: учебное пособие для студентов учреждений среднего профессионального образования / Н.В. Максимов, И.И. Попов. — М., 2010.
8. *Мержевич В.В.* HTML и CSS на примерах [Текст] / В.В. Мержевич. — СПб., 2005.
9. *Моримото Р.* Microsoft Windows Server 2012. Полное руководство [Текст] / Р. Моримото, К. Гардиньер, Ноэл, О. Драуби, Э. Аббейт [и др.]. — М., 2013.
10. *Мухамедзянов Р.Р.* Java. Серверные приложения [Текст] / Р.Р. Мухамедзянов. — М., 2010.
11. *Олифер В.Г.* Компьютерные сети. Принципы, технологии, протоколы [Текст]: учебник для вузов / В.Г. Олифер, Н.А. Олифер. — СПб., 2016.
12. *Стауфер Т.* Создание веб-страниц. Самоучитель [Текст] / Т. Стауфер. — СПб., 2003.
13. *Таненбаум Э.* Компьютерные сети [Текст] / Э. Таненбаум. — СПб., 2016.
14. *Шапиро Д.* Windows Server 2003, Библия пользователя [Текст] / Д. Шапиро, Д. Бойс. — М., 2006.

ОТВЕТЫ НА ТЕСТЫ

Глава \ Воорос	№ 1	№ 2	№ 3	№ 4	№ 5
Технология «клиент–сервер»	А, Б, Г	А, Г	Б	А, Б, Г	В
Сетевые операционные системы	А	В	Б	Г	Г
Прогрвммное обеспечение Web-систем	А	Б	В	Б	Г
Web-приложения и методы их разработки	А, Б	В, Г	Б	А	Б
Средства разработки Web-приложений	А, Б, Г	А	А	Г	В
Разработка Web-приложений	А	В	А	В	А
Создание приложений Web-сервера	А	А	Б	В	А, Б

ОСНОВНЫЕ ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Компьютерная сеть — это совокупность компьютеров, соединенных линиями связи, образованными кабелями, сетевыми адаптерами и другими коммуникационными устройствами.

Программные клиенты — модули, установленные на компьютерах-клиентах, предназначенные для получения доступа к ресурсам компьютера-сервера.

Программные серверы — модули, установленные на компьютерах-серверах, предназначенные для обслуживания запросов на доступ к ресурсам компьютера-сервера.

Программы-клиенты — предназначены для интерпретации разметки на языке HTML, интерпретации встроенных в HTML программ на одном из командных языков Web и т.п.

Программы-серверы — программы, принимающие запросы от Web-клиентов и отвечающие на них.

Программы подготовки публикаций — предназначены для создания и редактирования HTML-документов.

ASP (Active Server Pages) — активные серверные страницы, сценарий ASP-страницы формирует не полный HTML-документ, а лишь его часть, добавляемую к исходному документу, из которого производится вызов сценария.

CGI (Common Gateway Interface) — средство расширения возможностей Web-технологии путем написания прикладных пользовательских программ (CGI-скриптов).

CGI-скрипт — программа, написанная в соответствии с CGI на одном из языков программирования (C, Perl, Pascal и пр.) или командном языке ОС.

HTML (HyperText Markup Language) — язык гипертекстовой разметки документов.

HTTP (HyperText Transfer Protocol) — протокол обмена гипертекстовой информацией.

ISAPI (Internet Server Application Programming Interface) — прикладной программный интерфейс для интернет-серверов, представляет собой динамическую библиотеку (DLL, Data Link Library), загружаемую как программный поток, принадлежащий Web-серверу.

SGML (Standard Generalized Markup Language) — стандартный общий язык разметки, который был утвержден ISO в 1980-х гг. в качестве стандарта.

URL (Universal Resource Locator, uniform resource identificator) — универсальный способ адресации ресурсов в сети.

Web-клиент — удаленный компьютер Web-системы, использующий ресурсы другого компьютера, называемого сервером.

Web-сервер — компьютер, предназначенный для обработки клиентских запросов, дальнейшей их пересылки и формирования на основе запросов обращений к внешним источникам данных.

XML (Extensible Markup Language) — язык разметки, описывающий целый класс объектов данных, называемых XML-документами, используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов.

ОГЛАВЛЕНИЕ

По вопросам приобретения книг обращайтесь:
Отдел продаж «ИНФРА-М» (оптовая продажа):
127214, Москва, ул. Полярная, д. 31В, стр.1
Тел. (495) 280-33-86 (доб. 218, 222)
E-mail: bookware@infra-m.ru

•

Отдел «Книга—почтой»:
тел. (495) 280-33-86 (доб. 222)

ФЗ № 436-ФЗ	Издание не подлежит маркировке в соответствии с п. 1 ч. 4 ст. 11

Учебное издание

Исаченко Олег Вячеславович

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

УЧЕБНОЕ ПОСОБИЕ

Оригинал-макет подготовлен в НИЦ ИНФРА-М

ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29
E-mail: books@infra-m.ru http://www.infra-m.ru

Подписано в печать 09.09.2020.
Формат 60×90/16. Бумага офсетная. Гарнитура Petersburg.
Печать цифровая. Усл. печ. л. 9,88.
ППТ100. Заказ № 00000
ТК 156650-1189344-291119

Отпечатано в типографии ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29